

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

R. Sekar Arun K. Pujari (Eds.)

Information Systems Security

4th International Conference, ICISS 2008
Hyderabad, India, December 16-20, 2008
Proceedings

Volume Editors

R. Sekar

Stony Brook University, Department of Computer Science

Stony Brook, NY 11794-4400, USA

E-mail: sekar@cs.sunysb.edu

Arun K. Pujari

University of Hyderabad, Department of Computer and Information Sciences

Hyderabad 500 046, India

E-mail: akpcs@uohyd.ernet.in

Library of Congress Control Number: 2008940485

CR Subject Classification (1998): C.2, E.3, D.4.6, K.6.5, K.4.4, H.2

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743

ISBN-10 3-540-89861-1 Springer Berlin Heidelberg New York

ISBN-13 978-3-540-89861-0 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2008

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12577231 06/3180 5 4 3 2 1 0

Preface

The 4th International Conference on Information System Security (ICISS 2007) was held December 16–20, 2008 at the Jawaharlal Nehru Technological University (JNTU) in Hyderabad, India. Although this conference is held in India, it is a decidedly international conference, attracting papers from all around the world. This year, there were 81 submissions from 18 different countries. The final program contained papers from Australia, Austria, France, Germany, India, Poland, UK, and USA.

From the 81 submissions, the Program Committee accepted 15 full papers, 4 short papers, and 2 ongoing research reports. The accepted papers span a wide range of topics, including access control, cryptography, forensics, formal methods and language-based security, intrusion detection, malware defense, network and Web security, operating system security, and privacy.

The conference featured four keynote talks, with written papers accompanying most of them. We would like to thank the speakers Somesh Jha, Basant Rajan, Amit Sahai, and Dawn Song for accepting our invitation to deliver keynote talks at this year's conference. The conference was preceded by two days of tutorials.

We would like to thank JNTU for hosting the conference, and EasyChair (<http://www.easychair.org/>) for providing conference management services to handle the paper review and selection process. Lastly, we wish to express our deepest thanks to the members of the Program Committee who give their personal free time to perform the often thankless job of reviewing many papers under extremely short deadlines, and to the external reviewers, volunteers and local assistants who made this program a success.

December 2008

R. Sekar
Arun Pujari

Organization

Advisory Committee Chair	Sushil Jajodia George Mason University, USA
Program Co-chairs	Arun K. Pujari LNM IIT, India R. Sekar Stony Brook University, USA
Submissions and Website Co-chairs	Alok Tongaonkar Stony Brook University, USA Subrat Kumar Dash University of Hyderabad, India
Tutorial Co-chairs	Vineet P. Nair University of Hyderabad, India V.N. Venkatakrishnan University of Illinois at Chicago, USA
Organizing Chair	G. Vijaya Kumari JNTU, Hyderabad, India
Finance Chair	P. Radha Krishnan Infosys, Hyderabad, India

Steering Committee

Sushil Jajodia	George Mason University, USA, Chair
Vijay Atluri	Rutgers University, USA
Aditya Bagchi	Indian Statistical Institute, India
A.K. Chakrabarti	Dept. of IT, Govt. of India
Prem Chand	Mahindra British Telecom, India
Shyam K. Gupta	IIT Delhi, India
Arun K. Majumdar	IIT Kharagpur, India
Chandan Mazumdar	Jadavpur University, Kolkata, India
Patrick McDaniel	Pennsylvania State University, USA
Arun K. Pujari	LNMITT Jaipur, India
Pierangela Samarati	University of Milan, Italy
R. Sekar	Stony Brook University, USA
N. Sitaram	CAIR, India
Vijay Varadharajan	Maquarie University, Australia
N. Vijayaditya	Controller of Certifying Authorities, India

Program Committee

Vijay Atluri	Rutgers University, USA
Aditya Bagchi	Indian Statistical Institute, India
Kevin Butler	Pennsylvania State University, USA
Chakravarthy Bhagvat	University of Hyderabad, India
Lorenzo Cavallaro	University of California, Santa Barbara, USA
Hao Chen	University of California, Davis, USA
Shuo Chen	Microsoft Research, Redmond, USA
Marc Dacier	Symantec Research Labs Europe
Saumya Debray	University of Arizona, USA
Subrat Kumar Dash	University of Hyderabad, India
Vinod Ganapathy	Rutgers University, USA
Jonathon Giffin	Georgia Institute of Technology, USA
S.K. Gupta	Indian Institute of Technology, Delhi, India
Sushil Jajodia	George Mason University, USA
Somesh Jha	University of Wisconsin, Madison, USA
Shrikant Ojha	DRDO, Government of India, India
Engin Kirda	Technical University Vienna, Austria
P. Radha Krishna	Infosys Technologies Ltd., Hyderabad, India
G. Vijaya Kumari	JNTU, Hyderabad, India
Zhenkai Liang	National University of Singapore, Singapore
Arun K. Majumdar	Indian Institute of Technology, Kharagpur, India
Anish Mathuria	DA-IICT, Gandhinagar, India
Chandan Mazumdar	Jadhavpur University, Kolkata, India
Nasir Memon	Polytechnic University, Brooklyn, USA
Peng Ning	North Carolina State University, USA
Atul Prakash	University of Michigan, USA
R. Ramanujam	Institute of Mathematical Sciences, Chennai, India
Sanjay Rawat	Infosys Technologies Ltd., India
Pierangela Samarati	University of Milan, Italy
Amitabh Saxena	University of Trento, Italy
Radu Sion	Stony Brook University, USA
Anil Somayaji	Carleton University, Canada
Robin Sommer	ICSI/LBNL, USA
Gaurav Tandon	Florida Institute of Technology, USA
Vijay Varadharajan	Macquarie University, Australia
V.N. Venkatakrishnan	University of Illinois at Chicago, USA
Poorvi Vora	George Washington University, USA
Xiaofeng Wang	Indiana University
Wei Xu	VMWare, Inc., USA

External Reviewers

Rajesh Asthana
Giuseppe Ateniese
A. Baskar
Sevinc Bayram
Kevin Borders
Liang Cai
Manik Lal Das
Anupam Datta
Saumya Debray
Ahmet Dirik
William Enck
Laura Falk
Matt Fredrickson
Romesh Kumar Gambhir
Sanchit Gupta
Francis Hsu
Ken Ingham
Bharat Lal Jangid
Dr. Shri Kant
Stefan Katzenbeisser
Corrado Leita
Zhuowei Li
Subhamoy Maitra
Mandar Mitra

Thomas Moyer
Divya Muthukumaran
Anandrabratha Pal
Van Hau Pham
Stefan Popoveniuc
Arun Pujari
C.R. Ramakrishnan
A.V. Sarad
Amitabh Saxena
A. Prasad Sistla
Scott Stoller
S.P. Suresh
Yagiz Sutcu
Olivier Thonnard
Umut Topkara
Gene Tsudik
Uday Tupakula
Matthias Vallentin
Matthew Van Gundy
Kaushik Veeraraghavan
Rui Wang
Komminist Weldemariam
Haizhi Xu

Table of Contents

Keynote Address

BitBlaze: A New Approach to Computer Security via Binary Analysis	1
<i>Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena</i>	

Languages and Formal Methods

On the Decidability of Model-Checking Information Flow Properties	26
<i>Deepak D'Souza, Raveendra Holla, Janardhan Kulkarni, Raghavendra K. Ramesh, and Barbara Sprick</i>	
Synthesizing Enforcement Monitors wrt. the Safety-Progress Classification of Properties	41
<i>Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier</i>	
Implicit Flows: Can't Live with 'Em, Can't Live without 'Em	56
<i>Dave King, Boniface Hicks, Michael Hicks, and Trent Jaeger</i>	

Protocols

A Robust Reputation Scheme for Decentralized Group Management Systems	71
<i>Frédéric Cuppens, Nora Cuppens-Boulahia, and Julien A. Thomas</i>	
Complexity of Checking Freshness of Cryptographic Protocols	86
<i>Zhiyao Liang and Rakesh M. Verma</i>	
Secure Internet Voting Based on Paper Ballots	102
<i>Lukasz Nitschke</i>	

Short Papers

A Secure Round-Based Timestamping Scheme with Absolute Timestamps	116
<i>Duc-Phong Le, Alexis Bonnecaze, and Alban Gabillon</i>	

A Framework for Trustworthy Service-Oriented Computing	124
<i>Sasikanth Avancha</i>	
Revisiting Bluetooth Security	132
<i>Manik Lal Das and Ravi Mukkamala</i>	
A Verification Framework for Temporal RBAC with Role Hierarchy	140
<i>Samrat Mondal and Shamik Sural</i>	

Keynote Address

Computing on Encrypted Data (Extended Abstract)	148
<i>Amit Sahai</i>	

Ongoing Research

Identification of Cryptographically Strong and Weak Pseudorandom Bit Generators	154
<i>Neelam Verma, Prasanna R. Mishra, and Gireesh Pandey</i>	
Proxy Re-signature Schemes	156
<i>N.R. Sunitha and B.B. Amberker</i>	

Keynote Address

Fast Signature Matching Using Extended Finite Automaton (XFA)	158
<i>R. Smith, C. Estan, S. Jha, and I. Siahhan</i>	

Intrusion Detection

Real-Time Alert Correlation with Type Graphs	173
<i>Gianni Tedesco and Uwe Aickelin</i>	
Incorporation of Application Layer Protocol Syntax into Anomaly Detection	188
<i>Patrick Düssel, Christian Gehl, Pavel Laskov, and Konrad Rieck</i>	
A Parallel Architecture for Stateful, High-Speed Intrusion Detection	203
<i>Luca Foschini, Ashish V. Thapliyal, Lorenzo Cavallaro, Christopher Kruegel, and Giovanni Vigna</i>	

Biometrics, Forensics and Steganography

Indexing Multimodal Biometric Databases Using Kd-Tree with Feature Level Fusion	221
<i>Umarani Jayaraman, Surya Prakash, and Phalguni Gupta</i>	

Audio Watermarking Based on Quantization in Wavelet Domain	235
<i>Vivekananda Bhat K., Indranil Sengupta, and Abhijit Das</i>	
Overwriting Hard Drive Data: The Great Wiping Controversy	243
<i>Craig Wright, Dave Kleiman, and Shyaam Sundhar R.S.</i>	

Practical Applications

Optimizing the Block Cipher and Modes of Operations Overhead at the Link Layer Security Framework in the Wireless Sensor Networks ...	258
<i>Devesh Jinwala, Dhiren Patel, and Kankar Dasgupta</i>	
Privacy Management for Facebook	273
<i>Enkh-Amgalan Baatarjav, Ram Dantu, and Santi Phithakkitnukoon</i>	
HyDRo – Hybrid Development of Roles	287
<i>Ludwig Fuchs and Günther Pernul</i>	

Keynote Address

The Enlightened Era of Enterprise Security	303
<i>Basant Rajan</i>	
Author Index	307

BitBlaze: A New Approach to Computer Security via Binary Analysis

Dawn Song¹, David Brumley², Heng Yin^{1,2,3}, Juan Caballero^{1,2}, Ivan Jager²,
Min Gyung Kang^{1,2}, Zhenkai Liang², James Newsome², Pongsin Poosankam^{1,2},
and Prateek Saxena¹

¹ UC Berkeley

² Carnegie Mellon University

³ College of William and Mary

Abstract. In this paper, we give an overview of the BitBlaze project, a new approach to computer security via binary analysis. In particular, BitBlaze focuses on building a unified binary analysis platform and using it to provide novel solutions to a broad spectrum of different security problems. The binary analysis platform is designed to enable accurate analysis, provide an extensible architecture, and combines static and dynamic analysis as well as program verification techniques to satisfy the common needs of security applications. By extracting security-related properties from binary programs directly, BitBlaze enables a principled, root-cause based approach to computer security, offering novel and effective solutions, as demonstrated with over a dozen different security applications.

Keywords: Binary analysis, malware analysis and defense, vulnerability analysis and defense, reverse engineering.

1 Introduction

In BitBlaze, we propose a new approach to computer security via binary analysis. In particular, we make the simple and yet important observation that for many security problems, their root cause and the key to their solutions lie directly in the relevant programs (e.g., vulnerable programs and malicious code). Thus, by designing and developing techniques and tools to automatically extract security-related properties from programs and devise solutions based on them, we can enable a principled approach to many security problems, focusing on their root cause, and offer more effective solutions than previous approaches which rely on heuristics or symptoms of an attack.

To enable the above approach, one technical challenge is that for security applications, we often need to directly deal with binary code. For many programs such as common off-the-shelf (COTS) programs, users do not have access to their source code. For malicious code attacks, attackers simply do not attach the source code with the attack. And binary code is what gets executed, and hence analyzing binary code will give the ground truth important for security applications whereas analyzing source code may give the wrong results due to compiler errors and optimizations. However, analyzing binary code is commonly considered as an extremely challenging task due to its complexity and the lack of higher-level semantic information. As a result, very few tools

exist for binary analysis that are powerful enough for general security applications, and this has been a big hurdle preventing security researchers and practitioners from taking the aforementioned root-cause based approach.

Thus, the above observations have motivated us to conduct the *BitBlaze* project, building a unified binary analysis platform and using it to provide novel solutions to a broad spectrum of different security problems. In this paper, we give an overview of the two main research foci of BitBlaze: (1) the design and development of a unified, extensible binary analysis infrastructure for security applications; (2) novel solutions to address a spectrum of different security problems by taking a principled, root-cause based approach enabled by our binary analysis infrastructure.

In the rest of this paper, we describe the challenges and design rationale behind the BitBlaze Binary Analysis Platform and its overall architecture, and then describe its three main components *Vine*, *TEMU*, and *Rudder*. Finally, we give an overview of the different security applications that we have enabled using the BitBlaze Binary Analysis Platform and discuss some related work.

2 The Architecture of the BitBlaze Binary Analysis Platform

In this section, we first describe the challenges of binary analysis for security applications, then the desired properties of a binary analysis platform catering to security applications, and finally outline the architecture of the BitBlaze Binary Analysis Platform.

2.1 Challenges

There are several main challenges for binary code analysis, some of which are specific to security applications.

Complexity. The first major challenge for binary analysis is that binary code is complex. Binary analysis needs to model this complexity accurately in order for the analysis itself to be accurate. However, the sheer number and complexity of instructions in modern architectures makes accurate modeling a significant challenge. Popular modern architectures typically have hundreds of different instructions, with new ones added at each processor revision. Further, each instruction can have complex semantics, such as single instruction loops, instructions which behave differently based upon their operand values, and implicit side effects such as setting processor flags. For example, the IA-32 manuals describing the semantics of x86 weigh over 11 pounds.

As an example, consider the problem of determining the control flow in the following x86 assembly program:

```
// instruction dst, src
add a, b    // a = a+b
shl a, x    // a << x
jz target   // jump if zero to address target
```

The first instruction, `add a, b`, computes `a := a+b`. The second instruction, `shl a, x`, computes `a := a << x`. The last instruction, `jz a`, jumps to address `a` if the processor zero flag is set.

One problem is that both the `add` and `shl` instruction have implicit side effects. Both instructions calculate up to *six* other bits of information that are stored as processor status flags. In particular, they calculate whether the result is zero, the parity of the result, whether there is an auxiliary carry, whether the result is signed, and whether an overflow has occurred.

Conditional control flow, such as the `jz` instruction, is determined by the implicitly calculated processor flags. Thus, either the `add` instruction calculates the zero flag, or the `shl` will. However, which instruction, `add` or `shl`, determines whether the branch is taken? Answering this question is not straight-forward. The `shl` instruction behaves *differently* depending upon the operands: it only updates the zero flag if `x` is not zero.

Lack of Higher-Level Semantics. The second major challenge is that binary code is different than source code, and in particular, lacking higher-level semantics present in source code. Thus, we need to adapt and develop program analysis techniques and tools that are suitable for the setting of binary code (where debugging information is often unavailable). In particular, binary code lacks abstractions that are often fundamental to source code and source code analysis, as shown in the following examples:

- **No Functions.** The function abstraction does not exist at the binary level. Instead, control flow in a binary program is performed by jumps. For example, the x86 instruction `call x` is just shorthand for storing the current instruction pointer (register `eip`) at the address named by the register `esp`, decrementing `esp` by the architecture word size, then loading the `eip` with number `x`. Indeed, it is perfectly valid in assembly, and sometimes happens in practice, that code may call into the middle of a “function”, or have a single “function” separated into non-contiguous pieces.
- **Memory vs. Buffers.** Binary code does not have buffers, it has *memory*. While the OS may determine a particular memory page is not valid, memory does not have the semantics of a user-specified type and size. One implication of the difference between buffers and memory is that in binary code there is no such thing as a buffer overflow. While we may say a particular store violates a higher-level semantics given by the source code, such facts are inferences with respect to the higher-level semantics, not part of the binary code itself.
- **No Types.** New types cannot be created or used since there is no such thing as a type constructor in binary code. The only types available are those provided by the hardware: registers and memory. Even register types are not necessarily informative, since it is common to store values from one register type (e.g., 32-bit register) and read them as another (e.g., 8-bit register).

Overall, an assembly-specific approach is unattractive because writing analysis over modern complex instruction tends to be tedious and error-prone. Verifying a program analysis is correct over such a large and complicated instruction set seems even more difficult. Further, an assembly-specific approach is specific to a single architecture. All analysis would have to be ported each time we want to consider a new architecture. Thus, analysis could not take advantage of the common semantics across many different assembly languages.

Whole-System View. Many security applications requires the ability to analyze operations in the operating system kernel and interactions between multiple processes, thus require a whole-system view, presenting greater challenges than in traditional single-program analysis.

Code Obfuscation. Some security applications require analyzing malicious code. Malicious code may employ anti-analysis techniques such as code packing, encryption, and obfuscation to make program analysis difficult, posing greater challenges than analyzing benign programs.

2.2 Design Rationale

The goal of the BitBlaze Binary Analysis Platform is to design and develop techniques and the core utilities that cater the common needs of security applications and enable others to build upon and develop new solutions to security problems more easily and effectively. Given the aforementioned challenges, we have a few design guidelines motivating the architecture of the BitBlaze Binary Analysis Platform:

Accuracy. We would like to enable accurate analysis, motivating us to build precise, formal models of instructions that allow the tool to accurately model the program execution behavior symbolically.

Extensibility. Given the complexity of binary analysis, we would like to develop core utilities which can then be re-used and easily extended to enable other more sophisticated analysis on binaries, or easily re-targeted to different architectures.

Fusion of Static and Dynamic Analysis. Static and dynamic analysis both have advantages and disadvantages. Static analysis can give more complete results as it covers different execution paths, however, it may be difficult due to the complexity of pointer aliasing, the prevalence of indirect jumps, and the lack of types and other higher-level abstractions in binaries. Even telling what is code and what is data statically is an undecidable problem in general. Moreover, it is particularly challenging for static analysis to deal with dynamically generated code and other anti-static-analysis techniques employed in malicious code. Furthermore, certain instructions such as kernel and floating point instructions may be extremely challenging to accurately model. On the other hand, dynamic analysis naturally avoid many of the difficulties that static analysis needs to face, at the cost of analyzing one path at a time. Thus, we would like to combine static and dynamic analysis whenever possible to have the benefits of both.

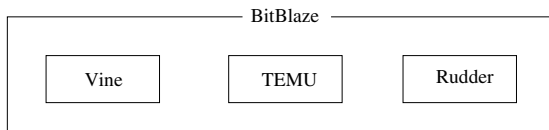


Fig. 1. The BitBlaze Binary Analysis Platform Overview

2.3 Architecture

Motivated by the aforementioned challenges and design rationale, the BitBlaze Binary Analysis Platform is composed of three components: *Vine*, the static analysis component, *TEMU*, the dynamic analysis component, and *Rudder*, the mixed concrete and symbolic analysis component combining dynamic and static analysis, as shown in Figure 1.

Vine translates assembly to a simple, formally specified intermediate language (IL) and provides a set of core utilities for common static analysis on the IL, such as control flow, data flow, optimization, symbolic execution, and weakest precondition calculation.

TEMU performs whole-system dynamic analysis, enabling whole-system fine-grained monitoring and dynamic binary instrumentation. It provides a set of core utilities for extracting OS-level semantics, user-defined dynamic taint analysis, and a clean plug-in interface for user-defined activities.

Rudder uses the core functionalities provided by Vine and TEMU to enable mixed concrete and symbolic execution at the binary level. For a given program execution path, it identifies the symbolic path predicates that symbolic inputs need to satisfy to follow the program path. By querying solvers such as decision procedures, it can determine whether the path is feasible and what inputs could lead the program execution to follow the given path. Thus, Rudder can automatically generate inputs leading program execution down different paths, exploring different parts of the program execution space. Rudder provides a set of core utilities and interfaces enabling users to snapshot and reload the exploration state and provide user-specified path selection policies.

3 Vine: The Static Analysis Component

In this section, we give an overview of Vine, the static analysis component of BitBlaze Binary Analysis Platform, describing its intermediate language (IL), its front end and back end components, and implementation.

3.1 Vine Overview

Figure 2 shows a high-level picture of Vine. The Vine static analysis component is divided into a platform-specific front-end and a platform-independent back-end. At the core of Vine is a platform-independent intermediate language (IL) for assembly. The IL is designed as a small and formally specified language that faithfully represents the assembly languages. Assembly instructions in the underlying architecture are lifted up to the Vine IL via the Vine front-end. All back-end analyses are performed on the platform-independent IL. Thus, program analyses can be written in an architecture-independent fashion and do not need to directly deal with the complexity of an instruction set such as x86. This design also provides extensibility—users can easily write their own analysis on the IL by building on top of the core utilities provided in Vine.

The Vine front-end currently supports translating x86 [23] and ARMv4 [4] to the IL. It uses a set of third-party libraries to parse different binary formats and produce assembly. The assembly is then translated into the Vine IL in a syntax-directed manner.

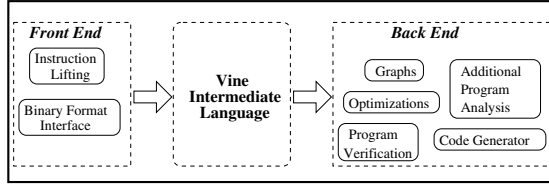


Fig. 2. Vine Overview

The Vine back-end supports a variety of core program analysis utilities. The back-end has utilities for creating a variety of different graphs, such as control flow and program dependence graphs. The back-end also provides an optimization framework. The optimization framework is usually used to simplify a specific set of instructions. We also provide program verification capabilities such as symbolic execution, calculating weakest preconditions, and interfacing with decision procedures. Vine can also write out lifted Vine instructions as valid C code via the code generator back-end.

To combine static and dynamic analysis, we also provide an interface for Vine to read an execution trace generated by a dynamic analysis component such as TEMU. The execution trace can be lifted to the IL for various further analysis.

3.2 The Vine Intermediate Language

The Vine IL is the target language during lifting, as well as the analysis language for back-end program analysis. The semantics of the IL are designed to be faithful to assembly languages. Table 1 shows the Vine IL.

The base types in the Vine IL are 1, 8, 16, 32, and 64-bit registers (i.e., n -bit vectors) and memories. A memory type is qualified by its endianness, which can be either `little` (e.g., for little-endian architectures like x86), `big` (e.g., for big-endian architectures such as PowerPC), or `norm` for *normalized memory* (explained later in this section). A memory type is also qualified by the index type, which must be a register type. For example `mem_t(little, reg32_t)` denotes a memory type which is little endian and is addressed by 32-bit numbers.

There are three types of values in Vine. First, Vine has numbers n of type τ_{reg} . Second, Vine has memory values $\{n_{a1} \rightarrow n_{v1}, n_{a2} \rightarrow n_{v2}, \dots\}$, where n_{ai} denotes a number used as an address, and n_{vi} denotes the value stored at the address. Finally, Vine has a distinguished value \perp . \perp values are not exposed to the user and cannot be constructed in the presentation language. \perp is used internally to indicate a failed execution.

Expressions in Vine are side-effect free. The Vine IL has binary operations \diamond_b (“&” and “|” are bit-wise), unary operations \diamond_u , constants, `let` bindings, and casting. Casting is used when the semantics requires a change in the width of a value. For example, the lower 8 bits of `eax` in x86 are known as `al`. When lifting x86 instructions, we use casting to project out the lower-bits of the corresponding `eax` register variable to an `al` register variable when `al` is accessed.

Table 1. The Vine Intermediate Language (Since ‘|’ is an operator, for clarity we use commas to separate all operator elements in the productions for \Diamond_b and \Diamond_u .)

<i>program</i>	::= <i>decl</i> * <i>instr</i> *
<i>instr</i>	::= <i>var</i> = <i>exp</i> <i>jmp exp</i> <i>cjmp exp,exp,exp</i> <i>halt exp</i> <i>assert exp</i> <i>label integer</i> <i>special id_s</i>
<i>exp</i>	::= <i>load(exp, exp, τ_{reg})</i> <i>store(exp, exp, exp, τ_{reg})</i> <i>exp \Diamond_b exp</i> <i>\Diamond_u exp</i> <i>const</i> <i>var</i> <i>let var = exp in exp</i> <i>cast(cast_kind, τ_{reg}, exp)</i>
<i>cast_kind</i>	::= <i>unsigned</i> <i>signed</i> <i>high</i> <i>low</i>
<i>decl</i>	::= <i>var var</i>
<i>var</i>	::= (<i>string</i> , <i>id_v</i> , τ)
\Diamond_b	::= +, −, *, /, / _s , mod, mod _s , <<, >>, >> _a , &, , ⊕, ==, ≠, <, ≤, < _s , ≤ _s
\Diamond_u	::= − (unary minus), ! (bit-wise not)
<i>value</i>	::= <i>const</i> { <i>n_{a1}</i> → <i>n_{v1}</i> , <i>n_{a2}</i> → <i>n_{v2}</i> , ... } : τ_{mem} \perp
<i>const</i>	::= <i>n</i> : τ_{reg}
τ	::= τ_{reg} τ_{mem} Bot Unit
τ_{reg}	::= <i>reg1_t</i> <i>reg8_t</i> <i>reg16_t</i> <i>reg32_t</i> <i>reg64_t</i>
τ_{mem}	::= <i>mem_t</i> (τ_{endian} , τ_{reg})
τ_{endian}	::= <i>little</i> <i>big</i> <i>norm</i>

In Vine, both *load* and *store* operations are pure. While *load* is normally pure, the semantics of *store* are often not. Each *store* expression must specify which memory to load or store from. The resulting memory is returned as a value. For example, a Vine store operation is written *mem1 = store(mem0, a, y)*, where *mem1* is the same as *mem0* except address *a* has value *y*. The advantage of pure memory operations in Vine notation is that it makes it possible to syntactically distinguish what memory is modified or read. One place where we take advantage of this is in computing Single Static Assignment (SSA) where both scalars and memory have a unique single static assignment location.

A program in Vine is a sequence of variable declarations, followed by a sequence of instructions. There are 7 different kinds of instructions. The language has assignments, jumps, conditional jumps, and labels. The target of all jumps and conditional jumps must be a valid label in our operational semantics, else the program halts with \perp . Note that a jump to an undefined location (e.g., a location that was not disassembled such as to dynamically generated code) results in the Vine program halting with \perp . A program can halt normally at any time by issuing the *halt* statement. We also provide *assert*, which acts similar to a C *assert*: the asserted expression must be true, else the machine halts with \perp .

A *special* in Vine corresponds to a call to an externally defined procedure or function. The *id* of a *special* indexes what kind of special, e.g., what system call. The

```

// x86 instr dst,src
1. mov [eax], 0xaabbccdd
2. mov ebx, eax
3. add ebx, 0x3
4. mov eax, 0x1122
5. mov [ebx], ax
6. sub ebx, 1
7. mov ax, [ebx]

```

(a)

address	memory	address	memory
eax	0xdd	eax	0xdd
eax+1	0xcc	eax+1	0xcc
eax+2	0xbb	eax+2	0xbb
eax+3	0xaa	eax+3	0x22
eax+4		eax+4	0x11

(b)

(c)

Fig. 3. An example of little-endian stores as found in x86 that partially overlap. (b) shows memory after executing line 1, and (c) shows memory after executing line 5. Line 7 will load the value 0x22bb.

semantics of `special` is up to the analysis; its operational semantics are not defined. We include `special` as an instruction type to explicitly distinguish when such calls may occur that alter the soundness of an analysis. A typical approach to dealing with `special` is to replace `special` with an analysis-specific summary function written in the Vine IL that is appropriate for the analysis.

Normalized Memory

The endianness of a machine is usually specified by the byte-ordering of the hardware. A little endian architecture puts the low-order byte first, and a big-endian architecture puts the high-order byte first. x86 is an example of a little endian architecture, and PowerPC is an example of a big endian architecture.

We must take endianness into account when analyzing memory accesses. Consider the assembly in Figure 3a. The `mov` operation on line 2 writes 4 bytes to memory in little endian order (since x86 is little endian). After executing line 2, the address given by `eax` contains byte 0xdd, `eax+1` contains byte 0xcc, and so on, as shown in Figure 3b. Lines 2 and 3 set `ebx = eax+2`. Line 4 and 5 write the 16-bit value 0x1122 to `ebx`. An analysis of these few lines of code needs to consider that the write on line 4 overwrites the last byte written on line 1, as shown in Figure 3c. Considering such cases requires additional logic in each analysis. For example, the value loaded on line 7 will contain one byte from each of the two stores.

We say a memory is *normalized* for a b -byte addressable memory if all loads and stores are exactly b -bytes and b -byte aligned. For example, in x86 memory is byte addressable, so a normalized memory for x86 has all loads and stores at the byte level. The normalized form for the write on Line 1 of Figure 3a in Vine is shown in Figure 4. Note the subsequent load on line 7 are with respect to the current memory `mem6`.

Normalized memory makes writing program analyses involving memory easier. Analyses are easier because normalized memory syntactically exposes memory updates that are otherwise implicitly defined by the endianness. The Vine back-end provides utilities for normalizing all memory operations.

```

1. mem4 = let mem1 = store(mem0, eax, 0xdd, reg8_t) in
    let mem2 = store(mem1, eax+1, 0xcc, reg8_t) in
    let mem3 = store(mem2, eax+2, 0xbb, reg8_t) in
    store(mem3, eax+3, 0xcc, reg8_t);
...
5. mem6 = let mem5 = store(mem4, ebx, 0x22, reg8_t) in
    store(mem5, ebx+1, 0x22, reg8_t)
...
7. value = let b1 = load(mem6, ebx, reg8_t) in
    let b2 = load(mem6, ebx+1, reg8_t) in
    let b1' = cast(unsigned, b1, reg16_t) in
    let b2' = cast(unsigned, b2, reg16_t) in
    (b2' << 8) | b1';

```

Fig. 4. Vine normalized version of the store and load from Figure 3a

3.3 The Vine Front-End

The Vine front-end is responsible for translating binary code to the Vine IL. In addition, the front-end interfaces with libraries such as the GNU Binary File Descriptor (libbfd) library for parsing the low-level details of binary files.

Translating binary code to the IL consists of three steps:

- **Step 1.** First the binary file is disassembled. Vine currently interfaces with three disassemblers: IDA Pro [19], a commercial disassembler, a research disassembler from Kruegel *et al.* [26] that can disassemble x86 obfuscated code, and our own linear-sweep disassembler built on top of GNU libopcodes. Interfacing with other disassemblers is straightforward.
- **Step 2.** The disassembly is passed to VEX, a third-party library which turns assembly instructions into the VEX intermediate language. The VEX IL is part of the Valgrind dynamic instrumentation tool [31]. The VEX IL is also similar to a RISC-based language. As a result, the lifted IL has only a few instruction types, similar to Vine. However, the VEX IL itself is inconvenient for performing program analysis because its information about side effects of instructions such as what EFLAGS are set by x86 instructions is implicit. This step is mainly performed in order to simplify the development of Vine: we let the existing tool take care of the task of reducing assembly instructions to a basic IL, then in step 3 expose all side-effects so that the analysis is faithful.
- **Step 3.** We translate the VEX IL to Vine. The resulting Vine IL is intended to be faithful to the semantics of the disassembled assembly instructions.

Translated assembly instructions have all side-effects explicitly exposed as Vine instructions. As a result, a single typical assembly instruction will be translated as a sequence of Vine instructions. For example, the `add eax, 0x2` x86 instruction is translated as the following instructions:

```

tmp1 = EAX; EAX = EAX + 2;
//eflags calculation
CF:reg1_t = (EAX<tmp1);
tmp2 = cast(low, EAX, reg8_t);
PF = (!cast(low,
              (((tmp2>>7)^(tmp2>>6))^(tmp2>>5)^(tmp2>>4)))^
              (((tmp2>>3)^(tmp2>>2))^(tmp2>>1)^(tmp2))))), reg1_t);
AF = (1==(16&(EAX^(tmp1^2))));
ZF = (EAX==0);
SF = (1==(1&(EAX>>31)));
OF = (1==(1&(((tmp1^(2^0xFFFFFFFF))&(tmp1^EAX))>>31)));

```

The translated instructions expose all the side-effects of the `add` instruction, including all 6 `eflags` that are updated by the operation. As another example, an instruction with the `rep` prefix is translated as a sequence of instructions that form a loop.

In addition to binary files, Vine can also translate an instruction trace to the IL. Conditional branches in a trace are lifted as `assert` statements to check that the executed branch is followed. This is done to prevent branching outside the trace to an unknown instruction. Vine and TEMU are co-designed so that TEMU currently generates traces in a trace format that Vine can read.

3.4 The Vine Back-End

In the Vine back-end, new program analyses are written over the Vine IL. Vine provides a library of common analyses and utilities which serve as building blocks for more advanced analyses. Below we provide an overview of some of the utilities and analyses provided in the Vine back-end.

Evaluator. Vine has an evaluator which implements the operational semantics of the Vine IL. The evaluator allows us to execute programs without recompiling the IL back down to assembly. For example, we can test a raised Vine IL for an instruction trace produced by an input by evaluating the IL on that input and verifying we end in the same state.

Graphs. Vine provides routines for building and manipulating control flow graphs (CFG), including a pretty-printer for the graphviz DOT graph language [2]. Vine also provides utilities for building data dependence and program dependence graphs [30].

One issue when constructing a CFG of an assembly program is determining the successors of jumps to computed values, called *indirect* jumps. Resolving indirect jumps usually requires program analyses that require a CFG, e.g., Value Set Analysis (VSA) [5]. Thus, there is a potential circular dependency. Note that an indirect jump may potentially go anywhere, including the heap or code that has not been previously disassembled.

Our solution is to designate a special node as a successor of unresolved indirect jump targets in the CFG. We provide this so an analysis that depends on a correct CFG can recognize that we do not know the subsequent state. For example, a data-flow analysis could widen all facts to the lattice bottom. Most normal analyses will first run an indirect jump resolution analysis in order to build a more precise CFG that resolves indirect jumps to a list of possible jump targets. Vine provides one such analysis, VSA [5].

Single Static Assignment. Vine supports conversion to and from single static assignment (SSA) form [30]. SSA form makes writing analysis easier because every variable is defined statically only once. We convert both memory and scalars to SSA form. We convert memories because then one can syntactically distinguish between memories before and after a write operation instead of requiring the analysis itself to maintain similar bookkeeping. For example, in the memory normalization example in Figure 3.2, an analysis can syntactically distinguish between the memory state before the write on line 1, the write on line 5, and the read on line 7.

Chopping. Given a source and sink node, a program chop [24] is a graph showing the statements that cause definitions of the source to affect uses of the sink. For example, chopping can be used to restrict subsequent analysis to only a portion of code relevant to a given source and sink instead of the whole program.

Data-flow and Optimizations. Vine provides a generic data-flow engine that works on user-defined lattices. Vine also implements several data-flow analysis. Vine currently implements Simpson's global value numbering [37], constant propagation and folding [30], dead-code elimination [30], live-variable analysis [30], integer range analysis, and Value set analysis (VSA) [5]. VSA is a data-flow analysis that over-approximates the values for each variable at each program point. Value-set analysis can be used to help resolve indirect jumps. It can also be used as an alias analysis. Two memory accesses are potentially aliased if the intersection of their value sets is non-empty.

Optimizations are useful for simplifying or speeding up subsequent analysis. For example, we have found that the time for the decision procedure STP to return a satisfying answer for a query can be cut in half by using program optimization to simplify the query first [9].

C Code Generator. Vine can generate valid C code from the IL. For example, one could use Vine as a rudimentary decompiler by first raising assembly to Vine, then writing it out as valid C. The ability to export to C also provides a way to compile Vine programs: the IL is written as C, then compiled with a C compiler.

The C code generator implements memories in the IL as arrays. A `store` operation is a store on the array, and a `load` is a load from the array. Thus, C-generated code simulates real memory. For example, consider a program vulnerable to a buffer overflow attack is raised to Vine, then written as C and recompiled. An out-of-bound write on the original program will be simulated in the corresponding C array, but will not lead to a real buffer overflow.

Program Verification Analyses. Vine currently supports formal program verification in two ways. First, Vine can convert the IL into Dijkstra's Guarded Command Language (GCL), and calculate the weakest precondition with respect to GCL programs [20]. The weakest precondition for a program with respect to a predicate q is the most general condition such that any input satisfying the condition is guaranteed to terminate (normally) in a state satisfying q . Currently we only support acyclic programs, i.e., we do not support GCL `while`.

Vine also interfaces with decision procedures. Vine can write out expressions (e.g., weakest preconditions) in CVC Lite syntax [1], which is supported by several decision procedures. In addition, Vine interfaces directly with the STP [22] decision procedure through calls from Vine to the STP library.

3.5 Implementation of Vine

The Vine infrastructure is implemented in C++ and OCaml. The front-end lifting is implemented primarily in C++, and consists of about 17,200 lines of code. The back-end is implemented in OCaml, and consists of about 40,000 lines of code. We interface the C++ front-end with the OCaml back-end using OCaml via IDL generated stubs.

The front-end interfaces with Valgrind’s VEX [31] to help lift instructions, GNU BFD for parsing executable objects, and GNU libopcodes for pretty-printing the disassembly.

The implemented Vine IL has several constructors in addition to the instructions in Figure 1:

- The Vine IL has a constructor for comments. We use the comment constructor to pretty-print each disassembled instruction before the IL, as well as a place-holder for user-defined comments.
- The Vine IL supports variable scoping via blocks. Vine provides routines to de-scope Vine programs via α -varying as needed.
- The Vine IL has constructs for qualifying statements and types with user-defined attributes. This is added to help facilitate certain kinds of analysis such as taint-based analysis.

4 TEMU: The Dynamic Analysis Component

In this section, we give an overview of TEMU, the dynamic analysis component of BitBlaze Binary Analysis Platform, describing its components for extracting OS-level semantics, performing whole-system dynamic taint analysis, its Plugins and implementation.

4.1 TEMU Overview

TEMU is a whole-system dynamic binary analysis platform that we developed on the basis of a whole-system emulator, QEMU [36]. We run an entire system, including the operating system and applications in this emulator, and observe in a fine-grained manner how the binary code of interest is executed. The whole-system approach in TEMU is motivated by several considerations:

- Many analyses require fine-grained instrumentation (i.e., at instruction level) on binary code. By dynamically translating the emulated code, the whole-system emulator enables fine-grained instrumentation.
- A whole-system emulator presents us a whole-system view. The whole-system view enables us to analyze the operating system kernel and interactions between multiple processes. In contrast, many other binary analysis tools (e.g., Valgrind,

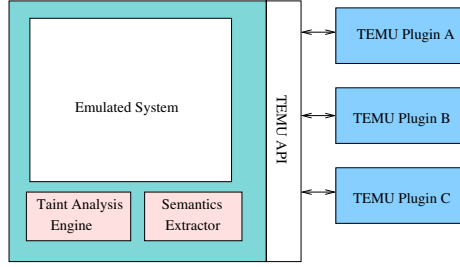


Fig. 5. TEMU Overview

DynamoRIO, Pin) only provide a local view (i.e., a view of a single user-mode process). This is particularly important for analyzing malicious code, because many attacks involve multiple processes, and kernel attacks such as rootkits have become increasingly popular.

- A whole-system emulator provides an excellent isolation between the analysis components and the code under analysis. As a result, it is more difficult for the code under analysis to interfere with analysis results.

The design of TEMU is motivated by several challenges and considerations:

- The whole-system emulator only provides us only the hardware-level view of the emulated system, whereas we need a software-level view to get meaningful analysis results. Therefore, we need a mechanism that can extract the OS-level semantics from the emulated system. For example, we need to know what process is currently running and what module an instruction comes from.
- In addition, many analyses require reasoning about how specific data depends on its data sources and how it propagates throughout the system. We enable this using whole-system dynamic taint analysis.
- We need to provide a well-designed programming interface (i.e., API) for users to implement their own plugins on TEMU to perform their customized analysis. Such an interface can hide unnecessary details from users and reuse the common functionalities.

With these considerations in mind, we have designed the architecture of TEMU, as shown in Figure 5. We build the *semantics extractor* to extract OS-level semantics information from the emulated system. We build the *taint analysis engine* to perform dynamic taint analysis. We define and implement an interface (i.e, TEMU API) for users to easily implement their own analysis modules (i.e. TEMU plugins). These modules can be loaded and unloaded at runtime to perform designated analyses. We implemented TEMU in Linux, and at the time of writing, TEMU can be used to analyze binary code in Windows 2000, Windows XP, and Linux systems. Below we describe these three components respectively.

4.2 Semantics Extractor

The semantics extractor is responsible for extracting OS-level semantics information of the emulated system, including process, module, thread, and symbol information.

Process and Module Information. For the current execution instruction, we need to know which process, thread and module this instruction comes from. In some cases, instructions may be dynamically generated and executed on the heap.

Maintaining a mapping between addresses in memory and modules requires information from the guest operating system. We use two different approaches to extract process and module information for Windows and Linux.

For Windows, we have developed a kernel module called *module notifier*. We load this module into the guest operating system to collect the updated memory map information. The module notifier registers two callback routines. The first callback routine is invoked whenever a process is created or deleted. The second callback routine is called whenever a new module is loaded and gathers the address range in the virtual memory that the new module occupies. In addition, the module notifier obtains the value of the CR3 register for each process. As the CR3 register contains the physical address of the page table of the current process, it is different (and unique) for each process. All the information described above is passed on to TEMU through a predefined I/O port.

For Linux, we can directly read process and module information from outside, because we know the relevant kernel data structures, and the addresses of relevant symbols are also exported in the `system.map` file. In order to maintain the process and module information during execution, we hook several kernel functions, such as `do_fork` and `do_exec`.

Thread Information. For windows, we also obtain the current thread information to support analysis of multi-threaded applications and the OS kernel. It is fairly straightforward, because the data structure of the current thread is mapped into a well-known virtual address in Windows. Currently, we do not obtain thread information for Linux and may implement it in future versions.

Symbol Information. For PE (Windows) binaries, we also parse their PE headers and extract the exported symbol names and offsets. After we determine the locations of all modules, we can determine the absolute address of each symbol by adding the base address of the module and its offset. This feature is very useful, because all windows APIs and kernel APIs are exported by their hosting modules. The symbol information conveys important semantics information, because from a function name, we are able to determine what purpose this function is used for, what input arguments it takes, and what output arguments and return value it generates. Moreover, the symbol information makes it more convenient to hook a function—instead of giving the actual address of a function, we can specify its module name and function name. Then TEMU will automatically map the actual address of the function for the user.

Currently, this feature is only available for PE binaries. Support for ELF (Linux) binaries will be available in future versions.

4.3 Taint Analysis Engine

Our dynamic taint analysis is similar in spirit to a number of previous systems [16, 35, 18, 38, 17]. However, since our goal is to support a broad spectrum of different applications, our design and implementation is the most complete. For example, previous approaches either operate on a single process only [17, 35, 38], or they cannot deal with memory swapping and disks [16, 18].

Shadow Memory. We use a shadow memory to store the taint status of each byte of the physical memory, CPU registers, the hard disk and the network interface buffer. Each tainted byte is associated with a small data structure storing the original source of the taint and some other book keeping information that a TEMU plugin wants to maintain. The shadow memory is organized in a page-table-like structure to ensure efficient memory usage. By using shadow memory for the hard disks, the system can continue to track the tainted data that has been swapped out, and also track the tainted data that has been saved to a file and is then read back in.

Taint Sources. A TEMU plugin is responsible for introducing taint sources into the system. TEMU supports taint input from hardware, such as the keyboard, network interface, and hard disk. TEMU also supports tainting a high-level abstract data object (e.g. the output of a function call, or a data structure in a specific application or the OS kernel).

Taint Propagation. After a data source is tainted, the taint analysis engine monitors each CPU instruction and DMA operation that manipulates this data in order to determine how the taint propagates. The taint analysis engine propagates taint through data movement instructions, DMA operations, arithmetic operations, and table lookups. Considering that some instructions (e.g., `xor eax, eax`) always produce the same results, independent of the values of their operands, the taint analysis engine does not propagate taint in these instructions.

Note that TEMU plugins may employ very different taint policies, according to their application requirements. For example, for some applications, we do not need to propagate taint through table lookups. For some applications, we want to propagate taint through an immediate operand, if the code region occupied by it is tainted. Therefore, during taint propagation, the taint analysis engine lets TEMU plugins determine how they want to propagate taint into the destination.

This design provides valuable flexibility to TEMU plugins. They can specify different taint sources, maintain an arbitrary record for each tainted byte, keep track of multiple taint sources, and employ various taint policies.

4.4 TEMU API and Plugins

In order for users to make use of the functionalities provided by TEMU, we define a set of functions and callbacks. By using this interface, users can implement their own plugins and load them into TEMU at runtime to perform analysis. Currently, TEMU provides the following functionalities:

- Query and set the value of a memory cell or a CPU register.
- Query and set the taint information of memory or registers.
- Register a hook to a function at its entry and exit, and remove a hook. TEMU plugins can use this interface to monitor both user and kernel functions.
- Query OS-level semantics information, such as the current process, module, and thread.
- Save and load the emulated system state. This interface helps to switch between different machine states for more efficient analysis. For example, this interface makes multiple path exploration more efficient, because we can save a state for a specific branch point and explore one path, and then load this state to explore the other path without restarting the program execution.

TEMU defines callbacks for various events, including (1) the entry and exit of a basic block; (2) the entry and exit of an instruction; (3) when taint is propagating; (4) when a memory is read or write; (5) when a register is read or written to; (6) hardware events such as network and disk inputs and outputs.

Quite a few TEMU plugins have been implemented by using these functions and callbacks. These plugins include:

- Panorama [43]: a plugin that performs OS-aware whole-system taint analysis to detect and analyze malicious code’s information processing behavior.
- HookFinder [42]: a plugin that performs fine-grained impact analysis (a variant of taint analysis) to detect and analyze malware’s hooking behavior.
- Renovo [25]: a plugin that extracts unpacked code from packed executables.
- Polyglot [14]: a plugin that make use of dynamic taint analysis to extract protocol message format.
- Tracecap: a plugin that records an instruction trace with taint information for a process or the OS kernel.
- MineSweeper [10]: a plugin that identifies and uncovers trigger-based behaviors in malware by performing online symbolic execution.
- BitScope: a more generic plugin that make use of symbolic execution to perform in-depth analysis of malware.
- HookScout: a plugin that infers kernel data structures.

4.5 Implementation of TEMU

The TEMU infrastructure is implemented in C and C++. In general, performance-critical code is implemented in C due to efficiency of C, whereas analysis-oriented code is written in C++ to leverage the abstract data types in STL and stronger type checking in C++. For example, the taint analysis engine insert code snippets into QEMU micro operations to check and propagate taint information. Since taint analysis is performance critical, we implemented it in C. On the other hand, we implemented the semantics extractor in C++ using `string`, `list`, `map` and other abstract data types in STL, to maintain a mapping between OS-level view and hardware view. The TEMU API is defined in C. This gives flexibility to users to implement their plugin in either C, C++, or both. The TEMU core consists of about 37,000 lines of code, excluding the code originally from QEMU (about 306,000 lines of code). TEMU plugins consist of about 134,000 lines of code.

5 Rudder: The Mixed Concrete and Symbolic Execution Component

In this section, we give an overview of Rudder, the mixed concrete and symbolic execution component of BitBlaze Binary Analysis Platform, describing its components for performing mixed execution and exploring program execution space and its implementation.

5.1 System Overview

We have designed and developed *Rudder*, to perform mixed concrete and symbolic execution at the binary level. Given a binary program and a specification of symbolic inputs, Rudder performs mixed concrete and symbolic execution and explores multiple execution paths whenever the path conditions are dependent on symbolic inputs. By doing so, Rudder is able to automatically uncover hidden behaviors that only exhibit under certain conditions.

Figure 6 shows a high level picture of Rudder. Rudder consists of the following components: the *mixed execution engine* that performs mixed concrete and symbolic execution, the *path selector* that prioritizes and determines the execution paths, and the *solver* that performs reasoning on symbolic path predicates and determines if a path is feasible. Rudder takes as inputs a binary program and a symbolic input specification. In TEMU, the binary program is executed and monitored. Rudder runs as a TEMU plugin to instrument the execution of the binary program. During the execution, Rudder marks some of the inputs as symbolic according to the symbolic input specification. Then the mixed execution engine symbolically executes the operations on symbolic inputs and data calculated from symbolic inputs. When a symbolic value is used in a branch condition, the path selector determines, with assistance of the solver, which branches are feasible and selects a branch to explore.

5.2 Mixed Execution Engine

Determine Whether to Symbolically Execution an Instruction. For each instruction, the mixed execution engine performs the following steps. First, it checks the source operands of that instruction, and answers whether they are concrete or symbolic. If all source operands are concrete, this instruction will be executed concretely on the emulated CPU. Otherwise, the mixed execution engine marks the destination operand as symbolic, and calculate symbolic expressions for the destination operand. To mark

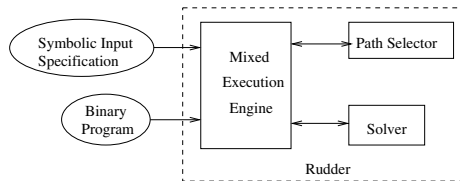


Fig. 6. Rudder Overview

and propagate symbolic data, the mixed execution engine relies on the dynamic taint analysis functionality provided by TEMU.

Formulate a Symbolic Program. Ideally, we would like to calculate symbolic expressions on the fly. However, this naive approach would incur significant performance overhead at runtime. Considering that many expressions would not be used in path predicates, we take a “lazy” approach. The basic idea is to collect necessary information in the symbolic machine during the symbolic execution. At a later time, when some symbolic variables are used in path predicates, we can extract the corresponding symbolic expressions from the symbolic machine.

More specifically, we enqueue this instruction, along with the relevant register and memory states into our symbolic machine. That is, if an instruction to be symbolically executed has any concrete operands, we must update those concrete values inside the symbolic machine. In the case of registers, this is trivial – we simply update their concrete values in the symbolic machine. Memory accesses with concrete addresses are handled similarly. However, we also have to deal with memory accesses where the memory address itself is symbolic, which is described below.

A symbolic memory address means that the data specifying which memory is to be read or written is symbolic. If this address is symbolic, we do not know which memory location is about to be accessed, because a symbolic address may potentially take any concrete value. To solve this problem, we use the solver to determine the range of possible values of this address. In some cases, the range that the solver returns is too large to be effective. In this case, we add a constraint to the system to limit its size, therefore limiting the complexity that is introduced. In practice, we found that most symbolic memory accesses are already constrained to small ranges, making this unnecessary. For example, consider code that iterates over an array. Each access to the array is bounded by the constraints imposed by the iteration itself. Note that this is a conservative approach, meaning that all solutions found are still correct. Once a range is selected, we update the symbolic machine with the necessary information.

We leverage the functionality of Vine to perform symbolic execution. That is, when symbolically executing an instruction, we lift it into Vine IL and formulate a symbolic program in form of Vine IL.

Extract Symbolic Expressions. Given the symbolic program and a symbolic variable of interest, we can extract a symbolic expression for it. Extracting a symbolic expression takes several steps. First, we perform dynamic slicing on the symbolic program. This step removes the instructions that the symbol does not depend upon. After this step, the resulted symbolic program is reduced drastically. Then we generate one expression by substituting intermediate symbols with their right-hand-side expressions. Finally, we perform constant folding and other optimizations to further simplify the expression.

5.3 Path Selector

When a branch condition becomes symbolic, the path selector is consulted to determine which branch to explore. There is an interface for users to supply their own path selection priority function. As we usually need to explore as many paths that depend upon symbolic inputs as possible, the default is a breadth-first search approach.

To make the path exploration more efficient, we make use of the functionality of state saving and restoring provided by TEMU. That is, when a symbolic conditional branch is first encountered, the path selector saves the current execution state, and determines which feasible direction to explore. Later, when it decides to explore a different direction from this branch, the path selector restores the execution state on this branch and explores the other branch.

5.4 Solver

The solver is a theorem prover or decision procedure, which performs reasoning on symbolic expressions. In Rudder, the solver is used to determine if a path predicate is satisfiable, and to determine the range of the memory region with a symbolic address. We can make use of any appropriate decision procedures that are available. Thus, if there is any new progress on decision procedures, we can benefit from it. Currently in our implementation, we use STP as the solver [21].

5.5 Implementation of Rudder

Rudder is implemented in C, C++ and Ocaml. The implementation consists of about 3,600 lines of C/C++ code and 2,600 lines of Ocaml code. The C/C++ code is mainly for implementing a TEMU plugin, and marking and tracking symbolic values, and the Ocaml code is used to interface with Vine and perform symbolic analysis. We also have an offline mode for Rudder where we take an execution trace and then perform symbolic execution on the trace to obtain the path predicate and then solve for inputs for different branches to be taken.

6 Security Applications

In this section, we give an overview of the different security applications that we have enabled using the BitBlaze Binary Analysis Platform, ranging from automatic vulnerability detection, diagnosis, and defense, to automatic malware analysis and defense, to automatic model extraction and reverse engineering. For each security application, we give a new formulation of the problem based on the root cause in the relevant program. We then demonstrate that this problem formulation leads us to new approaches to address the security application based on the root cause. The results demonstrate the utility and effectiveness of the BitBlaze Binary Analysis Platform and its vision—it was relatively easy to build different applications on top of the BitBlaze Binary Analysis Platform and we could obtain effective results that previous approaches could not.

6.1 Vulnerability Detection, Diagnosis, and Defense

Sting: An Automatic Defense System against Zero-Day Attacks. Worms such as CodeRed and SQL Slammer exploit software vulnerabilities to self-propagate. They can compromise millions of hosts within hours or even minutes and have caused billions of dollars in estimated damage. How can we design and develop effective defense mechanisms against such fast, large scale worm attacks?

We have designed and developed Sting [40,33], a new end-to-end automatic defense system that aims to be effective against even zero-day exploits and protect vulnerable hosts and networks against fast worm attacks. Sting uses dynamic taint analysis to detect exploits to previously unknown vulnerabilities [35] and can automatically generate filters for dynamic instrumentation to protect vulnerable hosts [34].

Automatic Generation of Vulnerability Signatures. Input-based filters (a.k.a. signatures) provide important defenses against exploits before the vulnerable hosts can be patched. Thus, to automatically generate effective input-based filters in a timely fashion is an important task. We have designed and developed novel techniques to automatically generate accurate input-based filters based on information about the vulnerability instead of the exploits, and thus generating filters that have zero false positives and can be effective against different variants of the exploits [11, 13].

Automatic Patch-based Exploit Generation. Security patches do not only fix security vulnerabilities, they also contain sensitive information about the vulnerability that could enable an attacker to exploit the original vulnerable program and lead to severe consequences. We have demonstrated that this observation is correct—we have developed new techniques showing that given the patched version and the original vulnerable program, we can automatically generate exploits in our experiments with real world patches (often in minutes) [12]. This opens the research direction of how to design and develop a secure patch dissemination scheme where attacker cannot use information in the patch to attack vulnerable hosts before they have a chance to download and apply the patch.

6.2 Malware Analysis and Defense

Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. A myriad of malware such as keyloggers, Browser-helper Objects (BHO) based spyware, rootkits, and backdoors accesses and leaks users' sensitive information and breaches users' privacy. Can we have a unified approach to identify such privacy-breaching malware despite their widely-varied appearance? We have designed and developed Panorama [43] as a unified approach to detect privacy-breaching malware using whole-system dynamic taint analysis.

Renovo: Hidden Code Extraction from Packed Executables. Code packing is one technique commonly used to hinder malware code analysis through reverse engineering. Even though this problem has been previously researched, the existing solutions are either unable to handle novel packers, or vulnerable to various evasion techniques. We have designed and developed Renovo [25], as a fully dynamic approach for hidden code extraction, capturing an intrinsic characteristic of hidden code execution.

HookFinder: Identifying and Understanding Malware Hooking Behavior. One important malware attacking vector is its hooking mechanism. Malicious programs implant hooks for many different purposes. Spyware may implant hooks to be notified of the arrival of new sensitive data. Rootkits may implant hooks to intercept and tamper with critical system information to conceal their presence in the system. A stealth

backdoor may also place hooks on the network stack to establish a stealthy communication channel with remote attackers. We have designed and developed HookFinder [42] to automatically detect and analyze malware's hooking behaviors, by performing fine-grained impact analysis. Since this technique captures the intrinsic nature of hooking behaviors, it is well suited for identifying new hooking mechanisms.

BitScope: Automatically Dissecting Malware. The ability to automatically dissect a malicious binary and extract information from it is an important cornerstone for system forensic analysis and system defense. Malicious binaries, also called malware, include denial of service attack tools, spamming systems, worms, and botnets. New malware samples are uncovered daily through widely deployed honeypots/honeyfarms, forensic analysis of compromised systems, and through underground channels. As a result of the break-neck speed of malware development and recovery, automated analysis of malicious programs has become necessary in order to create effective defenses.

We have designed and developed BitScope, an architecture for systematically uncovering potentially hidden functionality of malicious software [8]. BitScope takes as input a malicious binary, and outputs information about its execution paths. This information can then be used by supplemental analysis designed to answer specific questions, such as what behavior the malware exhibits, what inputs activate interesting behavior, and the dependencies between its inputs and outputs.

6.3 Automatic Model Extraction and Analysis

Polyglot: Automatic Extraction of Protocol Message Format. Protocol reverse engineering, the process of extracting the application-level protocol used by an implementation (without access to the protocol specification) is important for many network security applications. Currently, protocol reverse engineering is mostly manual. For example, it took the open source Samba project over the course of 10 years to reverse engineer SMB, the protocol Microsoft Windows uses for sharing files and printers [39].

We have proposed that binary program analysis can be used to aid automatic protocol reverse engineering [15]. The central intuition is that the binary itself encodes the protocol, thus binary analysis should be a significant aide in extracting the protocol from the binary itself.

Automatic Deviation Detection. Many network protocols and services have several different implementations. Due to coding errors and protocol specification ambiguities, these implementations often contain deviations, i.e., differences in how they check and process some of their inputs. Automatically identifying deviations can enable the automatic detection of potential implementation errors and the automatic generation of fingerprints that can be used to distinguish among implementations of the same network service. The main challenge in this project is to automatically find deviations (without requiring access to source code). We have developed techniques for taking two binary implementations of the same protocol and automatically generating inputs which cause deviations [7].

Replayer: Sound Replay of Application Dialogue. The ability to accurately replay application protocol dialogs is useful in many security-oriented applications, such as

replaying an exploit for forensic analysis or demonstrating an exploit to a third party. A central challenge in application dialog replay is that the dialog intended for the original host will likely not be accepted by another without modification. For example, the dialog may include or rely on state specific to the original host such as its host name or a known cookie. In such cases, a straight-forward byte-by-byte replay to a different host with a different state (e.g., different host name) than the original dialog participant will likely fail. These state-dependent protocol fields must be updated to reflect the different state of the different host for replay to succeed. We have proposed the first approach for soundly replaying application dialog where replay succeeds whenever the analysis yields an answer [32].

7 Related Work

In this section, we briefly describe some related work on other static and dynamic binary analysis platforms.

Static Binary Analysis Platforms. There are several static binary analysis platforms, however they are not sufficient for our purposes mainly because they were motivated by different applications and hence did not need to satisfy the same requirements.

Phoenix is a compiler program analysis environment developed by Microsoft [28]. One of the Phoenix tools allows code to be raised up to a register transfer language (RTL). A RTL is a low-level IR that resembles an architecture-neutral assembly. Phoenix differs from Vine in several ways. First, Phoenix can only lift code produced by a Microsoft compiler. Second, Phoenix requires debugging information, thus is not a true binary-only analysis platform. Third, Phoenix lifts assembly to a low-level IR that does not expose the semantics of complicated instructions, e.g., register status flags, as part of the IR [29]. Fourth, the semantics of the lifted IR, as well as the lifting semantics and goals, are not well specified [29], thus not suitable for our research purposes.

The CodeSurfer/x86 platform [6] is a proprietary platform for analyzing x86 programs. However, it was mainly designed for other applications such as slicing and does not provide some of the capabilities that we need.

Dynamic Binary Analysis Platforms. Tools like DynamoRIO [3], Pin [27], and Valgrind [41] support fine-grained instrumentation of a user-level program. They all provide a well-defined interface for users to implement plugins. However, as they can only instrument a single user-level process, they are unsuitable to analyze the operating system kernel and applications that involve multiple processes. In addition, these tools reside in the same execution environment with the program under instrumentation. Some of them even share the same memory space with the program under instrumentation, and change the memory layout of the program. In consequence, the analysis results may be affected. In contrast, TEMU provides a whole-system view, enabling analysis of the OS kernel and multiple processes. Moreover, as it resides completely out of the analyzed execution environment, TEMU can provide more faithful analysis results.

8 Conclusion

In this paper, we have described the BitBlaze project, its binary analysis platform and applications to a broad spectrum of different security applications. Through this project, we have demonstrated that our binary analysis platform provides a powerful arsenal that enables us to take a principled, root-cause based approach to a broad spectrum of security problems. We hope BitBlaze's extensible architecture will enable others to build upon and enable new solutions to other applications.

Acknowledgements

We would like to thank Cody Hartwig, Xeno Kovah, Eric Li, and other colleagues and collaborators for their help and support to the project.

References

1. CVC Lite documentation (Page checked 7/26/2008), <http://www.cs.nyu.edu/acsys/cvcl/doc/>
2. The DOT language (Page checked 7/26/2008), <http://www.graphviz.org/doc/info/lang.html>
3. On the run - building dynamic modifiers for optimization, detection, and security. Original DynamoRIO announcement via PLDI tutorial (June 2002)
4. ARM. ARM Architecture Reference Manual (2005) Doc. No. DDI-0100I
5. Balakrishnan, G.: WYSINWYX: What You See Is Not What You eXecute. PhD thesis, Computer Science Department, University of Wisconsin at Madison (August 2007)
6. Balakrishnan, G., Gruian, R., Repts, T., Teitelbaum, T.: Codesurfer/x86 - a platform for analyzing x86 executables. In: Proceedings of the International Conference on Compiler Construction (April 2005)
7. Brumley, D., Caballero, J., Liang, Z., Newsome, J., Song, D.: Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation. In: Proceedings of the USENIX Security Symposium, Boston, MA (August 2007)
8. Brumley, D., Hartwig, C., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Song, D.: Bitscope: Automatically dissecting malicious binaries. Technical Report CS-07-133, School of Computer Science, Carnegie Mellon University (March 2007)
9. Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Poosankam, P., Song, D., Yin, H.: Automatically identifying trigger-based behavior in malware. In: Lee, W., Wang, C., Dagon, D. (eds.) Botnet Detection. Countering the Largest Security Threat Series: Advances in Information Security, vol. 36, Springer, Heidelberg (2008)
10. Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Song, D., Yin, H.: Towards automatically identifying trigger-based behavior in malware using symbolic execution and binary analysis. Technical Report CMU-CS-07-105, Carnegie Mellon University School of Computer Science (January 2007)
11. Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S.: Towards automatic generation of vulnerability-based signatures. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy, pp. 2–16 (2006)

12. Brumley, D., Poosankam, P., Song, D., Zheng, J.: Automatic patch-based exploit generation is possible: Techniques and implications. In: *Proceedings of the 2008 IEEE Symposium on Security and Privacy* (2008)
13. Brumley, D., Wang, H., Jha, S., Song, D.: Creating vulnerability signatures using weakest pre-conditions. In: *Proceedings of Computer Security Foundations Symposium* (July 2007)
14. Caballero, J., Yin, H., Liang, Z., Song, D.: Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In: *Proceedings of the 14th ACM Conferences on Computer and Communication Security (CCS 2007)* (October 2007)
15. Caballero, J., Yin, H., Liang, Z., Song, D.: Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In: *Proceedings of the ACM Conference on Computer and Communications Security* (October 2007)
16. Chow, J., Pfaff, B., Garfinkel, T., Christopher, K., Rosenblum, M.: Understanding data lifetime via whole system simulation. In: *Proceedings of the 13th USENIX Security Symposium (Security 2004)* (August 2004)
17. Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., Barham, P.: Vigilante: End-to-end containment of internet worms. In: *Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP 2005)* (2005)
18. Crandall, J.R., Chong, F.T.: Minos: Control data attack prevention orthogonal to memory model. In: *Proceedings of the 37th International Symposium on Microarchitecture (MICRO 2004)* (December 2004)
19. DataRescue. IDA Pro. (Page checked 7/31/2008), <http://www.datarescue.com>
20. Dijkstra, E.W.: *A Discipline of Programming*. Prentice Hall, Englewood Cliffs (1976)
21. Ganesh, V., Dill, D.: STP: A decision procedure for bitvectors and arrays, <http://theory.stanford.edu/~vganesh/stp>
22. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) *CAV 2007*. LNCS, vol. 4590, pp. 524–536. Springer, Heidelberg (2007)
23. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual, Volumes 1-5 (April 2008)
24. Jackson, D., Rollins, E.J.: Chopping: A generalization of slicing. Technical Report CS-94-169, Carnegie Mellon University School of Computer Science (1994)
25. Kang, M.G., Poosankam, P., Yin, H.: Renovo: A hidden code extractor for packed executables. In: *Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM 2007)* (October 2007)
26. Kruegel, C., Robertson, W., Valeur, F., Vigna, G.: Static disassembly of obfuscated binaries. In: *Proceedings of the USENIX Security Symposium* (2004)
27. Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: Building customized program analysis tools with dynamic instrumentation. In: *Proceedings of the ACM Conference on Programming Language Design and Implementation* (June 2005)
28. Microsoft. Phoenix framework (Paged checked 7/31/2008), <http://research.microsoft.com/phoenix/>
29. Microsoft. Phoenix project architect posting (Page checked 7/31/2008) (July 2008), <http://forums.msdn.microsoft.com/en-US/phoenix/thread/90f5212c-05a-4aea-9a8f-a5840a6d101d>
30. Muchnick, S.S.: *Advanced Compiler Design and Implementation*. Academic Press, London (1997)
31. Nethercote, N.: *Dynamic Binary Analysis and Instrumentation or Building Tools is Easy*. PhD thesis, Trinity College, University of Cambridge (2004)
32. Newsome, J., Brumley, D., Franklin, J., Song, D.: Replayer: Automatic protocol replay by binary analysis. In: Write, R., De Capitani di Vimercati, S., Shmatikov, V. (eds.) *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 311–321 (2006)

33. Newsome, J., Brumley, D., Song, D.: Sting: An end-to-end self-healing system for defending against zero-day worm attacks. Technical Report CMU-CS-05-191, Carnegie Mellon University School of Computer Science (2006)
34. Newsome, J., Brumley, D., Song, D.: Vulnerability-specific execution filtering for exploit prevention on commodity software. In: Proceedings of the 13th Annual Network and Distributed Systems Security Symposium, NDSS (2006)
35. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS 2005) (February 2005)
36. Qemu, <http://fabrice.bellard.free.fr/qemu/>
37. Simpson, L.T.: Value-Driven Redundancy Elimination. PhD thesis, Rice University Department of Computer Science (1996)
38. Suh, G.E., Lee, J.W., Zhang, D., Devadas, S.: Secure program execution via dynamic information flow tracking. In: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2004) (October 2004)
39. Tridgell, A.: How samba was written (Checked on 8/21/2008) (August 2003), http://www.samba.org/ftp/tridge/misc/french_cafe.txt
40. Tucek, J., Newsome, J., Lu, S., Huang, C., Xanthos, S., Brumley, D., Zhou, Y., Song, D.: Sweeper: A lightweight end-to-end system for defending against fast worms. In: Proceedings of the EuroSys Conference (2007)
41. Valgrind, <http://valgrind.org>
42. Yin, H., Liang, Z., Song, D.: HookFinder: Identifying and understanding malware hooking behaviors. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS 2008) (February 2008)
43. Yin, H., Song, D., Manuel, E., Kruegel, C., Kirda, E.: Panorama: Capturing system-wide information flow for malware detection and analysis. In: Proceedings of the 14th ACM Conferences on Computer and Communication Security (CCS 2007) (October 2007)

On the Decidability of Model-Checking Information Flow Properties

Deepak D'Souza¹, Raveendra Holla¹, Janardhan Kulkarni¹,
Raghavendra K. Ramesh¹, and Barbara Sprick²

¹ Department of Computer Sc. & Automation, Indian Institute of Science, India
{deepakd,raveendra,janardhan,raghavendrkr}@csa.iisc.ernet.in

² Department of Computer Science, Modeling and Analysis of Information Systems,
TU Darmstadt, Germany
sprick@mais.informatik.tu-darmstadt.de

Abstract. Current standard security practices do not provide substantial assurance about information flow security: the end-to-end behavior of a computing system. Noninterference is the basic semantical condition used to account for information flow security. In the literature, there are many definitions of noninterference: Non-inference, Separability and so on. Mantel presented a framework of *Basic Security Predicates* (BSPs) for characterizing the definitions of noninterference in the literature. Model-checking these BSPs for finite state systems was shown to be decidable in [8]. In this paper, we show that verifying these BSPs for the more expressive system model of pushdown systems is undecidable. We also give an example of a simple security property which is undecidable even for finite-state systems: the property is a weak form of non-inference called WNI, which is not expressible in Mantel's BSP framework.

1 Introduction

Current standard security practices do not provide substantial assurance that the end-to-end behavior of a computing system satisfies important security policies such as confidentiality. Military, medical and financial information systems, as well as web based services such as mail, shopping and business-to-business transactions are all applications that create serious privacy concerns. The standard way to protect data is (discretionary) access control: some privilege is required in order to access files or objects containing the confidential data. Access control checks place restrictions on the release of information but not its propagation. Once information is released from its container, the accessing program may, through error or malice, improperly transmit the information in some form. Hence there is a lack of end-to-end security guarantees in access control systems.

Information flow security aims at answering end-to-end security. Most often the concept is studied in the setting of multi-level security [5] with data assigned levels in a security lattice, such that levels higher in the lattice correspond to data of higher sensitivity. A flow of information from a higher level in the security lattice to a lower one could breach confidentiality and a flow from a lower level

to a higher one might indicate a breach of integrity. Information flow security has been a focal research topic in computer security for more than 30 years. Nevertheless, the problem of securing the flow of information in systems is far from being solved. The two main research problems are, firstly, finding adequate formal properties of a “secure” system and, secondly, developing sound and efficient verification techniques to check systems for these properties.

Noninterference is the basic semantical condition used to account for information flow security. It requires that high-level behavior should not interfere with low-level observations. There are several semantical models for which noninterference has been studied. In [20], the state of the art was surveyed for approaches to capture and analyze information flow security of concrete programs. A broad spectrum of notions has been developed for noninterference at the level of more abstract specifications, in particular event systems (e.g., [13,17]), state based system (e.g., [10]) and process algebras (e.g., [9]).

In this article, we look at information flow security at the level of abstract specifications. A system is viewed as generating traces containing “confidential” and “visible” events. The assumption is that any “low level-user”, e.g. an attacker, knows the set of all possible behaviors (traces) but can observe only visible events. The information flow properties specify restrictions on the kind of traces the system may generate, so as to restrict the amount of information a low-level user can infer from his observations about confidential events having taken place in the system. For example, the “non-inference” [18,17,24] property states that for every trace produced by the system, its projection to only visible events must also be a possible trace of the system. Thus if a system satisfies the non-inference information flow property, a low-level user who observes the visible behavior of the trace is unable to infer whether or not any high-level behavior has taken place. There are other security properties defined in the literature: *separability* [17] (which requires that every possible low-level behavior interleaved with every possible high-level behaviour must be a possible behaviour of a system), *generalized noninterference* [16] (which requires that for every possible trace and every possible perturbation there is a correction to the perturbation such that the resulting trace is again a possible trace of the system), *nondeducability* [22], *restrictiveness* [16], the *perfect security property* [24] etc.

In [15] Mantel provides a framework for reasoning about the various information flow properties presented in the literature, in a modular way. He identifies a set of basic information flow properties which he calls “Basic Security Predicates” or BSPs, which are shown to be the building blocks of most of the known trace-based properties in the literature. The framework is modular in that BSPs that are common to several properties of interest for the given system need only be verified once for the system.

In this paper we consider the problem of model-checking information flow properties – that is, given a system model and a security property, can we algorithmically check whether the system satisfies the property? The most popular verification techniques are security type systems and program logics at the level of programs (see, e.g. [4]) and the unwinding technique at the level of

specifications (see, e.g., [14,11,7]). Other approaches for abstract systems are the more recent “model-checking” technique in [8] and the algorithmic approach in [23].

The unwinding technique is based on identifying structural properties of the system model which ensure the satisfaction of the information flow property. The method is not complete in general, in that a system could satisfy the information flow property but fail the unwinding condition. In [15] Mantel gives unwinding conditions for most of the BSPs he identifies. However, finding a useful unwinding relation is not trivial and remains up to the verifier.

The model-checking approach in [8] on the other hand is both sound and complete for all information flow properties that can be characterised in terms of BSPs and relies on an automata-based approach (when the given system is finite-state) to check language-theoretic properties of the system. Meyden and Zhang [23] also develop algorithms for model-checking information flow properties for finite-state systems and characterize the complexity of the associated verification problem.

In this article we investigate the problem of model-checking BSPs for systems modelled as “pushdown” systems. These systems can make use of an unbounded stack, and are useful in modelling programs with procedure calls but no dynamic memory allocation (“boolean programs” [2]), or recursive state machines [1]. We show that this problem is undecidable for all the BSPs, and hence we cannot hope to find an algorithmic solution to the problem. This result does not immediately tell us that verifying the non-interference properties in literature (which Mantel showed can be expressed as conjunctions of his BSP’s) is undecidable for pushdown systems. However, we can conclude that (a) it is not possible to algorithmically check BSPs – which are useful security properties in their own right – for pushdown system models; and (b) Mantel’s framework is not beneficial for the purpose of algorithmic verification of noninterference properties for pushdown systems, unlike the case of finite-state systems.

Systematic approaches investigating the decidability of noninterference properties for infinite state systems are, to the best of our knowledge, still missing. In the context of language based notions of noninterference, [4] shows that the notion of strong low bisimulation as defined by [21] is decidable for a simple parallel while language. Alternative notions of noninterference, defined for example in [3], turn out to be undecidable.

In the second part of the paper we consider a natural security property we call *Weak Non-inference* or *WNI*. The property essentially says that by observing a visible system trace a low-level user cannot pinpoint the exact sequence of confidential events that have taken place in the system. The property is thus in the the same spirit as the BSPs and noninterference properties as they all say that by seeing a visible trace a low-level user cannot infer “too much” about the confidential events that have taken place in the system. In fact, *WNI* can be seen to be weaker than all these properties. We show that the problem of model-checking *WNI* is undecidable not just for pushdown systems but for *finite-state* systems as well.

This result is interesting for a couple of reasons: Firstly, it follows from our results that *WNI*, an interesting noninterference property, is *not* definable in Mantel’s BSP framework. Secondly, this result sheds some light on a broader question: Is there a natural first-order logic which can express properties like the BSPs *and* which can be model-checked for finite-state systems? We note that BSPs make use of the operations of concatenation (implicitly) and projection to a subset of the alphabet. By earlier undecidability results in the literature ([19]) a general FO logic that allows concatenation is necessarily undecidable. By our undecidability result for *WNI*, it follows that a FO logic that uses projection (and negation, which BSPs don’t use) is necessarily undecidable.

The rest of the paper is organized as follows. Section 2 defines the technical terms used in the paper. Section 3 proves the undecidability of model-checking BSPs for pushdown systems. Section 4 introduces a property called “Weak Non Inference”, which cannot be characterized in terms of BSPs and gives its decidability results. Finally Section 5 concludes the article.

2 Preliminaries

By an alphabet we will mean a finite set of symbols representing *events* or *actions* of a system. For an alphabet Σ we use Σ^* to denote the set of finite strings over Σ , and Σ^+ to denote the set of non-empty strings over Σ . The null or empty string is represented by the symbol ϵ . For two strings α and β in Σ^* we write $\alpha\beta$ for the concatenation of α followed by β . A *language* over Σ is just a subset of Σ^* .

We fix an alphabet of events Σ and assume a partition of Σ into V, C, N , which in the framework of [13] correspond to events that are *visible*, *confidential*, and *neither* visible nor confidential, from a particular user’s point of view.

Let $X \subseteq \Sigma$. The projection of a string $\tau \in \Sigma^*$ to X is written $\tau|_X$ and is obtained from τ by deleting all events that are not elements of X . The projection of the language L to X , written $L|_X$, is defined to be $\{\tau|_X \mid \tau \in L\}$.

A (*labelled*) *transition system* over an alphabet Σ is a structure of the form $\mathcal{T} = (Q, s, \longrightarrow)$, where Q is a set of states, $s \in Q$ is the start state, and $\longrightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation. We write $p \xrightarrow{a} q$ to stand for $(p, a, q) \in \longrightarrow$, and use $p \xrightarrow{w}^* q$ to denote the fact that we have a path labelled w from p to q in the underlying graph of the transition system \mathcal{T} . If some state q has an edge labelled a , then we say a is enabled at q .

The language generated by \mathcal{T} is defined to be

$$L(\mathcal{T}) = \{\alpha \in \Sigma^* \mid \text{there exists a } t \in Q \text{ such that } s \xrightarrow{\alpha}^* t\}.$$

We begin by recalling the *basic security predicates* (BSPs) of Mantel [15]. It will be convenient to use the notation $\alpha =_Y \beta$ where $\alpha, \beta \in \Sigma^*$ and $Y \subseteq \Sigma$, to mean α and β are the same “modulo a correction on Y -events”. More precisely, $\alpha =_Y \beta$ iff $\alpha|_{\bar{Y}} = \beta|_{\bar{Y}}$, where \bar{Y} denotes $\Sigma - Y$. By extension, for languages L and M over Σ , we say $L \subseteq_Y M$ iff $L|_{\bar{Y}} \subseteq M|_{\bar{Y}}$.

In the definitions below, we assume L to be a language over Σ .

1. L satisfies R (*Removal of events*) iff for all $\tau \in L$ there exists $\tau' \in L$ such that $\tau' \upharpoonright_C = \epsilon$ and $\tau' \upharpoonright_V = \tau \upharpoonright_V$.
2. L satisfies D (*stepwise Deletion of events*) iff for all $\alpha\beta \in L$, such that $c \in C$ and $\beta \upharpoonright_C = \epsilon$, we have $\alpha'\beta' \in L$ with $\alpha' =_N \alpha$ and $\beta' =_N \beta$.
3. L satisfies I (*Insertion of events*) iff for all $\alpha\beta \in L$ such that $\beta \upharpoonright_C = \epsilon$, and for every $c \in C$, we have $\alpha'c\beta' \in L$, with $\beta' =_N \beta$ and $\alpha' =_N \alpha$.
4. Let $X \subseteq \Sigma$. Then L satisfies IA (*Insertion of Admissible events*) w.r.t X iff for all $\alpha\beta \in L$ such that $\beta \upharpoonright_C = \epsilon$ and for some $c \in C$, there exists $\gamma c \in L$ with $\gamma \upharpoonright_X = \alpha \upharpoonright_X$, we have $\alpha'c\beta' \in L$ with $\beta' =_N \beta$ and $\alpha' =_N \alpha$.
5. L satisfies BSD (*Backwards Strict Deletion*) iff for all $\alpha\beta \in L$ such that $c \in C$ and $\beta \upharpoonright_C = \epsilon$, we have $\alpha\beta' \in L$ with $\beta' =_N \beta$.
6. L satisfies BSI (*Backwards Strict Insertion*) iff for all $\alpha\beta \in L$ such that $\beta \upharpoonright_C = \epsilon$, and for every $c \in C$, we have $\alpha c\beta' \in L$, with $\beta' =_N \beta$.
7. Let $X \subseteq \Sigma$. Then L satisfies $BSIA$ (*Backwards Strict Insertion of Admissible events*) w.r.t X iff for all $\alpha\beta \in L$ such that $\beta \upharpoonright_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \upharpoonright_X = \alpha \upharpoonright_X$, we have $\alpha c\beta' \in L$ with $\beta' =_N \beta$.
8. Let $X \subseteq \Sigma$, $V' \subseteq V$, $C' \subseteq C$, and $N' \subseteq N$. Then L satisfies FCD (*Forward Correctable Deletion*) w.r.t V', C', N' iff for all $\alpha c v \beta \in L$ such that $c \in C'$, $v \in V'$ and $\beta \upharpoonright_C = \epsilon$, we have $\alpha \delta v \beta' \in L$ where $\delta \in (N')^*$ and $\beta' =_N \beta$.
9. Let, $V' \subseteq V$, $C' \subseteq C$, and $N' \subseteq N$. Then L satisfies FCI (*Forward Correctable Insertion*) w.r.t C', V', N' iff for all $\alpha v \beta \in L$ such that $v \in V'$, $\beta \upharpoonright_C = \epsilon$, and for every $c \in C'$ we have $\alpha c \delta v \beta' \in L$, with $\delta \in (N')^*$ and $\beta' =_N \beta$.
10. Let $X \subseteq \Sigma$, $V' \subseteq V$, $C' \subseteq C$, and $N' \subseteq N$. Then L satisfies $FCIA$ (*Forward Correctable Insertion of admissible events*) w.r.t. X, V', C', N' iff for all $\alpha v \beta \in L$ such that: $v \in V'$, $\beta \upharpoonright_C = \epsilon$, and there exists $\gamma c \in L$, with $c \in C'$ and $\gamma \upharpoonright_X = \alpha \upharpoonright_X$; we have $\alpha c \delta v \beta' \in L$ with $\delta \in (N')^*$ and $\beta' =_N \beta$.
11. L satisfies SR (*Strict Removal*) iff for all $\tau \in L$ we have $\tau \upharpoonright_{\overline{C}} \in L$.
12. L satisfies SD (*Strict Deletion*) iff for all $\alpha\beta \in L$ such that $c \in C$ and $\beta \upharpoonright_C = \epsilon$, we have $\alpha\beta \in L$.
13. L satisfies SI (*Strict Insertion*) iff for all $\alpha\beta \in L$ such that $\beta \upharpoonright_C = \epsilon$, and for every $c \in C$, we have $\alpha c\beta \in L$.
14. Let $X \subseteq \Sigma$. L satisfies SIA (*Strict Insertion of Admissible events*) w.r.t X iff for all $\alpha\beta \in L$ such that $\beta \upharpoonright_C = \epsilon$ and there exists $\gamma c \in L$ with $c \in C$ and $\gamma \upharpoonright_X = \alpha \upharpoonright_X$, we have $\alpha c\beta \in L$.

We say a Σ -labelled transition system \mathcal{T} satisfies a BSP iff $L(\mathcal{T})$ satisfies the BSP.

3 BSPs and Pushdown Systems

The model-checking problem for BSPs is the following: Given a BSP P and a system modelled as transition system \mathcal{T} , does \mathcal{T} satisfy P ? The problem was shown to be decidable when the system is presented as a finite-state transition system [8]. In this section we show that if the system is modelled as a pushdown system,

the model-checking problem becomes undecidable. We use a reduction from the emptiness problem of Turing machines, which is known to be undecidable.

This section is organized as follows. First, we introduce Pushdown Systems and show that they accept exactly the class of prefix-closed context-free languages. Then we show for the BSP D that verifying D for Pushdown systems is undecidable by reducing the emptiness problem for Turing machines to this problem. More concretely, for a given Turing machine M we construct a prefix closed context-free language L_M such that L_M satisfies BSP D iff the language $L(M)$ of the turing machine M is empty. By adjusting the prefix-closed context-free language L_M appropriately, we can show the same for all other BSPs defined in Mantel's MAKs framework. We will give all the respective languages for the other BSPs in this chapter. The detailed undecidability proofs can be found in the technical report [6].

We assume that Σ , the set of events, contains two visible events v_1 and v_2 , and one confidential event c .

A *pushdown system (PDS)* given by $P = (Q, \Sigma_P, \Gamma_P, \Delta, s, \perp)$ consists of a finite set of control states Q , a finite input alphabet Σ_P , a finite stack alphabet Γ_P , and a transition relation $\Delta \subseteq ((Q \times (\Sigma_P \cup \{\epsilon\}) \times \Gamma_P) \times (Q \times \Gamma_P^*))$, a starting state $s \in Q$ and an initial stack symbol $\perp \in \Gamma_P$. If $((p, a, A), (q, B_1 B_2 \cdots B_k)) \in \Delta$, this means that whenever the machine is in state p reading input symbol a on the input tape and A on top of the stack, it can pop A off the stack, push $B_1 B_2 \cdots B_k$ onto the stack (such that B_1 becomes the new stack top symbol), move its read head right one cell past the a , and enter state q . If $((p, \epsilon, A), (q, B_1 B_2 \cdots B_k)) \in \Delta$, this means that whenever the machine is in state p and A on top of the stack, it can pop A off the stack, push $B_1 B_2 \cdots B_k$ onto the stack (such that B_1 becomes the new stack top symbol), leave its read head unchanged and enter state q .

A configuration of pushdown system P is an element of $Q \times \Sigma_P^* \times \Gamma_P^*$ describing the current state, the portion of the input yet unread and the current stack contents. For example, when x is the input string, the start configuration is (s, x, \perp) . The next configuration relation \longrightarrow is defined for any $y \in \Sigma_P^*$ and $\beta \in \Gamma_P^*$, as $(p, ay, A\beta) \longrightarrow (q, y, \gamma\beta)$ if $((p, a, A), (q, \gamma)) \in \Delta$ and $(p, y, A\beta) \longrightarrow (q, y, \gamma\beta)$ if $((p, \epsilon, A), (q, \gamma)) \in \Delta$. Let \longrightarrow^* be defined as the reflexive transitive closure of \longrightarrow . Then P accepts w iff $(s, w, \perp) \longrightarrow^* (q, \epsilon, \gamma)$ for some $q \in Q$, $\gamma \in \Gamma_P^*$.

Pushdown systems also appear in other equivalent forms in literature. For example *Recursive state machines (RSM's)* [1] and *Boolean programs* [2]. Pushdown systems then capture an interesting class of systems. The class of languages accepted by pushdown systems is closely related to context-free languages.

Lemma 1. *The class of languages accepted by pushdown systems is exactly the class of prefix closed context-free languages.*

Proof. Pushdown systems can be seen as a special case of pushdown automata with all its control states treated as final states. Hence pushdown systems generate a context-free language. It is easy to see that if a PDS accepts a string w , then it also accepts all the prefixes of w .

Conversely, let L be a prefix closed context-free language. Then there exists a context-free grammar (CFG) G generating L . Without loss of generality, we assume that G is in Greibach Normal Form with every non-terminal deriving a terminal string. A nondeterministic pushdown automata P with a single state q , accepting by empty stack, with S (starting non-terminal in G) as the initial stack symbol, accepting exactly $L(G)$ can be constructed [12]. The idea of the construction is that for each production $A \rightarrow cB_1B_2 \cdots B_k$ in G , a transition $((q, c, A), (q, B_1B_2 \cdots B_k))$ is added to P . We now observe that if P has a run on w with γ as the string of symbols in stack, then there exists a left-most sentential form $S \xrightarrow{*} w\gamma$ in G . Then every terminal prefix of a left-most sentential form in G has an extension in L (since every non-terminal derives a string). Now view P as a PDS P' (ignoring the empty stack condition for acceptance). Clearly P' accepts all the strings in L . Further, if P' has a run on some string w with γ as the string of symbols (corresponds to non-terminals in G) in stack, then w corresponds to a prefix of a left-most sentential form in G , and hence has an extension in L . Since L is a prefix closed, w also belongs to L . Hence $L(P') = L$.

We use the emptiness problem of Turing machines to show the undecidability of verifying BSPs for pushdown systems. Let M be a Turing machine defined as $M = (Q, \Sigma_M, \Gamma_M, \vdash, \sqcup, \delta, s, t, r)$, where Q is a finite set of states, Σ_M is a finite input alphabet, Γ_M is a finite tape alphabet containing Σ_M , $\vdash \in \Gamma_M \setminus \Sigma_M$ is the left endmarker, $\sqcup \in \Gamma_M \setminus \Sigma_M$ is the blank symbol, $\delta : Q \times \Gamma_M \rightarrow Q \times \Gamma_M \times \{L, R\}$ is the transition function, $s \in Q$ is the start state, $t \in Q$ is the accept state and $r \in Q$ is the reject state with $r \neq t$ and no transitions defined out of r and t . The configuration x of M at any instant is defined as a triple (q, y, n) where $q \in Q$ is a state, y is a non-blank finite string describing the contents of the tape and n is a non negative integer describing the head position. The next configuration relation \rightsquigarrow is defined as $(p, a\beta, n) \rightsquigarrow (q, b\beta, n+1)$, when $\delta(p, a) = (q, b, R)$. Similarly $(p, a\beta, n) \rightsquigarrow (q, b\beta, n-1)$, when $\delta(p, a) = (q, b, L)$.

We can encode configurations as finite strings over the alphabet $\Gamma_M \times (Q \cup \{-\})$, where $- \notin Q$. A pair in $\Gamma_M \times (Q \cup \{-\})$ is written vertically with the element of Γ_M on top. We represent a computation of M as a sequence of configurations x_i separated by $\#$. It will be of the form $\#x_1\#x_2\#\cdots\#x_n\#$. Each letter in $(\Gamma_M \times (Q \cup \{-\})) \cup \{\#\}$ can be encoded as a string of v_1 's and a string (computation) in $(\Gamma_M \times (Q \cup \{-\})) \cup \{\#\}$ can be represented as a string of v_1 's and v_2 's, with v_2 used as the separator. Recall the assumption that Σ contains v_1, v_2 (visible events) and c (confidential event). For a computation w , we use $enc(w)$ to denote this encoding.

We now show that the problem of verifying BSP D for pushdown systems is as hard as checking the language emptiness problem of a Turing machine. To do so, we construct a language L_M out of a given Turing machine M and show, that L_M satisfies BSP D iff $L(M)$ is empty. Afterwards we show, that L_M can be generated by a Pushdown system and is hence a prefix closed context-free language.

Let $M = (Q, \Sigma_M, \Gamma_M, \vdash, \sqcup, \delta, s, t, r)$ be a Turing machine. Consider the language L_M to be the prefix closure of $L_1 \cup L_2$ where

$$L_1 = \{c \cdot \text{enc}(\#x_1\#x_2 \cdots x_n\#) \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration} \end{array}\}$$

$$L_2 = \{\text{enc}(\#x_1\#x_2 \cdots x_n\#) \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration,} \\ \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition} \end{array}\}$$

Lemma 2. L_M satisfies BSP D iff $L(M) = \emptyset$.

Proof. (\Leftarrow): Let us assume $L(M) = \emptyset$. Consider any string containing a confidential event in L_M . The string has to be of the form $c\tau$ in L_1 or a prefix of it. τ cannot be a valid computation (encoded) of M , since $L(M) = \emptyset$. So, if we delete the last c , we will get τ , which is included in L_2 . Also, all the prefixes of $c\tau$ and τ will be in L_M , as it is prefix closed. Hence L_M satisfies D .

(\Rightarrow): If $L(M)$ is not empty then there exists some string τ which is a valid computation (encoded). L_1 contains $c\tau$. If we delete the last c , the resulting string τ is not present in L_M . Hence D is not satisfied. Hence $L(M)$ is empty, when L_M satisfies the BSP D .

To complete the reduction, we need to show, that L_M is a prefix-closed context-free language, and we do this by translating M into a PDS accepting L_M .

Since CFLs are closed under prefix operation (adding the prefixes of the strings in the language) and as prefix closed CFLs are equivalent to PDS (from Lemma 1), it is enough to show that $L_1 \cup L_2$ is a CFL.

Let $T = (\Gamma_M \times (Q \cup \{-\})) \cup \{\#\}$. Consider the above defined language $L_2 \subseteq T^*$ (with the strings being unencoded versions).

$$L_2 = \{\#x_1\#x_2\# \cdots \#x_n\# \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration,} \\ \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition} \end{array}\}$$

L_2 can be thought of as the intersection of languages A_1, A_2, A_3, A_4 and A_5 , where

$$\begin{aligned} A_1 &= \{w \mid w \text{ begins and ends with } \#\} \\ A_2 &= \{w \mid \text{the string between any two } \# \text{ s must be a valid configuration}\} \\ A_3 &= \{\#x\#w \mid x \in (T \setminus \{\#\})^* \text{ is a valid starting configuration}\} \\ A_4 &= \{w\#x\# \mid x \in (T \setminus \{\#\})^* \text{ is a valid accepting configuration}\} \\ A_5 &= \{\#x_1\# \cdots \#x_n\# \mid \text{exists some } i, \text{ with } x_i \rightsquigarrow x_{i+1} \text{ not a valid transition}\} \end{aligned}$$

We now show that the languages from A_1 to A_4 are regular and A_5 is a CFL. Since CFLs are closed under intersection with regular languages, L_2 will be shown to be context-free. Note that L_1 (unencoded) is the intersection of A_1, A_2, A_3 and A_4 , and hence regular. Since CFLs are closed under union, $L_1 \cup L_2$ will also be context-free.

The language A_1 can be generated by the regular expression $\#(T \setminus \{\#\})^* \#$. For A_2 , we only need to check that between every $\#$'s there is exactly one symbol with a state q on the bottom, and \vdash occurs on the top immediately after each $\#$ (except the last) and nowhere else. This can be easily checked with a finite automaton. The set A_3 can be generated by the regular expression $\#(\vdash, s)K^* \#T^*$, with $K = \Gamma \setminus \{\vdash\} \times \{-\}$. To check that a string is in A_4 , we only need to check that t appears someplace in the string. This can be easily checked by an automaton. For A_5 , we construct a nondeterministic pushdown automaton (NPDA). The NPDA nondeterministically guesses i for x_i and then it scans to the three-length substring in x_i with a state in the middle component of the three-length string and checks with the corresponding three-length substring in x_{i+1} using the stack. Then, these substrings are checked against the transition relation of M , accepting if it is not a valid transition. Interested readers are referred to [12], for detailed proofs of above languages to be regular and context-free languages. Now, it follows that L_1 is regular and L_2 is context-free. As languages accepted by pushdown systems (Lemma 1) are equivalent to prefix closed context-free languages, L_M is a language of a pushdown system.

We have shown, that the prefix closed context-free language L_M satisfies BSP D iff the language $L(M)$ of Turing machine M is empty. Since the emptiness problem of Turing machines is undecidable, we get the following theorem.

Lemma 3. *The problem of verifying BSP D for pushdown systems is undecidable.*

Undecidability for the rest of the BSPs can be shown in a similar fashion. For the BSPs R , SR , SD , BSD we can use the same language L_M and get

Lemma 4. *L_M satisfies BSP R (and SR and SD and BSD) iff $L(M) = \emptyset$.*

For the other BSPs we need to adapt the languages appropriately as shown in the following. The detailed proofs for these languages can be found in the technical report [6].

To show undecidability of BSPs I , BSI , and SI , we consider L_M^I to be the prefix closure of $L_1 \cup L_2$ where

$$L_1 = \{ \text{enc}(\#x_1\#x_2 \cdots x_n\#) \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration} \end{array} \}$$

$$L_2 = \{ w \in (\{v_1, v_2\} \cup C \cup N)^* \mid \begin{array}{l} w \upharpoonright_{\{v_1, v_2\}} = \text{enc}(\#x_1\#x_2 \cdots x_n\#), \\ x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration,} \\ \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition} \end{array} \}$$

Lemma 5. *L_M^I satisfies BSP I (and SI and BSI) iff $L(M) = \emptyset$.*

To show undecidability of BSPs IA^X , $BSIA^X$, and SIA^X with $X \subseteq \Sigma$, we consider $L_M^{IA^X}$ to be the prefix closure of $L_1 \cup L_2 \cup L_3$ where

$$\begin{aligned}
 L_1 &= \{enc(\#x_1\#x_2 \cdots x_n\#) \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration} \end{array}\} \\
 L_2 &= \{w \in (\{v_1, v_2\} \cup C)^* \mid \begin{array}{l} w \upharpoonright_{\{v_1, v_2\}} = enc(\#x_1\#x_2 \cdots x_n\#), \\ x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration,} \\ \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition} \end{array}\} \\
 L_3 &= X^* \cdot C
 \end{aligned}$$

Lemma 6. $L_M^{IA^X}$ satisfies BSP IA^X (and SIA^X and $BSIA^X$) iff $L(M) = \emptyset$.

To show undecidability of BSP FCD we assume $V' \subseteq V, N' \subseteq N, C' \subseteq C$ to be given and $v_2 \in V'$ and $c \in C'$. We consider language L_M^{FCD} to be the prefix closure of $L_1 \cup L_2$ where

$$\begin{aligned}
 L_1 &= \{cv_2 \cdot enc(x_1x_2 \cdots x_n) \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration} \end{array}\} \\
 L_2 &= \{v_2 \cdot enc(x_1x_2 \cdots x_n) \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration,} \\ \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition} \end{array}\}
 \end{aligned}$$

Lemma 7. L_M^{FCD} satisfies BSP FCD iff $L(M) = \emptyset$.

To prove undecidability of BSP FCI we assume $V' \subseteq V, N' \subseteq N, C' \subseteq C$ to be given and $v_2 \in V'$. We consider language L_M^{FCI} to be the prefix closure of $L_1 \cup L_2$ where

$$\begin{aligned}
 L_1 &= \{v_2 \cdot enc(x_1x_2 \cdots x_n) \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration} \end{array}\} \\
 L_2 &= \{w \in (\{v_1, v_2\} \cup C')^* \mid \begin{array}{l} w \upharpoonright_{\{v_1, v_2\}} = v_2 \cdot enc(x_1x_2 \cdots x_n), \\ x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration,} \\ \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition} \end{array}\}
 \end{aligned}$$

Lemma 8. L_M^{FCI} satisfies FCI iff $L(M) = \emptyset$.

Finally, to show undecidability of BSP $FCIA^X$, where $X \subseteq \Sigma$ we assume $V' \subseteq V, N' \subseteq N, C' \subseteq C$ to be given with $v_2 \in V'$. We consider language $L_M^{FCIA^X}$ to be the prefix closure of $L_1 \cup L_2 \cup L_3$ where

$$\begin{aligned}
L_1 &= \{v_2 \cdot \text{enc}(x_1 x_2 \cdots x_n) \mid \begin{array}{l} x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration} \end{array}\} \\
L_2 &= \{w \in (\{v_1, v_2\} \cup C')^* \mid \begin{array}{l} w \upharpoonright_{\{v_1, v_2\}} = v_2 \cdot \text{enc}(x_1 x_2 \cdots x_n), \\ x_1 \text{ is a starting configuration,} \\ x_n \text{ is an accepting configuration,} \\ \text{exists } i : x_i \rightsquigarrow x_{i+1} \text{ invalid transition} \end{array}\} \\
L_3 &= X^* \cdot C'
\end{aligned}$$

Lemma 9. L_M^{FCIAX} satisfies BSP $FCIA^X$ iff $L(M) = \emptyset$.

The proofs of undecidability for these BSPs are given in the technical report [6]. We have shown that the BSPs defined by Mantel are undecidable for pushdown systems. Hence we get the following theorem.

Theorem 1. *The problem of model-checking any of the BSPs for pushdown systems is undecidable.*

4 Weak Non-Inference

In this section we introduce a natural information flow property which we call *Weak Non-Inference* (WNI) as it is weaker than most of the properties proposed in the literature. We show that model-checking this property even for finite-state transition systems is undecidable.

A set of traces L over Σ is said to satisfy WNI iff

$$\forall \tau \in L, \exists \tau' \in L : (\tau \upharpoonright_C \neq \epsilon \Rightarrow (\tau \upharpoonright_V = \tau' \upharpoonright_V \wedge \tau \upharpoonright_C \neq \tau' \upharpoonright_C)).$$

The classical non-inference property can be phrased as $\forall \tau \in L, \exists \tau' \in L : (\tau' = \tau \upharpoonright_V)$. Thus it is easy to see that any language that satisfies non-inference also satisfies WNI.

We show that checking whether a finite-state transition system satisfies this property is undecidable, by showing a reduction from Post's Correspondence Problem (PCP). We recall that an instance of PCP is a collection of dominos $P = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ where each $x_i, y_i \in \Delta^+$ where Δ is a finite alphabet (recall that Δ^+ is the set of non-empty words over Δ). A *match* in P is a sequence i_1, i_2, \dots, i_l such that $x_{i_1} \cdots x_{i_l} = y_{i_1} \cdots y_{i_l}$. The PCP problem is to determine if a given instance P has a match, and this problem is known to be undecidable.

We construct a transition system $\mathcal{T}_P = (Q, s_1, \rightarrow)$ on the alphabet $\Sigma' = V \cup C$, where $V = \{v_1, \dots, v_n\}$, $C = \Delta$ and the transition relation \rightarrow is as shown in Fig. 1. The states s_2 and s_3 have n loops each on them: state s_2 has loops on the strings $v_1 x_1 v_1$ through $v_n x_n v_n$, and s_3 has loops on the strings $v_1 y_1 v_1$ through $v_n y_n v_n$. We choose each v_i to be different from the symbols in Δ .

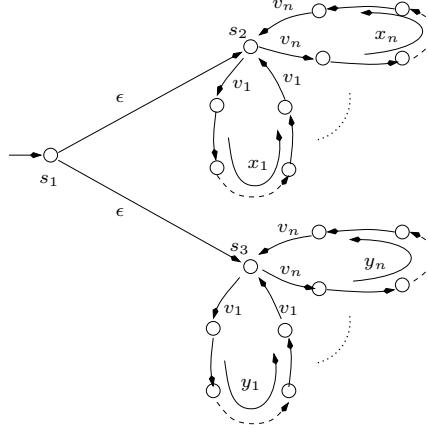


Fig. 1.

We call a string “interesting” if it is of the form $v_{i_1} w_{i_1} v_{i_1} v_{i_2} w_{i_2} v_{i_2} \cdots v_{i_k} w_{i_k} v_{i_k}$ for some i_1, \dots, i_k with $k \geq 1$, and w_{i_j} in Δ^* – in other words, the string must be non-empty and each visible alphabet occurs twice in succession. Thus every interesting string generated by \mathcal{T}_P will end up in state s_2 or in state s_3 . We observe that for every interesting string z in $L(\mathcal{T}_P)$, there is exactly one other string z' in $L(\mathcal{T}_P)$ with the same projection to V , which is moreover also an interesting string. If z passes through state s_2 , then z' is the string which mimics the run of z but through s_3 . Any other string will have a different projection to V .

Lemma 10. *A PCP instance P has a match iff \mathcal{T}_P does not satisfy WNI.*

Proof. (\Leftarrow :) Suppose \mathcal{T}_P does not satisfy WNI. Then there must exist a string τ in $L(\mathcal{T}_P)$ such that there is no other string in $L(\mathcal{T}_P)$ with the same projection to V and different projection to C . Now τ must be an interesting string – all uninteresting strings trivially satisfy the WNI condition since we can append or remove one confidential event from each of these strings. By our earlier observation, we know that there exists an interesting string τ' in $L(\mathcal{T}_P)$ which takes the symmetric path in \mathcal{T}_P and has the same projection to V as τ . Now by our choice of τ , the projection of τ' on C is the same as the projection of τ on C . But this means that we have found a match for P , given by the indices corresponding to the loops traces out by τ and τ' .

(\Rightarrow :) Let i_1, i_2, \dots, i_l be a match for P . Thus $x_{i_1} \cdots x_{i_l} = y_{i_1} \cdots y_{i_l}$. Consider the interesting string $\tau = v_{i_1} x_{i_1} v_{i_1} \cdots v_{i_l} x_{i_l} v_{i_l}$ in $L(\mathcal{T})$. Now there is exactly one other string τ' with the same projection to V , given by $v_{i_1} y_{i_1} v_{i_1} \cdots v_{i_l} y_{i_l} v_{i_l}$. However, as the indices chosen are a match for P , the projections of τ and τ' to C are identical. Thus \mathcal{T} does not satisfy WNI. \square

This completes the reduction of PCP to the problem of model-checking *WNI* for finite-state systems, and we conclude:

Theorem 2. *The problem of model-checking the property WNI for finite-state systems is undecidable.* \square

We note that if the property *WNI* were expressible as a boolean combination of BSPs, the decision procedure for model-checking BSPs for finite-state systems in [8] would imply that model-checking *WNI* for finite-state systems is decidable. Hence we can conclude that:

Corollary 1. *The property WNI is not expressible as a boolean combination of Mantel's BSPs.* \square

However, we can show that a restricted version of the problem is decidable. If we have a system model (finite-state or pushdown) that uses only *one* confidential event and *one* visible event, the problem of checking *WNI* becomes decidable.

Theorem 3. *The problem of model-checking WNI for pushdown systems when $|V| = 1$ and $|C| = 1$, is decidable.*

Proof. Consider an alphabet $\Sigma = \{v, c\}$, where v is the only visible event and c is the only confidential event. Recall that the *Parikh vector* of a string x over Σ , denoted $\psi(x)$, is the vector (n_v, n_c) where n_v is the number of occurrences of event v in x and n_c is the number of occurrences of event c in x . For a language L over Σ , its Parikh map $\psi(L)$ is defined to be the set of vectors $\{\psi(x) \mid x \in L\}$. By Parikh's theorem (see [12]), we know that whenever L is context-free, its Parikh map $\psi(L)$ forms a “semi-linear” set of vectors. To explain what this means, let us first introduce some notation. For a finite set of vectors $X = \{(n_1, m_1), \dots, (n_k, m_k)\}$ we denote the set of vectors *generated* by X , with initial vector (n_0, m_0) , to be the set $gen_{(n_0, m_0)}(X) = \{(n_0, m_0) + t_1(n_1, m_1) + \dots + t_k(n_k, m_k) \mid t_1, \dots, t_k \in \mathbb{N}\}$. Then $\psi(L)$ is semi-linear means that there exist finite non-empty sets of vectors X_1, \dots, X_l , and initial vectors $(n_0^1, m_0^1), \dots, (n_0^l, m_0^l)$, such that

$$\psi(L) = \bigcup_{i \in \{1, \dots, l\}} gen_{(n_0^i, m_0^i)}(X_i).$$

Now let L be the given context-free language over the alphabet $\Sigma = \{v, c\}$, with Parikh map given by the sets X_1, \dots, X_l , and initial vectors $(n_0^1, m_0^1), \dots, (n_0^l, m_0^l)$. Let each $X_i = \{(m_1^i, n_1^i), \dots, (m_{k_i}^i, n_{k_i}^i)\}$. To verify *WNI* property for L , we need to check that for every vector in $\psi(L)$ there exists another vector which is in $gen_{(n_0^i, m_0^i)}(X_i)$ for some i , such that their visible event count is same, and confidential event count is different. For $l = 1$, L satisfies *WNI* iff the following formula in Presburger arithmetic¹ is valid:

¹ Presburger arithmetic is the first-order theory of natural numbers with addition which is known to be decidable in double-exponential time.

$$\begin{aligned}
& \forall t_1, \dots, t_{k_1} \in \mathbb{N}, \exists t'_1, \dots, t'_{k_1} \in \mathbb{N} \\
& \left(n_0^1 + t_1 n_1^1 + \dots + t_{k_1} n_{k_1}^1 > 0 \implies \right. \\
& \left(m_0^1 + t_1 m_1^1 + \dots + t_{k_1} m_{k_1}^1 = m_0^1 + t'_1 m_1^1 + \dots + t'_{k_1} m_{k_1}^1 \wedge \right. \\
& \left. \left. n_0^1 + t_1 n_1^1 + \dots + t_{k_1} n_{k_1}^1 \neq n_0^1 + t'_1 n_1^1 + \dots + t'_{k_1} n_{k_1}^1 \right) \right).
\end{aligned}$$

The validity of this formula can be found using the decision procedure for Presburger arithmetic. The formula is extended in the expected way for higher values of l .

In fact, any property which just asks for the counting relationship between visible and confidential events can be decided using the above technique.

5 Conclusions

In this paper we have studied the problem of model-checking Mantel's Basic Security Predicates for systems modelled as pushdown systems. We have shown that unlike the case of finite-state system models, this problem is undecidable. We also studied the model-checking problem for a property called *WNI* which is a weak form of the property of non-inference studied earlier in the literature. We show that this problem is undecidable, not just for pushdown systems but also for finite-state systems. It follows from this result that *WNI* is *not* expressible in Mantel's BSP framework. We also show that for a restricted class of systems (with only one visible and confidential event) this property is decidable for both finite-state and pushdown system models.

In future work we plan to see if similar reductions can be used to argue that the model-checking problem for noninterference properties in the literature (of which Mantel showed the BSPs to be the basic building blocks) – are also undecidable. Another open question is whether the model-checking problem continues to be undecidable even when we consider *deterministic* pushdown systems.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Analysis of recursive state machines. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 207–220. Springer, Heidelberg (2001)
2. Ball, T., Rajamani, S.K.: Bebop: A symbolic model checker for boolean programs. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN 2000. LNCS, vol. 1885, pp. 113–130. Springer, Heidelberg (2000)
3. Boudol, G., Castellani, I.: Noninterference for concurrent programs and thread systems. Theoretical Computer Science 1-2(281), 109–130 (2002)
4. Dam, M.: Decidability and proof systems for language-based noninterference relations. In: Proceedings POPL 2006, Charleston, South Carolina (2006)
5. Denning, D.E.: A lattice model of secure information flow. Commun. ACM 19(5), 236–243 (1976)

6. D'Souza, D., Holla, R., Kulkarni, J., Raghavendra, K.R., Sprick, B.: On the decidability of model-checking information flow properties. Technical Report IISc-CSA-TR-2008-2 (2008)
7. D'Souza, D., Raghavendra, K.R.: Checking unwinding conditions for finite state systems. In: Proceedings of the VERIFY 2006 workshop, pp. 85–94 (2006)
8. D'Souza, D., Raghavendra, K.R., Sprick, B.: An automata based approach for verifying information flow properties. In: Proceedings of the second workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2005). ENTCS, vol. 135, pp. 39–58 (2005)
9. Focardi, R., Gorrieri, R.: A classification of security properties for process algebras. *Journal of Computer Security* 1, 5–33 (1995)
10. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proc. IEEE Symp. on Security and Privacy, April 1982, pp. 11–20 (1982)
11. Goguen, J.A., Meseguer, J.: Unwinding and inference control. In: Proc. IEEE Symp. on Security and Privacy, pp. 75–86 (April 1984)
12. Kozen, D.C.: Automata and Computability. Springer, Heidelberg (1997)
13. Mantel, H.: Possibilistic Definitions of Security – An Assembly Kit. In: Proceedings of the 13th IEEE Computer Security Foundations Workshop, Cambridge, UK, July 3–5, 2000, pp. 185–199. IEEE Computer Society, Los Alamitos (2000)
14. Mantel, H.: Unwinding Possibilistic Security Properties. In: Cuppens, F., Deswarte, Y., Gollmann, D., Waidner, M. (eds.) ESORICS 2000. LNCS, vol. 1895, pp. 238–254. Springer, Heidelberg (2000)
15. Mantel, H.: A Uniform Framework for the Formal Specification and Verification of Information Flow Security. PhD thesis, Universität des Saarlandes (2003)
16. McCullough, D.: Specifications for multilevel security and a hookup property. In: Proc. 1987 IEEE Symp. Security and Privacy (1987)
17. McLean, J.: A general theory of composition for trace sets closed under selective interleaving functions. In: Proc. IEEE Symposium on Research in Security and Privacy, pp. 79–93. IEEE Computer Society Press, Los Alamitos (1994)
18. O'Halloran, C.: A calculus of information flow. In: Proceedings of the European Symposium on Research in Computer Security, ESORICS 1990 (1990)
19. Quine, W.V.: Concatenation as a basis for finite arithmetic. *J. Symbolic Logic* 11(4) (1946)
20. Sabelfeld, A., Myers, A.: Language-based information-flow security. *IEEE Journal on Selected Areas in Communications* 21(1) (2003)
21. Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation* 1(14), 59–91 (2001)
22. Sutherland, D.: A model of information. In: Proceedings of the 9th National Computer Security Conference (1986)
23. van der Meyden, R., Zhang, C.: Algorithmic verification of noninterference properties. *Electron. Notes Theor. Comput. Sci.* 168, 61–75 (2007)
24. Zakinthinos, A., Lee, E.S.: A general theory of security properties. In: SP 1997: Proceedings of the 1997 IEEE Symposium on Security and Privacy, Washington, DC, USA, p. 94. IEEE Computer Society Press, Los Alamitos (1997)

Synthesizing Enforcement Monitors wrt. the Safety-Progress Classification of Properties

Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier

VERIMAG, Université Grenoble I, INPG, CNRS
Firstname.Lastname@imag.fr

Abstract. Runtime enforcement is a powerful technique to ensure that a program will respect a given security policy. We extend previous works on this topic in several directions. Firstly, we propose a generic notion of enforcement monitors based on a memory device and finite sets of control states and enforcement operations. Moreover, we specify their enforcement abilities w.r.t. the *general* safety-progress classification of properties. It allows a fine-grain characterization of the space of enforceable properties. Finally, we propose a *systematic* technique to produce an enforcement monitor from the Streett automaton recognizing a given safety, guarantee, obligation or response security property.

1 Introduction

The growing complexity of nowadays programs and systems induces a rise of needs in validation. With the enhancement of engineering methods, software components tend to be more and more reusable. Although this trend clearly improves programmers productivity, it also raises some important security issues. Indeed, each individual component should guarantee some individual properties at run-time to ensure that the whole application will respect some given security policy. When retrieving an external component, the question of how this code meets a set of proper requirements raises. Using formal methods appears as a solution to provide techniques to regain the needed confidence. However, these techniques should remain practical enough to be adopted by software engineers.

Runtime monitoring falls in this category. It consists in supervising at runtime the execution of an underlying program against a set of expected properties. With an appointed monitor, one is able to detect any occurrence of specific property violations. Such a detection might be a sufficient assurance. However, for certain kind of systems a misbehavior might be not acceptable. To prevent this, a possible solution is then to *enforce* the desired property: the monitor not only observe the current program execution, but it also controls it in order to ensure that the expected property is fulfilled. Such a control should usually remain *transparent*, meaning that it should leave any original execution sequence unchanged when already correct, or output its *longest correct prefix* otherwise.

Runtime enforcement monitoring was initiated by the work of Schneider [1] on what has been called *security automata*. In this work the monitors watch the current execution sequence and halt the underlying program whenever it deviates from the desired property. Such security automata are able to enforce the class of safety properties [2], stating that *something bad can never happen*. Later, Viswanathan [3] noticed that the class of enforceable properties is impacted by the computational power of the enforcement monitor: since the enforcement mechanism cannot implement more than computable functions, only decidable properties can be enforced. More recently [4,5], Ligatti and al. showed that it is possible to enforce at runtime more than safety properties. Using a more powerful enforcement mechanism called *edit-automata*, it is possible to enforce the larger class of *infinite renewal properties*, able to express some kinds of *obligations* used in security policies. More than simply halting an underlying program, edit-automata can also “suppress” (i.e., froze) and “insert” (frozen) actions in the current execution sequence. To better cope with practical resource constraints, Fong [6] studied the effect of memory limitations on enforcement mechanisms. He introduced the notion of *Shallow History Automata* which are only aware of the occurrence of past events, and do not keep any information about the order of their arrival. He showed that such a “shallow history” indeed leads to some computational limitations for the enforced properties. However, many interesting properties remain enforceable using shallow history automata.

In this paper, we propose to extend these previous works in several directions. Firstly, we study the enforcement capabilities relatively to the so-called *safety-progress* hierarchy of properties [7,8]. This classification differs from the more classical safety-liveness classification [9,10] by offering a rather clear characterization of a number of interesting kinds of properties (*e.g.* obligation, accessibility, justice, etc.), particularly relevant in the security context. Moreover this classification features properties, such as *transactional properties* (i.e, an action pattern to be repeated infinitely), which are neither safety nor liveness properties. Thus, using this classification as a basis provides a finer-grain classification of enforceable properties. Moreover, in this safety-progress hierarchy, each property φ can be characterized by a particular kind of (finite state) recognizing automaton \mathcal{A}_φ . Secondly, we show how to generate an enforcement monitor for φ in a *systematic way*, from a recognizing automaton \mathcal{A}_φ . This enforcement monitor is based on a *finite set of control states*, and an *auxiliary memory*. This general notion of enforcement monitor encompasses the previous notions of security automata, edit-automata and “shallow history” automata. The companion report [11] exposes more details and complete proofs of the theorems of this paper.

The remainder of this article is organized as follows. The Sect. 2 introduces some preliminary notions for our work. In Sect. 3 we recall briefly the necessary elements from the safety-progress classification of properties. Then, we present our notion of enforcement monitor and their properties in Sect. 4. The Sect. 5 studies the enforcement capability wrt. the safety-progress classification. Sect. 6 discusses some implementation issues. Finally, the Sect. 7 exposes some concluding remarks.

2 Preliminaries and Notations

This section introduces some preliminary notations, namely the notions of *program execution sequences* and *program properties* we will consider in the remainder of this article.

2.1 Sequences, and Execution Sequences

Sequences. Considering a finite set of elements E , we define notations about sequences of elements belonging to E . A sequence σ containing elements of E is formally defined by a function $\sigma : \mathbb{N} \rightarrow E$ where \mathbb{N} is the set of natural numbers. We denote by E^* the set of finite sequences over E (partial function from \mathbb{N}), and by E^ω the set of infinite sequences over E (total function from \mathbb{N}). The set $E^\infty = E^* \cup E^\omega$ is the set of all sequences (finite or not) over E . The empty sequence is denoted ϵ . The length (number of elements) of a finite sequence σ is noted $|\sigma|$ and the $(i + 1)$ -th element of σ is denoted by σ_i . For two sequences $\sigma \in E^*, \sigma' \in E^\infty$, we denote by $\sigma \cdot \sigma'$ the concatenation of σ and σ' , and by $\sigma \prec \sigma'$ (resp. $\sigma' \succ \sigma$) the fact that σ is a strict prefix of σ' (resp. σ' is a strict suffix of σ). The sequence σ is said to be a strict prefix of σ' when $\forall i \in \{0, \dots, |\sigma| - 1\} \cdot \sigma_i = \sigma'_i$. When $\sigma' \in E^*$, we note $\sigma \preceq \sigma' \stackrel{\text{def}}{=} \sigma \prec \sigma' \vee \sigma = \sigma'$. For $\sigma \in E^\infty$, we will need to designate its subsequences.

Execution sequences. A program \mathcal{P} is considered as a generator of execution sequences. We are interested in a restricted set of operations the program can perform. These operations influence the truth value of properties the program is supposed to fulfill. We abstract these operations by a finite set of *events*, namely a vocabulary Σ . We denote by \mathcal{P}_Σ a program for which the vocabulary is Σ . The set of execution sequences of \mathcal{P}_Σ is denoted $Exec(\mathcal{P}_\Sigma) \subseteq \Sigma^\infty$. This set is *prefix-closed*, that is $\forall \sigma \in Exec(\mathcal{P}_\Sigma), \sigma' \in \Sigma^* \cdot \sigma' \preceq \sigma \Rightarrow \sigma' \in Exec(\mathcal{P}_\Sigma)$.

Such execution sequences can be made of access events on a secure system to its resources, or kernel operations on an operating system. In a software context, these events may be abstractions of relevant instructions such as variable modifications or procedure calls.

2.2 Properties

Properties as sets of execution sequences. In this paper we aim to enforce properties on program. A property φ is defined as a set of execution sequences, *i.e.* $\varphi \subseteq \Sigma^\infty$. Considering a given execution sequence σ , when $\sigma \in \varphi$ (noted $\varphi(\sigma)$), we say that σ *satisfies* φ . A consequence of this definition is that properties we will consider are restricted to *single* execution sequences, excluding specific properties defined on powersets of execution sequences.

Reasonable properties. As noticed in [4], a property can be effectively enforced at runtime only if it is *reasonable* in the following sense: it should be satisfied by the empty sequence, and it should be decidable. This means that a program

which performs no action should not violate this property, and that deciding whether any finite execution sequence satisfies or not this property should be a computable function ([3]).

3 A Safety-Progress Classification of Properties

This section recalls and extends some results about the safety-progress [7,8,12] classification of properties. In the original papers this classification introduced a hierarchy between properties defined as *infinite* execution sequences. We extend here this classification to deal also with finite-length execution sequences.

3.1 Generalities about the Classification

The safety-progress classification is constituted of four basic classes defined over infinite execution sequences:

- *safety* properties are the properties for which whenever a sequence satisfies a property, *all its prefixes* satisfy this property.
- *guarantee* properties are the properties for which whenever a sequence satisfies a property, *there are some prefixes* (at least one) satisfying this property.
- *response* properties are the properties for which whenever a sequence satisfies a property, *an infinite number of its prefixes* satisfy this property.
- *persistence* properties are the properties for which whenever a sequence satisfies a property, *all its prefixes* continuously satisfy this property from a certain point.

Furthermore, two extra classes can be defined as (finite) boolean combinations of basic classes.

- The *obligation class* is the class obtained by positive boolean combinations of safety and guarantee properties.
- The *reactivity class* is the class obtained by boolean combinations of response and persistence properties.

Example 1. Let consider an operating system where a given operation *op* is allowed only when an authorization *auth* has been granted before. Then,

- the property φ_1 stating that “an authorization grant *grant_auth* should precede any occurrence of *op*” is a *safety* property;
- the property φ_2 stating that “the next authorization request *req_auth* should be eventually followed by a grant (*grant_auth*) or a deny (*deny_auth*)” is a *guarantee* property;
- the property φ_3 stating that “the system should run forever, unless a *deny_auth* is issued, meaning that every user should be disconnected and the system should terminate” is an *obligation* property;
- the property φ_4 stating that “each occurrence of *req_auth* should be first written in a log file and then answered either with a *grant_auth* or a *deny_auth* without any occurrence of *op* in the meantime” is a *response* property;

- the property φ_5 stating that “an incorrect use of operation op should imply that any future call to req_auth will always result in a $deny_auth$ answer” is a *persistence* property.

The safety-progress classification is an alternative to the more classical safety-liveness [9,10] dichotomy. Unlike this later, the safety-progress classification is a hierarchy and not a partition. It provides a finer-grain classification, and the properties of each class can be characterized according to four *views* [7]. We shall consider here only the so-called *automata view*.

3.2 The Automata View

First, we define a notion of *property recognizer* using a variant of deterministic and complete Streett automata (introduced in [13] and used in [7]).

Definition 1 (Streett Automaton). *A deterministic Streett automaton is a tuple $(Q, q_{\text{init}}, \Sigma, \longrightarrow, \{(R_1, P_1), \dots, (R_m, P_m)\})$ defined relatively to a set of events Σ . The set Q is the set of automaton states, where $q_{\text{init}} \in Q$ is the initial state. The total function $\longrightarrow: Q \times \Sigma \rightarrow Q$ is the transition function. In the following, for $q, q' \in Q, e \in \Sigma$ we abbreviate $\longrightarrow(q, e) = q'$ by $q \xrightarrow{e} q'$. The set $\{(R_1, P_1), \dots, (R_m, P_m)\}$ is the set of accepting pairs, in which for all $i \leq m$, $R_i \subseteq Q$ are the sets of recurrent states, and $P_i \subseteq Q$ are the sets of persistent states.*

We refer an automaton with m accepting pairs as a m -automaton. When $m = 1$, a 1-automaton is also called a *plain*-automaton, and we refer R_1 and P_1 as R and P . In the following (otherwise mentioned) $\sigma \in \Sigma^\omega$ designates an execution sequence of a program, and $\mathcal{A} = (Q^\mathcal{A}, q_{\text{init}}^\mathcal{A}, \Sigma, \longrightarrow_\mathcal{A}, \{(R_1, P_1), \dots, (R_m, P_m)\})$ a deterministic Streett m -automaton.

The *run* of σ on \mathcal{A} is the sequence of states involved by the execution of σ on \mathcal{A} . It is formally defined as $run(\sigma, \mathcal{A}) = q_0 \cdot q_1 \cdot \dots$ where $\forall i \cdot (q_i \in Q^\mathcal{A} \wedge q_i \xrightarrow{\sigma_i} q_{i+1}) \wedge q_0 = q_{\text{init}}^\mathcal{A}$. The *trace* resulting in the execution of σ on \mathcal{A} is the unique sequence (finite or not) of tuples $(q_0, \sigma_0, q_1) \cdot (q_1, \sigma_1, q_2) \cdot \dots$ where $run(\sigma, \mathcal{A}) = q_0 \cdot q_1 \cdot \dots$. We denote by $vinf(\sigma, \mathcal{A})$ (or $vinf(\sigma)$ when clear from context) the set of states appearing infinitely often in $run(\sigma, \mathcal{A})$. This set is formally defined as follows: $vinf(\sigma, \mathcal{A}) = \{q \in Q^\mathcal{A} \mid \forall n \in \mathbb{N}, \exists m \in \mathbb{N} \cdot m > n \wedge q = q_m \text{ with } run(\sigma, \mathcal{A}) = q_0 \cdot q_1 \cdot \dots\}$.

The following definition tells whether an execution sequence is accepted or not by a Streett automaton:

Definition 2 (Acceptance Condition of a Street Automaton)

For an infinite execution sequence $\sigma \in \Sigma^\omega$, we say that \mathcal{A} accepts σ if $\forall i \in \{1, \dots, m\} \cdot vinf(\sigma, \mathcal{A}) \cap R_i \neq \emptyset \vee vinf(\sigma, \mathcal{A}) \subseteq P_i$.

For a finite execution sequence $\sigma \in \Sigma^$ such that $|\sigma| = n$, we say that the m -automaton \mathcal{A} accepts σ if either $\sigma = \epsilon$ or $(\exists q_0, \dots, q_n \in Q^\mathcal{A} \cdot run(\sigma, \mathcal{A}) = q_0 \cdot \dots \cdot q_n \wedge q_0 = q_{\text{init}}^\mathcal{A} \text{ and } \forall i \in \{1, \dots, m\} \cdot q_n \in P_i \cup R_i)$.*

Note that this definition of acceptance condition for finite sequences matches the definition of finitary properties defined in [7] (see [11] for more details).

The hierarchy of automata. Each class of the safety-progress classification is characterized by setting syntactic restrictions on a deterministic Streett automaton.

- A *safety automaton* is a plain automaton such that $R = \emptyset$ and there is no transition from a state $q \in \overline{P}$ to a state $q' \in P$.
- A *guarantee automaton* is a plain automaton such that $P = \emptyset$ and there is no transition from a state $q \in R$ to a state $q' \in \overline{R}$.
- An *m-obligation automaton* is an *m*-automaton such that for each i in $\{1, \dots, m\}$:
 - there is no transition from $q \in \overline{P}_i$ to $q' \in P_i$,
 - there is no transition from $q \in R_i$ to $q' \in \overline{R}_i$,
- A *response automaton* is a plain automaton such that $P = \emptyset$
- A *persistence automaton* is a plain automaton such that $R = \emptyset$.
- A *m-reactivity automaton* is any unrestricted *m*-automaton.

Automata and properties. We say that a Streett automaton \mathcal{A}_φ defines a property φ (defined as a subset of Σ^∞) if and only if the set of execution sequences accepted by \mathcal{A}_φ is equal to φ . Conversely, a property $\varphi \subseteq \Sigma^\infty$ is said to be *specifiable* by an automaton if the set of execution sequences accepted by the automaton is φ . A property φ that is specifiable by an automaton is a κ -property iff the automaton is a κ -automaton, where $\kappa \in \{\text{safety, guarantee, obligation, response, persistence, reactivity}\}$. In the following we note $\text{Safety}(\Sigma)$ (resp. $\text{Guarantee}(\Sigma)$, $\text{Obligation}(\Sigma)$, $\text{Response}(\Sigma)$, $\text{Persistence}(\Sigma)$, $\text{Reactivity}(\Sigma)$) the set of safety (resp. guarantee, obligation, response, persistence, reactivity) properties defined over Σ .

4 Property Enforcement Via Enforcement Monitors

A program \mathcal{P} is considered as a generator of execution sequences. We want to build an enforcement monitor (EM) for a property φ such that the two following constraints hold:

Soundness: any execution sequence allowed by the EM should satisfy φ ;

Transparency: execution sequences of \mathcal{P} should be modified in a minimal way, namely if a sequence already satisfies φ it should remain unchanged, otherwise its *longest prefix* satisfying φ should be allowed by the EM.

4.1 Enforcement Monitors

We define now the central notion of enforcement monitor. Such a runtime device monitors a target program by observing relevant events and performing some enforcement operation depending on its internal state.

Definition 3 (Enforcement Monitor (EM)). An enforcement monitor \mathcal{A}_\downarrow is a 4-tuple $(Q^{\mathcal{A}_\downarrow}, q_{\text{init}}^{\mathcal{A}_\downarrow}, \text{Stop}^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow})$ defined relatively to a set of events Σ and a set of enforcement operations Ops . The finite set $Q^{\mathcal{A}_\downarrow}$ denotes the control

states, $q_{\text{init}}^{\mathcal{A}_\downarrow} \in Q^{\mathcal{A}_\downarrow}$ is the initial state and $\text{Stop}^{\mathcal{A}_\downarrow}$ is the set of stopping states ($\text{Stop}^{\mathcal{A}_\downarrow} \subseteq Q^{\mathcal{A}_\downarrow}$). The partial function (but complete wrt. $Q^{\mathcal{A}_\downarrow} \times \Sigma$) $\longrightarrow_{\mathcal{A}_\downarrow} : Q^{\mathcal{A}_\downarrow} \times \Sigma \times \text{Ops} \rightarrow Q^{\mathcal{A}_\downarrow}$ is the transition function. In the following we abbreviate $\longrightarrow_{\mathcal{A}_\downarrow} (q, a, \alpha) = q'$ by $q \xrightarrow{a/\alpha}_{\mathcal{A}_\downarrow} q'$. We also assume that outgoing transitions from a stopping state only lead to another stopping state: $\forall q \in \text{Stop}^{\mathcal{A}_\downarrow} \cdot \forall a \in \Sigma \cdot \forall \alpha \in \text{Ops} \cdot \forall q' \in Q^{\mathcal{A}_\downarrow} \cdot q \xrightarrow{a/\alpha}_{\mathcal{A}_\downarrow} q' \Rightarrow q' \in \text{Stop}^{\mathcal{A}_\downarrow}$.

Notions of *run* and *trace* (see Sect. 3.2) are naturally transposed from Streett automata. In the remainder of this section, $\sigma \in \Sigma^\infty$ designates an execution sequence of a program, and $\mathcal{A}_\downarrow = (Q^{\mathcal{A}_\downarrow}, q_{\text{init}}^{\mathcal{A}_\downarrow}, \text{Stop}^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow})$ designates an EM.

Typical enforcement operations allow the EM either to halt the target program (when the current input sequence irreparably violates the property), or to store the current event in a *memory device* (when a decision has to be postponed), or to dump the content of the memory device (when the target program comes back to a correct behavior). We first give a more precise definition of such enforcement operations, then we formalize the way an EM reacts to an input sequence provided by a target program through the standard notions of *configuration* and *derivation*.

Definition 4 (Enforcement Operations Ops). *Enforcement operations take as inputs an event and a memory content (i.e., a sequence of events) to produce a new memory content and an output sequence: $\text{Ops} \subseteq 2^{((\Sigma \cup \{\epsilon\}) \times \Sigma^*) \rightarrow (\Sigma^* \times \Sigma^*)}$. In the following we consider a set $\text{Ops} = \{\text{halt}, \text{store}, \text{dump}\}$ defined as follows: $\forall a \in \Sigma \cup \{\epsilon\}, \forall m \cdot \Sigma^*$*

$$\text{halt}(a, m) = (\epsilon, m) \quad \text{store}(a, m) = (\epsilon, m.a) \quad \text{dump}(a, m) = (m.a, \epsilon)$$

In the following we assume that outgoing transitions from a stopping state are all labeled with an *halt* operation. That is: $\forall q \in \text{Stop}^{\mathcal{A}_\downarrow} \cdot \forall a \in \Sigma \cdot \forall \alpha \in \text{Ops} \cdot \forall q' \in Q^{\mathcal{A}_\downarrow} \cdot q \xrightarrow{a/\alpha}_{\mathcal{A}_\downarrow} q' \Rightarrow \alpha = \text{halt}$.

Definition 5 (EM Configurations and Derivations). *For an EM $\mathcal{A}_\downarrow = (Q^{\mathcal{A}_\downarrow}, q_{\text{init}}^{\mathcal{A}_\downarrow}, \text{Stop}^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow})$, a configuration is a triplet $(q, \sigma, m) \in Q^{\mathcal{A}_\downarrow} \times \Sigma^* \times \Sigma^*$ where q denotes the current control state, σ the current input sequence, and m the current memory content.*

We say that a configuration (q', σ', m') is derivable in one step from the configuration (q, σ, m) and produces the output $o \in \Sigma^$, and we note $(q, \sigma, m) \xrightarrow{o}_{\mathcal{A}_\downarrow} (q', \sigma', m')$ if and only if $\sigma = a.\sigma' \wedge q \xrightarrow{a/\alpha}_{\mathcal{A}_\downarrow} q' \wedge \alpha(a, m) = (o, m')$.*

We say that a configuration C' is derivable in several steps from a configuration C and produces the output $o \in \Sigma^$, and we note $C \xRightarrow{o}_{\mathcal{A}_\downarrow} C'$, if and only if there exists $k \geq 0$ and configurations C_0, C_1, \dots, C_k such that $C = C_0, C' = C_k, C_i \xrightarrow{o_i}_{\mathcal{A}_\downarrow} C_{i+1}$ for all $0 \leq i < k$, and $o = o_0 \cdot o_1 \cdots o_{k-1}$.*

Besides, the configuration C is derivable from itself in one step and produces the output ϵ , we note $C \xRightarrow{\epsilon}_{\mathcal{A}_\downarrow} C$.

4.2 Enforcing a Property

We now describe how an EM can enforce a property on a given program, namely how it transforms an input sequence σ into an output sequence o by performing derivation steps from its initial state. For the upcoming definitions we will distinguish between finite and infinite sequences and we consider an EM $\mathcal{A}_\downarrow = (Q^{\mathcal{A}_\downarrow}, q_{\text{init}}^{\mathcal{A}_\downarrow}, \text{Stop}^{\mathcal{A}_\downarrow}, \longrightarrow_{\mathcal{A}_\downarrow})$.

Definition 6 (Sequence Transformation). *We say that:*

- The sequence $\sigma \in \Sigma^*$ is transformed by \mathcal{A}_\downarrow into the sequence $o \in \Sigma^*$, which is noted $(q_{\text{init}}^{\mathcal{A}_\downarrow}, \sigma) \Downarrow_{\mathcal{A}_\downarrow} o$, if $\exists q \in Q^{\mathcal{A}_\downarrow}, m \in \Sigma^*$ such that $(q_{\text{init}}^{\mathcal{A}_\downarrow}, \sigma, \epsilon) \xrightarrow{o}_{\mathcal{A}_\downarrow} (q, \epsilon, m)$.
- The sequence $\sigma \in \Sigma^\omega$ is transformed by \mathcal{A}_\downarrow into the sequence $o \in \Sigma^\infty$, which is noted $(q_{\text{init}}^{\mathcal{A}_\downarrow}, \sigma) \Downarrow_{\mathcal{A}_\downarrow} o$, if $\forall \sigma' \in \Sigma^* \cdot \sigma' \prec \sigma \cdot (\exists o' \in \Sigma^* \cdot (q_{\text{init}}^{\mathcal{A}_\downarrow}, \sigma') \Downarrow_{\mathcal{A}_\downarrow} o' \wedge o' \preceq o)$.

Definition 7 (Property-Enforcement). *Let consider a property φ , we say that \mathcal{A}_\downarrow enforces the property φ on a program \mathcal{P}_Σ (noted $\text{Enf}(\mathcal{A}_\downarrow, \varphi, \mathcal{P}_\Sigma)$) iff*

- $\forall \sigma \in \text{Exec}(\mathcal{P}_\Sigma) \cap \Sigma^*, \exists o \in \Sigma^* \cdot \text{enforced}(\sigma, o, \mathcal{A}_\downarrow, \varphi)$, where the predicate $\text{enforced}(\sigma, o, \mathcal{A}_\downarrow, \varphi)$ is the conjunction of the following constraints:

$$(q_{\text{init}}^{\mathcal{A}_\downarrow}, \sigma) \Downarrow_{\mathcal{A}_\downarrow} o \tag{1}$$

$$\varphi(o) \tag{2}$$

$$\varphi(\sigma) \Rightarrow \sigma = o \tag{3}$$

$$\neg \varphi(\sigma) \Rightarrow \left(\exists \sigma' \prec \sigma \cdot \varphi(\sigma') \wedge o = \sigma' \wedge (\nexists \sigma'' \succ \sigma' \cdot \varphi(\sigma'') \wedge \sigma'' \preceq \sigma) \right) \tag{4}$$

- $\forall \sigma' \in \text{Exec}(\mathcal{P}_\Sigma) \cap \Sigma^\omega, \forall \sigma \prec \sigma', \exists o \in \Sigma^* \cdot \text{enforced}(\sigma, o, \mathcal{A}_\downarrow, \varphi)$.

(1) stipulates that the sequence σ is transformed by \mathcal{A}_\downarrow into a sequence o , (2) states that o satisfies the property φ , (3) ensures transparency of \mathcal{A}_\downarrow , i.e. if σ satisfied already the property then it is not transformed, and (4) ensures in the case where σ does not satisfy φ that o is the longest prefix of σ satisfying the property.

Example 2 (Enforcement monitor). We provide on Fig. 1 two EMs to illustrate the enforcement of some of the properties introduced in example 1.

- The left-hand side of Fig. 1 is an EM $\mathcal{A}_{\downarrow \varphi_1}$ for the safety property φ_1 . We assume here that the set Σ of relevant events is $\{op, grant_auth\}$ (other events are ignored by the EM). $\mathcal{A}_{\downarrow \varphi_1}$ has one stopping state, $\text{Stop} = \{2\}$, and its initial state is 1. From this initial state it simply *dumps* a first occurrence of *grant_auth* and moves to state 3, where all events of Σ are allowed (i.e., *dumped*). Otherwise, if event *op* occurs first, then it moves to state 2 and halts the underlying program forever.

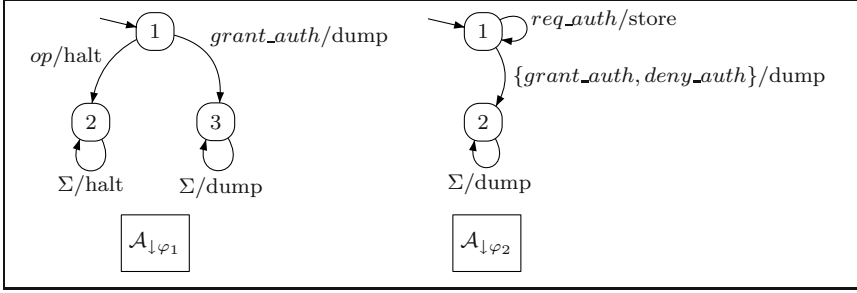


Fig. 1. Two examples of EMs

- The right-hand side of Fig. 1 is an EM $\mathcal{A}_{\downarrow\varphi_2}$ for the guarantee property φ_2 . We assume here that the set Σ of relevant events is $\{req_auth, grant_auth, deny_auth\}$. The initial state of $\mathcal{A}_{\downarrow\varphi_2}$ is state 1, and it has no stopping states. Its behavior is the following: occurrences of *req_auth* are *stored* in memory as long as *grant_auth* or *deny_auth* does not occur, then the whole memory content is *dumped*. This ensures that the output sequence always satisfies the property under consideration.

5 Enforcement wrt. the Safety-Progress Classification

We now study how to practically enforce properties of the safety-progress hierarchy (Sect. 3). More precisely, we show which classes of properties can be effectively enforced by an EM, and more important, we provide a *systematic construction* of an EM enforcing a property φ from the Streett automaton defining this property. This construction technique is specific to each class of properties.

5.1 From a Recognizing Automaton to an Enforcement Monitor

We define four general operations whose purpose is to transform a Streett automaton recognizing a safety (resp. guarantee, obligation, response) property into an enforcement monitor enforcing the same property.

Definition 8 (Safety Transformation). *Given a Streett safety-automaton $\mathcal{A}_\varphi = (Q^{\mathcal{A}_\varphi}, q_{\text{init}}^{\mathcal{A}_\varphi}, \Sigma, \longrightarrow_{\mathcal{A}_\varphi}, (\emptyset, P))$ recognizing a reasonable safety property φ defined over a language Σ , we define a transformation (named TransSafety) of this automaton into an enforcement monitor $\mathcal{A}_{\downarrow\varphi} = (Q^{\mathcal{A}_{\downarrow\varphi}}, q_{\text{init}}^{\mathcal{A}_{\downarrow\varphi}}, \text{Stop}^{\mathcal{A}_{\downarrow\varphi}}, \longrightarrow_{\mathcal{A}_{\downarrow\varphi}})$ such that:*

- $Q^{\mathcal{A}_{\downarrow\varphi}} = Q^{\mathcal{A}_\varphi}$, $q_{\text{init}}^{\mathcal{A}_{\downarrow\varphi}} = q_{\text{init}}^{\mathcal{A}_\varphi}$, with $q_{\text{init}}^{\mathcal{A}_\varphi} \in P$
- $\text{Stop}^{\mathcal{A}_{\downarrow\varphi}} = \overline{P}$
- the transition relation $\longrightarrow_{\mathcal{A}_{\downarrow\varphi}}$ is defined from $\longrightarrow_{\mathcal{A}_\varphi}$ as the smallest relation verifying the following rules:

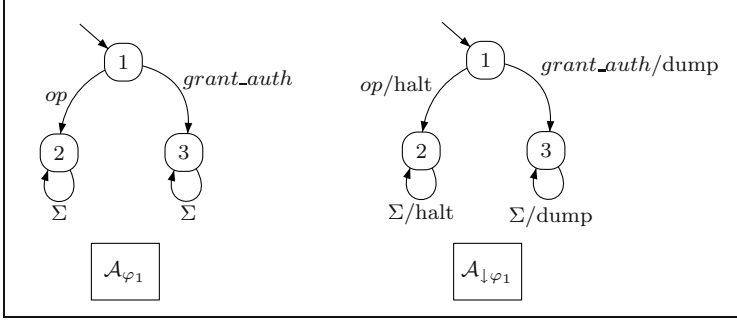


Fig. 2. Recognizing automaton and EM for the safety property φ_1

- $q \xrightarrow{a/dump} \mathcal{A}_{\downarrow\varphi} q'$ if $q' \in P \wedge q \xrightarrow{a} \mathcal{A}_{\varphi} q'$
- $q \xrightarrow{a/halt} \mathcal{A}_{\downarrow\varphi} q'$ if $q' \notin P \wedge q \xrightarrow{a} \mathcal{A}_{\varphi} q'$

Note that there is no transition (in the Streett automaton) from $q \notin P$ to $q' \in P$, and $R = \emptyset$. We note $\mathcal{A}_{\downarrow\varphi} = \text{TransSafety}(\mathcal{A}_{\varphi})$.

Informally, the behavior of an EM $\mathcal{A}_{\downarrow\varphi}$ obtained from $\text{TransSafety}(\mathcal{A}_{\varphi})$ can be understood as follows. While the current execution sequence satisfies the underlying property (i.e while \mathcal{A}_{φ} remains in P -states), it dumps each input event. Once the execution sequence deviates from the property (i.e., when \mathcal{A}_{φ} reaches a \overline{P} -state), then it halts immediately the underlying program with a *halt* operation. The following example illustrates this principle.

Example 3 (Safety Transformation). Fig. 2 (left-hand side) depicts a Streett automaton defining the safety property φ_1 of example 1. Its set of states is $\{1, 2, 3\}$, the initial state is 1, and we have $R = \emptyset$ and $P = \{1, 3\}$. The right-hand side shows the corresponding EM obtained using transformation TransSafety .

We now define a similar transformation for *guarantee* properties. The TransGuarantee transformation uses the set $\text{Reach}_{\mathcal{A}_{\varphi}}(q)$ of reachable states from a state q . Given a Street automaton \mathcal{A}_{φ} with a set of states $Q^{\mathcal{A}_{\varphi}}$, we have $\forall q \in Q^{\mathcal{A}_{\varphi}} \cdot \text{Reach}_{\mathcal{A}_{\varphi}}(q) = \{q' \in Q^{\mathcal{A}_{\varphi}} \mid \exists (q_i)_i, (a_i)_i, \cdot q \xrightarrow{a_0} \mathcal{A}_{\varphi} q_0 \xrightarrow{a_1} \mathcal{A}_{\varphi} q_1 \cdots q'\}$.

Definition 9 (Guarantee Transformation). Let consider a Streett guarantee-automaton $\mathcal{A}_{\varphi} = (Q^{\mathcal{A}_{\varphi}}, q_{\text{init}}^{\mathcal{A}_{\varphi}}, \Sigma, \longrightarrow_{\mathcal{A}_{\varphi}}, (R, \emptyset))$ recognizing a property $\varphi \in \text{Guarantee}(\Sigma)$. We define a transformation TransGuarantee of this automaton into an EM $\mathcal{A}_{\downarrow\varphi} = (Q^{\mathcal{A}_{\downarrow\varphi}}, q_{\text{init}}^{\mathcal{A}_{\downarrow\varphi}}, \text{Stop}^{\mathcal{A}_{\downarrow\varphi}}, \longrightarrow_{\mathcal{A}_{\downarrow\varphi}})$ such that:

- $Q^{\mathcal{A}_{\downarrow\varphi}} = Q^{\mathcal{A}_{\varphi}}, q_{\text{init}}^{\mathcal{A}_{\downarrow\varphi}} = q_{\text{init}}^{\mathcal{A}_{\varphi}},$
- $\text{Stop}^{\mathcal{A}_{\downarrow\varphi}} = \{q \in Q^{\mathcal{A}_{\downarrow\varphi}} \mid \nexists q' \in \text{Reach}_{\mathcal{A}_{\varphi}}(q) \wedge q' \in R\}$
- the transition relation $\longrightarrow_{\mathcal{A}_{\downarrow\varphi}}$ is defined from $\longrightarrow_{\mathcal{A}_{\varphi}}$ as the smallest relation verifying the following rules:

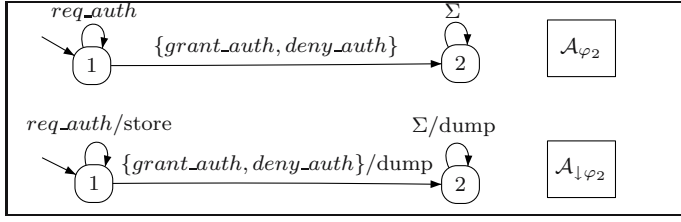


Fig. 3. A guarantee-automaton and the corresponding EM for property φ_2

- $q \xrightarrow{a/dump} \mathcal{A}_{\downarrow \varphi} q'$ if $q' \in R \wedge q \xrightarrow{a} \mathcal{A}_{\varphi} q'$
- $q \xrightarrow{a/halt} \mathcal{A}_{\downarrow \varphi} q'$ if $q' \notin R \wedge q \xrightarrow{a} \mathcal{A}_{\varphi} q' \wedge \nexists q'' \in R \cdot q'' \in Reach_{\mathcal{A}_{\varphi}}(q')$
- $q \xrightarrow{a/store} \mathcal{A}_{\downarrow \varphi} q'$ if $q' \notin R \wedge q \xrightarrow{a} \mathcal{A}_{\varphi} q' \wedge \exists q'' \in R \cdot q'' \in Reach_{\mathcal{A}_{\varphi}}(q')$

Note that there is no transition from $q \in R$ to $q' \in \overline{R}$. And, as $P = \emptyset$, we do not have transition from $q \in P$ to $q' \in P$. We note $\mathcal{A}_{\downarrow \varphi} = \text{TransGuarantee}(\mathcal{A}_{\varphi})$.

An EM $\mathcal{A}_{\downarrow \varphi}$ obtained from $\text{TransGuarantee}(\mathcal{A}_{\varphi})$ behaves as follows. While the current execution sequence does not satisfy the underlying property (i.e., while \mathcal{A}_{φ} remains in \overline{R} -states), it stores each entered event in its memory. Once, the execution sequence satisfies the property (i.e., when \mathcal{A}_{φ} reaches an R -state), it dumps the content of the memory and the current event. The following example illustrates this principle.

Example 4 (Guarantee Transformation). Fig. 3 (up) shows a Streett automaton recognizing the guarantee property φ_2 of example 1. Its set of states is $\{1, 2\}$, the initial state is 1, and we have $R = \{2\}$ and $P = \emptyset$. At the bottom is depicted the EM enforcing this same property, obtained by the TransGuarantee transformation. One can notice that this EM has no stopping state.

We now define a transformation for obligation properties. Informally the TransObligation transformation combines the effects of the two previously introduced transformations on using information of each accepting pair.

Definition 10 (Obligation Transformation). Let consider a Streett obligation-automaton $\mathcal{A}_{\varphi} = (Q^{\mathcal{A}_{\varphi}}, q_{\text{init}}^{\mathcal{A}_{\varphi}}, \Sigma, \rightarrow_{\mathcal{A}_{\varphi}}, \{(R_1, P_1), \dots, (R_m, P_m)\})$ recognizing an obligation property $\varphi \in \text{Obligation}(\Sigma)$ defined over a language Σ . We define a transformation TransObligation of this automaton into an EM $\mathcal{A}_{\downarrow \varphi} = (Q^{\mathcal{A}_{\downarrow \varphi}}, q_{\text{init}}^{\mathcal{A}_{\downarrow \varphi}}, \text{Stop}^{\mathcal{A}_{\downarrow \varphi}}, \rightarrow_{\mathcal{A}_{\downarrow \varphi}})$ such that:

- $Q^{\mathcal{A}_{\downarrow \varphi}} = Q^{\mathcal{A}_{\varphi}}, q_{\text{init}}^{\mathcal{A}_{\downarrow \varphi}} = q_{\text{init}}^{\mathcal{A}_{\varphi}},$
- $\text{Stop}^{\mathcal{A}_{\downarrow \varphi}} = \bigcup_{i=1}^m (\overline{P_i} \cap \{q \in Q^{\mathcal{A}_{\downarrow \varphi}} \mid \nexists q' \in R_i \wedge q' \in Reach_{\mathcal{A}_{\varphi}}(q)\})$
- the transition relation $\rightarrow_{\mathcal{A}_{\downarrow \varphi}}$ is defined from $\rightarrow_{\mathcal{A}_{\varphi}}$ as the smallest relation verifying the following rule:
 $q \xrightarrow{a/\alpha} \mathcal{A}_{\downarrow \varphi} q'$ if $q \xrightarrow{a} \mathcal{A}_{\varphi} q'$ and $\alpha = \prod_{i=1}^m (\sqcup (\beta_i, \gamma_i))$ where

- \sqcap, \sqcup designate respectively the infimum and the supremum with respect to the complete lattice (Ops, \sqsubseteq) , where $halt \sqsubseteq store \sqsubseteq dump$ (\sqsubseteq is a total order),
- and the β_i and γ_i are obtained in the following way:
 - * $\beta_i = dump$ if $q' \in P_i$
 - * $\beta_i = halt$ if $q' \notin P_i$
 - * $\gamma_i = dump$ if $q' \in R_i$
 - * $\gamma_i = halt$ if $q' \notin R_i \wedge \nexists q'' \in R_i \cdot q'' \in Reach_{\mathcal{A}_\varphi}(q')$
 - * $\gamma_i = store$ if $q' \notin R_i \wedge \exists q'' \in R_i \cdot q'' \in Reach_{\mathcal{A}_\varphi}(q')$

Note that there is no transition from $q \in R_i$ to $q' \in \overline{R_i}$, and no transition from $q \in \overline{P_i}$ to $q' \in P_i$.

Finding a transformation for a *response* property φ needs to slightly extend the definition of TransGuarantee to deal with transitions of a Streett automaton leading from states belonging to R to states belonging to \overline{R} (since such transitions are absent when φ is a *guarantee* property). Therefore, we introduce a new transformation called TransResponse obtained from the TransGuarantee transformation (Def. 9) by adding a rule to deal with the aforementioned difference.

Definition 11 (Response Transformation). Let consider a Streett response-automaton $\mathcal{A}_\varphi = (Q^{\mathcal{A}_\varphi}, q_{init}^{\mathcal{A}_\varphi}, \Sigma, \longrightarrow_{\mathcal{A}_\varphi}, (R, \emptyset))$ recognizing a response property $\varphi \in Response(\Sigma)$ defined over a language Σ . We define a transformation TransResponse of this automaton into an enforcement monitor $\mathcal{A}_{\downarrow\varphi} = (Q^{\mathcal{A}_{\downarrow\varphi}}, q_{init}^{\mathcal{A}_{\downarrow\varphi}}, Stop^{\mathcal{A}_{\downarrow\varphi}}, \longrightarrow_{\mathcal{A}_{\downarrow\varphi}})$ using the transformations of the TransResponse transformation and adding the following rules to define $\longrightarrow_{\mathcal{A}_{\downarrow\varphi}}$ from $\longrightarrow_{\mathcal{A}_\varphi}$:

$$\begin{aligned}
 q &\xrightarrow{a/store} \mathcal{A}_{\downarrow\varphi} q' \text{ if } q \in R \wedge q' \notin R \wedge q \xrightarrow{a}_{\mathcal{A}_\varphi} q' \wedge \exists q'' \in R \cdot q'' \in Reach_{\mathcal{A}_\varphi}(q') \\
 q &\xrightarrow{a/halt} \mathcal{A}_{\downarrow\varphi} q' \text{ if } q \in R \wedge q' \notin R \wedge q \xrightarrow{a}_{\mathcal{A}_\varphi} q' \wedge \nexists q'' \in R \cdot q'' \in Reach_{\mathcal{A}_\varphi}(q')
 \end{aligned}$$

An EM $\mathcal{A}_{\downarrow\varphi}$ obtained from TransGuarantee(\mathcal{A}_φ) behaves as follows. Informally the principle is similar to the one of guarantee enforcement, except that there might be an alternation in the run between states of R and \overline{R} . While the current execution sequence does not satisfy the underlying property (the current state of \mathcal{A}_φ is in \overline{R}), it stores each event of the input sequence. Once, the execution sequence satisfies the property (the current state of \mathcal{A}_φ is in R), it dumps the content of the memory and the current event.

Example 5 (Response Transformation). Fig. 4 (left-hand side) shows a Streett automaton recognizing the response property φ_4 introduced in example 1. Its set of states is $\{1, 2, 3, 4\}$, the initial state is 1, and we have $R = \{1\}$ and $P = \emptyset$. The right-hand side shows the EM enforcement the same property, obtained by the TransResponse transformation. One can notice there is one stopping state 4.

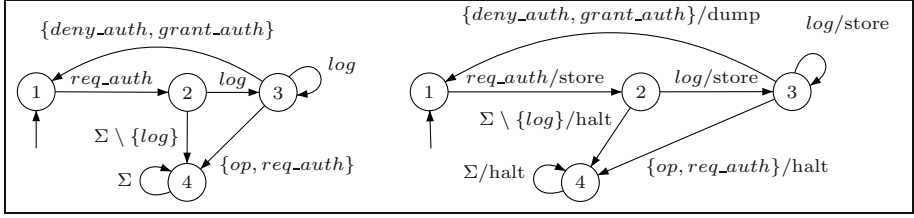


Fig. 4. A response-automaton and the corresponding EM for property φ_4

5.2 Enforcement wrt. the Safety-Progress Classification

Using the aforementioned transformations it is possible to derive an EM of a certain property from a recognizing automaton for this (enforceable) property. In the following, we characterize the set of enforceable properties wrt. the safety-progress classification.

Enforceable Properties. Now, we delineate the class of enforceable properties. Notably, the safety (resp. guarantee, obligation and response) properties are enforceable. Given *any* safety (resp. guarantee, obligation, response) property φ , and a Streett automaton recognizing φ , one can *synthesize* from this automaton an enforcing monitor for φ using systematic transformations. This also proves the correctness of these transformations.

Theorem 1. *Given a program \mathcal{P}_Σ , a reasonable safety (resp. guarantee, obligation, response) property $\varphi \in \text{Safety}(\Sigma)$ (resp. $\varphi \in \text{Guarantee}(\Sigma), \varphi \in \text{Obligation}(\Sigma), \varphi \in \text{Response}(\Sigma)$) is enforceable on $\mathcal{P}(\Sigma)$ by an EM obtained by the application of the safety (resp. guarantee, obligation, response) transformation on the automaton recognizing φ . More formally, given \mathcal{A}_φ recognizing φ , we have:*

$$\begin{aligned}
 (\varphi \in \text{Safety}(\Sigma) \wedge \mathcal{A}_{\downarrow\varphi} &= \text{TransSafety}(\mathcal{A}_\varphi)) \Rightarrow \text{Enf}(\mathcal{A}_{\downarrow\varphi}, \varphi, \mathcal{P}_\Sigma), \\
 (\varphi \in \text{Guarantee}(\Sigma) \wedge \mathcal{A}_{\downarrow\varphi} &= \text{TransGuarantee}(\mathcal{A}_\varphi)) \Rightarrow \text{Enf}(\mathcal{A}_{\downarrow\varphi}, \varphi, \mathcal{P}_\Sigma). \\
 (\varphi \in \text{Obligation}(\Sigma) \wedge \mathcal{A}_{\downarrow\varphi} &= \text{TransObligation}(\mathcal{A}_\varphi)) \Rightarrow \text{Enf}(\mathcal{A}_{\downarrow\varphi}, \varphi, \mathcal{P}_\Sigma). \\
 (\varphi \in \text{Response}(\Sigma) \wedge \mathcal{A}_{\downarrow\varphi} &= \text{TransResponse}(\mathcal{A}_\varphi)) \Rightarrow \text{Enf}(\mathcal{A}_{\downarrow\varphi}, \varphi, \mathcal{P}_\Sigma).
 \end{aligned}$$

Complete proofs for each class of properties are provided in [11].

Non-enforceable Properties. Persistence properties are not enforceable by our enforcement monitors. Such a property is φ_5 introduced in example 1 stating that “an incorrect use of operation *op* should imply that any future call to *req_auth* will always result in a *deny_auth* answer”. One can understand the enforcement limitation intuitively using the following argument: if this property was enforceable it would imply that an enforcement monitor could decide from a certain point that the underlying program will always produce the event *deny_auth* in response to a *req_auth*. However such a decision can never be taken

without reading and memorizing first the entire execution sequence. This is of course unrealistic for an infinite sequence.

As a straightforward consequence, properties of the reactivity class (containing the persistence class) are not enforceable by our enforcement monitors.

6 Discussion

An important question not yet addressed in this paper concerns the practical issues and possible limitations raised by the approach we propose. These limitations fall in several categories.

First, it is likely the case that not all events produced by an underlying program could be freely observed, suppressed, or inserted. This leads to well-known notions of *observable* and/or *controllable* events, that have to be taken into account by the enforcement mechanisms. To illustrate such a limitation, consider a system in which there is a data-flow dependence between two actions. It seems in this case that the enforcement ability is impacted since the first action cannot be frozen (otherwise, the second action could not be executed). Another example is that some actions, important from the security point of view, may not be visible from outside the system (and hence from any enforcement mechanism). Solving these kinds of issues means either further refining the class of enforceable properties (taking these limitations into account), or being able to replace non-controllable or non-observable actions by “equivalent” ones.

Moreover, it could be also necessary to limit the memory resources consumed by the monitor. A possible solution is to store only an *abstraction* of the sequence of events observed (e.g. using a *bag* instead of a FIFO queue, or a set as in [6]). From a theoretical point of view, this means defining enforcement up to some *abstraction preserving trace equivalence relations*. We strongly believe that our notion of enforcement monitors (with a generic memory device) is a suitable framework to study and implement this feature.

Furthermore, an other working direction concerns confidence indebted to the implementation of such enforcement monitors. Such an implementation should remain in a size which permits to prove the adequacy between the enforced property and the semantics of the original underlying program. Such a concern follows the well-known principle of minimizing the trusted computing base.

7 Conclusion

In this paper our purpose was to extend in several directions previous works on security checking through runtime enforcement. Firstly, we proposed a generic notion of finite-state enforcement monitors based on generic memory device and enforcement operations. Moreover, we specified their enforcement abilities wrt. the general safety-progress classification of properties. It allowed a fine-grain characterization of the space of enforceable properties, which encompasses previous results on this area. Finally, we proposed a set of (simple) transformations to produce an enforcement monitor from the Streett automaton recognizing a given

safety, guarantee, obligation or response security property. This feature is particularly appealing, since it allows to automatically generate enforcement monitors from high-level property definitions, like *property specification patterns* [14] commonly used in system verification.

Several perspectives can be foreseen from this work, but the most important issue would be certainly to better demonstrate its practicability, as discussed in Sect. 6. A prototype tool is currently under development, and we plan to evaluate it on relevant case studies in a near future.

References

1. Schneider, F.B.: Enforceable security policies. *ACM Trans. Inf. Syst. Secur.* 3, 30–50 (2000)
2. Hamlen, K.W., Morrisett, G., Schneider, F.B.: Computability classes for enforcement mechanisms. *ACM Trans. Program. Lang. Syst.* 28, 175–205 (2006)
3. Viswanathan, M.: Foundations for the run-time analysis of software systems. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, Supervisor-Sampath Kannan and Supervisor-Insup Lee (2000)
4. Ligatti, J., Bauer, L., Walker, D.: Runtime Enforcement of Nonsafety Policies. ACM, New York (2007)
5. Ligatti, J., Bauer, L., Walker, D.: Enforcing non-safety security policies with program monitors. In: de Capitani di Vimercati, S., Syverson, P.F., Gollmann, D. (eds.) *ESORICS 2005*. LNCS, vol. 3679, pp. 355–373. Springer, Heidelberg (2005)
6. Fong, P.W.L.: Access control by tracking shallow execution history. *sp 00*, 43 (2004)
7. Chang, E., Manna, Z., Pnueli, A.: The safety-progress classification. Technical report, Stanford University, Dept. of Computer Science (1992)
8. Chang, E.Y., Manna, Z., Pnueli, A.: Characterization of temporal property classes. *Automata, Languages and Programming*, 474–486 (1992)
9. Lamport, L.: Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng.* 3, 125–143 (1977)
10. Alpern, B., Schneider, F.B.: Defining liveness. Technical report, Cornell University, Ithaca, NY, USA (1984)
11. Falcone, Y., Fernandez, J.C., Mounier, L.: Synthesizing Enforcement Monitors wrt. the Safety-Progress Classification of Properties. Technical Report TR-2008-7, Verimag Research Report (2008)
12. Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: *PODC 1990: Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pp. 377–410. ACM, New York (1990)
13. Streett, R.S.: Propositional dynamic logic of looping and converse. In: *STOC 1981: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pp. 375–383. ACM, New York (1981)
14. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: *FMSP 1998: Proceedings of the second workshop on Formal methods in software practice*, pp. 7–15. ACM, New York (1998)

Implicit Flows: Can't Live with 'Em, Can't Live without 'Em

Dave King¹, Boniface Hicks², Michael Hicks³, and Trent Jaeger¹

¹ The Pennsylvania State University
`{dhking,tjaeger}@cse.psu.edu`

² Saint Vincent College
`fatherboniface@acm.org`

³ University of Maryland, College Park
`mwh@cs.umd.edu`

Abstract. Verifying that programs trusted to enforce security actually do so is a practical concern for programmers and administrators. However, there is a disconnect between the kinds of tools that have been successfully applied to real software systems (such as taint mode in Perl and Ruby), and information-flow compilers that enforce a variant of the stronger security property of noninterference. Tools that have been successfully used to find security violations have focused on *explicit flows* of information, where high-security information is directly leaked to output. Analysis tools that enforce noninterference also prevent *implicit flows* of information, where high-security information can be inferred from a program's flow of control. However, these tools have seen little use in practice, despite the stronger guarantees that they provide.

To better understand why, this paper experimentally investigates the explicit and implicit flows identified by the standard algorithm for establishing noninterference. When applied to implementations of authentication and cryptographic functions, the standard algorithm discovers many real implicit flows of information, but also reports an extremely high number of false alarms, most of which are due to conservative handling of unchecked exceptions (e.g., null pointer exceptions). After a careful analysis of all sources of true and false alarms, due to both implicit and explicit flows, the paper concludes with some ideas to improve the false alarm rate, toward making stronger security analysis more practical.

1 Introduction

The last decade has seen a proliferation of static analysis tools that analyze commodity software to discover potential security vulnerabilities. For example, tools have been developed, in research and industry, to uncover SQL injection vulnerabilities [16,9], missed access control checks [25], user-kernel pointer bugs [13], and format string vulnerabilities [6,21]. At their core, these tools are quite similar in that they track the flow of security-relevant data through the program. For example, many programs receive untrusted input from the filesystem or the network. If such data is unduly trusted it can be used for malicious purposes,

e.g., to construct an unexpected SQL query string or `printf` format string that leaks secrets or compromises the program. An analysis may track the flow of input data to ensure it is santized or verified before it is trusted. An analysis may dually check whether secret data, such as a cryptographic key or password, may flow to a public channel; if so, this constitutes an information leak [3].

The last decade has also seen the development of security-typed programming languages [20] (such as Jif [18], a variant of Java) that enforce flavors of the security property of noninterference [11,19]. Like the tools mentioned above, such languages ensure that no untrusted data *explicitly* flows to trusted contexts (and dually, that no secret data flows to public contexts). However, unlike these tools, security-typed languages also ensure that there are no illegal *implicit* flows, which arise due to control-dependent assignments. For example, given two boolean-typed variables `H` and `L`, the code fragments `L := H` and `if (H) then L := true; else L := false` are semantically equivalent. In the first, `H` explicitly flows to `L` via an assignment. In the second, no direct assignment takes place, but nevertheless `H` has the same value as `L` at the conclusion of execution; thus we say that `H` *implicitly* flows to `L`.

To our knowledge, implicit flow checking is not performed by mainstream tools. On the upside, tracking implicit flows could reveal more security vulnerabilities. On the downside, an analysis that tracks implicit flows could waste developer time, either by being too inefficient or producing too many false alarms (warnings that do not correspond to actual problems). A natural question is “do the benefits of implicit flow analysis outweigh the costs?”

This paper presents the results of a study we performed toward answering this question. We analyzed several libraries for information leaks using JLIft, an interprocedural extension of the Jif security-typed language [14] and carefully analyzed the results to see how often JLIft alarms real implicit information flows, and how often they were false alarms. The code we analyzed implemented security-critical functions: three different authentication methods in the J2SSH Java SSH library and three different cryptographic libraries from the Java implementation of the Bouncy Castle Cryptography API. We chose these applications for two reasons. First, they are security-critical, so understanding their information leaks is important. Second, password checking and encryption directly manipulate sensitive data, and are known to induce illegal flows implicitly. Therefore, the prevalence of true leaks in these routines suggests the best case for finding potential leaks due to implicit flows, while the prevalence of false alarms points to the human costs of analyzing an implicit flow checker’s results when doing so is likely to be worthwhile.

To perform the analysis, we labeled sensitive data, such as passwords or cryptographic keys, as *secret*, and labeled output channels as *public*, so that JLIft emits an alarm each time it discovers secret information could be inferred over a public channel. For our benchmarks, implicit flows caused 98% (870 out of 887) of the alarms, and of the 162 alarms identifying true information leaks, 145 of these (89%) were due to implicit flows. On the other hand, there was a tremendous number of false alarms (725), all of which were due to implicit flows (i.e., 83% of

the implicit flow alarms, or 725 out of 870, could not arise at runtime). Examining these further we find that the overwhelming majority (845 out of 870) of implicit flow alarms arise from the potential to throw an exception after examining sensitive data. Moreover, while the false alarm rate for flows induced by normal conditional statements is 30%, the rate of false alarms for exception-induced flows is much higher, at 85%. The false alarm rate from *unchecked* exceptions, such as `NullPointerException` and `ArrayIndexOutOfBoundsException`, is higher still: 757 of the 845 exception-induced flows were due to unchecked exceptions, and 706 of these (or 93.2%) were false alarms.

We draw two conclusions from these results. On the one hand, implicit flow checking can be valuable: JLIft emitted 145 alarms that correspond to true implicit flows of secret information. On the other hand, the human cost of dealing with the many false alarms is likely to be quite high, and could well be the reason that most tools perform no implicit flow checking. However, because the high rate of false alarms comes from a few identifiable sources (notably exceptions, particularly unchecked exceptions), we see much potential for improvement. In particular, we suggest that implicit flow checking tools: (1) employ more powerful, secondary analyses for ruling out spurious alarms due to infeasible unchecked exceptions; (2) rank the errors reported, beginning with explicit flows, followed by implicit flows not from exceptions, and finally reporting implicit flows due to exceptions; (3) employ annotations so that programmers can indicate code that has been checked by an alternative method, such as a human code review or extensive testing.

2 Program Analysis for Security

A large body of work has explored the use of programming languages and analyses, both static and dynamic, to detect security vulnerabilities. Many analyses aim to discover *source-sink* properties, or *explicit flows*. For example, to discover a possible format string vulnerability, a programmer could use the CQUAL [10] static analyzer to give `printf` the type `printf(untainted char* fmt, ...)`. This type specifies that `printf`'s first argument must not come from an untrusted source. Input functions are then labeled as producing potentially tainted data, e.g., the type `tainted char* getenv(char *name)` indicates that the value returned from `getenv` is possibly tainted, since it could be controlled by an attacker. Given this specification, CQUAL can analyze a source program to discover whether data from a source with type `tainted char*` is able to explicitly flow, via a series of assignments or function calls, to a sink of type `untainted char*`. If such a flow is possible, it represents a potential format string vulnerability.

Explicit flow analysis tools [9,16,24] can be used to detect a variety of integrity violations, such as SQL injection vulnerabilities [16,9], format string vulnerabilities [6,21], missed access control checks [25], and user-kernel pointer bugs [13]. These tools can also be used to check confidentiality-oriented properties, to ensure that secret data is not inadvertently leaked. In this case, the programmer would label sensitive data, such as a private key, as `secret` and arguments to

output functions as public, and the analyzer would ensure that secret data never flows to public channels (except under acceptable circumstances, e.g., after an access control check). Scrash [3] uses CQUAL in this way to identify sensitive information that should be redacted from program crash reports.

While common, explicit flows are but one way in which an attacker is able to gain information about a program's execution for malicious purposes. The other possibility is to use an *implicit flow* by which, in the case of confidentiality properties, the attacker learns secret information indirectly, via a control channel. As a simple illustration, consider the following program:

```
secret int x;
void check(public int y) {
    if (x > y) then printf("greater\n");
    else printf("smaller\n");
}
```

CQUAL would not report a violation for this program: the secret integer `x` is not leaked via an explicit flow to the public console. However, *information* about `x` is leaked, in particular whether or not `x` is greater than the attacker-controlled variable `y`. Invoked repeatedly, with different values of `y`, such a function could ultimately leak all of `x`. It has been shown that cryptosystem implementations can end up leaking the private key if they contain certain kinds of implicit flows, e.g., by reporting the stage in which a decryption failed [1,2,23]. Conversely, an attacker's input can influence a program to corrupt trusted data, e.g., by turning a `setuid`-invoking program into a confused deputy [5].

These problems are violations of the more general information flow security property of noninterference [11]. For confidentiality, a program that enjoys noninterference will not leak secret information to public outputs in any manner, whether via implicit or explicit flows. *Security-typed programming languages* [20] such as Jif [18] enforce noninterference via type checking. Program types are annotated with security labels like the secret and public qualifier annotations above, and security correctness is guaranteed through type checking: type-correct programs do not leak secret information on public channels. Thus, using a security-typed language, the `check` function above would be correctly flagged as a potential leak of information.

3 Security Type Checking and Implicit Flows

The question we address in this paper is whether implicit flow checking as done by security-typed languages is efficacious—is the signal worth the noise? This section describes the standard security-type checking algorithm used to detect implicit flows and considers possible sources of imprecision. The next section examines the number and sources of true and false alarms when using this algorithm to analyze some security-critical Java libraries.

In a security-typed language, types are annotated with *security labels*, analogous to type qualifiers: the integer type `int` can be labeled with the label `high` as `int{high}` to signify a high-secrecy integer. The labels are ordered, inducing

```

1 int{Public} authenticate(AuthenticationServer auth,
2                           SshAuthRequest msg) {
3   Key{Secret} key = reader.readBinaryString();
4   if (checkKey == 0) {
5     if (verify.acceptKey(msg.getUsername(), key)) {
6       SshAuthPKOK reply = new SshAuthPKOK(key);
7       auth.sendMessage(reply);
8       return READY; }
9   else return FAILED; } }

```

Fig. 1. Example Authentication Code From J2SSH

a subtyping relation \preceq on labeled types. In particular, because **low** is less secret than **high**, **int**{**low**} is a subtype of **int**{**high**}, written **int**{**low**} \preceq **int**{**high**}. If **h** and **l** are respectively high and low secrecy variables, then the assignment statement **h** := **l** is allowed, as **int**{**low**} \preceq **int**{**high**}. On the other hand, **l** := **h** is *not* allowed, preventing an illegal explicit flow, as **int**{**high**} $\not\preceq$ **int**{**low**}.

The statement **if** **h** == 0 **then** **l** := 1 **else** **l** := 0 contains an implicit flow of information from **h** to **l**, as by observing **l** we learn information about the value of **h**. The standard way [20] to check for implicit flows is to maintain a security label PC_ℓ for the *program counter*. This label contains the information revealed by knowing which statement in the program is being executed at the current time. PC_ℓ is determined by the labels of guards of any branches taken: if the program branches on a condition that examines a high-security value, PC_ℓ is set to **high** while checking the code in the branches. When a variable with label *m* is assigned, we require $PC_\ell \preceq m$. In the above statement, the branch condition (**h** == 0) causes the PC_ℓ to be **high** when checking the branches, meaning that any assignment to **l** is illegal and so the code is rejected.

For a more realistic example, consider the code in Figure 1, taken from the J2SSH implementation of public-key authentication. We use the label **Secret** to represent **high** security, and **Public** to represent **low** security. This code is executed when a user is attempting to authenticate using an SSH key. The `acceptKey` method checks if the specified key is an accepted key for the user that is attempting to log in. As this contains information about the system, `acceptKey` returns a **Secret** boolean value, causing PC_ℓ to become **Secret**. As such, the value **READY** returned from the branch must also be considered **Secret**, but notice that we have annotated the return value of `authenticate` as **int**{**Public**}, so this constitutes an implicit information leak.

While sound, the use of PC_ℓ to find implicit flows is conservative. Consider the program **if** **h** == 0 **then** **l** := 1 **else** **l** := 1. This program leaks no information—the value of **l** is always 1, no matter what the value of **h**—but the type system rejects it as an illegal flow. A more pernicious source of false alarms is the throwing of exceptions. For example, the Java code `obj.equals(otherObj)` throws a `NullPointerException` at run time if `obj` is `null`. If such an exception occurs while PC_ℓ is **high**, the resulting termination of the program (assuming the exception is not caught) is publicly-visible, and thus is an implicit flow of

information. In the example in Figure 1, such an exception thrown within the call to `sendMessage` could reveal whether the given key was accepted. Therefore, if the type system cannot prove that `obj` is always non-`null`, then dereferencing it must be considered as possibly throwing a null pointer exception. An analogous situation arises with other exceptions, such as array bounds violations and class cast exceptions.

Jif's type checker makes some attempt to prove that certain objects cannot be `null` and that some array accesses are legal, but its analysis is fairly simplistic. For example, the analysis operates only on local variables (not fields or formals), and is only intraprocedural. Programmers are thus forced to work within the limits of the analysis to limit false alarms. For example, it is a common practice to copy fields and formal method arguments into local variables to check if the objects are `null` before invoking methods on them. Redundant null checks and empty `try/catch` blocks (where the programmer presumes no exception can actually be thrown but the analysis cannot detect this) are also common.

While Jif's analysis could be improved, determining whether a runtime exception could occur or not is undecidable [15], so no analysis can be perfect. To show the general difficulty of the problem, consider the following code (taken from the public key authentication routine in J2SSH):

```
1 byte[] decode(byte[] source, int off, int len) {  
2     int len34 = len * 3 / 4;  
3     byte[] outBuff = new byte[len34];  
4     ...  
5 }
```

In this code, the statement `new byte[len34]` could throw a `NegativeArraySizeException`, meaning that the program attempted to create a byte array with a negative size (if `len34` is negative). As this code is invoked after checking whether a username is valid on the system, the program crashing here after a negatively-sized array is created could reveal this information to the requester. While the passed-in value `len` is meant to always be positive, there is no easy way, in general, for a program analysis to determine this. In our manual analysis, we determined that the only call to `decode` by the public key authentication method was with the variable `bytes.length`, where `bytes` was an array returned by `String.getBytes()`, a quantity that will always be non-negative.

4 Experiments

To determine the impact of implicit program flows in analyzing real code, we analyzed six implementations of security functions: (1) three different authentication methods in the J2SSH Java SSH library:¹ and (2) three different cryptographic libraries from the Java implementation of the Bouncy Castle

¹ J2SSH is available from <http://sourceforge.net/projects/sshtools>

Program	Security Function	# Alarms			False Alarms		False Alarm Rate
		Total	Explicit	Implicit	Explicit	Implicit	
J2SSH	Password	7	0	7	0	3	42.86 %
	Keyboard Interactive	23	0	23	0	19	82.61 %
	Public Key	170	0	170	0	111	65.29 %
BouncyCastle	RSA	218	3	215	0	186	86.51 %
	MD5	209	4	205	0	199	97.07 %
	DES	260	10	250	0	207	82.80 %

Fig. 2. False alarm rates for the three authentication methods present in J2SSH. The first and column columns give the application and security function that was analyzed. The third column gives the total number of alarms, with the fourth and fifth columns containing the number of alarms raised by both potential explicit and implicit flows. The sixth and seventh column gives the number of false alarms (subdivided between explicit and implicit flows), while the seventh column gives the overall percentage of false alarms among all of the alarms reported by JLIft.

Cryptography API.² The J2SSH authentication methods that we investigated were password-based, keyboard-interactive, and public-key-based. The Bouncy Castle implementations that we investigated were RSA Encryption (using SHA1 Digests), MD5 Hashing, and DES Encryption. To gather statistics about the existing runtime exceptions in these codebases, we used JLIft, an interprocedural extension of the Jif security-typed language [14].

The results from our investigations are summarized in Figure 2. Our experiments show the following:

- Implicit flows corresponded to most of the leaks of information in these applications. In the SSH authentication methods, there were no explicit information leaks, while there were only a handful of explicit leaks in the cryptography libraries. In total, leaks corresponding to explicit flows made up less than 2% of the reported alarms, and none of the explicit flows corresponded to a false positive.
- The implicit flows arising from unchecked runtime exceptions dominate all other flows. Specifically, 757 out of 870 (87 %) of alarms caused by implicit flows were due to the five types of runtime exceptions (Null Pointer, Array Out of Bounds, Class Cast, Negative Array, Arithmetic). These results are summarized in Figure 4.
- Most flows due to unchecked exceptions were caused by program paths that could not be executed at runtime. Specifically, we manually verified that 706 alarms (out of 824 total exceptional alarms) could not occur at run time.

4.1 Methodology

We chose to analyze two different codebases for information leaks: an SSH server and a cryptographic library. These programs are security-critical, and thus

² BouncyCastle implementations for Java and C# are available at <http://www.bouncycastle.org/>

constitute a worthy target of security analysis. While it would be unsurprising to find implicit (and explicit) flows in these programs since they manipulate secret information in an observable way (e.g., by outputting the result of a password check or emitting ciphertext), our interest is of effectiveness: does the standard algorithm, when applied to mature code written in a natural style, identify the true explicit and implicit information flows without producing a deluge of false alarms?

4.2 Analysis Methodology

Details of Security Analysis. To perform an information-flow analysis without explicitly converting the code into Jif as well as investigating how a implicit flow checker behaved when applied to code written in a natural style, we used JLift [14], a modification of the Jif compiler to perform a whole-program information-flow analysis on Java programs. From a set of seed labels, JLift propagates security information across procedure calls by determining a set of summary constraints for each method, a standard method in static analysis [22]. JLift relies on the Jif compiler to generate these constraints: in particular, it annotates types with a security label and uses a program counter to detect implicit flows. We ran JLift using a context-sensitive treatment of methods, associating a unique set of summary constraints to each instance of a method call. Our experience has been that context sensitivity is required for an accurate information-flow analysis [14].

Annotation of JLift. To analyze these programs for security errors, we labeled protected data as high security and analyzed the resulting flows. As JLift does not report all possible program paths that lead to a violation, we ran the tool multiple times, suppressing earlier warnings by adding some “declassification” annotations (which indicate that certain flows should be ignored). To efficiently catalogue exceptions that could be thrown at runtime that would affect the program counter, we first used the analysis to determine which methods could be invoked from a high program counter location. We then had JLift enumerate the runtime exceptions, determined by an interprocedural propagation analysis, that could possibly be thrown by those methods. We investigated each of these runtime exceptions to determine if they could be thrown at run time.

Following Jif's behavior, we did not directly analyze the program flows that could occur when invoking a library function at runtime. For conservativity, we assigned each instance of a class defined in a library a label variable L and treated each value passed to or returned from methods or fields as having label L . This prevented secure information from being laundered by being passed through library functions.

False Alarms. We define a *false alarm* as an alarm that the analysis tool reports that corresponds to an infeasible program path. For example, if the variable v is always not **null**, then an alarm raised by the statement $v.foo()$ attributed to a `NullPointerException` at this point would be false. We marked an alarm as true when we could manually confirm the flagged path could actually execute

at run time; all other alarms were considered false. As an example, Jif’s null pointer analysis is only performed on local variables; all fields are considered to be possibly null. As such, many times a field dereference was incorrectly flagged as problematic even though all feasible paths initialized the field (e.g., in the constructor) before dereferencing it.

Several places in the code explicitly throw an exception: for example, if a block to encrypt was too large for the RSA cipher, the program would throw a `DataLengthException`. We did not count explicitly thrown exceptions or exceptions from library calls (for example, `IOException` on file open) as false alarms. The code for MD5 hashing explicitly threw exceptions at only three locations, as opposed to DES, which threw exceptions at nineteen locations: this accounts for some of the difference in false positive rate between the two.

4.3 J2SSH

J2SSH is a Java SSH library that contains an SSH server implementation that has three different authentication methods: password, keyboard-interactive, and public-key authentication. This code most directly leaks information to potential attackers. If the programmer is not careful, information leaks in an authentication system risk giving away important information about the system, such as legal user names or even passwords (perhaps subject to a brute-force attack).

J2SSH handles authentication by allowing the host operating system to provide a native authentication mechanism. This object represents the operating system mechanism for checking and changing passwords. Our experiment was thus to see how the J2SSH server code protected the native authentication mechanism. We marked information associated with the native authentication mechanism as `Secret` and assumed that the authentication method returned a `Public` result.

There were only three means of communicate with the user in the J2SSH server: (1) the returned result from authentication (a `boolean`), (2) messages sent by the server back to the client, and (3) abnormal termination of the SSH server. There were no explicit flows of information in these codebases, as each of the authentication methods behaved similarly to the code from Figure 1: based on the result of a secret query, a value is returned to the user, a message is sent, or the program terminates.

4.4 Bouncy Castle Cryptographic Implementations

We also ran our analysis on the implementations of three cryptographic implementations from the Java implementations of the Bouncy Castle Cryptographic APIs: RSA asymmetric-key encryption (with SHA1 digests), MD5 hashing, and DES symmetric-key encryption.

For these codebases, we labeled both the keys and the data being encrypted or hashed as secret. For each function, the code performing the encryption contained many low-level bitwise operations, including accesses to pre-defined constant arrays and bit-shifts. This lead to many more possible array access errors reported in the Bouncy Castle cryptographic libraries than in J2SSH. This also

```

1 int newKey[32], boolean pc1m[56], boolean pcr[56];
2 for (int j = 0; j < 56; j++) {
3     int l = pc1[j];
4     pc1m[j] = (key[l >>> 3] & bytebit[1 & 7]) != 0; }
5 for (int i = 0; i < 16; i++) {
6     int l, m, n;
7     if (encrypting) { m = i << 1; } else { m = 15 - i << 1; }
8     n = m + 1; newKey[m] = (newKey[n] = 0);
9     for (int j = 0; j < 28; j++) {
10        l = j + totrot[i];
11        if (l < 28) { pcr[j] = pc1m[l]; } else { pcr[j] = pc1m[l - 28]; }
12    } for (int j = 28; j < 56; j++) {
13        l = j + totrot[i];
14        if (l < 56) { pcr[j] = pc1m[l]; } else { pcr[j] = pc1m[l - 28]; }
15    } for (int j = 0; j < 24; j++) {
16        if (pcr[pc2[j]]) { newKey[m] |= bigbyte[j]; }
17        if (pcr[pc2[j + 24]]) { newKey[n] |= bigbyte[j]; } } }

```

Fig. 3. Code from the DES key scheduling code, where the original secret key is split into two keys to operate on half of a 64-bit encryption string. Nearly every line of the above code was flagged as possibly throwing an exception.

accounts for the slightly higher false positive rate in the Bouncy Castle cryptographic implementations as compared to J2SSH.

Figure 3 shows a typical example of code where it is difficult to rule out impossible runtime exceptions. The code was taken from the implementation of DES and contained a very large false alarm rate: nearly every line of the algorithm was flagged as possibly throwing at least one exception. Automatically verifying that this code cannot throw an exception is difficult: it requires knowledge of the contents custom-built arrays `bytebit`, `totrot`, `bigbyte`, knowing that these arrays cannot change and carefully maintaining knowledge of possible bounds on each variable. With some effort, we were able to manually verify that each line of the above code could not throw a runtime exception.

4.5 Discussion

Figure 4 contains statistics about implicit flows, exceptional flows, and alarms raised by the five most commonly occurring runtime exceptions. From this figure, we can see that most alarms were caused by implicit flows, and that most of these implicit flows were caused by exceptions. Of these, five exceptions make up the bulk of the false alarms reported by the Jif program analysis: null pointer errors, array bounds errors, class cast errors, negative array errors, and arithmetic errors (divide by zero). The applications that we looked at had no explicit flows that corresponded to infeasible runtime paths; this may be due to the small number of explicit flows that these applications contained.

Our experimental results suggest that developers analyzing code for information-flow security should follow an iterative process: first analyze

Exception	# Alarms	False Alarms	False Alarm Rate	% of Total Alarms
all flows	887	725	81.74 %	100 %
all implicit	870	725	83.33 %	98.08 %
all exceptions	824	706	79.59 %	92.90 %
null pointer	475	455	95.79 %	53.78 %
array bounds	233	204	87.55 %	26.27 %
non-exceptional implicit flows	46	14	30.43 %	5.19 %
class cast	24	22	91.67 %	2.78 %
negative array	20	20	100.0 %	2.76 %
arithmetic exception	5	5	100.0 %	0.56 %

Fig. 4. Rates of false alarms per exception for the most commonly-occurring runtime exceptions. The first column gives the type of exception, the second gives the number of alarms, the third gives the number of those that were due to an code path that is not realizable at runtime, and the fourth gives the rate of false alarms. The fifth column shows the percentage of total alarms (number of alarms from this category divided by total alarms) that were caused by this specific exception type.

explicit flows, then analyze implicit flows that occur because of non-exceptional program paths, and finally analyze the remaining implicit flows.

Most of the implicit flows reported by the analysis that were not false alarms corresponded to technical violations of noninterference. For example, in DES, the eventual size of the output buffer for ciphertext (publicly available) depended on the size of the data to be encrypted. There were also several authentication flows that revealed information about whether a username was valid or not, but these occurred after the password was successfully verified, making these flows difficult to exploit. For example, when a login attempt was valid but the user was required to change his or her password, the server would send a message with this information.

A handful (3) of implicit flows that we marked as false alarms were not caused by the type-based nature of the analysis. These alarms occurred because the J2SSH KBI authentication method assigned three variables to both high security and low security values and the Jif program analysis treated all occurrences of those variables as high security data.

We make no judgment about the value of true alarms. Each of the true alarms represented a violation of noninterference. Methods for determining the severity of these alarms (for example, statically determining for each alarm a quantitative upper bound of number of bits leaked, as in recent work [17]) are beyond the scope of this paper.

One may wonder how our results with JLife speak to Jif programming as it is done today, as Jif has been used to build several substantial systems, including JPMail, a mail client [12] and a Civitas, remote voting system [7]. The answer is simple: the programmer must work around the imprecision of various analyses in order get the program to type check. As mentioned before, programmers often copy fields into local variables because the null pointer analysis in the

compiler is only done on local variables. Empty catch statements, signaling the programmer's belief that a runtime exception should not affect the security of the program counter, are also quite common. To quantify these observations, we examined the Civitas code base, which contains over 13000 lines of Jif Code. We found 568 empty catch statements to handle unexpected runtime exceptions. Of these, 429 of these caught exceptions were given the variable name `imposs`, indicating that the programmers believed that it was impossible for these errors to occur. Of the remainder, 13 of them were given the variable name `unlikely`, indicating that the programmers thought it was not likely that these errors would occur. While programmer belief does not guarantee all of these exceptions are impossible or unlikely, these coding constructions speak to the need for a sound but less burdensome way for developers to handle implicit flows arising from runtime exceptions.

5 Towards Painfree Noninterference

In this section we suggest several approaches for handling the high rate of false alarms.

Modern programming languages support robust features for error handling and recovery. As an imperative object-oriented language, Java gives more control over errors than previous languages like C and C++: if an error occurs in code, the Java runtime can catch and handle it, rather than simply terminating. However, the possibility for failure at every method call or array access in Java makes it difficult to perform an accurate information-flow analysis. Once the program counter has been raised to secret by a high-security conditional or thrown exception, even one that might itself be a false alarm, a sound static analysis must treat the execution of the rest of the program as conditional, meaning that every possible exception raised after this must be reported and handled.

From Figure 4, it is clear that better program analyses would aid in reducing the number of false alarms. Currently, most null pointer accesses cannot be proven safe by Jif, and as almost any line of Java code can throw a `NullPointerException`, this leads to an unacceptably high number of false alarms. Improved analyses could reduce this false alarm rate, but due to the fundamental undecidability of determining if a null pointer access is safe [15] combined with the difficulty of maintaining invariants for every line of code (as seen in Figure 3), these analyses will likely remain imperfect.

One way to improve the power of these analyses is to rely more on programmer annotations. For example, a programmer could annotate when an object could possibly be null, rather than the opposite: this would shift the burden of dealing with null pointer analyses to annotating which variables could possibly store the null value. Security-typed languages could also be integrated with tools such as ESC/Java [4,8] that can verify these annotations and so prove properties of variables and array bounds while reducing the number of false positives reported by the security analysis. Adding a new block primitive to the Jif programming language that indicated that the enclosed code could not terminate abnormally

would allow Jif programmers to make trusted sections of code more explicit without relying on empty **try/catch** blocks, as in the Civitas codebase.

As we have seen, a sound security analyses must therefore consider a large number of runtime program paths, not all of which can occur at runtime. Flow Caml [19] is a security-typed language based on Caml Lite, a member of the ML family of functional programming languages. As ML does not have the concept of a null pointer, Flow Caml does not require a null pointer analysis for programmers to use the language. It may be possible to, by changing the semantics of the underlying Java language, modify the kinds of flows that can occur at runtime and so reduce the number of false positives reported by a static analysis. Another possibility is to change our security model to insert a global system declassifier so that certain classes of implicit flows (those with a high noise rate) are allowed to leak information about secret data. If this is adopted, care must be taken to ensure that these flows cannot be repeatedly exploited to gain a substantial amount of information from the system.

Finally, different programming models could aid in reducing the number of false alarms. If a section of code can be encapsulated as only taking input and returning an output (rather than possibly terminating in the middle of its computation), then any implicit flows that occur during this computation can be folded into the input (through a **try/catch** block, for example). If every operation that is performed inside of a high program counter region will not throw an exception, then the only implicit flows that can occur during a program are updates to the system's global state performed by these operations (such as message sends and receives). It may be possible to automate this approach.

6 Conclusion

In this paper, we have outlined the impact of considering *implicit flows* in a security analysis. Our experiments show that when checking mature, security-critical code, the standard type-based algorithm reports a strikingly large number of implicit flows as false alarms. Most of these implicit flows are induced by unchecked exceptions that a static analysis is not able to prove are infeasible at runtime. If we are to make security analysis with implicit flows practical, better tools and different programming techniques are necessary.

References

1. Black, J., Urtubia, H.: Side-channel attacks on symmetric encryption schemes: The case for authenticated encryption. In: Proceedings of the 11th USENIX Security Symposium (2002)
2. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 1–12. Springer, Heidelberg (1998)
3. Broadwell, P., Harren, M., Sastry, N.: Scrash: a system for generating secure crash information. In: Proceedings of the 12th conference on USENIX Security Symposium (2003)

4. Chalin, P., Kiniry, J.R., Leavens, G.T., Poll, E.: Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2., pp. 342–363. Springer, Heidelberg (2006)
5. Chen, H., Wagner, D., Dean, D.: Setuid demystified. In: Proceedings of the 11th USENIX Security Symposium, pp. 171–190. USENIX Association, Berkeley (2002)
6. Chen, K., Wagner, D.: Large-scale analysis of format string vulnerabilities in Debian Linux. In: Proceedings of the 2007 workshop on Programming languages and analysis for security (2007)
7. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: IEEE Symposium on Security and Privacy, pp. 354–368 (2008)
8. Flanagan, C., Leino, K.R.M., Lillibridge, M., Nelson, G., Saxe, J.B., Stata, R.: Extended static checking for Java. In: PLDI, vol. 37, pp. 234–245 (June 2002)
9. FORTIFY SOFTWARE. Fortify, <http://www.fortify.com/>
10. Foster, J.S., Fähndrich, M., Aiken, A.: A theory of type qualifiers. In: PLDI, pp. 192–203 (1999)
11. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, pp. 11–20 (1982)
12. Hicks, B., Ahmadizadeh, K., McDaniel, P.: From Languages to Systems: Understanding Practical Application Development in Security-typed Languages. In: Jesshope, C., Egan, C. (eds.) ACSAC 2006. LNCS, vol. 4186. Springer, Heidelberg (2006)
13. Johnson, R., Wagner, D.: Finding user/kernel pointer bugs with type inference. In: SSYM 2004: Proceedings of the 13th conference on USENIX Security Symposium, p. 9. USENIX Association, Berkeley (2004)
14. King, D., Jaeger, T., Jha, S., Seshia, S.A.: Effective blame for information-flow violations. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086. Springer, Heidelberg (2008)
15. Landi, W.: Undecidability of static analysis. ACM Letters on Programming Languages and Systems 1(4), 323–337 (1992)
16. Martin, M., Livshits, B., Lam, M.S.: Finding application errors and security flaws using PQL: a program query language. In: OOPSLA, pp. 365–383. ACM, New York (2005)
17. McCamant, S., Ernst, M.D.: Quantitative information flow as network flow capacity. In: PLDI, pp. 193–205 (2008)
18. Myers, A.C.: JFlow: Practical mostly-static information flow control. In: POPL, pp. 228–241 (January 1999)
19. Pottier, F., Simonet, V.: Information flow inference for ML. In: POPL, pp. 319–330. ACM, New York (2002)
20. Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1) (2003)
21. Shankar, U., Talwar, K., Foster, J.S., Wagner, D.: Detecting format string vulnerabilities with type qualifiers. In: Proceedings of the 10th conference on USENIX Security Symposium (2001)
22. Sharir, M., Pnueli, A.: Two approaches to interprocedural dataflow analysis. In: Program Flow Analysis: Theory and Applications, pp. 189–234. Prentice-Hall, Englewood Cliffs (1981)
23. Vaudenay, S.: Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS.. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 534–546. Springer, Heidelberg (2002)

24. Xie, Y., Aiken, A.: Saturn: A scalable framework for error detection using boolean satisfiability. *ACM Transactions on Programming Languages and Systems* 29(3) (2007)
25. Zhang, X., Edwards, A., Jaeger, T.: Using CQUAL for static analysis of authorization hook placement. In: *Proceedings of the 11th USENIX Security Symposium*, pp. 33–48. USENIX Association, Berkeley (2002)

A Robust Reputation Scheme for Decentralized Group Management Systems

Frédéric Cuppens, Nora Cuppens-Boulahia, and Julien A. Thomas

IT/TELECOM Bretagne, SERES team, 2 rue de la Châtaigneraie, CS 17607, 35576
Cesson Sévigné Cedex

Abstract. In the literature, reputation systems are used to evaluate other entities behavior and have many applications such as, for instance, the detection of malicious entities. The associated models are based on mathematic formulae, in order to formally define elements such as the reputation evaluation and evolution and the reputation propagation between peers. Current proposals describe the behaviors of their models by examples, with few (if not no) formal analyses. In this article, we state the basic security properties such systems require and we show that current systems may not satisfy them on specific scenarios, which can be used by malicious entities to take advantage of the system. We also present a new reputation scheme, designed to satisfy these properties, and we compare it to existing research works.

1 Introduction

In the literature, reputation systems are used to evaluate behaviors of subjects, processes or systems and for instance detect malicious entities. The associated models [1,2,3], based on mathematic formulae, generally take into account two actions: the operations to perform when an incorrect behavior is detected and the operations to perform when no malicious behavior is detected during a defined interval of time. These models rely on two main variables, α and β , which respectively refer to the reputation increase and decrease rates, in case of correct (respectively incorrect) behaviors.

The global process can be summarized by the following formulae:

- when no malicious behavior is detected for a node n_i , between two reputation checkpoints, the current node n_c increases its reputation of α , which means $rep_{n_c}(n_i) = rep_{n_c}(n_i) + \alpha$.
- When a node n_i has a bad behavior, its reputation decreases of β . If several malicious behaviors are detected between two reputation checkpoints, some proposals consider them as a single bad behavior ($rep_{n_c}(n_i) = rep_{n_c}(n_i) - \beta$), while others consider them as several ones ($rep_{n_c}(n_i) = rep_{n_c}(n_i) - nb_{detection} \cdot \beta$).

An example of application of the reputation systems is the management of groups [4,5], for instance in ad hoc networks[6]. In this context, groups are used to join

together nodes which can then share information and communicate. Inside these groups, which are sometimes considered as communities inside undefined and potentially malicious environments, the notion of trust is important as it can be used to detect malicious nodes. Stating security properties such as *the collusion of malicious nodes must not engender an eviction of a correct node* and asserting that they will be respected is thus important. As the design of the reputation system and the values of its parameters, such as α and β , are linked to the assertion of the security properties, the system must be defined by taking these properties into account. In current research works, formal analyses of the system parameters and assertions of such security properties are not performed.

In this article, we thus propose a formal method to evaluate the system parameters, in order to define a robust reputation scheme. In section 2, we first present the notions bounded to the reputation systems and we analyze the existing approaches. We then introduce in section 3 our reputation and recommendation system, with the security properties it must satisfy. In section 4, we present our formal analyses and specify the values of our system parameters that satisfy these properties. We then present our simulations, performed on NS-2 [7], and we compare our results with existing research works. The last section concludes the article.

2 Limits of Existing Approaches

Many studies [1,2,3,8,9] have proposed reputation systems which rely on two basic systems: the reputation and recommendation based systems and the referees based reputation systems. In this section, we thus present these systems and we analyze existing proposals that rely on them in order to show their limits.

2.1 Reputation and Recommendation Based Systems

In the recommendation and reputation system proposed by Jinshan Liu and Valérie Issarny [1], several parameters (which are summed up in the table 1) are associated with each node to evaluate other nodes' quality. Among these parameters, $SExp$ is the reputation derived from direct interactions between the current node and the analyzed one and $SRep$ is a node reputation derived from personnel evaluations and third nodes information. Moreover, each node has information about other nodes recommendation quality ($RRep$). $RRep$ is used as a weighting coefficient in the reputation evaluations. Finally, Rec is the reputation declared by a node about a peer and is the only value shared in the network. For a correct node, $Rec_a(o) = SRep_a(o)$.

An important aspect of this study is the distinction between recommendation and reputation: when a node provides a correct service, it can always be used, even if its recommendations are not correct and thus can be ignored.

Reputation evolution: For a node n_c , a node's reputation is based on three parameters: its old reputation, its new reputation according to n_c (which are both represented by $SExp_a(o)^t$) and the other nodes declared reputations $RRep$,

Table 1. Recommendation and Reputation System Parameters

$SRep_a(o)^t$	node o 's reputation, declared by a , at time t
$RRep_a(o)^t$	o 's recommendation, about a , at time t
$SExp_a(o)^t$	Immediate experience of a about o
$Rec_a(o)^t$	Recommendation made by a about o , at time t .
ρ_e, ρ_c	weighting coefficient of the reputation and recommendation functions

where all these parameters are weighted by credibility and freshness coefficients. The reputation of a node o , according to a node a , is defined as follows:

$$SRep_a(o)^t = \rho_e \cdot SExp_a(o)^t + (1 - \rho_e) \cdot \frac{\sum_p (RRep_a(p) \cdot Rec_p(o))}{\sum_p RRep_a(p)}$$

Recommendation evolution: For a node a , the recommendation quality of a node p relies on the differences between the recommendation made by p (Rec_p) and its personal evaluation ($SExp_a$) for each node $o \in N$. We thus have the basic formula: $diff_1(o) = |Rec_p(o) - SExp_a(o)|$. However, the differences between the values of two nodes can be due to analyses of different data (i.e. different contexts). In order to solve this problem, they use the notion of tolerance threshold δ_a . We then have a difference evaluation $diff = \frac{1 - diff_1}{\delta_a}$.

The recommendation evolution mechanism satisfies the following principle: the recommendation of p at time t relies on the precedent recommendation at time t' and the evaluation differences in this interval $\Delta t = t - t'$: $RRep_a(p)^t = RRep_a(p)^{t'} \cdot \rho_c^{(t-t')} + diff \cdot (1 - \rho_c^{(t-t')})$.

2.2 Referees Based Reputation Systems

In the research work by Conrad and al. [2], the notion of reputation is studied in order to first mimic the human trust formation and secondly to have a lightweight approach. They use the notions of subjective trust and distrust to apply their reputation system to e-services and on-line transactions, as the results are quite binary: either the result is correct, or not.

As for many studies, the reputation analysis is based on two principal components: the node which performs the evaluation and the others. The reputation function they suggest is $reputation(c) = experience(c) \cdot p + (1 - p) \cdot hearsay(c)$ where p is the value to assign to our own credibility ($p = selfConfidence(c)$).

The notion of self-experience is based on two parameters: prior experiences and immediate experiences. No weighting is made between these two parameters and we thus have $experience(c) = \frac{immediateExperience(c) + experience(c)}{2}$. Another interesting aspect in this study is the way the hearsay parameter is evaluated: contrary to the previous study, the nodes do not take into account the information from all the nodes of the network. We have a notion of referees R that are used to analyze a service reputation: $hearsay(c) = \frac{\sum_{r \in R} reputation_r(c)}{|R|}$. The choice of a correct value of $|R|$ is important: if we have a too small value, few analyses will be used and the result may not be representative while with a too

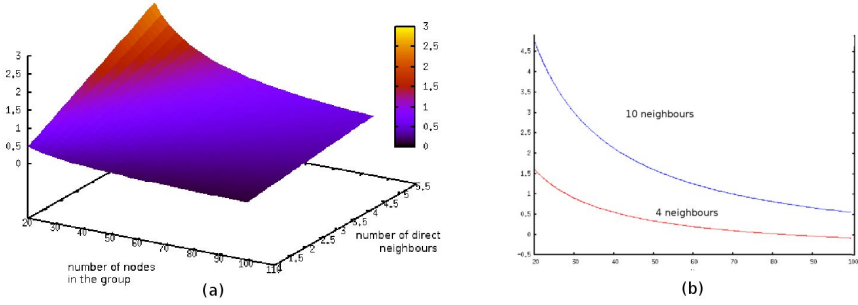


Fig. 1. Reputation evolution in Jinshan Liu and Valérie Issarny approach

big value, the reputation system becomes too slow. By performing simulations, they chose $|R| = 10$ and $selfConfidence(c) = 30\%$.

2.3 Analysis of Existing Proposals

The differences between the reputation and the recommendation is important. A node can have a quite bad behavior in the group (due to energy problem, for instance), but always a correct recommendation. In the opposite, an attack would consist in acting well, in order to avoid attack detection mechanisms, and lying about the reputation of other nodes, in order for example to obtain privileges.

In Jinshan Liu and Valérie Issarny study and in others which are similar, such as [8,9], the reputation and recommendation systems have some flaws: calculi are based on all the nodes of the networks. The first and most obvious issue is the scalability problem. However, a more important problem happens when the detection of malicious behaviors can be performed only in local area: when the group size increases, no significant reputation decrease may occur. Consider the following example:

- the detection of malicious behaviors can be performed only on direct neighbors, which is often the case for the lowest levels of the ISO model
- we have N nodes, and we consider that each node has k neighbors
- we assume that each node gives correct recommendations

In the figure 1a, we can see that the reputation mechanism suffers from scalability problems, when the number of nodes increases. In the referee based approach [2], the authors suggest to take into account only the nodes that belong to the referees R . This prevents the case illustrated in the figure 1 from occurring. However, no distinction is made between the reputation and the recommendation. This study is thus relevant to detect incorrect behaviors for the services, using neighbors' cooperation, but cannot be used to detect malicious nodes inside the network.

We have seen that existing proposals fail when some conditions occur, such as when the number of nodes increases while the detection region remains the

same. These problems arise because formal analyses of the system have not been performed. For instance, the notion of local region R presented in the referee-based system is not fully described: how can we evaluate R ? What are the impact of the size of R on the reputation mechanism? As these questions have not been answered, flaws may be discovered in the future. We can conclude that the definition of a reputation system requires a formal analysis of the system and the environment, which is not performed in current proposals.

3 Formal Model for Reputation and Recommendation Functions

As described in the previous section, reputation systems must be developed using a formal approach. In this section, we describe our reputation functions. An example of application of the reputation systems is the management of groups. We thus present the group decision principle and finally the security properties reputation systems must satisfy, based on these decisions. Formal analyses are presented in the next section.

3.1 Definitions of Our Reputation Model

As presented in the section 2, we are able to have a scalable mechanism by using local region. However, local regions engender local reputations. In our case, as we want to be able to take the same decisions on the whole group (property 4 of the section 3.2), we must have global reputations. This thus prevents nodes from waiting the acknowledgement of their decisions.

As the notion of reputation is bound to the recommendation, the way the recommendation is evaluated is also not correct for group decisions. In fact, the recommendation we need is a group recommendation, and not a node-dependent recommendation, as presented in Jinshan Liu and Valérie Issarny research work. In this study, the recommendation is defined by

$$rec_k(i) = rec_{k-1}(i) \cdot \rho_{rec} + (1 - \rho_{rec}) \cdot \frac{\sum_{j=0}^n diff(rep_{k-1}(j, i), rep_{k-1}(j))}{n}$$

where $rep_k(i, j)$ is the reputation of i declared by j at the step k . In this formula, the function $diff$ is used to evaluate the differences between the recommendations made by the current node and the ones made by the node i .

In order to have a global recommendation, we must evaluate the difference of the node's evaluations with all the other nodes' evaluations. Our notion of group reputation is defined as the following:

$$group_reputation_k(i) = \frac{\sum_{j \in R_i} rec_{k-1}(j) \cdot rep_k(i, j)}{\sum_{j \in R_i} rec_{k-1}(j)} \quad (1)$$

Using this formula, we get the following initial group recommendation:

$$rec_k(i) = rec_{k-1}(i) \cdot \rho_{rec} +$$

$$(1 - \rho_{rec}) \cdot \frac{\sum_{j=0}^n \text{diff}(\text{rep}_k(j, i), \text{group_reputation}_k(j))}{n} \quad (2)$$

Note that the recommendation function is studied in section 4.3, as it does not affect the evaluation of our reputation function in the worst cases.

Finally, the reputation is similar to the one presented in [2]:

$$\text{rep}_k(i) = \frac{100 \cdot \text{experiences} + \sum_{j \in R_i \wedge j \neq \text{myself}} \text{rec}_{k-1}(j) * \text{rep}_{k-1}(i, j)}{100 + \sum_{j \in R \wedge j \neq \text{myself}} \text{rec}_{k-1}(j)} \quad (3)$$

3.2 Group Decisions Principle

In a group management algorithm, we can find two groups of operations for group management protocols: group operations and group agreements. The first group describes all the basic decisions, such as «a request to add a node» while the second one describes all the group decisions, such as «the group adds a node». This distinction is important as the first operations can be decided by a single node while the second ones have to be decided by the whole group.

Group Operations. As described above, these operations are made by a single node: depending on several parameters, a node may want to authorize a new node to join a group, or may want to evict a node from the group.

Adding a node: A node n_i sends an adding message to the group if the local reputation of the node to add is higher than or equals to threshold_{Add} .

Removing a node: As for adding a node, a node sends an eviction message about the node n_m if the node n_m has a reputation lower than or equal to threshold_{Evict} .

Group Agreements. An important aspect of the group agreements is to have common decisions: if a node starts a removing or adding operation at the protocol group layer, all nodes in the network must do it too. In order to have stable group decisions, we define several functional properties. They rely on the variables τ_{add} , $\tau_{eviction}$ and $\text{minimal_recommendation}$ which respectively refer to the minimal number of nodes to take an adding message into account, the minimal number of nodes to take an eviction message into account and the minimal recommendation to consider a node's message as trustworthy. Finally, the variable τ is linked to security of the systems : $\tau - 1$ is the maximal number of malicious nodes the system supports. Thus, we have $\tau \leq \tau_{add}$ and $\tau \leq \tau_{eviction}$.

For the group decisions, there are mainly four functional properties:

Property 1: In order to start an adding operation, a node must have received τ_{add} adding messages from distinct nodes in the network.

Property 2: In order to start an eviction, a node must have received $\tau_{eviction}$ eviction messages from distinct nodes among the network.

Property 3: A node message should be taken into account only if the node recommendation is higher than or equal to $\text{minimal_recommendation}$.

Property 4: Upon receiving a group management operation, each node of the group must take the same decision.

3.3 Basic Security Properties

In the previous section, we have presented the functional properties our reputation system must satisfy. However, in order to develop a robust system, we must also state the security properties our system must satisfy.

The first one deals with the impact of the reputation increase rate.

Security Property 1: the collusion of malicious nodes must not engender an eviction of a correct node

For the reputation decrease scenario, two security properties are defined.

Security Property 2: A collusion of malicious nodes must not prevent a malicious node from having a decrease of its reputation.

Security Property 3: The group must be able to evict a malicious node, according to the functional properties, when its reputation exceeds a defined threshold.

Finally, in order to prevent malicious nodes from interfering with correct information about a node, their recommendation must decrease. This is expressed by the forth security property:

Security Property 4: a node recommendation must decrease if it acts maliciously.

4 Theoretical Quantification of the Model's Parameters

In the previous section, we described our reputation and recommendation functions. In this section, we analyse the parameters of these functions and the impact of their values on the reputation system and the assessment of the security properties. In subsections 4.1 and 4.2, we introduce the global ideas about the reputation functions evaluation and our solution, which solves three principal problems: what is the value of the reputation increase rate if a node acts well? what is the value of the reputation decrease rate if a malicious node is detected? How can we define the local region R of a node? Finally, the evaluation of the recommendation function is given in section 4.3.

The evaluation of the different parameters is made by first formulating the worst cases that can occur. We then specify values that satisfy our security properties. Due to space limitation, complete demonstrations of the mathematical equations are not given in this paper but one can refer to [10].

4.1 Reputation Increase Assessment

Worst Case 1: Incorrect Eviction. The usual worst case is related to the eviction by malicious nodes of a node acting well. This can be represented by the following scenario: $\tau - 1$ malicious nodes declare a reputation of 0 for this node while others increase its reputation by α .

Table 2. Minimal value of α depending on $Evic_{threshold}$, $|R| = 4 \cdot \tau$

$Evic_{threshold}$	α_{min}	$Evic_{threshold}$	α_{min}	$Evic_{threshold}$	α_{min}	$Evic_{threshold}$	α_{min}
10	4	30	10	20	7	40	14

According to the Group Agreement Property 2, the eviction of a node occurs if $\tau_{eviction}$ nodes send an eviction message. As $\tau_{eviction} \geq \tau$, this means that at least one «correct» node has to send an eviction message. Thus, to satisfy the Security Property 1, we must ensure that no correct node sends an eviction message. This can be ensured by the following requirements:

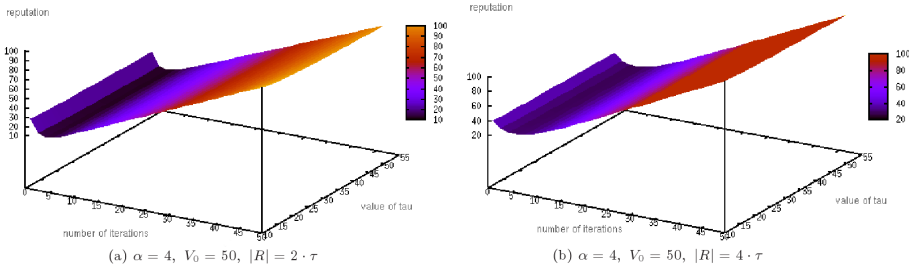
- the reputation does not go under the eviction threshold $Evic_{threshold}$ (*Req1*)
- the reputation is still able to increase (*Req2*)

To satisfy the first requirement, we must assure that there is no $i \in N$ such that $rep_i < Evic_{threshold}$. At the n th round, the reputation of the attacked node is given by (rep_0 is the initial reputation): $rep_n = rep_0 \cdot a^n + b \cdot \sum_{i=0}^{n-1} a^i$ where $a = \frac{|R|-\tau+1}{|R|}$ and $b = \alpha \cdot \frac{|R|-\tau+1}{|R|}$. Based on this formula and considering different eviction thresholds $Evic_{threshold}$, the table 2 illustrates the different values of α_{min} that satisfy *Req1*.

For the second requirement (the reputation is still able to increase), we can analyze the impacts of the reputation system parameters with several scenarios. We considered the following ones, where V_0 is the initial value of the reputation:

- $\{\alpha = 4, V_0 = 50, |R| = 2 \cdot \tau\}$ (figure 2a)
- $\{\alpha = 4, V_0 = 50, |R| = 4 \cdot \tau\}$ (figure 2b)
- $\{\tau = 20, V_0 = 50, |R| = 4 \cdot \tau\}$ (figure 3)

We can see that as τ is proportional to $|R|$, its value does not interact with the reputation increase rate. However, the way $|R|$ is evaluated does interact with the reputation increase rate. For instance, with $|R| = 4 \cdot \tau$, we manage to get a maximal reputation (i.e. 100) faster than with $|R| = 2 \cdot \tau$. The choice of $|R| = 4 \cdot \tau$ is due to several reasons. First, the increase rate is more important than with $|R| = 2 \cdot \tau$, which means that correct nodes will reach the maximal (and thus the best) reputation faster. Secondly, if one decide to take R such that $|R| = 6 \cdot \tau$

**Fig. 2.** Reputation increase - minimal rates

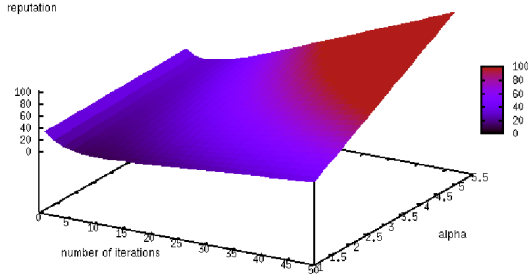


Fig. 3. Reputation increase according to α 's value, with $|R| = 4 \cdot \tau$

or $|R| = 2^\tau$, we would have better results but the size of $|R|$ would increase very quickly, which means that τ_{max} would be far less important and that nodes would have to keep a watch on more nodes. Obviously, as for R , several values for α can be taken into account. We decide to consider $\alpha = 4$, as the increase rate is correct (and $4 > \alpha_{min}$ for $|R| = 4 \cdot \tau$).

Worst Case 2: Incorrect Increase Rate. Another problem occurred when malicious nodes cooperate in order to quickly increase a node's reputation: all of them decide to give a value of 100 to the reputation. This is represented by the formula $rep_k = \frac{100 \cdot (\tau - 1) + (rep_{k-1} + \alpha) \cdot (|R| - \tau + 1)}{|R|}$. In this case, we must choose a value of α which leads to a correct reputation increase. The formula can be represented by $rep_k = rep_0 \cdot a^n + b \cdot \sum_{i=0}^{n-1} a^i$, where $a = \frac{|R| - \tau + 1}{|R|}$ and $b = \frac{100 \cdot (\tau - 1) + \alpha \cdot (|R| - \tau + 1)}{|R|}$.

As we can see in the figure 4, the reputation of the malicious node evolves very quickly, no matter the value of τ : with $|R| = 4 \cdot \tau$, five iterations are needed to get the maximal reputation, starting from a value of 50 while it is of three for $|R| = 2 \cdot \tau$. A solution to this problem is to find a way to decrease in all the cases the reputation of the malicious nodes.

Case 3: Common Case. Finally, the common case is when each node increases the reputation of α . We must choose parameters values such that the increase rate is not too fast, in order to prevent malicious nodes from recovering a good reputation too quickly. In this case, the evolution formula is $rep_k = \frac{(rep_{k-1} + \alpha) \cdot |R|}{|R|} = (rep_{k-1} + \alpha)$. So, the increase is equal to α . With a value of 4 for α , 13 iterations are needed to get a maximal reputation, starting from a reputation of 50.

According to the different possible cases, we can see that a value of 4 for α and a value of $4 \cdot \tau$ for $|R|$ are interesting.

4.2 Reputation Decrease Assessment

Standard Reputation Decrease. The worst case of the reputation decrease scenario is the following one: all the malicious nodes cooperate in order to prevent

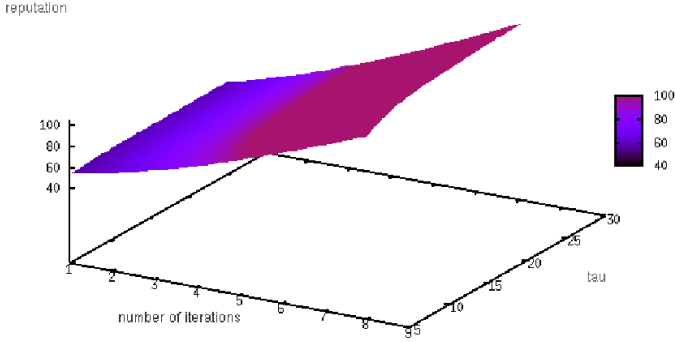


Fig. 4. Reputation increase - maximal increase rate with $\alpha = 4$

the decreases of a malicious node reputation. They send a reputation of 100 and others decrease the malicious node's reputation of β .

In this scenario, we must choose the size of R and β so that the reputation will still decrease. Moreover, we must choose a value of β that decreases in a significant way the malicious node reputation, in order to increase the time this node requires to recover the maximal reputation (also called the recovering time). With β a constant, we have the following worst case:

$$rep_k = \frac{100 \cdot (\tau - 1) + (rep_{k-1} - \beta) \cdot (|R| - \tau + 1)}{|R|}, \text{ where } rep_0 = 100$$

According to the section 4.1, we can analyze several values of β , which are described in the table 5 (with $\tau = 25$, $|R| = 4 \cdot \tau$). The main idea is to choose a correct value of β that implies a long recovering time, which tends to decrease the number of bad behaviors. For instance, with $\beta = 25$, a malicious node has to wait for five iterations before getting its maximal reputation back. If it acts maliciously during each reputation update intervals, its reputation will decrease and it would have a reputation of 50 after 5 iterations and a reputation of 30 after 15 iterations.

However, a drawback is that we may not be able to get a reputation of $threshold_{Evict}$ for malicious nodes, depending on τ and β . Moreover, if we use usual equations, a special case cannot be taken into account: a malicious node acts badly, waits for its reputation to increase and restarts to act badly. We need a group history to take this case, namely the Moral Hazard [11] (byzantine behavior), into account. Thus, the current reputation function does not satisfy the security property 3 and has to be modified.

decrease rate	β	recovering time (nb of iterations)	decrease rate	β	recovering time (nb of iterations)
10	7	3	30	43	8
20	25	5	40	60	10

Fig. 5. Influence of β on the reputation decrease

Dynamic reputation decrease. In a study made by Ba & Pavlou [12], an analysis of ebay's reputation mechanism has been made. Based on ebay's reputation results, they modelled the ebay trust system with a correlation between positive rates (PR) and negative rates (NR). It is given by the following formula: $Trust = \beta_0 + \beta_1 \cdot \text{Log}(PR) + \beta_2 \cdot \text{Log}(NR)$.

In our situation, positive rates are implicit: a node increases the reputation of the other nodes at each check, if this node does not have a bad behavior. Our current equation takes negative rates into account with the variable β , in which $\beta = f(NR)$. Using NR , we obtain $\beta(NR) = \beta_0 + f(NR) \cdot \beta_1$. The principal scheme of f is that $f(0) = 0$ and $f(NR_{max}) = \frac{100}{\beta_1} - \beta_0$ (as $\beta(NR_{max}) = 100$). A common aspect of $\beta(NR)$ would be that its value is reduced by 2 at each bad behavior. Thus, with $\beta(NR) = \frac{100}{2^{NR_{max}}} \cdot 2^{NR}$, we have a reputation decrease that satisfies the Security Property 2 and 3.

4.3 Evaluation of the Recommendation Functions

In existing studies, the recommendation function is the following:

$$rec_k(i) = rec_{k-1}(i) \cdot \rho_{rec} + (1 - \rho_{rec}) \cdot \frac{\sum_{j=0}^n diff(rep_{k-1}(j, i), X_reputation_k(j))}{n} \quad (4)$$

where $X_reputation_k(j)$ can be $group_reputation_k(j)$ or the node reputation, for node-oriented reputation mechanisms. As for the reputation mechanism, we can see that this function is linked to the size of the group. Thus, a simple attack from the malicious nodes would be to target a single node. As shown with the simulation results (section 5.1), we got a stabilized state where the malicious nodes' recommendation are still high (94%) while the attacked node's reputation is low. In this case, the Security Property 1 is not satisfied.

By analyzing these drawbacks of the recommendation mechanism, we first propose the function 5, which is not group size-dependent. By using the *multiply* operation instead of the *sum* one, isolated lies are not hidden and the cumulation of lies amplify the recommendation decrease.

$$rec_k(i) = rec_{k-1}(i) \cdot \rho_{rec} + (1 - \rho_{rec}) \cdot \frac{\prod_{j=0}^n diff(rep_{k-1}(j, i), group_reputation_k(j))}{n} \quad (5)$$

This function is thus robust against the mentioned attack. However, if we consider intelligent malicious nodes, similar drawbacks remain: in this function, the decrease rate is directly associated to the difference between what the malicious node says and the $group_reputation$ value. Thus, by sending reputation values that are lower than the $group_reputation$, but not so far, malicious nodes can still lie about others' reputation and the decrease of their recommendation will not be important. Moreover, advanced attacks such as the binary state {correct, malicious} can impact the reputation mechanism. So, though attacks need to be more sophisticated, the mechanism may still be affected by the collusion of malicious nodes.

In our case, we made a strong assumption which has not been taken into account yet: we decided to choose R such that $R = 4 \cdot \tau$, with $\tau - 1$ being the maximal number of malicious nodes our system has to support. Thus, we assume that at least 75% of the nodes among R are not malicious. Moreover, the choice of R (and τ) is made such that each node among R is able to detect if a node is acting maliciously or not at the group layer. We are thus assured that most of the reputation values are correct. So, we can compare a node recommendation with the majority value, instead of the group value with the following function (in which $majority_reputation(k)$ refers the majority value for the reputation about the node k):

$$rec_k(i) = rec_{k-1}(i) \cdot \rho_{rec} + (1 - \rho_{rec}) \cdot lieValue(i)$$

$$lieValue(i) = \begin{cases} 0 & \text{if } \exists j \in R/rep_k(j, in) \neq majority_reputation_k(j) \\ 100 & \text{otherwise} \end{cases} \quad (6)$$

As when a node lies its recommendation is set to 0, no matter how much incorrect information it provides, it is obvious that the Security Property 4, *a node recommendation must decrease if it acts maliciously*, is satisfied.

5 Simulations

5.1 Results and Comparisons

In the previous section, we have presented several recommendation evaluations. In order to compare them, we have used the NS-2 [7] simulator with the UM-OLSR [13] implementation of the OLSR Ad hoc routing protocol.

In the simulation, we have compared the different recommendation functions:

- $\sum 10$ and $\sum 25$ refer to the equation 4 with τ equal to 10% and 25%
- $\Pi 10$ and $\Pi 25$ refer to the equation 5 with τ equal to 10% and 25%
- $\Pi 10b$ refers to the equation 5, with $\tau = 10\%$ and the attack which consists in alternating correct and malicious behaviors.
- *lying10* and *lying25* refer to the equation 6 with τ respectively equal to 10% and 25%

We compared the functions by using the following issues: when do the malicious nodes' recommendation (figure 6a) and the attacked node reputation (figure 6b) are stabilized? What are the stabilized recommendation (figure 6c) and reputation (figure 6d)? According to the security property 4, the recommendation value of the malicious nodes must be null. It is easy to see that this property is not assured by the standard recommendation evaluations. With the updated recommendation evaluation we suggest (equation 5), the recommendation evaluation and the reputation evaluation are correct in the case of basic malicious nodes, with $\tau = 10\%$. However, when we reach the extreme case $\tau = 25\%$, the attacked node's reputation is impacted. With $\tau = 10\%$ and advanced malicious nodes, the attacked node's reputation is not really malicious and the malicious

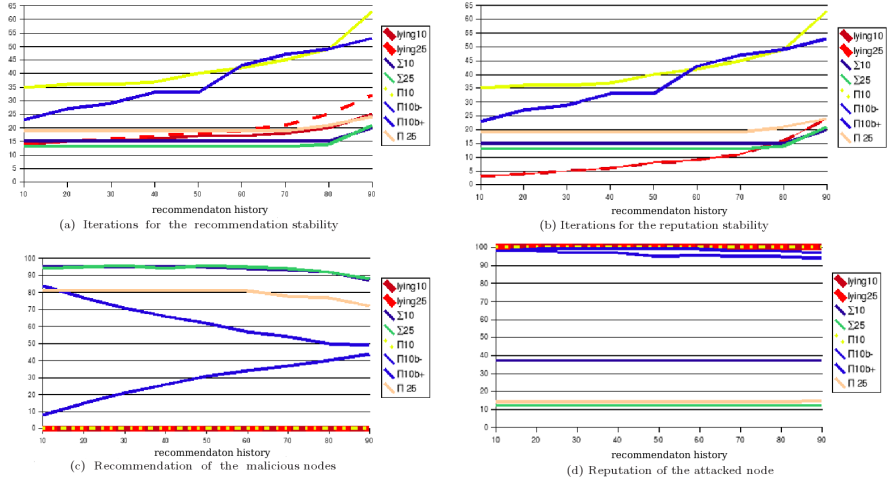


Fig. 6. Comparison of recommendation evaluation functions

nodes' recommendations are neither considered as good nor bad. Finally, we can see that the lying method provides really good results, as malicious nodes recommendations are always null and the stabilized states are quickly reached. Thus, our proposals respect our security properties and the system stability is quickly reached, which is important in ad hoc networks.

5.2 Evaluation of the History Parameter ρ_{rec}

The parameter ρ_{rec} defines the importance of the history and thus will have consequences and the system's evolution. In this case, the choice of ρ_{rec} is important. For instance, with $\rho_{rec} \sim 0$, an incorrect behavior will have immediate repercussion, while it is not the case with $\rho_{rec} \sim 1$.

The table 3 illustrates the importance of ρ_{rec} in several cases which are parts of the worst cases presented in section 4.1:

- $stability_1$ illustrates the recommendation decrease of a malicious node in the worst case 1 of the reputation increase
- $stability_2$ illustrates the recommendation increase rate in the common case
- $stability_3$ illustrates the recommendation decrease of correct nodes in the reputation decrease case, starting with a reputation of 100
- $stability_4$ illustrates the recommendation decrease of malicious nodes in the reputation decrease case, starting with a reputation of 100

With $\rho_{rec} \sim 1$, the recommendations of the malicious nodes and the attacked nodes decrease very slowly. This is the opposite in the case of no recommendation history. By choosing $\rho_{rec} = 0.2$, we limit the recommendation decreases of the correct nodes, and we also reduce the increase rate in common states, which prevents malicious nodes from alternating correct and malicious behaviors.

Table 3. Influence of ρ_{rec} on the reputation mechanism, $\tau = 25\%$

ρ_{rec}	<i>stability</i> ₁ (%)		<i>stability</i> ₂ (%)	<i>stability</i> ₃			<i>stability</i> ₄		
	<i>H</i>	lying		$\beta = 20$	$\beta = 40$	lying	$\beta = 20$	$\beta = 40$	lying
0	14	100	100	5	10	0	15	30	100
0.1	12.6	90	90	4.5	9	0	13.5	27	90
0.2	11.2	80	80	4	5	0	12	24	80
0.5	7	50	50	2.5	5	0	7.5	21	50
best	max	max	max	min	min	min	max	max	max

6 Conclusion

In this article, we have shown that designing a reputation and recommendation mechanism at the group layer requires to develop a reputation shared between the nodes and not a local reputation, as proposed in existing studies. This kind of system relies on many parameters, such as update rates, synchronization intervals and thresholds, which are linked together in complex ways. We have defined basic security properties (such as *the collusion of malicious nodes must not engender an eviction of a correct node*) whose enforcement requires a correct setting of the system parameters. We have analyzed the system parameters and determined values that satisfy our security properties.

Moreover, as the recommendation aspect is as important as the reputation aspect, we have studied the existing recommendation evaluation. We have shown that the basic principle *a node recommendation must decrease if it acts maliciously* is not assured in the worst cases, which may engender incorrect stabilized states. We have then proposed two modifications of the evaluation scheme: a recommendation function that improves the existing function and a new one, designed under hypotheses about the group environment, whose results are even better.

Our reputation system may be used in different contexts, such as the group management in ad hoc networks, as a reinforcement of existing proposals such as [6], and the reinforcement of routing protocol with misbehaviors detection [14].

References

1. Liu, J., Issarny, V.: Enhanced reputation mechanism for mobile ad hoc networks. In: Jensen, C., Poslad, S., Dimitrakos, T. (eds.) iTrust 2004. LNCS, vol. 2995, pp. 48–62. Springer, Heidelberg (2004)
2. Conrad, M., French, T., Huang, W., Maple, C.: A lightweight model of trust propagation in a multi-client network environment: To what extent does experience matter? In: ARES 2006: Proceedings of the First International Conference on Availability, Reliability and Security (ARES 2006), pp. 482–487. IEEE Computer Society, Washington (2006)
3. Guha, R., Kumar, R., Raghavan, P., Tomkins, A.: Propagation of trust and distrust. In: WWW 2004: Proceedings of the 13th international conference on World Wide Web, pp. 403–412. ACM Press, New York (2004)

4. Wong, C.K., Gouda, M.G., Lam, S.S.: Secure group communications using key graphs. In: Proceedings of the ACM SIGCOMM 1998 conference on Applications, technologies, architectures, and protocols for computer communication, pp. 68–79 (1998)
5. Sherman, A.T., McGrew, D.A.: Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering* 29(5), 444–458 (2003)
6. Cuppens, F., Cuppens-Boulahia, N., Thomas, J.A.: STGDH: An enhanced group management protocol. In: Proceedings of the CRISIS Conference, Marocco, Marrakech (July 2007)
7. Fall, K., Varadhan, K.: The ns Manual, <http://www.isi.edu/nsnam/ns/doc/>
8. Anantyaloe, T., Wu, J.: Reputation-based system for encouraging the cooperation of nodes in mobile ad hoc networks. In: IEEE ICC, June 24–28 (2007)
9. Carbone, M., Nielsen, M., Sassone, V.: A formal model for trust in dynamic networks. In: Cerone, A., Lindsay, P. (eds.) Proceedings of Int. Conf. on Software Engineering and Formal Methods, SEFM 2003, pp. 54–61. IEEE Computer Society, Los Alamitos (2003)
10. Cuppens, F., Cuppens-Boulahia, N., Thomas, J.A.: Malevolence detection and reactions in ad hoc networks. Technical report (June 2007)
11. Dembe, A.E., Boden, L.I.: The story of the moral. In: New Solutions, pp. 257–279 (2002)
12. Ba, S., Pavlou, P.A.: Evidence of the effect of trust building technology in electronic markets: Price premiums and buyer behavior. *MIS Quarterly* 26(3) (2002)
13. Ros, F.J., Ruiz, P.M.: Implementing a New Manet Unicast Routing Protocol in NS2. Technical report, Dept. of Information and Communications Engineering University of Murcia (December 2004)
14. Cuppens, F., Cuppens-Boulahia, N., Ramard, T., Thomas, J.A.: Misbehaviors detection to ensure availability in olsr. In: Zhang, H., Olariu, S., Cao, J., Johnson, D.B. (eds.) MSN 2007. LNCS, vol. 4864, pp. 799–813. Springer, Heidelberg (2007)

Complexity of Checking Freshness of Cryptographic Protocols*

Zhiyao Liang and Rakesh M. Verma

Computer Science Department, University of Houston,
Houston TX 77204-3010, USA
zliang@cs.uh.edu, rmverma@cs.uh.edu

Abstract. Freshness is a central security issue for cryptographic protocols and is the security goal violated by replay attacks. This paper is the first to formally define freshness goal and its attacks based on role instances and the attacker's involvement, and is the first work to investigate the complexity of checking freshness. We discuss and prove a series of complexity results of checking freshness goals in several different scenarios, where the attacker's behavior is restricted differently, with different bounds on the number of role instances in a run.

Keywords: Cryptographic protocols, freshness, replay attack, challenge response, model checker, undecidability, NP-completeness, Athena.

1 Introduction

Security of communication protocols is critical in this age when computer communication is ubiquitous. An important research direction in verifying communication protocols is checking attacks while assuming perfect cryptography and a dominant attacker in the network, commonly referred as the Dolev and Yao attacker model [1]. Many researchers follow Dolev and Yao attacker model. Much research on the complexity of checking security goals has focused on checking secrecy [2] [3] [4] [5] [6] [7] [8].

Freshness is a central and fundamental issue of communication protocols [9]. The common ways to maintain freshness of terms of a protocol, without arguing the exact definition of freshness, are by using timestamps or by challenge-response [10]. Intuitively, in a protocol run a term may be considered as “fresh” in two aspects: how the term is created, and how the term is received. When a term is created we may say it is fresh, or it is new or not stale, if it is not created before a certain time, while time can be measured by the timestamps of terms. In [5] freshness means uniqueness, which means when a fresh term is created it should not have appeared in the run before, and this approach may also be categorized as emphasizing the freshness of a term on the creation aspect. The freshness of terms on the reception aspect, if implemented by timestamps, could

* Research supported in part by NSF grants CCF 0306475 and CNS 0755500.

mean that only terms with timestamps after a certain time points can be received in a certain situation. When a regular agent A participates in a protocol run, it is guaranteed that the A will create really fresh terms such as nonces if we assume unbounded creation of fresh terms, but what can cause security failure are the terms that are received by A , which could be not “fresh” when they are supposed to be “fresh”. Therefore we think to define the freshness of terms on the reception aspect is what really matters and deserves more attention. Timestamps have the limitation of relying on a precise global clock. On the contrary to timestamps, the challenge-response mechanism indirectly restricts how a term can be received and accepted, i.e. a challenge must be passed in order to let a term be accepted. Comparing the challenge-response approach and the timestamp approach to implement and define freshness, we consider the former has wider coverage of cryptographic protocols. Obviously the former is more complicate to analyze. In this paper we address the freshness goal that may be implemented by challenge-response. Further discussions on the challenge-response mechanism and its relationship with freshness are provided in [11].

The contributions of this paper are as follows.

- We define freshness goal based on role instances, the terms that are supposed to be fresh, and the attacker’s involvement.
- We address different scenarios where the attacker’s behavior in a run is restricted differently. We define three kinds of replay attacks that violate freshness goals, called direct, restricted and general replay attacks.
- We address three bounds on the number of role instances in a run (NRI), including fixed, individually bounded, and unbounded. This paper is the first to clarify the difference between the three bounds.
- We address and prove a series of the complexity results of checking freshness for DRA , RRA , and GRA , when NRI is fixed, individually bounded, or unbounded. These results are non-trivial to prove. For example, the proof of Theorem 5, which shows the NP-completeness of checking RRA when NRI is fixed, is quite delicate.
- We analyze the performance of the model checker Athena [12] [13]. We improve the presentation, semantics, and efficiency of the algorithm of Athena.

To the best of our knowledge, the closest definition of a freshness goal that is independently defined by other researchers appears in [14]. However our work is significantly different from [14], and cannot be covered by [14] for the following reasons. First, in [14] the authors demonstrate that the freshness goal can be expressed using the constraint solving system, but there is no complexity investigation. It is obvious that the freshness goal, which is defined later in this paper, can be expressed in different systems, since its definition is simple and clear. Second, the attacker’s involvement is not discussed in [14] for the definition of the freshness goal. Third, the definition of a freshness goal in [14] is less generally applicable than the one defined in this paper, which is discussed later. Fourth, the discussion on freshness in [14] is sketchy and is not the focus of [14].

We do not need to follow the detailed behavior of challenge and response in order to investigate the complexity of checking freshness. Therefore our approach is rather different from, and cannot be covered by other papers that design rules or logic to address the details of challenges and responses, such as [15] and [16], and they do not address complexity issues.

One approach of studying the complexity of checking freshness is to reduce the problem of checking freshness to the problem of checking secrecy or the other way, since there are published results of the complexity of checking secrecy (but not much for authentication). But this approach requires that convincing proofs of the corresponding complexity results of checking secrecy are available.

Based on our works in [7] and [8], which improves the work of [3], of proving undecidability results by direct reductions from the well-known reachability problem of 2-counter machines, we think this direct approach is convenient to prove the undecidability results of checking freshness and may be easier for readers to verify, especially when the ideal proof of checking secrecy for the corresponding setting is not obvious or not easily available. Therefore the proof of Theorem 1 is by this direct reduction.

Reduction from checking secrecy to checking freshness can be done, and the idea is demonstrated in the proof of Theorem 2. Such a reduction may be useful to show that checking freshness is undecidable when the number of role instances (NRI) is unbounded, and NP-hard when NRI is fixed, since we believe checking secrecy has the same complexity results correspondingly. We present two different proofs for Theorem 2, one by a direct reduction from 2-counter machine, and the other by a reduction from secrecy. Details of these proofs are provided in [11]. The NP-hardness proof of checking secrecy we have noticed is provided by [4], which is by reduction from 3-SAT. However that proof assumes protocols as non-matching roles instead of communication sequences, and the secret term is declared in a non-realistic way, which are unconvincing aspects as discussed in [17] and [8]. Therefore in this paper we prove the NP-hardness of checking freshness in Theorem 5 by a direct reduction from 3-SAT.

Reduction from freshness to secrecy can also be done, as demonstrated in [11], although it may not be obvious. Such a reduction may show that checking freshness is NP when NRI is fixed since we believe checking secrecy is NP with fixed NRI, and this complexity result have been addressed in [4]. However, besides the significant contributions made by [4], we think there is an error in the proof of [4] to show checking secrecy is NP. More specifically the error is due to an assumption in the proof of Theorem 1 in [4] that the DAG size of a substitution of a term is no less than the DAG size of the term, which is incorrect. More details of the error is described in [11]. Currently we are studying another related paper [18]. Therefore we prove the NP result of checking freshness by directly analyzing a model checker, not by reduction from secrecy.

For space limit most of the detailed proofs are in the technical report [11], while in the paper we present the essential definitions and proving methods.

2 Notations and Modeling

[Notations]. Notations are chosen in a style that is commonly used in the literature. More details of notations and modeling can be found in [11]. A *term* is either an atomic term or a composite term. An *atomic term* is a *variable* (represented by a symbol with at least one upper case letter), or constant (a symbol without any upper case letter). The special constant I is the name of the attacker. Asymmetric keys are atomic terms. The established public key and private key of an agent A is k_A^1 and k_A^0 respectively. A *composite term* is a list, or an asymmetric encryption, or a symmetric encryption. A *list* has the form of $[X, Y, \dots]$, where X and Y are terms and the list contains finite number of member terms. A list is a simpler representation of a sequence of nested pairs. For example $[W, X, Y, Z]$ is the same as $[W, [X, [Y, Z]]]$. When a message is a list, the top level enclosing $[]$ is omitted. A term constructed by encryption algorithms is called an *encryption*. An *asymmetric encryption* has the form of $\{T\}_{k_A^i}^{\rightarrow}$, $i \in \{0, 1\}$, where T is called the *text*, and k_A^i , for $i \in \{0, 1\}$, is the atomic encryption key, and it can be decrypted using the key k_A^{1-i} . A *symmetric encryption* has the form of $\{T\}_Y^{\leftrightarrow}$, where T is the text and Y is any term working as the encryption key. When a list, say $[X, Y, Z, \dots]$ is encrypted symmetrically or asymmetrically, the enclosing square brackets are removed within “ $\{ \}$ ”. The word *ground* means variable free.

The set of **blocks** of a term T , denoted as $blocks(T)$, is defined as follows:

- If T is an encryption or an atomic term, $blocks(T) = \{T\}$.
- If $T = [X, Y]$, then $blocks(T) = blocks(X) \cup blocks(Y)$.

An **action step** can have one of the following three forms. A term sent or received in an action step is called a **message**.

- If agent P generates a set of (at least one) fresh terms, and then sends a message Msg to agent B , then the action step has the form

$$\#_P(T_1, T_2 \dots) + (P \Rightarrow B) : Msg.$$

Here $\#_P(T_1, T_2 \dots)$ is the fresh term generation action of P to generate the fresh terms T_1, T_2 etc. These fresh terms should appear as subterms in Msg .

- If agent P sends a message Msg to B without generating any fresh terms, then the action step is denoted as $+(P \Rightarrow B) : Msg$.
- If P receives a message Msg , and by the context of the communication or by analyzing Msg , P considers the supposed sender of Msg should be B , then the action step has the form $-(B \Rightarrow P) : Msg$.

A **communication step** can also have three possible forms, which are the same as an action step, except the $+$ and $-$ signs are not used. A communication step implies two corresponding action steps, one is the message sending, maybe with nonce generation, by the sender, and the other is the message receiving by the receiver. A **communication sequence**, or simply **CS**, is a sequence of communication steps numbered sequentially starting from 1. A protocol is

commonly described as a CS accompanied with other information including the initial knowledge patterns of agents.

[Model of protocol run]. Here we present the essential model of protocol run. A verbose model can be found in [11]. We assume the *free term algebra*, which means that two different symbolic terms must represent two different bit-strings of the real world. This assumption is commonly for the Dolev-Yao model. We assume *unbounded fresh nonce generation*, which means that when a nonce is generated, it is always different from other nonces and all the terms appeared in the run before its generation or initially known to some agent.

A **role** is a sequence of action steps executed by the same agent A obtained by parsing the CS of a protocol, which is called A 's role.

An **event** is a tuple $\langle act, time \rangle$. act is a ground action step, described earlier. The $time$ field of an event e is referred as $e.time$, which is a positive real number representing when the event occurs after the start of the run.

A **role instance** r of role R , such that the action steps of the sequence of events in r instantiate a prefix of the sequence of action steps, not necessarily all of the action steps, of R , by a ground substitution. For two events ev and ev' in r , if their message numbers in the corresponding role R are n and n' respectively, and $n < n'$, then $ev.time < ev'.time$.

In a run the **attacker** is associated with a set of ground terms that are initially known to I , denoted as $I.init$. The attacker can analyze and synthesize terms and create nonces by following a set of standard rules, as discussed by [19] and [4]. Given a certain set E of events that have occurred in a run, based on $I.init$ and the messages sent by E , $know_I(E)$ represents the (infinite) set of terms that can be derived following these standard rules.

A **run** is a tuple: $\langle Pro, D, R, AN, E \rangle$ Pro is the protocol. D is the initial knowledge pattern of the attacker who is involved in the run. R is a set of role instances that are executed honestly by regular agents. AN is the set of ground names of the agents who participate in the assumed perfect initial knowledge establishing stage of the run, including all of the regular agents and sometimes the attacker. The agents in AN are insiders. E is a set of events that occur in the run, including, nothing more and nothing less, all of the events of in the role instances in R . The set of **time points** of run is defined as $\{t \mid t = \eta.time, \eta \in run.E\} \cup \{0\}$. Given a time point t , we define $E_{<t}$ as the set of events $\{\eta \mid \eta.time < t, \eta \in run.E\}$. The following conditions should be satisfied: For any event η in E , if η receives a message msg then $msg \in know_I(E_{<\eta.time})$. The set of all possible runs of a protocol Pro and with some specific initial knowledge pattern D of the attacker, is denoted as $Runs^{D:Pro}$. Note that we allow two events to occur at the same time in a run, which is different from the trace based models like [19] and [6], but agrees with the strand space model [20].

[Definition of Freshness and Its Attacks]

Definition 1. Given a certain pattern D of the attacker's initial knowledge, and a protocol Pro , where a role of A receives a term X which should be freshly generated by B 's role such as a nonce variable, the **freshness** of X to A 's role is that it is impossible to have a run , $run \in Runs^{D:Pro}$, where there are two

different role instances r and r' of A 's role, such that the following two conditions are satisfied:

- In r and r' , B is not instantiated by I , which is the attacker's name.
- The same ground term, say c , instantiates X in both r and r' . □

Note that in the above definition r and r' do not need to be executed by the same agent, i.e. A may be instantiated by different agents in r and r' , which is more generally applicable than the freshness defined in [14]. We require that B is not instantiated by I , since if the nonce X is supposed to be generated by the attacker, then the attacker can obviously send the same X to both r_1 and r_2 .

Freshness, as defined above, is a necessary condition of authentication. Lowe provided some well-known definitions of authentication goals in [21], and the strongest definition is the follows. The protocol, which is a communication sequence, implies a set of variables that appear in both A 's role and B 's role, which is called the set of shared data. The authentication goal of B 's role to A 's role, for any two agent variables A and B , is that in every run of the protocol, when a role instance r of B finishes execution, there is a one-to-one correspondence between r and another role instance r' of A 's role such that in r and r' the shared variables are instantiated by the same values. In order to implement the one-to-one correspondence commonly a nonce N_A is created by A and received by B , and N_A is shared by A 's role and B 's role. If the authentication goal of B 's role to A 's role is satisfied then the freshness goal of N_A to B 's role is obviously satisfied ([11] explains more). However the freshness goal is not sufficient for the authentication goal. Even when all of the freshness goals to B 's role of the nonce variables shared between A and B are satisfied, the authentication goal of B 's role to A 's role may still not be satisfied, since it is possible that the values of these shared variables in a role instance r of B 's role does not appear together in a single role instance r' of A 's role. The idea of authentication goal is to describe a condition that should not be violated in the normal situation, which is a one-to-one correspondence between two role instances. The freshness goal defined above extends this idea to described s a one-to-one correspondence between a role instance and a nonce that normally should be satisfied.

A unique **location** is assigned to each occurrence of a term in the messages of a communication sequence of a protocol as follows.

- The message with message number i , $1 \leq i$, has location i .
- If an encryption $\{X\}_{\vec{Y}}$ or $\{X\}_{\vec{Y}}$ has the location L , then X is located at $L.\alpha$, and Y is located at $L.\beta$.
- If term $[X, Y, \dots]$ has location L , then the members $X, Y \dots$ have the locations, respectively, $L.1, L.2, \dots$.

Since roles are parsed from a CS, the location of an occurrence of a term T in a role is the location of the corresponding term occurrence in the CS.

Definition 2. We consider the following restrictions on the attacker's behavior in a protocol run.

1. In a run when a regular agent receives a block T in a message, T must have appeared as a block in some message that has already been sent earlier by some regular agent.
2. In any message Msg received in a regular role instance, if T is the block in Msg with location L , then T must be a block with the same location L in some message sent earlier by a regular role instance.

For an attack (a run of the protocol) that violates a freshness goal, if the attacker's behavior is restricted by

- both restrictions 1 and 2, the attack is called a **direct replay attack** (DRA).
- only restriction 1, the attack is called a **restricted replay attack** (RRA).
- no restrictions, the attack is called a **general replay attack** (GRA). \square

It is not obvious how to formally describe the three replay attacks defined above using the taxonomy discussed in [22].

[Bounding the Number of Role Instances in a Run]. We consider the *number of role instances in a run*, call it **NRI** for short, for different problem settings of checking freshness. Note that every run has a finite number of role instances. NRI has been used to analyze the complexity of checking secrecy in the literature [4] [2] [5] [3] [6] [7]. We clarify different notions of bounding NRI, depending on different settings of the inputs to the algorithm, as follows.

- We say **NRI** is **bounded by an individual number**, or simply, is **individually bounded** if the problem to be decided is a tuple $\langle Pro, D, R, V, N \rangle$ where Pro is the protocol, whose size is measured by the size of its CS , and the freshness goal of variable V to the role R needs to be checked, and N is a natural number representing the bound on NRI. Only the runs with the number of role instances no more than N are considered. Note that for different problem instances N could be different.
- We say **NRI** is **bounded by a fixed number**, or simply, is **fixed**, if the problem to be decided is $\langle Pro, D, R, V, n \rangle$, for some fixed number n and all of the instances of the problem shares the same n .
- We say **NRI** is **unbounded**, if the problem to be decided is just $\langle Pro, D, R, V \rangle$, where there is neither fixed nor individual bound considered on **NRI**, there could be any finite number of role instances in a run.

The advantage of clarifying these different settings of bounds is to avoid possible confusion in understanding the terms for bounds including bounded, fixed, unbounded, finite and infinite that appear in the literature. Note that to check DRA or RRA, the intruder's initial knowledge pattern D is irrelevant.

3 Complexity Results on Checking Freshness

[Undecidability Results]

Theorem 1. Checking RRA for a freshness goal is undecidable when the number of role instances in a run (**NRI**) is unbounded.

Proof. We reduce the reachability problem of a deterministic 2-counter machine to a problem of checking RRA for a freshness goal. We have used this approach similarly in [8]. In the reduction the attacker cannot construct any blocks that can be received by a regular agent since these blocks must be encrypted by a symmetric key K which is unknown to the attacker. Reachable configuration of a 2-counter machine M are encoded by special terms called configuration terms in a protocol run. The reduction ensures that a certain freshness goal is violated in a run if and only if a configuration term is known to I which encodes a final configuration reachable to M . Detailed proof is in [11]. \square

Theorem 2. Checking GRA for a freshness goal is undecidable when the number of role instances in a run (NRI) is unbounded.

Proof. There are two ways to prove this theorem. First, we can do reduction from the reachability problem of a deterministic 2-counter machine. The protocol used in the reduction is the same as the one used in [8], which has been specially designed so that the attacker I is an insider and I has to construct blocks in order to commit an attack. The freshness goal chosen for the reduction is the one of the term $C1_{final}$ or $C2_{final}$ to the role R_{final} executed by agent B .

Second, we can do reduction from secrecy to freshness, as sketched below. Given a protocol Pro of the secrecy problem, we construct a protocol Pro' for the freshness problem by adding the following lines in some proper way into the CS of Pro .

$$\begin{aligned} \#_A(N_1) (A \Rightarrow B) : \{m, A, B, N_1, SECRET\}_{k_B}^{\rightarrow} \\ \#_B(N_2) (B \Rightarrow A) : \{m+1, B, A, N_1, N_2\}_{SECRET}^{\leftrightarrow} \\ (A \Rightarrow B) : \{m+2, A, B, N_2\}_{SECRET}^{\leftrightarrow} \end{aligned}$$

Here N_1 and N_2 are fresh nonces created by A and B respectively. $SECRET$ is a term that is supposed to be shared between A and B . The three numbers (constants) m , $m+1$ and $m+2$, for some integer m , are used to distinguish the three messages from other encryptions appearing in the protocol to avoid confusion. Then the freshness goal to be checked is the one of N_1 to B 's role. \square

[Decidability Results]. We obtain several decidability results based on analyzing the performance of the model checker Athena [12] [13]. We introduce the notions and algorithm of Athena first, which are adapted for the modeling of this paper. We arrange the presentation and the proof of Athena differently for simplicity and clarity.

A **strand** is a sequence of action steps formed by instantiating a role by some substitution. There are two differences between a strand and a role instance. First in a strand variables can appear, while in a role instance all terms are ground. Second a role instance includes events, where action steps are associated with *time* fields, but strand includes only action steps. During the reasoning of the model checker a strand represents a role instance of a run.

Every strand is associated with a unique identifier in the reasoning. A **node** of a strand is a pair $\langle r, L \rangle$ where r is the unique identifier of the strand, and L is the location of a block in a message of the strand, which has been introduced in Section 2. Each node can be used as a unique identifier of a block occurrence. The **term of node** nd , for $nd = \langle r, L \rangle$, denoted as $term(nd)$, means the term (the block) appearing at location of L of the strand r . A node in a message received in a strand is called a **negative node**, otherwise a node in a message sent in a strand is called a **positive node**. A **state** is a tuple $\langle strands, binding, counter \rangle$, where *strands* is a set of strands, *binding* is a binary relationship mapping from one negative node in *strands* to a positive node in *strands*, and *counter* is a natural number corresponding to the number of strands in a state. The notation $\langle r, L \rangle \rightarrow \langle r', L' \rangle$ means that the negative node $\langle r, L \rangle$, which is called the **goal**, binds to the positive node $\langle r', L' \rangle$, which is called the **binder**, where $r, r' \in strands$. *counter* is used to check if the bounds on *NRI* is satisfied in a state or not. *counter* can also be used to name a new strand that is introduced into a state and to name the variables of the new strand.

The **causally precedence** \lesssim , also called **causally earlier** relationship between two nodes $nd = \langle r, L \rangle$ and $nd' = \langle r', L' \rangle$ is defined as follows.

- if $r == r'$, i.e. they refer to the same strand, if nd appears in a message with message number m , and nd' appears in a message with number m' , and $m' < m$, then $nd' \lesssim nd$.
- if $nd \rightarrow nd'$ then $nd' \lesssim nd$. Note that \lesssim is opposite to \rightarrow .
- if $nd' \lesssim nd''$ and $nd'' \lesssim nd$, for some node nd'' , then $nd' \lesssim nd$. This condition means that \lesssim is transitive.

In [13] \lesssim is defined to be reflexive, but not in this paper since it is not necessary. In a state, node nd' is not causally earlier than node nd is denoted as $nd' \not\lesssim nd$.

\lesssim is obviously extended for action steps of the strands in a state. For two action steps stp and stp' appearing in a state S , $stp' \lesssim stp$ if one of the following conditions satisfies.

- If stp' and stp appear in the same strand with message number m' and m respectively and $m' < m$.
- If there is a node nd of stp and a node nd' of stp' such that $nd' \lesssim nd$.
- If $stp' \lesssim stp''$ and $stp'' \lesssim stp$, for some step stp'' in S .

The strand space model [20] describes a run as a bundle of strands, where each negative node is bound to a positive node. In [12] the author extended the strand space model of [20] for the model checker Athena by introducing variables and the unification mechanism and a set of new notions including semi-bundle and goal binding. A semi-bundle, which may have goals unbound, is expanded and updated during the computation of Athena and finally forms a bundle. We only use a subset of the notions used in [12] [13] that are enough for this paper, and in

some aspects these notions are presented in a different perspective since we want to clarify the relationship between the notions of strand space used by Athena and our model of protocol run. The advantage of our presentation of Athena is that the meaning and the correctness of the algorithm can be understood more clearly. Based on the improved understanding, the algorithm is also simplified. For example we directly introduce the goal binding relation \rightarrow for the model checker without using the \rightarrow relationship, which is defined and used in Athena [12] [13], since we consider \rightarrow is unnecessary or its meaning is unclear.

The semantics of goal binding can be explained more intuitively in our model. When checking RRA, note that the attacker's behavior of constructing blocks does not need to be considered due to the restriction, node nd_1 binds to node nd_2 means the follows: Let ev_1 and ev_2 be the two events that nd_1 and nd_2 belong to respectively. Then $term(nd_1)$ is sent by ev_2 as a block and $term(nd_1)$ cannot be sent at any event with time point earlier than $ev_2.time$ in the run. This semantics of goal binding can be extended for GRA, when the attacker's internal computation need to be described using term derivations.

The model checker has to ensure three properties, call them **correctness properties**, of a reachable state S during the reasoning.

1. The \lesssim relationship of the action steps and nodes in S is acyclic.
2. All of the negative nodes with the same term in S must bind to the same positive node in S .
3. If a node nd' is the binder of nd , i.e. $nd \rightarrow nd' \in S.binding$, then there is no node nd'' , which is different from nd' in S such that the $term(nd'') == term(nd') == term(nd)$ and $nd'' \lesssim nd'$.

The second correctness property is not introduced in [12] and [13], but we consider it can reduce redundant state exploration significantly in some situation. Even though there could be several nodes that possibly send the same term T at the earliest time, only one of these possible binders for T needs to be considered in a child state of S , and then let other children states of S to consider the other nodes as binders. Property 3 means that a goal should only bind to the (causally) earliest binder as described in [12] and [13].

In a state the variables are global across different strands, which means that if a variable X will be instantiated by Y , then all X appearing in all strands in the state will be replaced by Y . The function $unifiable(T, T')$ returns true, if T and T' are unifiable, otherwise false. Type information can be introduced for unification. For example if according to the protocol in a certain role a variable A is required to be some known agent name, then in a run A can only be unified with an agent name, and A cannot be unified with a nonce generated in the run. For two terms T and T' , $mgu(T, T')$ represents the most general unifier (MGU) of T and T' . For a substitution γ , $\gamma(X)$ means to apply γ to X , where X could be a term, a step, strand, or a state, in the obvious way.

RRAChecker(*Pro*, *R*, *V*, *N*), to check the freshness goal of a variable *V* to role *R* in a protocol *Pro*, no more than *N* role instances in a run, $N \geq 2$

- 1: Let r^1 and r^2 be two different strands of *R* formed as follows. r^1 and r^2 are the same as *R* except for two aspects: First, for every variable *X* in *R* that is not freshly generated in *R*, except *V* whose freshness needs to be checked, *X* is renamed as X^1 in r^1 , and as X^2 in r^2 . The variable *V* remains unchanged and appears in both r^1 and r^2 . Second, for each variable *Y* if *Y* is freshly generated in *R*, *Y* is renamed by a unique constant in r^1 , and by another unique constant in r^2 .
- 2: Let $state^0 := \{\{r^1, r^2\}, \emptyset, 2\}$; Let $STATES := \{state^0\}$.
- 3: **while** $STATES \neq \emptyset$ **do**
- 4: let *S* be an arbitrary state in $STATES$; $STATES := STATES - S$.
- 5: **if** for every negative node *nd* in *S*, there is some positive node nd' in *S* such that $nd \rightarrow nd' \in S.binding$ **then**
- 6: **print** BAD: an attack found, the freshness goal of *V* for *R* is violated.
- 7: Quit the algorithm.
- 8: **end if**
- 9: Let *nd* be an arbitrarily chosen negative node in *S* such that $nd \rightarrow nd' \notin S.binding$ for any positive node nd' . It means *nd* has not been bound (to any positive node) yet. Let $T := term(nd)$.
- 10: **for all** positive node nd' in $S.strands$ such that $nd' \not\prec nd$ **do**
- 11: **if** $unifiable(T, term(nd'))$ **then**
- 12: Let $\gamma := mgu(T, term(nd'))$.
- 13: Let state *S'* be a new state formed as
 $\langle \gamma(S.strands), S.binding \cup \{nd \rightarrow nd'\}, S.counter \rangle$.
- 14: **if** *S'* satisfies the three correctness properties as described earlier in this Section **then**
- 15: Let $STATES := STATES + S'$. { /* Insert *S'* into $STATES$. */ }
- 16: **end if** { /* *S* spawns *S'* */ }
- 17: **end if**
- 18: **end for**{ /* Finish trying to bind *nd* to nodes of existing strands */ }
- 19: **if** $S.counter < N$ **then**
- 20: **for all** blocks *T'* of the protocol, such that *T'* appears at location *L'* in a role *R'* in a sent message numbered with *m*, and $unifiable(T, T')$ **do**
- 21: Let $n = S.counter + 1$; Let r^n be a new strand formed as follows. r^n is the same as the prefix of the action steps of *R'* up to the message numbered with *m*, denote this prefix as $R'^{\uparrow m}$, except that for each variable *X* of $R'^{\uparrow m}$ if *X* is not freshly generated in *R'*, *X* is renamed with X^n in r^n . Otherwise if *X* is freshly generated in $R'^{\uparrow m}$, then *X* is replaced in r^n by a unique constant that has not appeared in *S* yet.
- 22: Let T'' be the block located at *L'* in r^n ; Let $\gamma := mgu(T, T'')$; Let nd' be the node $\langle r^n, L' \rangle$

```

23:      let  $S'$  be a new state formed as (note  $S$  does not change)
         $\langle \gamma(S.strands \cup \{r^n\}), S.binding \cup \{nd \rightarrow nd'\}, n \rangle$ 
24:      if  $S'$  satisfies the second correctness property, described earlier then
25:        insert  $S'$  into  $STATES$ .  $\{ /* S$  spawns  $S' */ \}$ 
26:      end if  $\{ /* No need to consider the correctness properties 1 and 3 */ \}$ 
27:    end for  $\{ /* Finish trying to bind  $nd$  to nodes of new strands.  $*/ \}$ 
28:  end if  $\{ /* Finish handling the state  $S */ \}$ 
29: end while
30: print Good: the freshness goal of  $V$  for  $R$  is satisfied.$$ 
```

Lemma 1. When NRI is individually bounded, $RRA_Checker$ terminates in 2-EXPTIME, and when NRI is fixed, $RRA_Checker$ terminates in EXPTIME.

Proof. Terms appear as bit-strings to the actual algorithm. Note that a bit-string here does not mean the data in physical network. The length of a bit-string Str is the number of bits and is denoted as $|Str|_{bit}$. The input of $RRA_Checker$, which is $\langle Pro, R, V, N \rangle$, can be considered as a bit-string and its size is measured as $\zeta = |\langle Pro, R, V, N \rangle|_{bit}$. We want to prove that when NRI is individually bounded by N , or NRI is fixed to n (N is replaced by n), $RRA_Checker$ terminates with time cost $O(2^{2^{\mathcal{P}(\zeta)}})$, or $O(2^{\mathcal{P}(\zeta)})$, respectively, where $\mathcal{P}(\zeta)$ is a polynomial function of ζ . Time is measured by the number of instructions executed.

First we consider the case that NRI is individually bounded by N . The states that can be reached in a computation of $RRA_Checker$ can be viewed as a tree. The top state is s^0 . If a state S' is created from a state S (by the line 15 or 25) then S' is a child state of S . Let $\psi = |Pro|_{bit} \times N$. It is obvious that $N < 2^{|N+1|_{bit}} = 2^{O(|N|_{bit})}$. The number of occurrences of subterms of a term T is no more than $|T|_{bit}$. Since each state can have at most N strands, and each strand can have no more than $|Pro|_{bit}$ nodes, the total number of nodes in a state is at most $|Pro|_{bit} \times N = \psi$. Each state has at most $O(\psi)$ children states, since for the single arbitrarily chosen negative node nd of S (see line 9), there are at most ψ positive nodes in the existing strands or new strands that can be the possible binders for nd . The depth from the top state s^0 to the bottom of the tree is at most ψ , since each child state has one more negative node bound (to some positive node) than its parent state, and there are at most ψ negative nodes in a state. So the tree of states is at most $O(\psi)$ branching and at most $O(\psi)$ deep. So the number reachable states is at most $O(\psi^\psi)$.

For efficiency purpose of unification, all terms are represented as DAGs [4] in the reasoning of $RRA_Checker$. A DAG of a term T is a tree where each subterm of T appears as a node of the tree exactly once. The subterms of T is defined in the common way as in [4]. Note that encryption key is considered as a subterm of an encryption. The DAG size of a term T is the number of subterms of T , denoted as $|T|_{DAG}$. Obviously $|T|_{DAG} \leq |T|_{bit}$. All messages sent or received in the protocol Pro is translated into DAG representation, which can be done in $O(|Pro|_{bit})$ time. $|Pro|_{DAG}$ is defined accordingly. Let \mathcal{M} be the number of all distinct subterms appearing in all messages sent or received in all strands in a state. Since for RRA a regular agent can only receive and accept a block that

is sent in a message by a regular agent, obviously only the messages sent in the strands contributes to \mathcal{M} . It is obvious to prove that for a strand r of role R , r can contribute at most $|R|_{DAG}$ to \mathcal{M} , and $|R|_{DAG} \leq |Pro|_{DAG}$. Since there are at most N strands, for any reachable state $\mathcal{M} \leq |Pro|_{DAG} \times N < \psi$. So for any message Msg in a strand of a state, $|Msg|_{DAG} < \psi$.

The time cost to generate a new state is $O(\psi^3)$ for the following reasons: First, for two terms T_1 and T_2 , $mgu(T_1, T_2)$ and $unifiable(T_1, T_2)$ (Line 12 and 21) have time cost $O(|T_1|_{DAG} + |T_2|_{DAG})$, and since $|T_1|_{DAG} < \psi$ and $|T_2|_{DAG} < \psi$, the time cost is $O(\psi)$. Second, the cost of applying a substitution to the strands in a state is $O(\psi^3)$ (line 23), since there are at most ψ action steps in a state, and there are at most ψ variables in an action step, and for the instantiation T of each variable $|T|_{DAG} = O(\psi)$. Third, with proper organization of the data structure for the \lesssim relationship, the cost to check the correctness properties for a state is also $O(\psi)$, which is the maximum number of nodes in a state.

Therefore *RRA_Checker* will terminate after

$$O(\psi^\psi \times \psi^3) = O(\{2^{\log_2 \psi}\}^\psi \times \psi^3) = O(2^{\log_2 \psi \times \psi + \log_2 \psi^3}) = O(2^{\psi^2})$$

instructions. Since $|Pro|_{bit}$ is at most ζ and N is at most 2^ζ ,

$$\psi = |Pro|_{bit} \times N < \zeta \times 2^\zeta = 2^{\log_2 \zeta + \zeta} < 2^{2\zeta}.$$

Therefore the algorithm terminates in no more than

$$O(2^{\psi^2}) = O(2^{(2^{2\zeta})^2}) = O(2^{2^{4\zeta}})$$

instructions, which is in 2-EXPTIME, since 4ζ is a polynomial function of ζ .

Now we analyze the other case. When *NRI* is a fixed number n , the input size of *RRA_Checker* is ζ , and $|Pro|_{bit} = O(\zeta)$. Then $\psi = |Pro|_{bit} \times n < n\zeta$. The time complexity is no more than $O(2^{\psi^2}) < O(2^{(n\zeta)^2}) = O(2^{n^2\zeta^2})$, which is in EXPTIME, since $n^2\zeta^2$ is a polynomial of ζ where n is a constant. \square

The soundness and completeness of Athena to check secrecy and authentication when *NRI* is bounded have been addressed in [13], and the soundness and completeness of *RRA_Checker* can be proved similarly. The soundness of the *RRA_Checker* is obvious: when *RRA_Checker* reports an attack, then there is an attack that violates the freshness goal. The completeness of *RRA_Checker* can be proved by induction on the iterations of the *RRA_Checker* to show that for any *run* of the protocol that violates the freshness goal, a subset of the role instances in the *run* can always be mapped to the strands of a reachable state during the computation of *RRA_Checker*. In [11] further details of the correctness of *RRA_Checker* are provided. Therefore we can prove the following theorem.

Theorem 3. Checking RRA for a freshness goal is 2-EXPTIME when *NRI* is individually bounded, and is EXPTIME when *NRI* is fixed.

When *NRI* is individually bounded or fixed, the complexity of checking DRA is no more than checking RRA, and we have the following corollary.

Corollary 1. Checking DRA for a freshness goal is 2-EXPTIME when *NRI* is individually bounded, and is EXPTIME when *NRI* is fixed.

Since DRA can be considered a special case of RRA, `RRA_Checker` can be adapted to check DRA. Then the goal binding mechanism is further restricted, so that a node nd received at location L can only bind to a node nd' sent at location L , according to the protocol. Suppose r' is the new strand introduced to a state by the binding from nd to nd' , where nd' appears in r' , and nd appears in a strand r already included in the state. Let m and m' be the largest message number of a receiving action step in r and r' respectively. Then it is true that $m' < m$. By this observation, it is not difficult to design a measure which is strictly decreasing in any branch of the state tree of `RRA_Checker`. Since `RRA_Checker` is finitely branching, its termination of checking DRA is obvious, even when NRI is unbounded. The soundness and completeness of the algorithm still hold for checking DRA. So we have the following theorem.

Theorem 4. Checking DRA for a freshness goal is decidable when the number of role instances in a run (*NRI*) is unbounded.

Theorem 5. Checking RRA for a freshness goal is NP-complete when NRI is fixed by a number n .

Proof. A non-deterministic algorithm can just guess a branch of the tree of states of `RRA_Checker`, which has at most $\psi = |Pro|_{bit} \times n$ states. Since the cost to generate a state is $O(\psi^3)$, and $\psi < n\zeta$, the total cost of a branch of `RRA_Checker` is $O(\psi) \times O(\psi^3) = O(\psi^4) = O(n^4\zeta^4) = O(\zeta^4)$. Since ζ^4 is a polynomial function of ζ , checking RRA for a freshness goal is NP when NRI is fixed. To prove NP-hardness we reduce the satisfaction problem of 3-SAT to a problem of checking freshness with NRI bounded to 2 ($n = 2$). The same proof can be applied for other cases where $n > 2$. Detailed proof is included in [11], which is delicate. \square

We have attempted to check DRA by a set of efficient reasoning rules that take advantage of the restrictions for the attacker. It seems that checking DRA is P (decidable in polynomial time) even NRI is unbounded. Adapting Athena to check DRA cannot get a polynomial time result since we have found some protocols that require exponential time for Athena to check DRA.

To show the NP-completeness of checking GRA following the approach of this paper will need to incorporate the attacker's internal computation (the attacker strands of Athena) into the model checker. Furthermore we have to prove that in a non-deterministic branch of the computation of the model checker, the DAG size of the substitution is polynomial to the DAG size of the protocol, as discussed in [4]. Since we have some doubts on the existing proofs of NP-completeness for checking secrecy [4] as mentioned earlier, we are trying to have a better approach to show the NP-completeness of secrecy and authentication, which we believe should cover the NP-completeness of checking GRA when NRI is fixed.

We summarize the complexity results of checking freshness in the following table. The results surrounded by two question marks are by postulation and have not been proved by this paper and are under investigation.

attack \ NRI	unbounded	individually bounded	fixed
DRA	Decidable, ?P?	2-EXPTIME, ?P?	EXPTIME, ?P?
RRA	Undecidable	2-EXPTIME	NP-complete
GRA	Undecidable	?2-EXPTIME?	?NP-complete?

4 Summary

In this paper we define freshness goal and its attacks and investigate the complexity of checking freshness, which is the first research on this topic. The techniques of modeling, reduction, and model checking have novel features and can be applied generally in this area. Currently we are investigating the polynomial time algorithm to check DRA, and we use an approach that achieves efficiency by tracing the mechanism of challenge-response, which is rather different from the approach of using a model checker in this paper. We are also studying an improved approach to prove NP-completeness of checking secrecy, which can also be applied to authentication and checking GRA. We expect to extend the NP-complete proof of this paper and to handle several demanding issues discussed at the end of Section 3. These substantial works are proper to be addressed in subsequent researches beyond the scope of this paper.

References

1. Dolev, D., Yao, A.C.C.: On the security of public key protocols. *IEEE Transactions on Information Theory* 29(2), 198–207 (1983)
2. Durgin, N.A., Lincoln, P., Mitchell, J.C.: Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security* 12(2), 247–311 (2004)
3. Ramanujam, R., Suresh, S.P.: Undecidability of secrecy for security protocols (manuscript) (July 2003)
4. Rusinowitch, M., Turuani, M.: Protocol insecurity with a finite number of sessions, composed keys is NP-complete.. *Theor. Comput. Sci.* 1-3(299), 451–475 (2003)
5. Tiplea, F.L., Enea, C., Birjoveanu, C.V.: Decidability and complexity results for security protocols. Technical Report TR 05-02, “Al.I.Cuza” University of Iași, Faculty of Computer Science (2005)
6. Millen, J.K., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: *ACM Conference on Computer and Communications Security*, pp. 166–175 (2001)
7. Liang, Z., Verma, R.M.: Secrecy Checking of Protocols: Solution of an Open Problem. In: *Automated Reasoning for Security Protocol Analysis (ARSPA 2007)*, pp. 95–112 (July 2007)
8. Liang, Z., Verma, R.M.: Improving Techniques for Proving Undecidability of Checking Cryptographic Protocols. In: *The Third International Conference on Availability, Security and Reliability, Barcelona, Spain*, pp. 1067–1074. *IEEE Computer Society Press, Los Alamitos* (2008); *Workshop on Privacy and Security by means of Artificial Intelligence (PSAI)*
9. Gong, L.: Variations on the themes of message freshness and replay—or the difficulty of devising formal methods to analyze cryptographic protocols. In: *Proceedings of the Computer Security Foundations Workshop VI*, pp. 131–136. *IEEE Computer Society Press, Los Alamitos* (1993)

10. Lam, K.-Y., Gollmann, D.: Freshness Assurance of Authentication Protocols. In: Deswarte, Y., Quisquater, J.-J., Eizenberg, G. (eds.) ESORICS 1992. LNCS, vol. 648, pp. 261–272. Springer, Heidelberg (1992)
11. Liang, Z., Verma, R.M.: Complexity of Checking Freshness of Cryptographic Protocols. Technical report, Computer Science Department, University of Houston, Texas, USA, UH-CS-08-14 (September 2008), <http://www.cs.uh.edu/preprint>
12. Song, D.X.: Athena: A new efficient automatic checker for security protocol analysis. In: CSFW, pp. 192–202 (1999)
13. Song, D.X., Berezin, S., Perrig, A.: Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security* 9(1/2), 47–74 (2001)
14. Corin, R., Etalle, S., Saptawijaya, A.: A logic for constraint-based security protocol analysis. In: SP 2006: Proceedings of the 2006 IEEE Symposium on Security and Privacy, Washington, DC, USA, pp. 155–168. IEEE Computer Society Press, Los Alamitos (2006)
15. Backes, M., Cortesi, A., Focardi, R., Maffei, M.: A Calculus of Challenges and Responses. In: Proceedings of 5th ACM Workshop on Formal Methods in Security Engineering (FMSE) (November 2007)
16. Guttman, J.D., Thayer, F.J.: Authentication tests. In: IEEE Symposium on Security and Privacy, pp. 96–109 (2000)
17. Froschle, S.: The insecurity problem: Tackling unbounded data. In: IEEE Computer Security Foundations Symposium 2007, pp. 370–384. IEEE Computer Society, Los Alamitos (2007)
18. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An np decision procedure for protocol insecurity with xor. *Theor. Comput. Sci.* 338(1-3), 247–274 (2005)
19. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. *Journal of Computer Security* 6(1-2), 85–128 (1998)
20. Thayer, F.J., Herzog, J.C., Guttman, J.D.: Strand spaces: Proving security protocols correct. *Journal of Computer Security* 7(1) (1999)
21. Lowe, G.: A hierarchy of authentication specifications. In: CSFW 1997: Proceedings of the 10th Computer Security Foundations Workshop (CSFW 1997), Washington, DC, USA, p. 31. IEEE Computer Society Press, Los Alamitos (1997)
22. Syverson, P.F.: A taxonomy of replay attacks. In: CSFW, pp. 187–191 (1994)

Secure Internet Voting Based on Paper Ballots^{*}

Lukasz Nitschke

Faculty of Mathematics and Computer Science, Adam Mickiewicz University,
Umultowska 87, 61-614 Poznań, Poland

Abstract. Internet voting will probably be one of the most significant achievements of the future information society. It will have an enormous impact on the election process making it fast, reliable and inexpensive. Nonetheless, so far the problem of providing security of remote voting is considered to be very difficult, as one has to take into account susceptibility of the voter's PC to various cyber-attacks. As a result, most of the research effort is put into developing protocols and machines for poll-site electronic voting. Although these solutions yield promising results, most of them cannot be directly applied to Internet voting because of the secure platform problem. However, the cryptographic components they utilize may be very useful in the context of remote voting, too. This paper presents a scheme based on combination of mixnets and homomorphic encryption borrowed from robust poll-site voting, along with techniques recommended for remote voting - code sheets and test ballots. The protocol tries to minimize the trust put in voter's PC by employing paper ballots distributed before elections. The voter uses the ballot to submit an encrypted vote, which is illegible for the potentially corrupt PC. The creation of paper ballots, as well as the decryption of votes, is performed by a group of cooperating trusted servers. As a result, the scheme is characterized by strong asymmetry - all computations are carried out on the server side. Hence, it does not require any additional hardware on the voter's side. Furthermore, the scheme offers distributed trust, receipt-freeness and verifiability.

1 Introduction

If we take a critical look at the traditional voting methods that we have been using for years, we can observe many opportunities for fraud along with the inability of citizens to verify election results. This gives a strong motivation for computer scientists to design *e*-tools that could realize voting, and that would not only prevent cheating and allow checking, but also lower the costs and increase availability. Unfortunately, such solutions, contrary to traditional voting, have to face an inherent threat that any security vulnerability may allow massive abuse (this is an example of a general phenomena well described by Schneier [1], and denoted as *class break*). Therefore, electronic voting should meet the following requirements.

^{*} Supported by Ministry of Science and Higher Education, grant N N206 2701 33, 2007-2010.

- Anonymity, privacy – voter’s individual choices should remain secret.
- Receipt-freeness – the voter should be unable to convince a third party of the vote decision and, as a consequence, should be unable to sell his or her vote. This property is also achieved if the voter has effective measures of deceiving a potential buyer.
- Verifiability – the voter should be able to check correctness of every stage of the protocol. He or she should be empowered to verify the tallying process (global verifiability) and check if his or her vote was included (individual verifiability). Usually individual verifiability requires a lookup in a public catalog, whereas global verifiability demands performing some computations. In consequence, an average voter delegates global verifiability to experts or watchdog organizations. Verifiability implies integrity of the election – no party is able to introduce illegal votes and all proper ones are counted.

Three approaches to the problem of electronic voting have been proposed so far (see: [2,3]).

- *Poll-site voting* – special voting machines with dedicated software are installed in voting booths at polling stations. Voters can cast votes by interacting with such a machine, and in some cases they can receive a receipt for verification. The terminal and the environment can be controlled. Moreover, some steps of the protocol may be performed by an election official, for instance the voter can be personally authorized.
- *Kiosk voting* – voting takes place through publicly available terminals (e.g. sophisticated ATMs or dedicated state-owned machines). In this scenario only the terminal can be controlled.
- *Voting via Internet* performed by a client-server application, run by voter’s personal computer (PC)/mobile phone/PDA/smart card, and on the server side, by trusted authority or authorities. Neither the terminal not the environment can be controlled. Internet voting is a special case of remote voting, which encompasses also mail-in elections or voting by phone. Although the presented protocol was designed for Internet voting, it can also handle other kinds of remote voting. Therefore, we also use the broader term in the rest of the paper.

The rest of the paper is organized as follows: section 2 covers the state-of-the-art voting protocols in the context of remote voting. Section 3 describes general ideas of the proposed scheme. It is followed by a presentation of the main building-blocks of the protocol. The fifth section explains the protocol in detail. Finally, we explain the main properties of the scheme and introduce further enhancements.

2 Related Works

2.1 Poll-Site Voting

Recently, the effort of researchers has mostly been put into developing protocols and machines for poll-site voting. Among the three above listed approaches, this

is the least demanding from the security perspective, as we can assume control of the voter's terminal, and on the environment (the officials are present at the poll-site). Several well designed solutions have been proposed, including: Chaum's voting system [4], Neff's voting system [5], W-voting [6], Scratch&Vote (S&V) [7], Prêt à Voter (PaV) [8], Punchscan (PS) [9] and Punchscan merged with Prêt à Voter (PS+PaV) [10]. The machines used at polling stations produce ballots – digital or paper means of casting a vote. The ballot contains either the encrypted vote or all possible encrypted votes, along with other values (identifiers, non-interactive zero-knowledge proofs, randomizing values, keys). Some of the values that prove the proper construction of the ballot may be hidden (e.g. proofs). A ballot for which the hidden values were revealed is a compromised ballot and may be used as evidence for vote selling. The voter verifies the creation process by choosing two ballots and then selecting one for validation. The selected ballot is compromised by providing additional hidden data (usually printing – see for example [4], or not destroying – see for example [7]). The validation procedure requires a computer program and equipment (e.g. scanner). In practice, the verification is meant to be carried out by watchdog organizations that collect the ballots at polling stations or somewhere else. As a consequence, the remaining ballot is believed to be properly constructed and is used to cast a vote. The non-compromised ballot (without additional data) cannot serve as evidence for vote selling purposes (receipt-freeness). Encrypted votes are published and processed by election authorities to obtain the final result. Every voter can check if his or her encrypted vote was counted. Correctness of the tallying process can be universally verified. Interactive testing of machine encryption and verification of the processing give strong assurance that cheating is impossible.

Among the listed solutions, we can distinguish protocols with pre-printed ballots – the ballot is not printed in interaction with the voter (S&V, PaV, PS, PS+PaV) and contains codes for all possible choices. The other protocols demand a voting decision during the creation of a ballot, which contains only one encrypted choice, and they are inappropriate for remote voting. This is a consequence of the following observations.

- We cannot replace dedicated machines with voters' personal computers. This is caused by the fact that a PC is not tamper-proof, and may leak secret information to the voter, enabling vote selling. For instance, interactive checking of ballots relies on the assumption that the voter does not learn verification data for the proper ballot.
- It is difficult and expensive to reduce the size of voting machines used in poll-site voting to make them portable (integrated printer and other implementation aspects).

The protocols with pre-printed ballots, do not require voting machines to be present at polling stations – the ballots may be prepared and distributed earlier. The vote is cast manually, and recorded (digitalized) by the election authority. Most of the protocols conform to the model with one election authority and many trusted printing parties. The model assumes distribution of trust between the parties that create the ballot and the parties that collect votes. Distribution

of trust is a property that gives the voter confidence that there exists no single authority (more generally, a proper subset of the set of authorities) that can breach either confidentiality or integrity of the system. The model seems to be applicable to remote voting if we assume that the voter obtains his ballot (or two for probabilistic testing) before elections, and then he or she sends relevant values from the ballot via Internet and it is assured that neither the printing authority nor the collecting authority is able to breach his or her privacy. These requirements are met by the Prêt à Voter system, which employs distributed printing – the process of printing is performed by two (see [8]) or more trusted parties (see a different extension of the system proposed in [11]). Although the distribution of trust is not provided by the solutions based on Punchscan, they offer unconditional privacy as a trade-off (assuming the usage of perfectly hiding commitments). It also worth mentioning that the Punchscan solutions have the simplest ballot (especially PS+PaV), without any ciphertexts and long strings. The table below gives a comparison of properties of the protocols with pre-printed ballots including the new one proposed in the paper.

Table 1. Comparison of voting protocols with pre-printed ballot (* supplied documentation does not convince, ** printing authority learns too much from the values published for verification, + no distributed printing)

	New	PaV	S&V	PS	PS+PaV	SureVote
Verifiability	Yes	Yes	Yes	Yes	Yes	No*
Distribution of trust	Yes+	Yes	No**	No	No	No*
Simple ballot	Yes	contains ciphertext scratch surface	contains ciphertext	Two layers	Yes	Yes
Unconditional privacy	No	No	No	Yes	Yes	No

2.2 Internet Voting

The remote voting approach is the most convenient and cost-effective. It also reflects the needs of the modern society. Nonetheless, it is considerably more challenging, as we have to take into account various cyber-attacks (possibly launched from a hostile country), and less control of the voter's terminal and environment [3,12]. The latter implies the possibility of external influence being exerted on the voter. Vulnerabilities of voters' PCs may open the door to many serious abuses, e.g. automated vote selling or malicious changing of votes. Several countermeasures have been proposed to minimize trust put in voter's PCs. Apart from the expensive ones (trusted hardware) and the idealistic ones (clean operating system), code sheets and test ballots seem to be promising [3]. Code sheets impose a complete asymmetry in the computational sense – no computations are done on the voter's side. This is achieved by providing voters with ballots that contain unique codes representing candidates (different set of codes for each ballot). Each candidate code has a verification code assigned to it. The PC is used to pass the entered code on to the election authority,

which returns the relevant verification code. The response is displayed by the PC to assure integrity of remotely cast vote. Honest election authority may prevent cyber-attacks, but a dishonest one can try to influence elections results or breach voter's privacy. SureVote system, described in [13], is an example of a code sheets scheme. On the other hand, test ballots is an approach that suggests introducing special ballots in the voting stage. The ballots should be unrecognizable for the tallying authorities, so that they are unable to predict for which ballots they will have to reveal processing after publication of the result. There are also attempts to solve the untrusted platform problem by utilizing trusted hardware (see [14,15,16]), but these solutions have to face a serious threat of malicious producers and kleptography [17].

3 Sketch of the New Protocol

3.1 Design Goals

Design of our protocol was motivated by the following aims.

- (*Distribution of trust*) The scheme, similarly to PaV, provides distribution of trust but it does not require distributed printing and all relating complications (scratch surface). In consequence, the implementation of the property is easier and less expensive.
- (*Verifiability*) Every stage of the protocol may be verified either directly by the voter or the validation may be delegated to independent organizations.
- (*Receipt-freeness*) Receipt-freeness is guaranteed in the same way as in PaV or Scratch&Vote – the voter receives two ballots, one of which is compromised and thoroughly checked. The other ballot is considered to be a valid one and is utilized. The key point is that for the valid ballot the voter possesses no digital evidence of its authenticity. In consequence, it is difficult to organize vote selling or coercion on a massive scale.
- (*Simple ballot*) The ballot does not contain any ciphertexts and is equivalent to the one used in PaV+PS. The voter does not need to enter long strings which significantly increases user-friendliness of the system compared to PaV.
- (*Trustworthiness*) Test ballots may be introduced into the system by an average voter to convince skeptics of correctness of the tallying process.
- (*Practicality*) Easy and inexpensive integration with traditional elections is possible. No additional hardware on the client side is required and remote voters are registered before traditional elections start to prevent unnoticed double voting. Remote voter might be allowed to cast their vote in a traditional way. This would cancel the remote vote. Such solution was implemented in Estonia [18] and solves the problem of a voter who loses trust in the remote voting system or a voter who is an object of malicious coercion.
- (*Corrupt PC resistance*) The protocol assumes reduction of trust put in the voter's PC and the software it runs, by assuring that voter's computer is unable to record and secretly change choices made by its user (neither randomly nor intentionally).

The main idea of the proposed protocol is that the voter encrypts his vote by performing a simple cyclic shift. The operation performed by the voter can be inverted by a group of cooperating servers that perform distributed computations. This is how the trust is distributed, and contrary to basic code sheets, the process of decrypting votes can be publicly controlled. Our model also includes distributed creation of paper ballots. The ballot can be interactively tested using similar techniques as in poll-site machines. In addition, the solution offers the voter user-friendliness and ability to introduce test ballots to the system. This may be the key factor for winning social acceptance, which is a problem in the context of electronic elections.

3.2 Actors

Apart from the voters, the protocol employs the following trusted authorities.

- EC_1 (*Election Committee 1*) - provides an on-line voting service, whose users are authorized using known protection mechanisms. Communication with EC_1 can be established through an authenticated (in both directions), private channel.
- EC_2 (*Election Committee 2*) - prints paper ballots used to encrypt user choice. EC_1 and EC_2 are in a conflict of interests.
- $A_1, A_2, \dots, A_\lambda$ - authorities that participate in the process of creating paper ballots, and in the process of decrypting votes, they also assure proper distribution of trust and audit.
- BB - bulletin board, provides an authenticated public channel - allows publishing and later access to signed messages.

3.3 The Protocol from the Voter's Perspective

1. *Registration.* Registration procedure should be similar to applying for an electronic bank account. A citizen fills in an application form and submits it personally to the local administration office where he is personally authorized (based on his signature and ID card). After a reasonable period of time the voter is able to receive his elections kit from EC_1 . The kit contains credentials which enable remote authentication of the user and access to the on-line voting service. The methods used here may be similar to the ones used in e-banking, (PINs, passwords, one-time passwords, tokens). A token integrated with an electronic ID card or signatures-enabled ID is considered to be the optimal solution, as the ID is tightly bound to the citizen.
2. *Obtaining the paper ballot.* A citizen who is willing to cast his vote via Internet is obliged to visit the local administration office in order to obtain a paper ballot, and to be personally authorized. It is supposed that it can be done during a reasonably long period of time before the elections. It is assumed also that the voter has already gained access to the election service (see: registration). The voter chooses two ballots, then decides which one should be verified. The selected ballot is recoded by an election official, and can be verified by the voter or by a civic committee. The other ballot is

1. Alice	3
2. Bob	0
3. Carol	1
4. Dave	2
sh=2	id=0001221

Fig. 1. The ballot. The shift value does not need to be printed explicitly.

separated from the part containing validation data, and serves as a proper means of casting a vote. The validation data is destroyed in the presence of the voter.

3. *Manual encryption.* Every ballot has a shift value sh and an identifier id assigned to it. The transformation is represented by a table, the first column of which consists of a list of candidate names in alphabetical order (or any other order approved for given elections). The second column contains encrypted votes – we encode candidate number v by the value $v + sh \bmod c$ (c is the number of candidates).
4. *Vote submission.* In a definite period of time (e.g. the week before traditional elections), when remote votes are collected, the voter, using EC_1 on-line voting site logs into the proper elections service, enters the id and the encrypted vote.
5. *Publication of encrypted votes.* EC_1 publishes voters' names along with encrypted votes on the BB (without the id 's).
6. *Verification.* Voter checks if his encrypted vote reached the bulletin board in an unchanged form.
7. *Vote results publication.* The votes are decrypted by trusted authorities $A_1, A_2, \dots, A_\lambda$ and published.

4 Building Blocks

Most of the building blocks we employ are based on the ElGamal public key cryptosystem. Let p be a large prime, g a generator in \mathbb{Z}_p and x a random element of \mathbb{Z}_p . We define: ElGamal private-public key as $(x; p, g, y)$, where $y = g^x \bmod p$, and ElGamal encryption and decryption functions as: $e_y(m) = (my^k, g^k)$, $d_x(a, b) = a/b^x$. Owner A of asymmetric keys $(x; p, g, y)$ can prove that a given ciphertext (a, b) is an encryption of message m using a non-interactive zero-knowledge proof of equality of discrete logarithm ($\log_b(a/m) = \log_g y$) [19]. We will denote the proof as $NIZK(m, (a, b))$.

4.1 Mix Networks

One of the most important branches in research of electronic voting are protocols based on mix networks. Mix network protocols allow shuffling a list of encrypted

messages in a distributed way by λ trusted parties (mix servers). Each party A_i sequentially permutes and transforms elements on the list. The resulting list is passed on to the next mix server via an authenticated public channel (BB). Transformations carried out by a single server obfuscate relations between input and output elements. Therefore, it is hard to determine the secret permutation of a single server, and in consequence the global permutation of the whole mixnet. We need two functions to perform distributed shuffling:

- $on_k(m)$ – creates an initial encrypted form (called an onion) of m that passes through the mix servers (k is a randomizing value);
- $t_{i,k}(c)$ – function of the i -th mix server that transforms the ciphertext c into c' so that: c' encrypts the same message as c , and it is difficult to prove this fact without the knowledge of the randomizing value k .

Depending on the transformation function, we can distinguish different types of mixnets. The protocol introduced in this paper employs partially decrypting mixnets. This type of mixnet is characterized by the fact that each server partially decrypts elements on its input list and the last server yields messages. Such a mixnet based on ElGamal can be build by defining the o and t functions as follows [20]: $on_k(m) = e_y(m)$ (where $y = y_1 \cdot y_2 \cdot \dots \cdot y_\lambda$), $t_{i,k}((a, b)) = (a(y_{i+1}y_{i+2}\dots y_\lambda)^k / b^{x_i}, b \cdot g^k)$, where $t_{i,k}, (x_i, y_i)$ are transformation function and asymmetric keys of A_i ($i = 1, 2, \dots, n$).

Randomized Partial Checking (RPC). A mix network protocol can be employed as a component of electronic voting if it can be guaranteed that none of the elements from the list was replaced or maliciously altered. This property called *robustness* is provided by additional checking. Randomized Partial Checking is a fairly simple and effective verifying technique which was introduced in [21]. The mix servers are obliged to reveal a random half of their input-output relations, with the assurance that no path of length greater than 2 can be uncovered. To achieve this property the servers are paired, and forced to uncover complementary halves of their transformations.

More precisely, RPC consists of the following steps (see [21]):

1. *Before shuffling.* The servers $A_1, A_2, \dots, A_\lambda$ publish commitments to their permutations ($pcommit(\pi_i) = (commit(\pi_i(1)), \dots, commit(\pi_i(n)))$), where π_i is A_i 's permutation of n element set $\{1, 2, \dots, n\}$,
2. *After shuffling.* The servers establish a fairly chosen value $r = r_1 \oplus r_2 \oplus \dots \oplus r_\lambda$ (\oplus is a XOR of a string of bits) – each server contributes its randomizing value r_i using commitments, so that no party is able to determine r . Then a value $q = hash(r, content(BB))$ is computed, and $q_i = hash(q, i)$ are derived. The q_i values determine transitions to be revealed in pair i . To prove the validity of a selected transition of j -th input server A_i publishes a value $validator(i, j)$ that may consist of $decommit(\pi_i(j)), k_{ij}$, where k_{ij} is the randomization value used in the j -th transformation.

4.2 Homomorphic Encryption and Re-encryption

The ElGamal scheme has the property that having two encrypted messages, one can calculate the ciphertext of multiplication of the two messages. This can be achieved by simply multiplying the two ciphertexts. This property is known as (\cdot, \cdot) -homomorphism. In this paper $(\cdot, +)$ -homomorphism is more useful. However, this requires a small modification of the original ElGamal.

$$\hat{e}_y(m_1) = (h^{m_1} \cdot y^{k_1}, g^{k_1}), \hat{e}_y(m_2) = (h^{m_2} \cdot y^{k_2}, g^{k_2})$$

$$\hat{e}_y(m_1) \cdot \hat{e}_y(m_2) = (h^{m_1} \cdot y^{k_1} \cdot h^{m_2} \cdot y^{k_2}, g^{k_1} \cdot g^{k_2}) = (h^{m_1+m_2} \cdot y^k, g^k) = \hat{e}_y(m_1+m_2)$$

h^m is obtained from $\hat{e}_y(m)$ by performing regular ElGamal decryption, then m is found through exhaustive search or lookup in a precomputed table. Note, that if we take p such that a large prime $q|p-1$ and a small $c|p-1$ then assuming that $\text{ord}(g) = q$ and $\text{ord}(h) = c$ we can perform encrypted additions in a cyclic group \mathbb{Z}_c . Because the ElGamal encryption offers semantic security this modification does not weaken the cryptosystem.

4.3 Computing Mixnet

If we combine mixnets with the idea of homomorphic encryption we obtain a protocol for distributed computation that has the property that it obfuscates the relations between input and output values. Computations performed by such a network can be used to anonymously invert the modulo addition operation (cyclic rotation) performed by the voter. We now obtain two new functions: $\hat{o}n_k(m) = \hat{e}_y(m)$, where $y = y_1 \cdot y_2 \dots y_\lambda$, and $\hat{t}_{i,k,l}((a, b)) = (a \cdot h^l \cdot (y_{i+1} y_{i+2} \dots y_\lambda)^k / b^{x_i}, b \cdot g^k)$, l is a value added in a given transformation of the i -th server.

5 The Protocol

5.1 Setup

Notation: n – number of paper ballots; c – number of candidates; p – secure, public prime, such that a large prime $q|p-1$ and $c|p-1$, g, h – generators in \mathbb{Z}_p of order q, c respectively; $(x_i; p, g, y_i)$ – asymmetric keys of A_i ; $(x_{EC_2}; p, g, y_{EC_2})$ – asymmetric keys of EC_2 . Before the start of the elections the following steps need to be fulfilled.

1. EC_1 chooses a permutation $\pi_0 : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$, and publishes the commitment.

$$EC_1 \longrightarrow BB : pcommit(\pi_0)$$

2. Each A_i :

- (a) chooses a permutation $\pi_i : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$, and a vector of small integers $(l_{ij} < c)$: $l_i = (l_{i,1}, l_{i,1}, \dots, l_{i,n})$;
- (b) publishes the commitment to its permutation and to the values l_{ij}

$$A_i \longrightarrow BB : pcommit(\pi_i), commit(l_i).$$

5.2 Actions of the Protocol

Creation of ballots. Creation of paper ballots involves sending n pairs of partially decrypting onions through the mixnet. The first onion $(c_{0,j})$ in a pair carries an identifier of the input position, while the second one $(\hat{c}_{0,j})$ uses homomorphic encryption to accumulate the sum sh that forms the cyclic rotation printed on the paper ballot.

$$c_{0,j} = e_y(\pi_0(j)), y = y_1 \cdot y_2 \cdot \dots \cdot y_\lambda \cdot y_{EC_2}, \hat{c}_{0,j} = (1, 1), j = 1, \dots, n$$

$$EC_1 \longrightarrow BB : (c_{0,j}), (\hat{c}_{0,j})$$

The pairs of onions $(c_{0,j}, \hat{c}_{0,j})$ are then being processed by the authorities A_i ($c_{i,j} = t_{i,k_{i,j}}(c_{i-1,j})$, $\hat{c}_{i,j} = \hat{t}_{i,k_{i,j},l_{i,j}}(\hat{c}_{i-1,j})$), $i = 1, 2, \dots, \lambda$ and passed on to the next authority through the bulletin board. Note that the resulting identifiers and values sh remain secret to the public audience, as they are still encrypted with EC_2 public key.

Ballot printing. Each pair on the output list is decrypted by EC_2 , and resulting identifiers and shift values are printed on the ballots (sh, id) . The shift value is used to create the second column of the table (see section 3.3). Every ballot is then completed with hidden values: the corresponding onion pair from the output list $(c_{\lambda,k}, \hat{c}_{\lambda,k})$ and non-interactive zero-knowledge proof of the correctness of the decryption of $c_{\lambda,k}$ ($NIZK(id, c_{\lambda,k})$) and $\hat{c}_{\lambda,k}$ ($NIZK(h^v, \hat{c}_{\lambda,k})$).

Distribution and checking of ballots. Every voter V personally obtains two paper ballots. He or she chooses one for verification and learns its validation values (the part of the ballot with validation values for the second ballot is separated and destroyed). The ballot identifier is scanned by an election official and marked as invalid by EC_2 on the BB . The values it contains can be verified. Note that the verification process has to include values available on-line on the bulletin board. The other ballot provides the voter with id, sh , which are used for voting.

Casting votes. Each voter V encrypts his or her vote for candidate number v with value from the second column, which is equal to $v + sh \pmod{c}$. The value is sent along with id to EC_1 through an authenticated channel.

$$V \xrightarrow{auth} EC_1 : id, v + sh \pmod{c}$$

The election authority publishes the following: position on the input list $p = \pi_0^{-1}(id)$, voter's identifier and his encrypted vote, the onion $\hat{c}'_{0,p} = \hat{\sigma}_{k'_{0,j}}(v + sh \pmod{c})$, and $k'_{0,j}$ as proof of the correctness of the onion. The id value of the ballot is not published.

$$EC_1 \longrightarrow BB : p, V, v + sh \pmod{c}, \hat{c}'_{0,p}, k'_{0,j}$$

Recovering and counting votes. The encrypted votes that have been available on the bulletin board enter the same mixnet, that uses the same l_{ij} values and the same permutations, but instead of adding they are now subtracted by A_i ($\hat{c}'_{\pi_i(j)} = \hat{t}_{i,k'_{i,j},-l_{i,j}}(\hat{c}'_{i-1,j})$). After the second stage of processing the mixnet outputs cleartext votes (published on the *BB*). The onions that reach positions on the output list marked as invalid are traced back. It means that in case a compromised ballot was used the fact can be easily discovered.

5.3 Mix-and-Compute Verification

For verification of the two stage process of creating ballots and recovering votes we also need a two stage validation technique. Splitting RPC directly into two phases (revealing 1/4 of transitions twice) is unacceptable – before the second stage of testing a malicious mix server would be able to pinpoint transitions that cannot be tested according to the rule that in a pair of servers no path of length two can be uncovered. Therefore, we propose a different two-stage version of RPC, in which the servers are grouped in 4-tuples consisting of two pairs.

1. (*After creation of ballots*) 1/4 of transitions of each server is revealed in a similar way as in the regular RPC procedure (no path of length two is uncovered in each pair).
2. (*After recovery of votes*) Within each 4-tuple one pair is selected to reveal remaining the 1/4 of mappings in the RPC fashion. The servers in the other pair reveal transformations independently (without hiding paths).

Now the probability that replacement of n onions will remain unnoticed is $(1/6)^n$. The transitions selected to be verified are determined in a way similar to regular RPC. However, the validating values also include elements of sums – l_{ij} . Transitions chosen to be revealed are uncovered in both stages.

6 Protocol Properties

6.1 Security

Confidentiality and distribution of trust. Confidentiality of the system is mainly preserved by the properties of the partially decrypting mixnets based on ElGamal asymmetric encryption. The operations carried out by the computing mixnet allow to produce shift values used by the voter to encrypt his vote by modulo addition (one-time pad). Computing mixnet is verified by the two-stage RPC without revealing complete paths – the anonymity is preserved. It has to be stressed that the security of the elections is preserved if the authorities EC_1 and EC_2 are in a conflict of interest – for instance they are controlled by the ruling and opposition party. Otherwise, they would be able to violate users' privacy and try to introduce fake votes. This is caused by the fact that one authority (EC_1) controls the input to the computing mixnet while the other party (EC_2) controls the output. EC_2 learns all the shift values, while EC_2 (based on the ballots' identifiers) knows how to position encrypted votes before they enter the mixnet in the second stage of mixing.

Receipt-freeness and resistance to coercion. Providing verifiability and receipt-freeness (inability to sell votes) is the biggest challenge in design of voting protocols. In our protocol the voter is given two paper ballots. He or she chooses one of them to reveal its validation values. The ballot is marked as compromised on the mixnet's output list and can be verified by a watchdog organization. As a result the voter believes that the proper ballot, whose validation values were destroyed, is also valid. But he or she is unable to prove it to anybody else (who was not present during interactive testing), and sell the vote. The voter simply has no digital evidence of the authenticity of the proper ballot, and is unable to convince a third party. Moreover, the voter is able to produce fake ballots based on the proper one, and claim to have voted as the potential buyer wishes. One can imagine open software for production of such ballots. A more active coercer may force absence of the voter or casting a random vote, but the remote voter always has the possibility to cast his vote traditionally.

End-to-end verifiability. To verify correctness and integrity of the elections the following steps deserve extraordinary scrutiny: distributed creation of the ballots, ballot decryption and printing, vote submission, vote positioning on the input list, distributed decryption of votes. Proof of the correctness of distributed creation of ballots and decryption of votes relies on the modified two-stage RPC procedure. The proof may be universally verified but requires special knowledge. As a result, verification will be delegated to independent experts. Printing and decryption of ballots is verified by interactive testing – choosing one ballot out of two for thorough investigation. The output position of the compromised ballot is then marked as compromised on the first stage mixing output list. The compromised ballots may be then utilized as test ballots. They are introduced into the mixnet by verification organizations or voters themselves to strengthen verification of the decryption process and the creation of the second stage input list. The decrypted votes that are output at compromised positions can be traced back, and voters who decided to check could verify their test ballot by checking how it was decrypted on the bulletin board. In this sense test ballots are a real trust increasing factor. Submission of the vote may be directly verified – the voter checks his encrypted vote on the list published on the bulletin board.

6.2 Further Enhancements

Code sheets. The scheme would certainly benefit from an immediate assurance that votes cast by the voters reached the election authority unchanged. To achieve this goal we can employ verification codes inserted in the identifier onion by EC_1 during creation of ballots. Truncated digital signatures of EC_1 are good candidates for the codes – EC_1 could provide on-demand authentication of a voting ballots. If the number of candidates exceeds the capacity of the onion, a seed value may be used instead.

K-out-of-L voting. In the basic setting the proposed scheme offers 1-out-of-L voting, which means that we can choose only one candidate. However, we can

easily extend it by adding multiple cyclic rotations to K-out-of-L or K-out-of-L-ordered voting. This requires increased number of homomorphic onions processed by the authorities and a slightly different ballot layout. The ballot contains a list of candidates and a table for each cyclic shift.

7 Conclusions

So far, no viable solution to the problem of remote electronic voting has been proposed. The computational model presented in this paper is a step toward overcoming vulnerabilities of operating systems, personal computers and the Internet. The solution also offers receipt-freeness and full verification of every step. Compared to existing solutions the protocol offers simpler ballot, user-friendliness and distribution of trust without a complex printing procedure. The crucial part of the verification can be carried out by every voter while the more complicated routines may be delegated to experts or independent organizations. The protocol can be regarded as a general solution and, in consequence, can be used at poll-stations. We also showed that autonomously computing mix servers may be a useful component of cryptographic protocols.

References

1. Schneier, B.: Beyond fear: Thinking sensibly about security in an uncertain world. Copernicus Books (2003)
2. CIVTF: A report on the feasibility of internet voting (2000), http://www.ss.ca.gov/executive/ivote/final_report.htm
3. Oppliger, R.: How to address the secure platform problem for remote internet voting. In: Proceedings 5th Conf. on Security in Information Systems (SIS 2002), pp. 153–173 (2002)
4. Chaum, D.: Secret-ballot: True voter verifiable elections. IEEE Security and Privacy, 38–47 (2004)
5. Neff, A.: A verifiable secret shuffle and its application to e-voting. In: Samarati, P. (ed.) ACM CCS 2001, pp. 116–125. ACM Press, New York (2001)
6. Klonowski, M., Kutylowski, M., Lauks, A., Zagórski, F.: A practical voting scheme with receipts. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 490–497. Springer, Heidelberg (2005)
7. Adida, B., Rivest, R.L.: Scratch and Vote: Self-contained paper-based cryptographic voting. In: Dingledine, R., Yu, T. (eds.) ACM Workshop on Privacy in the Electronic Society. ACM Press, New York (2006)
8. Ryan, P., Schneider, S.: Prêt à Voter with re-encryption mixes. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 313–326. Springer, Heidelberg (2006)
9. Popovenuic, S., Hosp, B.: An introduction to punchscan. In: IAVoSS Workshop On Trustworthy Elections, WOTE (2006)
10. van de Graaf, J.: Merging prêt à voter and punchscan (2007), <http://eprint.iacr.org/2007/269.pdf>
11. Xia, Z., Schneider, S., Heather, J.: Analysis, improvement and simplification of Prêt à Voter with paillier encryption. In: USENIX/ACCURATE Electronic Voting Technology Workshop (2008)

12. Rubin, A.: Security considerations for remote electronic voting over the internet (2001), <http://avirubin.com/e-voting.security.pdf>
13. SureVote: E-voting system - overview (2001), <http://www.vote.caltech.edu/wote01/pdfs/surevote.pdf>
14. Lee, B., Kim, K.: Receipt-free electronic voting scheme with a tamper-resistant randomizer. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 389–406. Springer, Heidelberg (2003)
15. Kutylowski, M., Zagórski, F.: Verifiable internet voting solving secure platform problem. In: Miyaji, A., Kikuchi, H., Rannenberg, K. (eds.) IWSEC 2007. LNCS, vol. 4752, pp. 199–213. Springer, Heidelberg (2007)
16. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: 2008 IEEE Symposium on Security and Privacy, pp. 354–368 (2008)
17. Young, A., Yung, M.: The dark side of black-box cryptography, or: Should we trust Capstone? In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109. Springer, Heidelberg (1996)
18. Estonian National Election Committee: E-voting system - overview (2006), <http://www.vvk.ee/elektr/docs/Yldkirjeldus-eng.pdf>
19. Chaum, D., Pedersen, T.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
20. Park, C., Itoh, Kh., Kurosawa, K.: Efficient anonymous channel and all/nothing election scheme. In: Helleseht, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 248–259. Springer, Heidelberg (1994)
21. Jakobsson, M., Juels, A., Rivest, R.L.: Making mix nets robust for electronic voting by randomized partial checking. In: USENIX Security Symposium, pp. 339–353 (2002)

A Secure Round-Based Timestamping Scheme with Absolute Timestamps (Short Paper)*

Duc-Phong Le¹, Alexis Bonnecaze², and Alban Gabillon³

¹ Laboratoire LIUPPA, IUT de Mont de Marsan, 40004 Mont de Marsan
dle@etud.univ-pau.fr

² Laboratoire IML, Université de la Méditerranée, 13288 Marseille cedex 09 France
alexis.bonnecaze@esil.univmed.fr

³ Laboratoire GePaSud, Université de la Polynésie Française, 98702 FAA'A - Tahiti -
Polynésie française
alban.gabillon@upf.pf

Abstract. The aim of timestamping systems is to provide a proof-of-existence of a digital document at a given time. Such systems are important to ensure integrity and non-repudiation of digital data over time. Most of the existing timestamping schemes use the notions of round (a period of time) and round token (a single value aggregating the timestamping requests received during one round). Such schemes have the following drawbacks: (i) Clients who have submitted a timestamping request must wait for the end of the round before receiving their timestamping certificate (ii) TimeStamping Authorities (TSA) based on such schemes are discrete-time systems and provide relative temporal authentication only, i.e. all the documents submitted during the same round are timestamped with the same date and time. (iii) the TSA can tamper timestamps before the round token is published in a widely distributed media. In this paper, we define a new timestamping scheme which overcomes these drawbacks.

Keywords: Timestamping scheme, Merkle tree, Chameleon hash function, Absolute timestamp.

1 Introduction

The use of digital documents is growing rapidly, nowadays. It thus becomes very important to ensure their security when they are stored/exchanged on the open network environment. Cryptographic primitives, including digital signatures, help to provide ongoing assurance of authenticity, data integrity, confidentiality and non-repudiation. Besides that, it is also important to be able to certify that an electronic document has been created at a certain date. Timestamping protocols, which prove the existence of a message/document at a certain time, are mandatory in many domains like patent submissions, electronic votes

* This work was supported by Conseil Général des Landes and the French Ministry for Research under Project ANR-07-SESU-FLUOR.

or electronic commerce, where frauds are related to monetary (or even political) interests. Moreover, timestamping services can serve as non-repudiation services. A digital signature is only legally binding if it was made when the user's certificate was still valid.

Timestamping schemes provide two types of temporal authentication: *absolute authentication* [1,14] and *relative authentication* [2,10,8,4]. Most of the existing timestamping schemes use the notions of round (a period of time or a number of requests) and round token (a single value aggregating the timestamping requests received during one round). Such schemes have the following drawbacks: (i) Clients who have submitted a timestamping request must wait for the end of the round before receiving their timestamping certificate (ii) TimeStamping Authorities (TSA) based on such schemes are discrete-time systems and provide relative temporal authentication only, i.e. all the documents submitted during the same round are timestamped with the same date and time. It is important to note that even totally ordered schemes are not continuous time schemes and comparing timestamps issued from different totally ordered TSS can be impossible. (iii) the TSA can tamper timestamps before the round token is published in a widely distributed media.

Note that there also exist distributed schemes (see [6,7] and [23]). They are secure (particularly against a Denial of Service attack) but heavy because of the huge number of interactions between the servers. Another drawback of these techniques is the lifetime of the signatures which were sent to clients by trusted servers (or by the users of service). If the keys pair of a signer is expired, valid timestamps cannot be verified.

In this paper, we define a new round-based timestamping scheme which overcomes these drawbacks. First, we use chameleon hash functions [17] to construct the authenticated data tree (in this paper, we use the Merkle tree) and to compute the round token at the beginning of the round. Second, our scheme is distributed, i.e. several servers cooperate to compute round tokens and timestamping certificates.

For a given round, our timestamping procedure consists of two phases. The first phase is performed off-line before the round begins (before receiving the timestamping requests) and the second phase is performed on-line after the beginning of the round (while receiving the requests). Using a chameleon hash function allows the TSA to pre-construct an authenticated data tree and pre-generate timestamps and round token in the off-line phase. Later, in the on-line phase, when the i^{th} request of the round arrives, the TSA only needs to find a collision for the timestamp at the position i in the authenticated data tree and returns the i^{th} timestamp to the client. Consequently, a timestamping certificate with absolute time can be returned to the client immediately after receiving his request. The same technical idea was used to construct on-line/off-line signatures [21].

The biggest danger of using a chameleon hash function lies in the possible compromise of the trapdoor key TK . We eliminate such a risk by distributing the trapdoor key in a network of servers. Each server knows only a fragment of

the trapdoor key TK . Therefore, any calculation requiring the trapdoor key can only be done by a collaboration of a given number of servers. This number λ is called the threshold. We need a collaboration of at least λ servers to get a result. Such a system is called a (λ, n) -threshold cryptographic system. In our scheme, if k is the number of corrupted servers, then the threshold must be greater than $2k$ in order for the calculation to succeed. The use of a threshold scheme has two objectives: (i) Avoiding the compromise of the trapdoor key. (ii) Proving the absolute time. The absolute time must be checked by at least λ servers. In our scheme, we need $\lambda = 2k + 1$, when the number of failed or malicious servers is at most k ¹.

Threshold secret sharing protocols frequently require one trusted dealer which generates and gives the secret to the n servers. In order to eliminate such a trusted dealer, we use a Distributed Key Generation (DKG) protocol to generate the hashing key HK and trapdoor key TK .

The remainder of this paper is organized as follows. Section 2 introduces some preliminaries. In Section 3, we present our scheme. In Section 4, we briefly analyze its security. Finally, we conclude the paper in Section 5.

2 Preliminary

2.1 Notations and Cryptographic Tools

We denote by $\{0, 1\}^*$ the set of all (binary) strings of finite length. If X is a string then $|X|$ denotes its length in bits. If X, Y are strings then $X||Y$ denotes an encoding from which X and Y are uniquely recoverable. If S is a set then $X \in_R S$ denotes that X is selected uniformly at random from S . For convenience, for any $k \in N$ we write $X_1, \dots, X_k \in_R S$ as shorthand for $X_1 \in_R S, \dots, X_k \in_R S$.

In this paper, we use the following cryptographic tools:

Authenticated Data Structure . Authenticated data structures provide cryptographic proofs that their answers are as accurate as the author intended, even if the data structure is being maintained by a remote host. The most known authenticated data structure proposed by Merkle [19] is called *Merkle tree*.

Chameleon Hash Function . The principal cryptographic tool we use in this paper is a *chameleon hash function*. Informally, a chameleon hash function is a special type of hash function, whose collision resistance depends on the user's state of knowledge: Without knowledge of the associated trapdoor, the chameleon hash function is resistant to the computation of pre-images and of collisions. However, with knowledge of the trapdoor, collisions are efficiently computable. See [17] for more detail.

Shamir Secret Sharing . Shamir's secret sharing scheme [20] is a threshold scheme based on polynomial interpolation. It allows a dealer D to distribute a secret value s to n players, such that at least $\lambda < n$ players are required to reconstruct the secret.

¹ We require $k < n/3$ to guarantee robustness.

2.2 Security Requirements

The security objectives of timestamping schemes is to guarantee the properties: *correctness*, *data integrity* and *availability*. The first property requires that the verification succeeds only if T_x is correct. The second means that an adversary cannot tamper the timestamp by back or forward dating it, or modify the request associated to it, or insert an old timestamp in the list of timestamps previously issued. The last property means that timestamping and verification must be available despite processes failures.

In general, there exist two major types of attacks on timestamping protocols: *back-dating* and *forward-dating* attacks. In the former attack, an adversary may try to “back-date” the valid time-stamp. This is a fatal attack for applications in which the priority is based on descendant time order (e.g. patents ...). The adversary may corrupt the TSA and may try to create a forged but valid timestamp token. In the later, an adversary may try to “forward-date” the timestamp without the approval of the valid requester. This is a fatal attack for applications in which the priority is based on ascendant time order (e.g. Will ...).

The forward-dating attacks can be prevented by requiring the client’s identity which allows us to determine who had timestamped the document. This type of attacks was analyzed in more details by Matsuo and Oguro in [18]. Thus, security problems of timestamping systems only concentrate on back-dating attacks. The strongest security condition against back-dating attacks for a timestamping scheme is presented by Buldas and Laur in [9].

3 Our Construction

In this section, we describe a round-based distributed timestamping scheme which provides absolute temporal authentication. The basic idea is to make use of the chameleon hash function to pre-construct a Merkle tree before each round. When the TSA receives the i^{th} timestamping request, it finds a collision at the i^{th} position in the Merkle tree and immediately returns a timestamping certificate to the client. In order to guarantee the secret of the trapdoor key of the chameleon hash function, our scheme uses a threshold secret sharing scheme which enables the trapdoor key to be shared among n servers such that a subset of them can find a trapdoor collision of the chameleon hash function without reconstructing the key. In other words, at most k of the n servers in a threshold timestamping scheme may be compromised without endangering the security of the timestamping scheme. Our timestamping scheme is robust with $k < n/3$ and can detect compromised servers when they try to generate a faulty partial trapdoor collision.

The timestamping service consists of two types of servers: a reception server and several hash servers. The former will receive requests, add time and return timestamps to clients. The later consists of n servers, each of them keeps a fragment of the trapdoor key of the TSA and will make a partial trapdoor collision of the chameleon hash function when a new timestamping request will arrive. The trapdoor key TK is generated and stored in distributed manner. This

key can be regularly renewed after some rounds or when a number of corrupted servers is superior to a threshold k . Before each round, (m'_i, r'_i) pairs are also generated and stored in a distributed manner in the hash servers. Then, the reception server generates hash digests $h_i = \mathcal{H}_r(m'_i, r'_i)$, constructs the Merkle tree and publishes the round token and the hashing key HK in a newspaper. When Alice sends a document she want to have timestamped to the TSA, a subset of $2k+1$ hash servers finds a partial trapdoor collision r_i and then returns a timestamp certificate to the client. In particular, the timestamping scheme works as follows:

Let \mathcal{P} be the set of n hash servers, l be a security parameter of system, r be a security parameter of random strings in the chameleon hash function, $\mathcal{H} : \{0, 1\}^* \mapsto \{0, 1\}^l$ be a collision-resistant hash function (e.g. SHA-1) and $\mathcal{H}_r : \{0, 1\}^l \times \{0, 1\}^r \mapsto \{0, 1\}^l$ be a chameleon hash function. In describing, we use the Merkle tree and the log-discrete-based chameleon hash function $\mathcal{H}_r(m, r) = \mathcal{H}_r(g^{r+xm}) = \mathcal{H}_r(g^r h^m)$, where $h = g^x$, x is trapdoor key and g, h are hash keys. This chameleon hash function was presented in [12], and its security relies on *one-more-discrete-logarithm* assumption. We also denote $\mathcal{M} = \{0, 1\}^l$ and $\mathcal{R} = \{0, 1\}^r$.

Key Generation. In describing of the key generation, we make use of the Distributed Key Generation (DKG) protocol for discrete-log-based threshold cryptosystems of Gennaro et al. [13] to securely generate keys and m'_i, r'_i pairs.

1. Use the DKG protocol to create $h = g^x$, where $x \in_R \mathbb{Z}_p$ is the trapdoor key and $P_j \in \mathcal{P}, 1 \leq j \leq n$ receives the share x_j for a degree k polynomial $p_x(y) \in \mathbb{Z}_p[y]$ such that $p_x(0) = x$.
2. Publish the hashing keys g, h . Each hash server $P_j \in \mathcal{P}$ retain x_j .

Setup. Suppose that 2^m be the number of requests pre-generated for each round. For $1 \leq i \leq 2^m$:

1. Use the DKG protocol to create $g^{r'_i}$, where $r'_i \in_R \mathcal{R}$. Each hash server $P_j \in \mathcal{P}$ receives the share r'_{ij} for a another degree k polynomial $p_{r'_i}(y) \in \mathbb{Z}_p[y]$ such that $p_{r'_i}(0) = r'_i$.
2. Use the DKG protocol to create $h^{m'_i}$, where $m'_i \in_R \mathcal{M}$. Each hash server $P_j \in \mathcal{P}$ receives the share m'_{ij} for a another degree k polynomial $p_{m'_i}(y) \in \mathbb{Z}_p[y]$ such that $p_{m'_i}(0) = m'_i$.
3. Use the DKG protocol to generate shares z_{ij} for each hash server $P_j \in \mathcal{P}$ of a degree $2k$ polynomial $p_0(y) \in \mathbb{Z}_p[y]$ such that $p_0(0) = 0$.
4. Now $g^{r'_i}$ and $h^{m'_i}$ are both known to the servers, so the hash digest $h_i = \mathcal{H}_r(m'_i, r'_i) = \mathcal{H}_r(g^{r'_i} h^{m'_i})$ can be computed. The reception server computes h_i for $i = 1, \dots, 2^m$.
5. The reception server constructs the Merkle tree whose leaves are above hash digests and computes tree root (round token). Then the TSA publishes the round token, the hashing key (g, h) of the chameleon hash function \mathcal{H}_r .
6. Generate a sequence of authentication paths $auth_i$, one for each leaf. These paths are stored by the reception server.

To compute parent's node value in constructing the Merkle tree from hash digests h_i , we use a cryptographic hash function \mathcal{H} (e.g. SHA-1).

Timestamping. The Stamping Protocol used to generate a timestamp works as follows:

1. Alice, the client i^{th} of the round sends her identity and the hash value of the document D_i she wants to have timestamped: she sends $ID_A, m_i (= \mathcal{H}(D_i))$.
2. The reception server adds the current time t to m_i , computes $m = \mathcal{H}(m_i || t)$ and then sends t, m_i, m to all of the servers.
3. Each hash server $P_j \in \mathcal{P}$ checks the correct time t (within reasonable limits of precision) and checks whether $m = \mathcal{H}(m_i || t)$. If so, P_j computes $c_{1j} = r'_{ij} - x_j m$ and $c_{2j} = x_j m'_{ij} + z_{ij}$ which is P_j 's share of the trapdoor collision. The role of the share z_{ij} in c_{2j} is to allow us make the polynomial random. Then P_j sends c_{1j}, c_{2j} to the reception server and to all of the other server in \mathcal{P} . After receiving $2k + 1$ shares² from a subset \mathcal{P}' of hash servers \mathcal{P} , the reception server:
4. Defines $f_j(y) = \prod_{P_l \in \mathcal{P}' \setminus P_j} \frac{l-y}{l-j}$, as in the definition of Lagrange interpolation. The trapdoor collision is computed as follows:

$$\begin{aligned}
 r_i &= \sum_{P_j \in \mathcal{P}'} (c_{1j} + c_{2j}) f_j(0) \\
 &= \sum_{P_j \in \mathcal{P}'} (r'_{ij} - x_j m + x_j m'_{ij} + z_{ij}) f_j(0) \\
 &= r'_i + x m'_i - x m.
 \end{aligned}$$

5. Each hash server $P_j \in \mathcal{P}'$ discards its share m'_{ij}, r'_{ij}, z_{ij} .
6. The TSA then returns the timestamp certificate $C_i = (i, ID_A, m_i, r_i, t, auth_i, t_r)$ to Alice, where i is the certificate serial number, and t the current date and time, t_r is the date and time for the round and $auth_i$ is the authentication path of the message m_i on the Merkle tree (for the purpose of reconstructing this timestamp). This is the timestamp for Alice's document D_i .
7. Alice receives the certificate and checks that it contains the hash of the document she asked a timestamp for and the correct time (within reasonable limits of precision).

Verification. A verifier who questions the validity of the timestamp C_i for the document D_i will:

1. Check that the hash value $m_i = \mathcal{H}(D_i)$ corresponds to the document D_i .
2. Compute the value of the leaf i : $h_i = \mathcal{H}_r(\mathcal{H}(m_i || t), r_i)$ check that it is part of the data that reconstructs the timestamp for the round.

² Our protocol requires a multiplication operation of two secrets x and m , so each share will be a degree $2k$ polynomial. For this reason, we need $2k + 1$ servers for each calculation.

Furthermore, the scheme is robust against dishonest hash servers. As pointed out in [12], we can verify values c_{1j}, c_{2j} for the purpose of detecting incorrect shares using zero-knowledge proofs for verification.

If any of the shares is deemed incorrect, then broadcast a complaint against P_j . If there are at least $k + 1$ complaints, then clearly P_i must be corrupt since with at most k malicious players, there can be at most k false complaints.

4 Efficiency and Security Analysis

From the point-of-view of efficiency, in our scheme the costs of our key generation and of our construction of Merkle tree are dominated by the cost of the DKG protocol [13]. Our timestamping phase only requires one round of communication between servers. Finally, our verification phase is efficient, it can be executed in an off-line manner by a verifier, i.e. there needs no interaction with the timestamping service.

The security of our scheme reduces directly to the security of chameleon hash functions, the security of the Merkle tree (that depends on the property “collision-resistant” of hash functions) and the security of the threshold scheme.

Our scheme tolerates the participation of at most $k < \frac{n}{3}$ corrupted servers and requires $2k + 1$ servers to construct a timestamping certificate³. When the number of corrupted servers reaches k , our scheme allows to change the pair of keys without influencing the correctness of previous timestamping certificates. One can always verify the correctness of a timestamping certificate even if the timestamping service is not available.

5 Conclusion

Our new round-based timestamping scheme delivers certificates instantaneously. Another interesting point is that each certificate provides a provable absolute time. Until now, schemes based on authenticated data structure like Merkle trees did not have this capability. This scheme can be easily implemented since the cryptographic bricks are already well known and partially implemented.

References

1. Adams, C., Cain, P., Pinkas, D., Zuccherato, R.: Internet X.509 Public Key Infrastructure Time-Stamp Protocol, TSP (2001)
2. Bayer, D., Haber, S., Stornetta, W.S.: Improving the efficiency and reliability of digital time-stamping. In: Sequences II: Methods in Communication, Security, and Computer Science, London, UK, pp. 329–334. Springer, Heidelberg (1993)
3. Benaloh, J., de Mare, M.: Efficient broadcast time-stamping. Technical Report 1 TR-MCS-91-1, Clarkson University Department of Mathematics and Computer Science (August 1991)

³ We require $2k + 1$ servers for one calculation and tolerate k corrupted servers, thus k should be inferior $n/3$.

4. Blibech, K., Gabillon, A.: A new timestamping scheme based on skip lists. In: Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3982, pp. 395–405. Springer, Heidelberg (2006)
5. Bonnetcaze, A.: A multi-signature for time stamping scheme. In: SAR/SSI 2006: The 1st Conference On Security in Network Architectures and Information Systems, Seignosse, France (June 2006)
6. Bonnetcaze, A., Liardet, P., Gabillon, A., Blibech, K.: Secure time-stamping schemes: A distributed point of view. *Annals of Telecommunications* 61(5-6), 662–681 (2006)
7. Bonnetcaze, A., Trebuchet, P.: Threshold signature for distributed time stamping scheme. *Annals of telecommunications* 62(11-12), 1353–1363 (2007)
8. Buldas, A., Laud, P., Lipmaa, H., Vilemson, J.: Time-stamping with binary linking schemes. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 486–501. Springer, Heidelberg (1998)
9. Buldas, A., Laur, S.: Do broken hash functions affect the security of time-stamping schemes? In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 50–65. Springer, Heidelberg (2006)
10. Buldas, A., Lipmaa, H.: Digital signatures, timestamping and the corresponding infrastructure. Technical report, Küberneetika AS (2000)
11. Chaum, D., Pedersen, T.P.: Wallet Databases with Observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
12. Crutchfield, C., Molnar, D., Turner, D., Wagner, D.: Generic on-line/off-line threshold signatures. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 58–74. Springer, Heidelberg (2006)
13. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.* 20(1), 51–83 (2007)
14. Haber, S., Stornetta, W.S.: How to Time-Stamp a Digital Document. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 437–455. Springer, Heidelberg (1991)
15. Haber, S., Stornetta, W.S.: Secure names for bit-strings. In: ACM Conference on Computer and Communications Security, pp. 28–35 (1997)
16. Just, M.: Some timestamping protocol failures. In: NDSS 1998: Proceedings of the Symposium on Network and Distributed Security, San Diego, CA, USA, pp. 89–96 (March 1998)
17. Krawczyk, H., Rabin, T.: Chameleon signatures. In: NDSS (2000)
18. Matsuo, S., Oguro, H.: User-side forward-dating attack on timestamping protocol. In: Proc. of the 3rd International Workshop for Applied Public Key Infrastructure (IWAP 2004), pp. 72–83 (2004)
19. Merkle, R.C.: Secrecy, authentication, and public key systems. PhD thesis (1979)
20. Shamir, A.: How to share a secret. *Commun. ACM* 22(11), 612–613 (1979)
21. Shamir, A., Tauman, Y.: Improved online/offline signature schemes. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 355–367. Springer, Heidelberg (2001)
22. Takura, A., Ono, S., Naito, S.: A secure and trusted time stamping authority. In: IWS 1999: Internet Workshop, Osaka, Japan, pp. 88–93. IEEE Computer Society, Los Alamitos (1999)
23. Tulone, D.: A secure and scalable digital time-stamping service (2006)

A Framework for Trustworthy Service-Oriented Computing (Short Paper)

Sasikanth Avancha

Systems Research Center, Intel Technology India Pvt. Ltd., Bangalore, India
`sasikanth.avancha@intel.com`

Abstract. Service-oriented computing requires un-trusted and trusted software to simultaneously execute on the same hardware platform. Trusted software protects a service provider’s business model and must execute in a high assurance environment. Increasingly, hardware mechanisms are required to create high-assurance closed environments to host trusted software on open platforms. In current approaches, independent hardware vendors (IHVs) design and implement closed environments with proprietary interfaces specific to mobile phones, PCs and servers, forcing independent software vendors (ISVs) and service providers to develop non-portable software. In this paper, we present an abstract closed environment architecture that exposes its facilities via implementation-independent canonical interfaces. IHVs can use this architecture to implement platform-specific closed environments, while ISVs and service providers develop applications to the canonical interface and build portable trusted software. We discuss example implementations of our framework to demonstrate the feasibility of building scalable solutions to support trustworthy service-oriented computing.

1 Introduction

According to the Service Oriented Architecture (SOA) Reference Model [1], SOA is “*a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains*”. The “distributed capabilities” include hardware platforms, systems software, applications and associated data, under the control of different “ownership and administrative domains”, including service providers, service brokers and service consumers. An implicit SOA requirement is *mandatory policy* enforcement on the use of each distributed capability, especially when it is utilized in a physical context not controlled by the capability-owner or policy-owner. Mandatory policies can only be enforced in a “closed” environment on an otherwise open platform, e.g., a network operator enforces its policy on a mobile phone using the *closed*, smart card environment on the Subscriber Identity Module (SIM), while the mobile phone owner is free to install any applications to run on the phone’s operating system, which is an *open* environment. Therefore, platforms required to support SOA usage models must be *hybrid*: open, general-purpose programmable environments alongside closed, special-purpose programmable environments supporting trusted software that

enforces *mandatory policy* on behalf of service providers. Indeed, we see a proliferation of these types of platforms in the form of devices such as the iPhone and other smartphones.

Current approaches to build hybrid platforms focus on enhancing conventional software and hardware architectures. For example, TrustZone® is an enhancement to ARM® processor cores; it is a mechanism to execute security-sensitive parts of an application in a secure mode (closed environment), which is separate from the normal mode (open environment) supporting standard applications. Other examples are Intel® TXT^{TM,1} and AMD-VTM, that enhance x86 processors to support creation of closed environments on open platforms. The IBM® 4758 [2] is a standalone, closed environment for server platforms. It is a cryptographic co-processor based on a physical-tamper-resistant PCI card, providing an OS independent software execution environment. It is rated at FIPS 140-1 Level 4, the highest that the National Institute of Standards and Technology (NIST) assigns to security products.

A problem with state-of-art approaches is that IHV's find it difficult to change closed environment designs based on these approaches without breaking applications; IHVs also do not expose a consistent set of security properties of their designs. Similarly, ISV's code cannot easily port to platform types (even with the same processor ISA), but with differing design approaches to support closed environments. For example, ARM processors are optimized for mobile phones and embedded systems, not PCs or servers. Therefore, it is practically infeasible to adopt TrustZone for PCs or servers based on other architectures. Conversely, Intel TXT and AMD-V are PC technologies, and it is not at all clear that they can be "ported" to mobile phones and servers without significant changes, potentially weakening their security properties.

The primary contribution of this paper is an architectural framework consisting of platform-agnostic mechanisms and principles for designing and exposing closed environments via a canonical set of interfaces on open platforms. Our framework consists of three components:

- An architectural construct that is rooted in hardware, is programmable, possesses a canonical set of security properties to be a closed environment and provides high assurance. We refer to this construct as a "secure vault" in this paper.
- A canonical programming and messaging interface that ISVs and service providers use to build services that interact with the secure vault and the OS environment
- A software run-time environment that allows hardware-agnostic software and services to execute in the secure vault

The rest of the paper is organized as follows: in Section 2 we present a brief overview of related work. Section 3 describes our framework in detail. In

¹ Intel, TXT and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Other names and brands may be claimed as the property of others.

section 4, we describe examples of implementations based on our framework. We state our conclusions in Section 5.

2 Related Work

While a significant body of work on implementation approaches [2,3,4,5] to create closed environments on open platforms exists, our focus is on work that takes a higher-level architectural approach to the problem. Therefore, in this section, we briefly describe two recent efforts on creating a security architecture with a canonical set of architectural components whose properties meet platform stakeholder requirements.

The TCG-MPRA (TCG Mobile Phone Reference Architecture) specification [6] describes a generic architecture that meets requirements of four stakeholders of a mobile phone platform: device manufacturer, network operator, 3rd party service provider and user. The TCG-MPRA defines a component called a Trusted Engine (TE) that each stakeholder owns on the platform; it contains data and services that the stakeholder provides to the other entities on the platform. The TCG-MPRA specifies two levels of isolation: between TEs and within TEs (between “trusted services” and “normal services”). Each TE exposes a set of interfaces to other TEs and un-trusted components on the platform, and requires the underlying mechanisms to enforce security policies at those interfaces.

Ashkenazi and Hartley [7] describe a security architecture and its integration with virtualization for embedded systems. The paper discusses key components of the security architecture required to provide assurance to “high-value services” executing on embedded systems. These components include Secure Boot, a hardware-based Run-Time memory Integrity Checker (RTIC), a tamper-resistant Secure RAM for run-time secrets with a Key Encryption Module (KEM) to store secrets on disk, a Secure Debug port and a VMM-based Trusted Execution Environment (TEE) to provide authenticity, integrity and confidentiality properties to software, data, and secrets.

While there are similarities between our approach and those described above, a key difference is that our approach applies to any type of platform - mobile, embedded, PC and server - whereas the above approaches are specific to mobile and embedded platforms. The rationale for our cross-platform approach is that the SOA requires all platform types - not only mobile devices or PCs, but also servers and possibly network routers - to enable trusted software enforce mandatory policy in a closed environment.

3 Proposed Framework

In this section, we describe the architectural constructs and security principles of our framework. Figure 1 shows the three components of our framework, their inter-relationship and the relationship of our framework to the OS environment. The Secure Vault is an *architectural construct* that provides a closed environment to trusted software for mandatory policy enforcement. IHVs can design

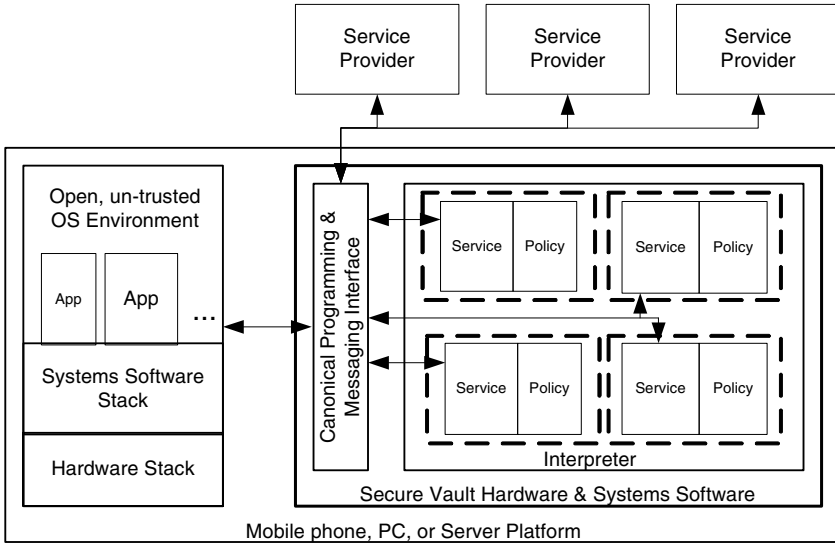


Fig. 1. Trustworthy Service-Oriented Computing Platform

and implement this component using any available approach (e.g., AEGIS [3], IBM 4758 [2], our approach discussed in section 4 or some other approach or combination), as long as it meets the security specifications discussed in section 3.1 and meets platform specifications for performance and power. The Canonical Programming and Messaging Interface in Fig.1 is based on industry standard protocols and formats, e.g., WS-* and SAML, and presents a uniform interface that remote parties can use to provision and manage services in the secure vault. The Interpreter, e.g., a JavaTM Virtual Machine, executes interpretable software in a sandboxed environment; the value of the interpreter in this scenario is to present a portable, cross-architecture, binary format for applications, and not for security. Therefore, a service running in a sandbox depends on the secure vault, not the sandbox, to enforce its mandatory policy. This fact is shown by a dashed box around each service in Fig.1.

3.1 Secure Vault

The Secure Vault (SV) is based on the principles [8] of **Minimal Trusted Computing Base (TCB)**, **Least Privilege Assignment** and **Separation of Privilege**. The SV must consist of exactly those (i.e., a minimal set) hardware, firmware and systems software components required for it to support the execution of trusted software, and no more. For example, it must not contain a driver for a device that trusted software does not need. The SV TCB must assign the least level of access privilege to each trusted software entity such that it can function correctly and be usable; the SV TCB must apply a *default-deny* access policy and grant access permissions only after evaluating each access

request using a pre-defined policy. The SV TCB must *logically* separate itself - using CPU privilege levels combined with its own security policy and enforcement mechanism or being encoded using type-safe languages - from non-TCB elements to ensure its protection.

A secure vault must possess the following security properties: **tamper resistance** - i.e., be resistant [2,3,5] to both software and hardware attacks, **high integrity** - i.e., contain hardware and software components that form the SV's root of trust [2,6,7] to prove its trustworthiness to a remote service provider, **safe execution environment** - i.e., provide a programmable run-time environment that is a root of trust for software entities executing in isolated protection domains (PDs) with the required confidentiality and integrity properties, **mediated communications** - i.e., be able to enforce policy-based inter-domain communication as well as communication between a PD and non-PD entity outside the vault, **trusted I/O** - i.e., be able to gain control of a set of input devices, e.g., keyboard, mouse, graphics, biometric device, on-demand and redirect security-sensitive input data from those devices to a software entity that requires this input and **access control** - i.e., be able to control access to secrets produced and owned by software entities executing in the vault. We observe that the "assurance property" of the SV may vary and how to consistently express its assurance properties to those with a need to know is an open, but difficult, research problem.

It is desirable for a secure vault to possess *dependability* properties in addition to security properties. A dependable SV can guarantee quality of *service execution* to service providers, who in turn, enter into appropriate service level agreements (SLA) with users. A dependable SV must possess the following properties: **high availability** - i.e., be available whenever the user attempts to initiate a service-oriented transaction with the provider, even if the OS on the open part of the platform is not available, and **high reliability** - i.e., possess a high Mean Time Before Failure (MTBF) by minimizing the number of critical components whose failure would cause the SV to fail and being completely independent of the OS in the open environment for functionality, e.g., memory services.

Independent hardware vendors will benefit significantly by using the concept of a secure vault proposed in this paper. First, they can employ the SV as a single way to expose different programmable closed environment designs. Therefore, by *standardizing* the manner in which closed environment designs are exposed, IHVs can cost-effectively implement a closed environment on different types of platforms. IHVs can also use the secure vault concept as a benchmark against which they can evaluate level of security assurance a closed environment may provide for a specific platform. They can use evaluation results to improve each design, and in turn, enable ISVs and service providers build more robust software.

3.2 Canonical Programming and Messaging Interface (CPMI)

The second component of our framework, CPMI, exposes a set of core functions to software executing within the SV and in the service provider's back-end infrastructure. Some of these functions include **provisioning** - to enable a service

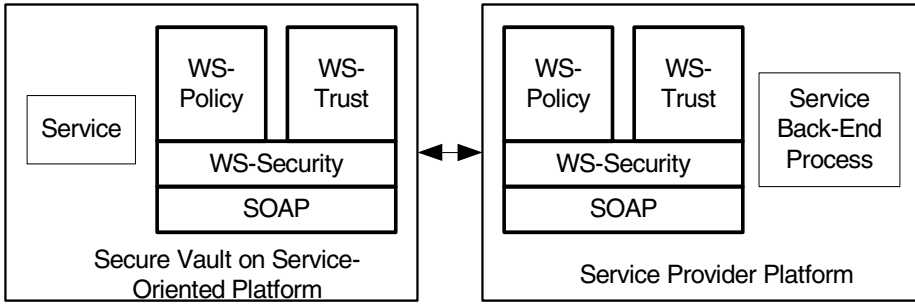


Fig. 2. Web Services as Canonical Programming & Messaging Interface

provider to deploy software into the secure vault, **remote attestation** - to enable the secure vault to prove its trustworthiness to the service, and functions for **lifecycle management** - to enable service providers to remotely manage their provisioned software. CPMI enables service providers to build back-end software and services agnostic to the specific technology used to implement the vault. Further, software executing in the secure vault can use the CPMI to request the vault to act on their behalf to perform certain functions or to interact with the un-trusted OS environment. The Web Services Framework [9,10] is an example of a CPMI. Figure 2 shows a conceptual model of interaction between a Service-Oriented Platform (i.e., a platform built using our framework) and a service provider using web services based CPMI. Thus, CPMI enables the IHVs to continually improve their implementation of the secure vault while supporting all externally-visible functions.

3.3 Interpreter

In this section we briefly discuss the need for, and the benefits of, an interpreter for service-oriented computing and its place in our framework. In general, an interpreter serves as a “virtual machine” environment, e.g., a Java Virtual Machine (JVMTM), managing the software run-time and translating bytecode into intermediate code or native instructions. Thus, the interpreter shields software and service developers from details of the hardware implementation. Further, interpreters provide mechanisms for isolating software modules from each other, by managing their stack and heap, and also tracking overflows and cross-references [11]. We expect an interpreter to play a similar role in our framework, albeit with a greater focus on providing efficient translation than access control. This is because the interpreter is an application layer component and as such, does not have the ability to enforce mandatory policies on behalf of different principals *at the system level*. For example, it cannot enforce an access control policy with respect to the TPM on behalf of a remote party service, because the *secure vault* owns the TPM and only it can control access to the TPM. Similarly, the interpreter cannot enforce isolation between remote party services it supports

and those executing in a different interpreter in the vault. Therefore, as shown in Figure 1, the secure vault enforces mandatory policy on behalf of each principal, whereas the interpreter provides a platform-independent run-time environment.

4 Example Implementations of Proposed Framework

In this section, we briefly describe examples of secure vault implementation based on the proposed framework.

The first example describes our ongoing effort to implement a secure vault using off-the-shelf components and enhance them such that the vault meets the overall security and dependability requirements discussed in section 3.1. The secure vault implementation in this example is based on a microkernel called L4 [12] running on an Intel TXT based CPU, which is on a separate motherboard connected to the main CPU board over a PCI link. Thus, the Intel CPU and L4 form the secure vault's hardware and systems software infrastructure. This infrastructure provides software assurance to the vault's TCB; the board has its own RAM and a TPM to provide roots of trust for measurement, storage and reporting - together, these components provide limited protection to the secure vault's TCB against hardware attacks. L4, in combination with the Intel TXT CPU provides *high integrity, separation of privilege, least privilege assignment* and *mediated communications* within the vault. Work on enabling access control and trusted I/O mechanisms in L4 is in progress. The CPMI in this example implementation is based on the Web Services Framework. Work on integrating a JVM-like interpreter into the L4 environment is in progress.

The second example describes how a secure vault can be implemented using a combination of SELinux and a technology such as AEGIS [3] or IBM 4758 [2] to provide tamper resistance against hardware attacks. This example is similar to the effort described in [13] in which SELinux forms the TCB and provides software assurance to each software entity executing as a Linux process or thread in its own domain. In [13], SELinux is used to provide *domain isolation* (by enforcing a fairly complex information flow policy) and is extended to *enforce integrity policy* on behalf of each domain. By design SELinux enforces *separation of privilege* on domains, assigns *least privilege* to each domain and enforces *mediated communications* between domains. As SELinux is the TCB, can enforce access control on domains to protect their secrets from other software; it can gain also access to I/O devices on behalf of domains. We are also investigating how an SELinux-based vault can be hidden beneath a Web-service interface and a JVM, a subject for a future paper.

5 Conclusions

The service-oriented computing paradigm will succeed only if end-points can enforce mandatory policies on behalf of remote parties. Current approaches to this problem employ platform-specific one-off solutions. IHVs create different designs for each platform; service providers are either forced to develop variants of the

same service to suit various platforms or worse, to deploy the service without requisite protections, resulting in violation of the mandatory policy associated with the service. The main contribution of this paper is to provide a framework that addresses these issues. Our framework enables IHVs to choose a desired level of platform security assurance and implement a platform-specific solution based on their choice. Software vendors and service providers can develop their software and services independent of the hardware vendor, and rest assured that their modules can be easily ported across platforms.

References

1. MacKenzie, C.M., et al.: Reference Model for Service Oriented Architecture 1.0. World Wide Web (October 2006), <http://docs.oasis-open.org/soa-rm/v1.0/>
2. Dyer, J.G., et al.: Building the IBM 4758 Secure Coprocessor. *IEEE Computer* 34, 57–66 (2001)
3. Suh, G.E., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: AEGIS: architecture for tamper-evident and tamper-resistant processing. In: *ICS 2003: Proceedings of the 17th annual international conference on Supercomputing*, pp. 160–171. ACM, New York (2003)
4. Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., Horowitz, M.: Architectural support for copy and tamper resistant software. *SIGPLAN Not.* 35(11), 168–177 (2000)
5. Alves, T., Felton, D.: TrustZone: Integrated Hardware and Software Security. World Wide Web (July 2004), <http://www.arm.com/pdfs/TZ.Whitepaper.pdf>
6. TCG: TCG mobile reference architecture specification version 1.0. World Wide Web (June 2007), <https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobilereference-architecture-1.0.pdf>
7. Ashkenazi, A., Hartley, D.: Securing Mobile Devices with Platform Independent Security Architecture. In: *Embedded Systems Conference* (April 2008)
8. Saltzer, J.H., Schroeder, M.D.: The Protection of Information in Computer Systems. *Proc. of the IEEE* 63(9), 1278–1308 (1975)
9. Chinnici, R., et al.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. World Wide Web (June 2007), <http://www.w3.org/TR/2007/REC-wsd120-20070626>
10. Nadalin, A., et al.: WS-Trust 1.3. World Wide Web (March 2007), <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc>
11. Ekberg, J., et al.: OnBoard Credentials Platform Design and Implementation. Technical Report NRC-TR-2008-001, Nokia Research Center (January 2008)
12. Liedtke, J.: On μ -kernel construction. In: *Proc. 15th ACM Symposium on OS Principles*, pp. 237–250 (December 1995)
13. Zhang, X., Acicmez, O., Seifert, J.P.: A trusted mobile phone reference architecture via secure kernel. In: *STC 2007: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pp. 7–14. ACM, New York (2007)

Revisiting Bluetooth Security (Short Paper)

Manik Lal Das¹ and Ravi Mukkamala²

¹ Dhirubhai Ambani Institute of Information and Communication Technology
Gandhinagar - 382007, India
`maniklal.das@daaiict.ac.in`

² Department of Computer Science
Old Dominion University, Norfolk, VA 23529, USA
`mukka@cs.odu.edu`

Abstract. Bluetooth technology is gaining increasing interest in the research community because of the convenience of exchanging information between wireless devices. As the communication medium is wireless, security is an important concern in this emerging technology. This paper discusses the basic security of Bluetooth technology, some of its shortcomings and presents two new proposals for securing Bluetooth technology. One of the proposals is based on passkey-authenticated key exchange, where security relies on keyed hash function, and the other one is on amplified passkey-authenticated key exchange, where security relies on elliptic curve discrete logarithms problem. The latter provides some additional security services, but with added cost compared to the former one. Both protocols provide mutual authentication, resist known and possible threats, and achieve efficiency compared to other protocols.

Keywords: Bluetooth technology, Bluetooth security, Authentication, Privacy, Wireless communications.

1 Introduction

Bluetooth technology [1,2] facilitates a simple way to communicate with a wide range of devices connecting to the public or private network without wires, cables and connectors. Over the years, the Bluetooth technology has emerged rapidly in electronic society and has been used in several interesting real-world applications. The Bluetooth technology operates in the unlicensed industrial, scientific and medical (ISM) radio band at 2.4GHz that allows two Bluetooth enabled devices within 10-100 meter range to share data [3]. To date, a variety of products available in market have short range Bluetooth radios installed, including mobile phones, printers, laptops, keyboards and so on. The major challenges facing Bluetooth technology are: (i) robust and efficient security solution; (ii) vendor independence and application interoperability; and (iii) quality of service. Although Bluetooth Special Interest Group (SIG) Specification defines security at the link level [4,5]; the security requirements for Bluetooth vary from application to application.

The basic Bluetooth security mechanism enforces a device to choose either “discoverable” mode or “non-discoverable” mode. When a Bluetooth device is in discoverable, it is easy to establish a session using a Personal Digital Assistant (or a Notebook PC) and exchange data. In contrast, the “non-discoverable” mode enables Bluetooth devices invisible to other Bluetooth device, instead, the device would be only visible to those devices who know its Bluetooth MAC address. Once pairing of devices has formed, they can communicate each other in secure and non-secure modes as per configuration. By default, the communication is set as non-secure mode. However, for some services, such as, dial-up account, voice gateway, or file transfer, security services such as authentication, authorization and (optional) encryption, are required. It is noted that the link layer security needs to be stronger to make the Bluetooth communication safe and application layer security independent.

1.1 Related Work

Bellovin and Merritt [6,7] proposed the seminal work, known as Encrypted Key Exchange (EKE), using the Diffie-Hellman [8] key exchange approach, which opens substantial research directions for authenticated key exchange using different crypto primitives. Jablon [9] and Kwon [10] added more lights on authenticated key exchange mechanism. MacKenzie [11] proposed the password authenticated key exchange (PAK) with tight security notion.

In 2005, the work in [12,13] articulates how a Bluetooth PIN can be cracked by brute force search. Wong et. al. [13] also mentioned the inability to use a 128-bit PIN at the baseband, as the PIN is represented by a variable-length character encoding scheme. Jakobsson and Wetzel [14] observed two other attacks, namely location attack and cipher attack. A group of researchers from industry and academia, initiated by the Bluetooth SIG, proposed a simple pairing whitepaper for Bluetooth security [4] using Diffie-Hellman like key exchange and keyed hash function. However, their proposal lacks a rigorous security proof. Recently, Kostakos [15] observed some more threats on Bluetooth security while comparing the Bluetooth with RFID [16] security.

1.2 Contributions

This paper aims to present authenticated key exchange protocols for Bluetooth security. One of the proposals is based on passkey-authenticated key exchange, where security relies on keyed hash function such as HMAC-SHA256 [17], and the other one is designed as an amplified passkey-authenticated key exchange, where security relies on elliptic curve discrete logarithm problem [18]. The salient features of the proposed protocols are: (i) both protocols provide mutual authentication of the communicating parties; (ii) secure data transmission through a mutually agreed session key; and (iii) resist known and possible threats.

The remainder of the paper is organized as follows: Section 2 reviews the basic security model of Bluetooth technology, Section 3 presents our protocols, Section 4 analyzes the protocols’ security and efficiency, and Section 5 concludes the paper.

2 Bluetooth Security

This section reviews the basic security of Bluetooth that the SIG specification supports followed by some of the prominent limitations of the specification.

2.1 The Basic Security Model of Bluetooth

The security architecture of Bluetooth [5] is divided into three modes: (1) non-secure; (2) service-level enforced security; and (3) link-level enforced security. In non-secure mode, a Bluetooth device does not initiate any security measures. Whereas in service-level and link-level enforced security modes, two Bluetooth devices can establish an asynchronous connection-less link. The difference between service-level and link-level enforced security is that in the latter, the Bluetooth device initiates security procedures before the channel is established. Bluetooth supports authentication and encryption per mode of configuration. Authentication is provided by a 128-bit link key. The algorithms for authentication and encryption are based on the SAFER+ [19] cipher. We briefly review both service and link levels security as follows:

When the pair is formed for the first time, they generate and share a link key $K_{init} = E_{22}[PIN, \mathbf{addr}_I, RAND_R]$, where \mathbf{addr}_I is the address of the initiator device, say I, $RAND_R$ is a random number chosen by the receiver device, say R, PIN is a shared secret number that the user manually enter into both devices, and E_{22} is the algorithm based on the SAFER+ cipher. Once two devices share the initial link key K_{init} , they can renew it latter and derive a new one, known as a combination link key (say, K_{IR}). The combination key is derived as follows:

Devices I and R generate random number $RANDLK_I$ and $RANDLK_R$, respectively. Each device masks the random number by XORing it with K_{init} and sends it to other device. Both devices individually hash each of the two random numbers with the Bluetooth address of the device, using the algorithm E_{21} which is also based on the SAFER+ cipher. The two hashes are then XORed to generate the combination key K_{IR} as $K_{IR} = E_{21}(RANDLK_I, \mathbf{addr}_I) \oplus E_{21}(RANDLK_R, \mathbf{addr}_R)$. The old link key is then discarded. The authentication process follows a challenge-response mechanism. The initiator I sends a challenge $RAND$ to the receiver R and then R responds to I with an authentication token $SRES = E_1(K_{IR}, RAND, \mathbf{addr}_R)$, where $E_1(\cdot)$ is the authentication algorithm based on SAFER cipher.

Observations. It is straightforward to see that the basic security model of Bluetooth depends primarily on the user's PIN (or passkey). In other words, if user's PIN is compromised then the secret link key is derived easily from the PIN and other parameters. It is also noted that the works [12] and [13] observe how an attacker can find the PIN used during the pairing process. In fact, the results show that a 4-digit PIN can be cracked in less 0.06 sec on a Pentium IV 3GHz computer [12].

3 The Proposed Protocols

We present two protocols, namely SPAK and APAK, that provide secure and efficient authenticated key exchange for securing data transmission between the communicating Bluetooth-enabled devices. The SPAK is based on user passkey and keyed hash approach, whereas, APAK is based on amplified passkey and Diffie-Hellman key exchange approach. The notation used throughout the paper is given in Table 1.

Table 1. Notation used in the protocols

I	Bluetooth-enabled Initiator device
R	Bluetooth-enabled Receiver device
$\text{PRF}(k; < m >)$	keyed hashed value of message m using key k
$h(\cdot), h1(\cdot)$	cryptographically secure hash functions
$[m]^l$	most significant l bits of string m
$a b$	concatenation of strings a and b
$a \oplus b$	bitwise exclusive-OR of strings a and b
$X \rightarrow Y: < m >$	X sends message m to Y over a public channel

The proposed protocols require an association step between two Bluetooth-enabled devices (a similar concept of forming pairing). The user of the devices could run the association step, **Asso-Step**, either each time the devices want to communicate or periodically in changing his/her passkey.

3.1 SPAK: Simply Passkey-Authenticated Key Exchange

Asso-Step: This step is executed when two devices want to communicate for the first time or renewing the verifier. The user enters his/her passkey pw (or PIN) and a two-character salt s into I and R manually. Both I and R compute $v = \text{PRF}(pw; < \text{addr}_I || \text{addr}_R || s >)$ and store it in their database. Here the salt is used to avoid the dictionary attack of stolen verifier.

Once the association of devices is formed, the SPAK works as follows:

- s1) I \rightarrow R: $< \text{addr}_I, c_1, n > ::$ I chooses two random numbers n and r_1 , computes $c_1 = \text{PRF}((v \oplus r_1); < \text{addr}_I || n >)$, where n is acted as a nonce to safeguard the protocol against replay attacks. I sends (addr_I, c_1, n) to R.
- s2) R \rightarrow I: $< \text{addr}_R, c_2, h_1 > ::$ R first validates n . If n is fresh (e.g., value of n is greater than the previously received n), computes $c_2 = \text{PRF}((v \oplus r_2); < \text{addr}_I || \text{addr}_R >)$ and $h_1 = v \oplus (r_2 || n)$, where r_2 is a random number chosen by R; else abort. Then R sends $(\text{addr}_R, c_2, h_1)$ to I.
- s3) I \rightarrow R: $< h_2 > ::$ I first extracts r_2 from h_1 as $(r_2 || n) = h_1 \oplus v$, and then checks whether $c_2 = \text{PRF}((v \oplus r_2); < \text{addr}_I || \text{addr}_R >)$. If it does hold then R is authenticated; else abort. I computes $h_2 = r_1 \oplus r_2$ and sends h_2 to R.
- s4) R extracts $r_1 = h_2 \oplus r_2$ and checks whether $c_1 = \text{PRF}((v \oplus r_1); < \text{addr}_I || n >)$. If it holds then I is authenticated; else abort.

Data Confidentiality. Once both I and R get authenticated, they establish a session key as $sk = h(r_1 || r_2 || n)$ through which they can exchange data encrypted under the key sk .

3.2 APAK: Amplified Passkey-Authenticated Key Exchange

The APAK is designed in a similar line of AMP [10], but with a fine-tuned version, which makes APAK more robust and efficient compared to AMP [10] and others [13].

Asso-Step: Both I and R compute an amplified passkey apw from user's passkey pw as $apw = [h(\text{addr}_I || \text{addr}_R || t || pw)]^l$, where t is a synchronized timestamp of I and R used to avoid dictionary attack on apw to get user's passkey pw . Then they compute a common secret parameter $V = apw \cdot G$, where G is a common base point of the elliptic curve chosen by communicating parties.

Once the association of devices is formed, the APAK works as follows:

- a1) I \rightarrow R: $\langle C_I, h_3, \hat{n} \rangle ::$ I chooses a random \hat{r}_1 and a nonce \hat{n} , computes $Q_I = \hat{r}_1 G$, $C_I = Q_I + V$ and $h_3 = h1(\hat{n} || \text{addr}_R || Q_I)$. Then I sends (C_I, h_3, \hat{n}) to R.
- a2) R \rightarrow I: $\langle C_R, h_4 \rangle ::$ R first validates \hat{n} . If \hat{n} is fresh then R obtains Q_I from C_I as $Q_I = C_I - V$ and checks whether $h_3 = h1(\hat{n} || \text{addr}_R || Q_I)$; else abort. If h_3 is found correct then I is authenticated; else abort. Then R chooses a random \hat{r}_2 , computes $Q_R = \hat{r}_2 G$, $C_R = Q_R + V$ and $h_4 = h1(\hat{n} || \text{addr}_I || Q_R)$. R sends (C_R, h_4) to I.
- a3) I obtains Q_R from C_R as $Q_R = C_R - V$ then checks whether $h_4 = h1(\hat{n} || \text{addr}_I || Q_R)$. If it does hold then R is authenticated; else abort.

Data Confidentiality. Once both I and R get authenticated, they establish a session key as $\hat{sk} = h(Q_I || Q_R || \hat{n})$ through which they can exchange data encrypted under the key \hat{sk} .

4 Analysis of SPAK and APAK

4.1 Security Analysis

The security strength of SPAK and APAK are explained in the context of known and possible threats as follows:

Association threat: Bluetooth devices have a 48-bit unique identifier and 24-bit device class descriptor. Each device can be assigned a user friendly device name. And the association threat can be tackled by the renaming solution (similar to self-assigned IP-address of peer-to-peer IEEE 802.11 communications).

Location threat: With frequent changing Bluetooth devices' identifier, the location of device as well as person's position can be safeguarded from being revealed and tracked.

Eavesdropping: A strong passkey is linked in generating session key for each session that prevents eavesdropping.

In SPAK, the parameters are addr_I , addr_R , c_1 , c_2 , h_1 , h_2 and n , in which addr_I , addr_R , n do not contain any secret information. And with c_1 , c_2 , h_1 and h_2 , the eavesdropper cannot execute either passive or active attacks, as the security relies on the keyed hash function such as HMAC-SHA256.

Similarly, the eavesdropper cannot gain anything on having C_I , C_R , h_3 and h_4 in APAK, as security is based on elliptic curve discrete logarithm problem, which is believed to be an intractable problem.

Guessing threat: In our protocols, the user passkey is not transmitted as a simple hashed passkey, instead, passkey is masked in a secret random number. On intercepting $\langle c_1, c_2, h_1, h_2 \rangle$ or $\langle C_I, C_R, h_3, h_4 \rangle$, the intruder cannot figure out pw or apw , as in every case there are two unknown parameters (pw or apw and a random number) to be guessed on each string and the intruder needs to guess pw/apw .

Freshness of Session key: The protocol run resists the freshness property of the session, as the intruder cannot guess a fresh session key on having some previously issued session keys. The session key is computed as $sk = h(r_1 || r_2 || n)$ in SPAK, and $\hat{sk} = h(Q_I || Q_R || \hat{n})$ in APAK, where r_1 , r_2 , Q_I and Q_R are all secret parameters. At the end of the protocol runs, the intruder will not be able to distinguish a fresh session key from any previously issued session keys.

Secure channel: Once both devices I and R authenticate each other, they establish a session key in which all subsequent data can be transmitted through the secure channel protected under the session key.

Forward secrecy: Forward secrecy ensures no previously established session key will be revealed if the link key gets compromised. We note that SPAK does not resist this threat, as on getting pw one can obtain r_1 as well as r_2 from the intercepted parameters. However, the APAK successfully resists this property, because the user passkey is get amplified with devices' system time and then both devices compute a common secret for authentication and session key establishment.

Mutual authentication: Both SPAK and APAK provide mutual authentication before accessing or sharing data.

Known key threat: Known key threat enables to compromise systems security by knowing a link key. Both SPAK and APAK resist this threat. Unlike SIG specification link key, our protocols authenticate the communicating devices dynamically in each session with the help of user's passkey. Consequently, knowing session key or intercepting other parameters cannot make any damages in our protocols.

Mutual key control: In SPAK, the session key is computed as $sk = h(r_1 || r_2 || n)$ and in APAK it is $\hat{sk} = h(Q_I || Q_R || \hat{n})$, where (r_1, Q_I) and (r_2, Q_R) chosen by I and R, respectively. Therefore, a device cannot be biased to a specific session key of its own choice.

Bluejacking: Bluejacking is the process of sending unsolicited messages (e.g., business card) to Bluetooth-enabled devices and gaining control of the device, if the receiver device accepts the message. This attack will not work in our protocols because of two-way authentication and message acceptance thereafter.

Replay Attack: Suppose the intruder intercepts a valid authentication request $\langle c_1, c_2, h_1, h_2 \rangle$ (or $\langle C_I, C_R, h_3, h_4 \rangle$) and tries to authenticate him/herself by replaying the same. The verification of this request fails, as each request is accompanying a nonce, n (or \hat{n}). If the intruder manipulates c_1 (or C_I) with a new random number and generates h_1 (or h_3) from that number, the recipient extracts a different number (as the user already picked a number which is masked with his/her pw (or apw) than the hashed value h_1 (or h_3) communicated by the intruder. Consequently, the intruder will not be succeeded in replay attack until s/he knows the user's passkey.

4.2 Performance Evaluation

In order to analyze the efficiency of our protocols, we compare the protocols with two related protocols [13], [20] for Bluetooth security based on passkey-authenticated key exchange. Performance of the protocols is shown in Table 2. From Table 2, it is easy to see that the computational cost of our protocols is

Table 2. Comparisons of the protocols t_e , t_s , t_h are computation time for modular exponentiation, symmetric key encryption, hash operation, respectively, and ex is the number of message exchange

	Computational cost		Communication cost	
	Device I	Device R	Device I	Device R
Wong et. al. [13]	$2t_e + 5t_s + 2t_h$	$2t_e + 5t_s + 2t_h$	2 ex	2 ex
Bluetooth SIG WG[20]	$2t_e + t_s + t_h$	$2t_e + t_s + t_h$	2 ex	2 ex
The SPAK	$6t_h$	$t_e + 7t_h$	2 ex	1 ex
The APAK	$6t_h$	$t_e + 7t_h$	2 ex	1 ex

significantly less than the protocols [13], [20]. Moreover, our protocol requires less bandwidth than others, as the message size and number of message exchange in our protocol are comparatively lesser than [13], [20]. Consequently, our protocols provide better efficiency in comparisons with [13], [20].

5 Conclusions

We discussed the security issues and implications of current Bluetooth technology. We then proposed two new protocols, SPAK and APAK, for device authentication and securing data transmissions. We have shown both protocols provide mutual authentication, establish secure session for data transmission, and achieves efficiency in comparisons with other protocols. The proposed protocols can be validated formally with some logic and/or process algebra based techniques, and the authors are in process to formalize its security as well.

References

1. Bluetooth Special Interest Group. Bluetooth Baseband Specification. Specifications of the Bluetooth System, 1.1 (2001)
2. IEEE P802.15 Working Group for WPANs, Cluster Tree Network (2001)
3. Bluetooth Special Interest Group. Bluetooth Core Specification plus Enhanced Data Rate. Specification of the Bluetooth System, 2.1 (2007)
4. Bluetooth Special Interest Group. Bluetooth Security WhitePaper, 1.0 (2002)
5. Bluetooth Special Interest Group. Bluetooth Security Specification. Specification of the Bluetooth System, 1.2 (2003)
6. Bellare, S.M., Meritt, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: Proc. of the IEEE Symposium on Research in Security and Privacy, pp. 72–74 (1992)
7. Bellare, S.M., Meritt, M.: Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise. In: Proc. of the ACM Conference on Computer and Communications Security, pp. 244–250 (1993)
8. Diffie, W., Hellman, M.E.: New directions in cryptography. *IEEE Transactions on Information Theory* IT-22(6), 644–654 (1976)
9. Jablon, D.: Strong Password-Only Authenticated Key Exchange. *Computer Communication Review* 26(5), 5–26 (1996)
10. Kwon, T.: Authentication and key agreement via memorable password. Contribution to the IEEE P1363 study group for Future PKC Standards (2000)
11. MacKenzie, P.: The PAK suite: Protocols for Password-Authenticated Key Exchange. DIMACS Technical Report 2002-46 (2002)
12. Shaked, Y., Wool, A.: Cracking the Bluetooth PIN. In: Proc. of the International Conference on Mobile systems, applications, and services, pp. 39–50. ACM Press, New York (2005)
13. Wong, F.L., Stajano, F., Clulow, J.: Repairing the Bluetooth pairing protocol. In: Proc. of the International Conference on Security Protocols. LNCS. Springer, Heidelberg (2005)
14. Jakobsson, M., Wetzel, S.: Security Weaknesses in Bluetooth. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020. Springer, Heidelberg (2001)
15. V. Kostakos. The privacy implications of Bluetooth. ArXiv (2008) (Retrieved on May 15, 2008), <http://arxiv.org/pdf/0804.3752>
16. Juels, A.: RFID Security and privacy: a research survey. *IEEE Journal On Selected Areas In Communications* 24(2), 381–394 (2006)
17. Frankel, S., Kelly, S.: The HMAC-SHA-256-128 Algorithm and Its Use With IPsec. draft-ietf-ipsec-ciph-sha-256-01.txt (2002)
18. Koblitz, N.: Elliptic curve cryptosystems. *Mathematics of Computation* 48, 203–209 (1987)
19. Massey, J., Khachatrian, G., Kuregian, M.: Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard. In: Proc. of the AES Candidate Conference (1998)
20. Bluetooth Special Interest Group. Simple Pairing Whitepaper. Core Specification Working Group, V10r00 (2006)

A Verification Framework for Temporal RBAC with Role Hierarchy (Short Paper)

Samrat Mondal and Shamik Sural

School of Information Technology
Indian Institute of Technology, Kharagpur
{samratm,shamik.sural}@sit.iitkgp.ernet.in

Abstract. In this paper a Timed Automata (TA) based verification framework is proposed for Temporal RBAC. Roles, users, permissions - three basic components of RBAC are modeled using TA. These components interact with each other through channel synchronization. A parallel composition of TA is used to construct the complete system. Temporal constraints on roles, user-role assignments and role-permission assignments are conveniently expressed in this representation. Furthermore, both role hierarchy and separation of duty (SoD) have been incorporated in the proposed framework. Security properties are specified using Computation Tree Logic (CTL) and verified by model checking.

Keywords: Temporal RBAC, Timed Automata, Verification, Model Checking, CTL.

1 Introduction

In Role Based Access Control (RBAC) [1], roles, users and permissions are the primary components of a system. In real application scenarios, it is often required to give access depending on when the request is being made or where the subject or object is. So temporal and spatial aspects are to be considered for performing authorization. Bertino and Bonatti have proposed a model for extending RBAC in the temporal domain, which is named as TRBAC [2]. It supports temporal constraints on role enabling and disabling. Later a new version of TRBAC is proposed, which is known as the Generalized Temporal Role Based Access Control (GTRBAC) model [3]. It is capable of expressing temporal constraints on user-role assignments and role-permission assignments as well. Also it makes a distinction between role enabling and activation.

With the development of different access control models, it is also required to provide a framework for formal verification of properties of systems implementing such models. Ahmed and Tripathi [4] have proposed a model checking mechanism for verification of security requirements in role based Computer Supported Cooperative Work (CSCW) systems. Li and Tripunitara [5] have worked on the security analysis of two variants of RBAC model— AATU (Assignment And Trusted Users) and AAR (Assignment And Revocation). Jha et al. have

compared the use of model checking and first order logic programming for the security analysis of RBAC system and concluded that model checking is a promising approach for security analysis [6].

Analysis of Temporal RBAC is to be done differently due to the presence of temporal constraints in the system. Bertino and Bonatti [2] have provided a polynomial time algorithm to verify whether specifications in TRBAC are free from ambiguities. Shafiq et al. have studied a Petri-Net based framework for verifying the correctness of event-driven RBAC policies in real time systems [7]. It considers the cardinality and separation of duty constraints. But it does not talk about temporal role hierarchy. A few other systems where time is considered to be a critical factor have also been analyzed by the researchers. Alur et al. [8] have used model checking for the analysis of real time systems. Furfaro and Nigro [9] have used timed automata (TA) for temporal verification of real time state machines. Recently, we [10] have suggested timed automata based approach for the security analysis of temporal RBAC but it only deals with the temporal constraints on user role assignment relation. Here we propose a verification framework for temporal RBAC which can handle temporal constraints on user-role assignment, role-permission assignment, role hierarchy and Separation of Duty (SoD).

2 Proposed Approach

The main objective here is to map the Temporal RBAC system to a TA based system. The full descriptions of a timed automaton can be found in [11]. We have considered the general framework of GTRBAC model [3] and we try to address the different temporal constraints on role enabling-disabling, role activation-deactivation, permission assignment, user assignment, role hierarchy and SoD. The mapping scheme is dependent on a given set of policies. Some desirable properties based on the given set of policies are used to verify the model. For verification of properties, we have used a TA based tool named Uppaal [12].

2.1 Algorithms for Constructing Timed Automata

Here we propose four different algorithms to construct TA. *Algorithm 1* gives the steps to construct role TA. The algorithm uses standard terminology of an RBAC system. It generates a timed automaton corresponding to each role. *Algorithm 2* illustrates the steps to construct controller timed automaton. It is dependent on a global clock variable. Synchronization actions are initiated from this automaton and the appropriate role gets enabled or disabled. *Algorithm 3* depicts the steps to construct the user TA. It searches *UA* relation to determine which user is associated with which role. *Algorithm 4* is used to build permission TA. For each permission an automaton is constructed.

2.2 Mapping from Temporal RBAC to Timed Automata

Here we explain how the algorithms work in different situations. A role is enabled if a user can acquire the permissions associated with it. Role enabled is a state

Algorithm 1. Constructing Role Timed Automata

```

for each role  $r$  such that  $r \in R$  do
  construct a timed automaton  $T_r = \langle L, l_0, C, A, E, I \rangle$  such that-
   $L = \{Disabled, Enabled\}$ ;  $l_0 = Disabled$ ;  $A = \{enable_{\mathcal{L}}, disable_{\mathcal{L}}\}$ ;
   $t \subseteq C$ ; where  $C$  is a finite set of clocks
  Let  $g$  be a guard which is a conjunction of boolean expressions involving clocks or
  some other variables
   $E = \{Disabled \xrightarrow{g, enable_{\mathcal{L}}?, t} Enabled, Enabled \xrightarrow{g, disable_{\mathcal{L}}?, t} Disabled\}$ ;
   $usercount = 0$ 
  for each user  $u$  such that  $(u \in U)$  and  $((r, u) \in UA)$  do
     $usercount++$ ;
  end for
  if  $usercount = 1$  then
     $L = L \cup \{Activated\}$ ;  $A = A \cup \{assigned_{\mathcal{L}}, unassigned_{\mathcal{L}}\}$ ;
     $E = E \cup \{Enabled \xrightarrow{g, assigned_{\mathcal{L}}!, t} Activated, Activated \xrightarrow{g, unassigned_{\mathcal{L}}!, t} Enabled\}$ ;
  else if  $usercount > 1$  then
     $L = L \cup \{Activated\}$ ;  $A = A \cup \{assigned_{\mathcal{L}}, unassigned_{\mathcal{L}}\}$ ;
     $E = E \cup \{Enabled \xrightarrow{g, assigned_{\mathcal{L}}!, t} Activated, Activated \xrightarrow{g, unassigned_{\mathcal{L}}!, t} Enabled, Activated \xrightarrow{g, assigned_{\mathcal{L}}!, t} Activated, Activated \xrightarrow{g, unassigned_{\mathcal{L}}!, t} Activated\}$ ;
  else
    no operation;
  end if
  for each permission  $p$  such that  $(p \in P)$  and  $((r, p) \in PA)$  do
     $L = L \cup \{p\}$ ;  $A = A \cup \{allow_{\mathcal{L}}, disallow_{\mathcal{L}}\}$ ;
     $E = E \cup \{Activated \xrightarrow{g, allow_{\mathcal{L}}!, t} p, p \xrightarrow{g, disallow_{\mathcal{L}}!, t} Activated\}$ ;
  end for
  for each role  $r'$  such that  $(r' \in R)$  and  $((r \succeq r') \in RH)$  do
    {here  $r$  is a senior role}
     $L = L \cup \{Activated_{\mathcal{L}}'\}$ ;
     $A = A \cup \{activate_{\mathcal{L}}', deactivate_{\mathcal{L}}'\}$ ;
     $E = E \cup \{Activated \xrightarrow{g, activate_{\mathcal{L}}'!, t} Activated_{\mathcal{L}}', Activated_{\mathcal{L}}' \xrightarrow{g, deactivate_{\mathcal{L}}'!, t} Activated\}$ ;
  end for
  for each role  $r'$  such that  $(r' \in R)$  and  $((r' \succeq r) \in RH)$  do
    {here  $r$  is a junior role}
     $L = L \cup \{ActivatedByr'\}$ ;  $A = A \cup \{activate_{\mathcal{L}}', deactivate_{\mathcal{L}}'\}$ ;
     $E = E \cup \{Enabled \xrightarrow{g, activate_{\mathcal{L}}'?, t} ActivatedByr', ActivatedByr' \xrightarrow{g, deactivate_{\mathcal{L}}'?, t} Activated\}$ ;
    for each permission  $p$  such that  $(p \in P)$  and  $((r', p) \in PA)$  do
       $L = L \cup \{pH\}$ ;
       $A = A \cup \{allow_{\mathcal{L}}, disallow_{\mathcal{L}}\}$ ;
       $E = E \cup \{ActivatedByr' \xrightarrow{g, allow_{\mathcal{L}}!, t} pH, pH \xrightarrow{g, disallow_{\mathcal{L}}!, t} ActivatedByr'\}$ ;
    end for
  end for
end for

```

Algorithm 2. Constructing Controller Timed Automaton

```

construct a timed automaton say  $T_c = \langle L, l_0, C, A, E, I \rangle$  such that
 $L = \{l_0\}$ ;
 $E = \phi$ ;
 $X = \phi$ ; {X is used to store the role enabling-disabling time instances}
for each role  $r$  such that  $r \in R$  do
   $X = X \cup \{x_r, x'_r\}$ ; here  $x_r$  is a role enabling time instance and  $x'_r$  is a role disabling
  time instance for role  $r$ .
end for
arrange the elements of  $X$  in ascending order.
for each  $x$  such that  $(x \in X)$  do
  if  $x$  is a role enabling instance of role  $r$  then
     $L = L \cup \{l_r\}$ ;  $E = E \cup \{l \xrightarrow{g, enable\_r^!, t} l_r\}$ ;  $l = l_r$ ;
  else
     $L = L \cup \{l'_r\}$ ;  $E = E \cup \{l \xrightarrow{g, disable\_r^!, t} l'_r\}$ ;  $l = l'_r$ ;
  end if
end for
 $E = E \cup \{l \xrightarrow{t=0} l_0\}$ ;

```

Algorithm 3. Constructing User Timed Automata

```

for each user  $u$  such that  $(u \in U)$  do
   $flag = 0$ ; {To keep track whether a user is associated with more than one role}
  for each role  $r$  such that  $(r \in R)$  and  $((r, u) \in UA)$  do
    if  $flag = 0$  then
      construct a timed automaton say  $T_u = \langle L, l_0, C, A, E, I \rangle$ .
       $L = L \cup \{Idle, Active\_r\}$ ;
       $E = E \cup \{Idle \xrightarrow{g, assigned\_r^?, t} Active\_r, Active\_r \xrightarrow{g, unassigned\_r^?, t} Idle\}$ ;
       $flag = 1$ ;
    else
       $L = L \cup \{Active\_r\}$ ;
       $E = E \cup \{Idle \xrightarrow{g, assigned\_r^?, t} Active\_r, Active\_r \xrightarrow{g, unassigned\_r^?, t} Idle\}$ ;
    end if
  end for
end for

```

Algorithm 4. Constructing Permission Timed Automata

```

for each permission  $p$  such that  $(p \in P)$  do
  construct a timed automaton say  $T_p = \langle L, l_0, C, A, E, I \rangle$  where
   $L = L \cup \{Inactive, Active\}$ ;  $l_0 = Inactive$ ;
   $E = E \cup \{Inactive \xrightarrow{g, allow\_p^?, t} Active, Active \xrightarrow{g, disallow\_p^?, t} Inactive\}$ ;
end for

```

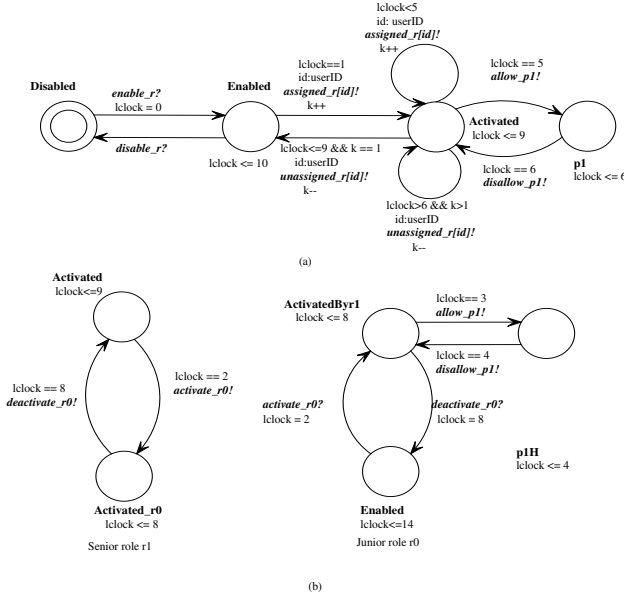


Fig. 1. (a) Example of a Role Timed Automaton (b) Senior Role r_1 activates Junior Role r_0

when it is ready for user assignment but no user is assigned yet [13]. A role is disabled if it is not ready for user assignment. An enabled role becomes active when a user acquires the permissions associated with it. So with the first user assignment an enabled role goes to the activated state. Any subsequent user assignment operation will not change its state. A role is deactivated when all the users are unassigned from that role. When the roles are in active state, the users assigned to the role can access some permissions. This is represented by permission assignment relation, which may also depend on some temporal constraints. In Figure 1 (a), we show a role timed automaton which incorporates the temporal constraints on role enabling, role activation and on permission assignment relation.

In some situations an active role (say a senior role) can activate another role (junior role). Joshi et al. [13] have shown the use of different types of temporal role hierarchies. In this paper, we represent the role hierarchy as $r_1 \succeq r_0$ where r_0 and r_1 are two roles such that r_1 is senior to r_0 . This means that the users associated with role r_1 can access the permissions associated with role r_0 and the users of r_1 can also activate role r_0 . In Figure 1 (b) we show how this hierarchical relation is represented using two role TA.

User timed automaton is primarily represented by two locations - *Idle* and *Active*. On receiving a synchronization action from a role timed automaton, it goes to *Active* location. This represents that the user is assigned to a specific role. With another synchronization action it may go to the *Idle* location. Separation of Duty constraint can be addressed using the user timed automaton. If a user u

can be assigned to two roles, $r0$ and $r1$ but not at the same time, then instead of one *Active* location, it will have two locations - *Active0* and *Active1*, representing the user assignment to role $r0$ and $r1$ respectively. If a user is assigned to the role $r0$ then it will move to *Active0* location; otherwise, it will move to *Active1*.

To model a complete Temporal RBAC system, another automaton is required to control role enabling and disabling using a global clock. This automaton is known as Controller Automaton. A global clock variable is used in each transition to specify the temporal constraints on role enabling and disabling. Also each transition is labeled with a synchronization action such as *enable_r0!* or *disable_r0!*. The role timed automaton with corresponding receiving synchronization label gets enabled or disabled at appropriate time. It may be noted that periodic time expression as suggested in [2] can also be handled by this automaton. However, due to page restrictions it is kept outside the scope of this paper.

3 Property Specification, Verification and Analysis

For property specification, we have used the branching notion of time – an infinite tree of states. Computation Tree Logic (CTL) supports such branching notion. CTL is sufficiently expressive for the formulation of properties such as *safety*, *liveness* and *reachability* properties [10]. Let us assume that there are three roles – $r0$, $r1$ and $r2$ where $r1 \succeq r2$ and $r0$ is enabled at time $t = 1$ and disabled at time $t = 16$. Let the permissions associated with the roles (i.e., the PA relation) be represented by $PA = \{ \langle r0, p0 \rangle, \langle r1, p1 \rangle, \langle r2, p2 \rangle \}$. Let there be nine users – $u0, u1, \dots, u8$. Among them users $u0, u1, \dots, u6$ can take the role $r0$, similarly users $u6, \dots, u8$ can take the role $r1$. So only the user $u6$ can be assigned to both the roles. Here we put one SoD constraint: whenever user $u6$ is assigned to role $r0$ then he will not be assigned to role $r1$ and vice versa. Now we specify some CTL properties desirable for this system.

- **Property A:** $A[] u0.Active \text{ imply } (r0.Activated \parallel r0.p0)$ – As permission $p0$ is only associated with $r0$, this indicates that in all possible states if $u0$ is active then role $r0$ is also active.
- **Property B:** $A[] r0.Enabled \text{ imply } t \geq 1 \ \&\& \ t \leq 16$ – This means that in all possible states $r0$ is enabled in between time 1 and 16.
- **Property C:** $A[] u6.Active0 \text{ imply not } u6.Active1$ – This represents that in all possible states if user $u6$ is assigned to role $r0$ then it cannot be assigned to role $r1$ at the same time. It is a typical SoD constraint.
- **Property D:** $A[] r2.ActivatedByr1 \text{ imply } r1.Activated.r2$ – This expresses that if role $r2$ is activated by $r1$ then $r1$ must be in activated state also. This shows a role hierarchical behavior.
- **Property E:** $A[] \text{ not deadlock}$ – This checks that in all paths and in all states whether there is any possibility of deadlock or not.
- **Property F:** $E <> r0.Activated$ – This asks whether $r0$ role will eventually be activated at some time.

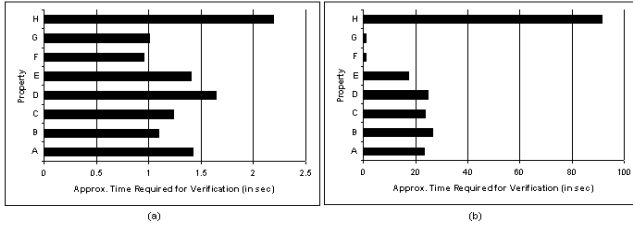


Fig. 2. Two Cases of Verification Time Analysis of Properties A - H

- **Property G:** $E <> p0.Active$ – This asks whether permission $p0$ will be accessible at some time.
- **Property H:** $r0.Enabled \rightarrow r0.Disabled$ – This asks whether an enabled role $r0$ will eventually be disabled at some time.

In Figure 2 we show the verification time required for each property. Two cases are considered. Figure 2(a) shows the time requirement for a system with three roles, nine users and three permissions. In Figure 2(b), the result is shown with three roles, fifteen users and three permissions. In the second case, six new users have been assigned to role $r2$. With this increase, the time required for the verification of properties A, B, C, D, E and H has increased drastically. This can be explained as follows. To verify properties A, B, C, D and E, all the possible states in all the paths are required to be explored. With increase in the number of user automata, the size of the state space grows automatically. So the state space search consumes a lot of time to explore all the states separately before concluding that the property is satisfiable or not. For the liveness property H, verification time is dependent on the two states under consideration. In property H, if we replace $r0.Enabled$ and $r0.Disabled$ with $u0.Active$ and $u0.Idle$ then we get much faster response time. This is because, in the latter case, less number of states is involved in between the states under consideration. Finally, the time required for reachability properties such as F and G is almost unchanged with increase of user automata. They look for a single state in the state space and whenever that state is reached the verification process can conclude that the property is satisfiable or not.

4 Conclusions

We have discussed how to represent a Temporal RBAC system using TA. Using the algorithms of Section 2.1, any Temporal RBAC model with different temporal constraints on role enabling-disabling, role activation-deactivation, permission assignment and user role assignment can be mapped to a TA based system. Even temporal constraints on role hierarchy and SoD constraints can be addressed using this mapping scheme. This helps to express a wide range of security policies. The verification framework can be useful in designing a correct and safe system with respect to a given set of security policies.

Acknowledgement

This work is partially supported by a research grant from the Department of Science and Technology, Government of India, under Grant No. SR/S3/EECE/082/2007.

References

1. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role based access control models. *IEEE Computer* 29(2), 38–47 (1996)
2. Bertino, E., Bonatti, P.A.: TRBAC: A temporal role based access control model. *ACM Transactions on Information and System Security* 4(3), 191–223 (2001)
3. Joshi, J.B.D., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering* 17(1), 4–23 (2005)
4. Ahmed, T., Tripathi, A.R.: Static verification of security requirements in role based CSCW systems. In: 8th ACM Symposium on Access Control Models and Technologies, Italy, pp. 196–203 (June 2003)
5. Li, N., Tripunitara, M.V.: Security analysis in role based access control. *ACM Transactions on Information System Security* 9(4), 391–420 (2006)
6. Jha, S., Li, N., Tripunitara, M., Wang, Q., Winsborough, W.: Towards formal verification of role based access control policies. *IEEE Transactions on Dependable and Secure Computing* (to appear, 2008)
7. Shafiq, B., Masood, A., Joshi, J., Ghafoor, A.: A role based access control policy verification framework for real time systems. In: 10th IEEE International Workshop on Object Oriented Real Time Dependable Systems, USA, pp. 13–20 (2005)
8. Alur, R., Courcoubetis, C., Dill, D.L.: Model checking for real time systems. In: 5th Symposium on Logic in Computer Science, USA, pp. 414–425 (1990)
9. Furfaro, A., Nigro, L.: Temporal verification of communicating real time state machines using Uppaal. In: IEEE International Conference on Industrial Technology, Slovenia, pp. 399–404 (2003)
10. Mondal, S., Sural, S.: Security analysis of Temporal-RBAC using timed automata. In: 4th International Conference on Information Assurance and Security, Italy, pp. 37–40 (2008)
11. Alur, R., Dill, D.: A theory of timed automata. *Theoretical Computer Science* 126(2), 183–235 (1994)
12. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time, Italy, pp. 200–236 (2004)
13. Joshi, J.B.D., Bertino, E., Ghafoor, A.: Hybrid role hierarchy for generalized temporal role based access control model. In: 26th Annual International Computer Software and Application Conference, England, pp. 951–956 (2002)

Computing on Encrypted Data

(Extended Abstract)

Amit Sahai*

University of California, Los Angeles
sahai@cs.ucla.edu

Abstract. Encryption secures our stored data but seems to make it inert. Can we process encrypted data without having to decrypt it first? Answers to this fundamental question give rise to a wide variety of applications. Here, we explore this question in a number of settings, focusing on how interaction and secure hardware can help us compute on encrypted data, and what can be done if we have neither interaction nor secure hardware at our disposal.

1 Introduction

The increased frequency of cyber-attacks as well as governmental regulations, such as HIPPA and GLBA in the United States, are driving enterprises to encrypt more and more of their data. In the near future, it is expected that encryption will be ubiquitous and the majority of enterprise data will be stored encrypted.

While protecting data with encryption provides clear benefits, it also has some significant drawbacks. Unlike cleartext data which can be edited and searched, encrypted data seems to support none of these operations. It is commonly believed that encrypted data cannot be manipulated without first decrypting it. This often leads to complex key management where multiple entities are given access to the decryption keys. The end result is reduced security and increased cost. To give some examples, consider a database that stores encrypted transactions. Locating all transactions within a certain range of dates is believed to be impossible without giving the database access to decryption keys. More generally, once a database column is encrypted it is typically impossible to issue general queries on that column. The same limitations apply to systems managing encrypted file servers and email. This leads to the following fundamental question:

Can we process encrypted data without having to decrypt it first?

Answers to this fundamental question give rise to a wide variety of applications. Here, we explore this question in a number of settings, focusing on how

* Research supported in part from NSF grants 0627781, 0716389, 0456717, and 0205594, a subgrant from SRI as part of the Army Cyber-TA program, an equipment grant from Intel, an Alfred P. Sloan Foundation Fellowship, and an Okawa Foundation Research Grant.

interaction and secure hardware can help us compute on encrypted data, and briefly discussing our recent work in this area. We conclude with some musings about what can be done if we have neither interaction nor secure hardware at our disposal.

2 Interaction

Somewhat surprisingly, over two decades ago a powerful positive answer was given to this fundamental question by making use of *interaction*, which led to the flourishing field of secure two-party and multiparty computation [24,11,3,6]. Secure multi-party computation allows users that hold different sets of secret data to collaborate and analyze all their data together, in such a way that no user learns anything about anyone else’s secret data except for whatever is revealed by the output of the analysis. Thus, even though the secrecy of other users’ data is guaranteed through encryption methods, arbitrarily complex functions of all the data can be jointly computed by the users.

The key to this is interaction. The intuition behind what makes this possible is the fact that the holder of the data in unencrypted form is available to “answer questions” from the other users, who only hold the data in some encrypted form. Of course, the challenge is the following: how can we allow others to process our data when we are only willing to answer questions that don’t reveal anything about our data? A number of fascinating and sophisticated techniques have been discovered to meet this challenge.

Secure two-party and multiparty computation remains a vast and exciting field. We will briefly mention here two works regarding secure computation that we were recently involved in. In recent joint work with Ishai and Prabhakaran [17], we show how to achieve a very high level of efficiency for secure two-party and multi-party computation – where the communication overhead of a secure two-party computation protocol for a function F is only a fixed constant factor larger than the circuit size of F . This is possible under standard and minimal computational assumptions. By making somewhat stronger and more non-standard intractability assumptions, in another recent work, joint with Ishai, Kushilevitz, and Ostrovsky [16], we gave the first steps towards achieving secure two-party computation with *constant computational overhead*. By this, we mean that the amount of computation (not just communication) necessary to securely compute some function F is only a fixed constant factor larger than the circuit size of F .

3 Secure Tamper-Proof Hardware

Another resource with a much more obvious application to our question is secure tamper-proof hardware. By “secure tamper-proof hardware,” we mean a device that implements some functionality in a “black-box” manner, so that the holder of the device can only query the device with some input and receive some output from the device. As such, it is quite intuitive that such devices could allow

for the processing of encrypted data. In particular, the device could have the decryption key built into it, so the device would take as input some encrypted values, decrypt them and process them, and reencrypt the result. Indeed, Goldreich and Ostrovsky [12] showed that using such devices, one can efficiently run “encrypted programs”. That is, the entire program to be executed is available only in encrypted form, and therefore nothing about the program is revealed except for roughly how long it runs, and what its outputs are on given inputs. This goal is often called program obfuscation. Even with trusted hardware, this goal is non-trivial to achieve because of the problem of “replay” attacks, where the result of some intermediate computation is maliciously reused at some other point of the program where it should not be used. Goldreich and Ostrovsky [12] solve this problem using secure hardware tokens that maintain state information. Recently, the work of Goyal and Venkatesan [15] achieved the same result using *stateless* hardware tokens, where more sophisticated cryptographic techniques are used to defend against the problem of replay attacks.

Perhaps the most natural question to ask in the context of secure tamper-proof hardware is whether it is reasonable to assume that we could really have secure tamper-proof hardware to begin with? This question has led to two very interesting recent lines of work:

- **Can we implement secure tamper-proof hardware out of insecure components?** In joint works with Ishai, Prabhakaran, and Wagner [19,18], we considered this question, and showed how to make secure hardware devices for implementing any function that can tolerate specific side channel attacks (namely bit probes) and specific tampering attacks (namely tampering with the values on individual wires inside the circuit). Interestingly, the techniques we needed to accomplish these goals are closely related to techniques from secure two-party and multiparty computation.

In other related work, Gennaro et al. [9] considered the question of whether a general secure hardware device could be separated into a readable but tamper proof part, and a separate unreadable but tamperable part. They gave a number of positive results for various specific functions.

- **What can we accomplish with a very simple secure hardware devices?** The recent work of Goldwasser, Kalai, and Rothblum [13] introduce a very simple hardware device that they call a “one-time memory” device (we call such devices “OT tokens” because they can be thought of as implementing the oblivious transfer [21,8] function). This device has built into it two secret strings s_0 and s_1 , takes a single bit t as input, outputs s_t and then self-destructs (i.e. becomes useless). Using a collection of such simple devices, and assuming that one-way functions exist, they show how to achieve a relaxation of program obfuscation, where the encrypted program may only be run *once* – a notion which they call *one-time programs*. (Note that this notion can be trivially generalized to allow the program to run a pre-specified k times.)

In recent joint work with Goyal, Ishai, and Wadia [14], we extend their results in three ways, two major and one minor. One minor improvement that

we offer is that our secure devices can be even simpler – a “bit OT token” where the two strings s_0 and s_1 are replaced with just two bits b_0 and b_1 , and the output is just the bit b_t . Our two major improvements are: (1) We strengthen the notion of one-time programs to allow the manufacturer of the program to specify guarantees about the encrypted program that the evaluator of the program can verify at run-time (*i.e.* when the owner of the tokens, Alice, runs the program on her secret input x_A , she could be assured that the program will simply compute $f(x_A, x_B)$, where f is a specific function known to Alice, and x_B is a secret input of the manufacturer). We call this stronger notion one-time programs with security against malicious sender. (2) We achieve this stronger notion of one-time programs *unconditionally*, without needing to make any unproven assumptions, such as the existence of one-way functions.

4 The “Plain” Model

Unfortunately, much less is known about computing on encrypted data in the “plain model,” where encrypted data is given to us, and we must non-interactively process it without the assistance of any secure hardware. Indeed, in joint work with Barak, Goldreich, Impagliazzo, Rudich, Vadhan, and Yang [2], we showed that in this situation the goal of program obfuscation (encrypted programs) is in general impossible to achieve, even when the programs to be obfuscated/encrypted are fairly simple. Nevertheless, this area is filled with a number of fascinating open questions.

One of the central open problems in cryptography today, called *doubly-homomorphic encryption*, was first posed by Rivest et al. [22] almost 30 years ago. The problem can be stated succinctly as follows. Let E be a (semantically) secure encryption system where plaintexts are integers in $\{0, \dots, n\}$ for some n . The system is said to be doubly homomorphic if given the encryption of two plaintexts $E_k(x)$ and $E_k(y)$ anyone can construct a new independent encryption of $E_k(x + y)$ and $E_k(x \cdot y)$, without knowledge of x or y . This property enables arbitrary computations on encrypted data, where the output and all intermediate computations remain in encrypted form. More precisely, E is doubly homomorphic if there are two efficient algorithms A_+ and A_* such that

$$\begin{aligned} A_+(E_k(x_1), E_k(x_2)) &=_p (E_k(\mathbf{x}_1 + \mathbf{x}_2), E_k(x_1), E_k(x_2)), \quad \text{and} \\ A_*(E_k(x_1), E_k(x_2)) &=_p (E_k(\mathbf{x}_1 \cdot \mathbf{x}_2), E_k(x_1), E_k(x_2)) \end{aligned}$$

where $=_p$ denotes indistinguishability of distributions.

Virtually nothing is known about the existence of such an E , other than some very weak impossibility results [5]. The **major open problem** is to build a semantically secure public-key encryption that is doubly homomorphic where, furthermore, algorithms A_+ and A_* are efficient and practical. Such a system will enable a host of magical applications, such as:

- **Searching.** In principle, E will enable a very rich set of search queries on encrypted data, as discussed in the previous sections. It will also enable very general searches with encrypted queries.
- **Minimally interactive distributed data-mining and secure computation.** E will enable simple minimally-interactive data-mining of databases distributed across multiple competing entities (e.g., multiple airlines, hospitals, or governments). The goal is to ensure that nothing other than the data-mining results is revealed. Currently, this requires highly interactive protocols based on secure computation techniques.

Several existing public-key systems, such as ElGamal [7] and Pallier [20], are singly-homomorphic — they support only one homomorphic operation. Previous attempts [23] at building doubly-homomorphic systems only applied to boolean operations and doubled the ciphertext size at every step. As a result, one could only perform few boolean operations before the ciphertext size became unmanageable. Recent work of Boneh, Goh, and Nissim [4] use techniques from elliptic curves to construct a system that allows for an arbitrary number of additions, and one multiplication. Surprisingly, this small additional homomorphic property already leads to a number of exciting new constructions. Thus, even seemingly minor progress on this fundamental open question can lead to significant payoffs.

5 Conclusions

The question of computing on encrypted data has spawned countless fascinating techniques as well as deep open problems. It remains a driving force behind much of the most exciting research in cryptography today.

Acknowledgements

We thank Dan Boneh, Vipul Goyal, Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, Ramarathnam Venkatesan, Akshay Wadia, David Wagner, and Brent Waters for several insightful conversations on these topics over the course of several collaborations.

References

1. Proc. 20th STOC. ACM, New York (1988)
2. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139. Springer, Heidelberg (2001)
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proc. 20th STOC [1], pp. 1–10

4. Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-dnf formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–342. Springer, Heidelberg (2005)
5. Boneh, D., Lipton, D.: Black box fields. In: Kobitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109. Springer, Heidelberg (1996)
6. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: Proc. 20th STOC [1], pp. 11–19
7. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
8. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM 28(6), 637–647 (1985)
9. Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., Rabin, T.: Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 258–277. Springer, Heidelberg (2004)
10. Goldreich, O.: Foundations of Cryptography: Basic Applications. Cambridge University Press, Cambridge (2004)
11. Goldreich, O., Micali, S., Wigderson, A.: How to play ANY mental game. In: ACM (ed.) Proc. 19th STOC, pp. 218–229. ACM, New York (1987); See [10, Chap. 7] for more details
12. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM 43(3), 431–473 (1996)
13. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)
14. Goyal, V., Ishai, Y., Sahai, A., Wadia, A.: Cryptography from tamper-proof hardware, revisited (manuscript, 2008)
15. Goyal, V., Venkatesan, R.: On obfuscation complete oracles (manuscript, 2008)
16. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: STOC, pp. 433–442. ACM, New York (2008)
17. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)
18. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.: Private circuits II: Keeping secrets in tamperable circuits. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 308–327. Springer, Heidelberg (2006)
19. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
20. Pallier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
21. Rabin, M.: How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory (1981)
22. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. Foundations of Secure Computation (1978)
23. Sander, T., Young, A., Yung, M.: Non-interactive CryptoComputing for NC^1 . In: Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS), New York, NY, USA, October 1999, pp. 554–567. IEEE Computer Society Press, Los Alamitos (1999)
24. Yao, A.C.: How to generate and exchange secrets. In: Proc. 27th FOCS, pp. 162–167. IEEE, Los Alamitos (1986)

Identification of Cryptographically Strong and Weak Pseudorandom Bit Generators

Neelam Verma, Prasanna R. Mishra, and Gireesh Pandey

Scientific Analysis Group, DRDO, Delhi-110054, India
{neelamverma123, prasanna.r.mishra}@gmail.com,
gp2_bhu@yahoo.com

Abstract. Cryptographically strong Psuedo Random Bit Generators (PRBGs) are used extensively for confidentiality of information. If these PRBGs are weak then the security of information is at stack. In this paper we are giving a model which will use confidence interval and multidimensional scaling for their identification. This statistical model can be used to estimate the strength of a new PRBGs.

Keywords: PRBG, BBS generator, Geffe generator, Feature extraction, Confidence Interval Estimation, Multidimensional Scaling.

1 Introduction

A PRBG [1] is a deterministic algorithm which, given a k -bit seed, outputs a bit-sequence of length $l \gg k$ which appears to be random.

We have considered one cryptographically strong PRBG (Blum-Blum-Shub with $p = 65371, q = 65587$) and a weak PRBG (Geffe generator with registers of length 17, 19 and 23), described in [1]. We have devised a statistical model for their identification and allocation of new PRBG to a class of strong PRBG or weak PRBG.

Binary sequences from PRBGs are converted into real values by feature extraction techniques [2]. These features are normalized and confidence interval(C.I.) for each generator are estimated at various levels of significance [3]. The multidimensional scaling (MDS) [4] of high dimensional feature set to two dimensional features has helped in graphical display of the two generators. The quantitative discrimination has been given by Minimum Distance Classifier (MDC) [5].

2 Identification Using C.I. Estimation and MDS

A set of M sequences from each of the PRBGs is taken. We divide each sequence into blocks of 32-bit and 64-bit and evaluate four measures described in [2] for each block and then their means were taken as the the four feature of the sequence. The M values of each of four features are normalized between 0 and 1. We have used two techniques viz., Confidence Interval Estimation in which C.I. of the means of both the generators at various levels of significance are found

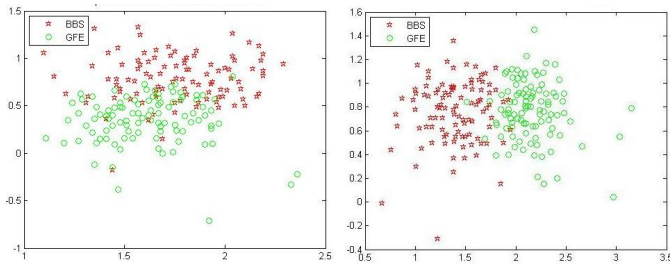


Fig. 1. Graphical display of 32-bit and 64-bit transformed 2D data

and Multidimensional Scaling, where four dimensional features are transformed to two dimensional features by a non-linear mapping such that the inter-vector distance or dissimilarities in four dimensional space approximate the inter-vector distance or the dissimilarity in two dimensional space. The graphical display of two dimensional data for 32-bit block and 64-bit block are shown in Fig. 1¹. For quantification of above figure MDC is used for both the generators using various learning and test sets.

3 Conclusion

It has been concluded that in our model, C.I. of means of four features in weak and strong generators are significantly different. Transformation of four dimensional features to two dimensional features through MDS results in almost non-overlapping clusters of weak and strong generator as visible in the graphical display. The clusters are quantified to more than 90% of identification through MDC. To estimate the strength of new PRBG, above model can be used by taking BBS as the reference generator.

References

1. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
2. Venimadhavan, C.E.: Statistical Techniques for Cryptanalysis and Steganalysis. In: Workshop on steganography, C-DAC, Kolkata (2004)
3. Neter, J., Wasserman, W., Whitmal, G.A.: Applied Statistics. Allyn and Bacon, Boston, Sydney, Toronto (1993)
4. Chien, Y.-T.: Interactive pattern recognition. Marcel-Dekker INC, New York (1978)
5. Theodoridis, S., Kautroumbas, K.: Pattern recognition. Academic Press, NY (1999)

¹ Extensive experiments with C.I., MDS and MDC for two sized block have been done for various data sets whose results are not shown here because of space constraint.

Proxy Re-signature Schemes

N.R. Sunitha¹ and B.B. Amberker²

¹ Department of Computer Science & Engg., Siddaganga Institute of Technology,
Tumkur, Karnataka, India

² Department of Computer Science & Engg., National Institute of Technology,
Warangal, Andhra Pradesh, India

Abstract. In a proxy re-signature scheme a semi-trusted proxy acts as a translator between Alice and Bob to translate a signature from Alice into a signature from Bob on the same message. In the 12th ACM CCS 2005, Ateniese and Hohenberger presented two secure proxy re-signature schemes based on bilinear maps and left as open challenges the design of multi-use unidirectional systems and determining whether or not a proxy re-signature scheme can be built that translates one type of signature scheme to another.

To address the first open problem, Benoit Libert and Damien Vergnaud have given one solution based on bilinear groups. We propose another solution for the same problem using the property of forward-security. With a minor change in resigning key, we can make the scheme to behave as a multi-use bidirectional scheme. To address the second open problem, we have come up with schemes to convert Schnorr/ElGamal Signatures to RSA signature.

Keywords: Signature conversion, Proxy re-signature, Proxy Signature, Proxy revocation, Proxy key.

1 Introduction

In Eurocrypt 98, Blaze, Bleumer, and Strauss (BBS)[3] proposed proxy re-signatures. Since the BBS proposal, the proxy re-signature primitive has been largely ignored. Ateniese and Hohenberger [1] re-opened the discussion of proxy re-signature and gave provably secure proxy re-signature constructions from bilinear maps. Nevertheless, they left open the following two problems: (1) The design of multi-use unidirectional proxy re-signature scheme. (2) Determining whether or not proxy re-signature scheme can be built that translate from one type of signature scheme to another i.e. like a scheme that translates Alice's Schnorr signatures into Bob's RSA based ones.

2 Solution to Open Problems in Proxy Re-signatures

To address the first open problem *i.e* for the design of multi-use (the translation of signatures can be performed in sequence and multiple times by distinct proxies) unidirectional (the re-signature key allows the proxy to turn Alice's signatures into Bob's, but not Bob's into Alice's) systems, Benoit Libert and Damien

Vergnaud [2] have given one solution based on bilinear groups. We propose another solution for the same problem using the property of forward-security [4]. Our forward-secure proxy re-signature scheme which is based on the hardness of factoring translates one person's signature to another person's signature and additionally facilitates the signers as well as the proxy to guarantee the security of messages signed in the past even if their secret key is exposed today (property of forward-security). With a minor change in resigning key, we can make the scheme to behave as a multi-use bidirectional scheme. Also, automatic revocation of re-signing occurs on expiry of the time period T for which the scheme is designed.

To address the second open problem, we have come up with a proxy re-signature schemes that translates from Alice's ElGamal/Schnorr signature scheme to Bob's RSA Signature. Following is the protocol:

- a) Key generation for ElGamal/Schnorr and RSA Signatures
- b) Re-Signature key generation for ElGamal to RSA conversion and Schnorr to RSA conversion
- c) ElGamal / Schnorr Signature generation
- d) ElGamal / Schnorr Signature verification
- e) Re-Sign Algorithm for ElGamal to RSA conversion and Schnorr to RSA conversion
- f) RSA Signature verification.

We have carefully generated the proxy re-sign keys by establishing communication among delegatee, proxy signer and the delegator. Signatures generated by the basic signature generation algorithm and re-signature algorithm are indistinguishable. Also, Proxy signer revocation is possible.

3 Conclusion

We have tried to give solutions for the open challenges in the area of proxy re-signatures *i.e* design of a multi-use unidirectional scheme and translation of one type of signature scheme to another type. In both the solutions proxy signer revocation is possible.

References

1. Ateniese, G., Hohenberger, S.: Proxy re-signatures: new definitions, algorithms, and applications. In: ACM CCS 2005, pp. 310–319. ACM Press, New York (2005)
2. Libert, B., Vergnaud, D.: Multi-Use Unidirectional Proxy Re-Signatures February 8 (2008) arXiv:0802.1113v1 [cs.CR]
3. Blaze, M., Bleumer, G., Strauss, M.J.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998)
4. Anderson, R.: Invited Lecture. In: Fourth Annual Conference on Computer and Communications Security, ACM, New York (1997)

Fast Signature Matching Using Extended Finite Automaton (XFA)

R. Smith¹, C. Estan¹, S. Jha¹, and I. Siahaan²

¹ University of Wisconsin, Madison, WI 53706, USA
{smithr, estan, jha}@cs.wisc.edu

² Università di Trento, Povo-Trento, TN 38100, Italy
siahaan@disi.unitn.it

Abstract. Automata-based representations and related algorithms have been applied to address several problems in information security, and often the automata had to be augmented with additional information. For example, extended finite-state automata (EFSA) augment finite-state automata (FSA) with variables to track dependencies between arguments of system calls. In this paper, we introduce extended finite automata (XFAs) which augment FSAs with finite scratch memory and instructions to manipulate this memory. Our primary motivation for introducing XFAs is signature matching in Network Intrusion Detection Systems (NIDS). Representing NIDS signatures as deterministic finite-state automata (DFAs) results in very fast signature matching but for several types of signatures DFAs can blowup in space. Nondeterministic finite-state automata (NFA) representation of NIDS signatures results in a succinct representation but at the expense of higher time complexity for signature matching. In other words, DFAs are time-efficient but space-inefficient, and NFAs are space-efficient but time-inefficient. Our goal is to find a representation of signatures that is both time and space efficient. In our experiments we have noticed that for a large class of NIDS signatures XFAs have time complexity similar to DFAs and space complexity similar to NFAs. For our test set, XFAs use 10 times less memory than a DFA-based solution, yet achieve 20 times higher matching speeds.

1 Introduction

Automata-based representations have found several applications in information security. In some of these applications automata are augmented with additional information. For example, extended finite state automata (EFSA) augment finite-state automata (FSA) with uninterpreted variables and are very useful for capturing dependencies between system calls [23]. A similar representation is used in STATL [8] to track dependencies between events. In this paper our primary goal is to improve the time and space efficiency of signature matching in network intrusion detection systems (NIDS).¹ To achieve our goal we

¹ A NIDS that performs misuse detection matches incoming network traffic against a set of signatures. This functionality of a NIDS is called signature matching.

introduce *extended finite automata (XFAs)* which augment traditional FSAs with a finite scratch memory used to remember various types of information relevant to the progress of signature matching. Since an XFA is an FSA augmented with finite scratch memory, it still recognizes a regular language, albeit more efficiently than an FSA. We demonstrate that representing signatures in NIDS as XFAs significantly improves time and space efficiency of signature matching. We also present algorithms for manipulating XFAs, such as constructing XFAs from regular expressions and combining XFAs.

In the past signatures in NIDS were simply keywords, which resulted in extremely efficient signature-matching algorithms. The Aho-Corasick algorithm [1], for example, finds all keywords in an input in time linear in the input size. Because of the increasing complexity of attacks and evasion techniques [19], NIDS signatures have also become complex. Therefore, current techniques for generating different types of signatures, such as vulnerability [32, 4] or session [28, 21] signatures, generate signatures that use the full power of regular expressions. Representing NIDS signatures as deterministic finite-state automata (DFAs) results in a time-efficient signature-matching algorithm (each byte of the input can be processed in $O(1)$ time), but for certain regular expressions DFAs blow up in space. Nondeterministic finite-state automata (NFAs) are succinct representations for regular expressions, but the time complexity of the signature-matching algorithm increases, *i.e.*, each byte of the input can take $O(m)$ time to process, where m is the number of states in the NFA. Therefore, *DFAs are time-efficient but space-inefficient, and NFAs are space-efficient but time-inefficient*. If signatures are represented as XFAs, the scratch memory has to be updated while processing some input bytes. However, since the scratch memory is very small it can be updated very efficiently (especially if it is cached). Moreover, for many signatures XFAs are also a very succinct representation. For a large class of NIDS signatures *XFAs have time complexity similar to DFAs and space complexity similar to or better than NFAs*. The larger the scratch memory we can use, the smaller the space complexity of the required automaton (but the time complexity of the operations for updating the scratch memory may increase).

Recall that XFAs augment traditional FSAs with a small scratch memory which is used to remember various types of auxiliary information. We will explain the intuition behind XFAs with a short example. Consider n signatures s_i ($1 \leq i \leq n$) where $s_i = . *k_i . *k'_i$ (k_i and k'_i are keywords or strings). Note that s_i matches an input if and only if it contains a keyword k_i followed by k'_i . DFA D_i for signature s_i is linear in the size of the regular expression $. *k_i . *k'_i$. However, if the keywords are distinct, the DFA for the combination of the signatures $\{s_1, \dots, s_n\}$ is exponential in n . The reason for this state-space blowup is that for each i ($1 \leq i \leq n$) the DFA has to “remember” if it has detected the keyword k_i in the input processed so far. The XFA for the set of signatures $\{s_1, \dots, s_n\}$ maintains a scratch memory of n bits (b_1, \dots, b_n) , where bit b_i remembers whether it has seen the keyword k_i or not. The space complexity of the XFA is linear in n and the time complexity is $O(n)$ because the bits have to be potentially updated after processing each input symbol, but this

worst case happens only if n of the keywords overlap in specific ways. For the actual signatures we evaluated, the time complexity for XFAs is much closer to DFAs. Further, the XFA for an individual signature s_i is not much smaller than the corresponding DFA, but the combined XFA for the entire signature set is much smaller than the combined DFA. The reason is not that we use a special combination procedure, but that the “shape” of the automata the XFAs are built on does not lead to blowup. We discuss this example in detail in Section 3.1.

This paper is largely based on our earlier paper that appeared in Oakland 2008 [26]. The Oakland 2008 paper introduced the concept of XFAs. A subsequent paper that appeared in Sigcomm 2008 [27] formalizes the notion of state space explosion and discussed some optimizations and migration of XFAs to hardware.

2 Related Work

String matching was important for early network intrusion detection systems as their signatures consisted of simple strings. The Aho-Corasick [1] algorithm builds a concise automaton (linear in the total size of the strings) that recognizes multiple such signatures in a single pass. Other software [3, 6, 9] and hardware solutions [29, 16, 31] to the string matching problem have also been proposed. However, evasion [19, 11, 24], mutation [14], and other attack techniques [22] require signatures that cover large classes of attacks but still make fine enough distinctions to eliminate false matches. Signatures have thus evolved from simple exploit-based signatures to richer session [28, 21, 33] and vulnerability-based [4, 32] signatures. These complex signatures can no longer be expressed as strings, and regular expressions are used instead.

NFAs can compactly represent multiple signatures but may require large amounts of matching time, since the matching operation needs to explore multiple paths in the automaton to determine whether the input matches any signatures. In software, this is usually performed via backtracking (which opens the NFA up to serious algorithmic complexity attacks [7]) or by maintaining and updating a “frontier” of states, both of which can be computationally expensive. However, hardware solutions can parallelize the processing required and achieve high speeds. Sidhu and Prasanna [25] provide an NFA architecture that updates the set of states during matching efficiently in hardware. Further work [30, 5] has improved on their proposal, but for software implementations the processing cost remains significant.

DFAs can be efficiently implemented in software, although the resulting state-space explosion often exceeds available memory. Sommer and Paxson [28] propose on-the-fly determinization for matching multiple signatures, which keeps a cache of recently visited states and computes transitions to new states as necessary during inspection. This approach can be subverted by an adversary who can repeatedly invoke the expensive determinization operations. Yu *et al.* [34] propose combining signatures into multiple DFAs instead of one DFA, using simple heuristics to determine which signatures should be grouped together. The

procedure does reduce the total memory footprint, but for complex signature sets the number of resulting DFAs can be large. The cost of this approach is increased inspection time, since payloads must now be scanned by multiple DFAs. The D^2FA technique [15] performs edge compression to reduce the memory footprint of individual states. It stores only the difference between transitions in similar states, and in some sense, extends the string-based Aho-Corasick algorithm to DFAs. The technique does not address state space explosion and thus is orthogonal to our technique which focuses on reducing the number of states required. The two techniques could be combined to obtain further reductions in memory usage.

Pre-filter-based solutions such as those used by Snort [20] can achieve good average-case performance. The pre-filter performs string matching on subparts of a signature, invoking the matching procedure for the full regular expression only when a subpart has been matched. Our preliminary results show that this approach is vulnerable to algorithmic complexity attacks. By sending traffic crafted to defeat Snort's pre-filter and to cause expensive regular expression processing, an attacker can slow it down by as much as a factor of 5000.

Other extensions to automata have been proposed in the context of information security. *Extended Finite State Automata (EFSA)* extend traditional automata to assign and examine values of a finite set of variables. Sekar and Upuluri [23] use EFSAs to monitor a sequence of system calls. Extensions, such as EFSAs, fundamentally broaden the language recognized by the finite-state automata, *e.g.*, EFSAs correspond to regular expression for events (REEs). On the other hand, XFAs can be viewed as an optimization of a regular DFA, but XFAs do not enhance the class of languages that can be recognized. It will be interesting to consider XFA-type optimizations to EFSAs.

Eckmann *et al.* [8] describe a language STATL, which can be thought of as finite-state automata with transitions annotated with actions that an attacker can take. The motivation for STATL was to describe attack scenarios rather than improve the efficiency of signature matching. Automata augmented with various objects, such as timed automata [2] and hybrid automata [12], have also been investigated in the verification community. For example, hybrid automata, which combine discrete transition graphs with continuous dynamical systems, are mathematical models for digital systems that interact with analog environments. As with EFSAs, these automata (which are usually infinite-state) fundamentally enhance the languages they recognize.

Space-time or time-memory tradeoff is a technique where the memory use can be reduced at the cost of slower program execution, or vice versa, the computation time can be reduced at the cost of increased memory use. In complexity theory researchers investigate whether addition of a restriction on the space inhibits one from solving problems in certain complexity class within specific time bounds. For example, time-space tradeoff lower bounds for SAT were investigated by Fortnow [10]. Time-space tradeoffs have also been explored in the context of attacks [18,17]. We are not aware of existing work on time-space tradeoffs in the context of signature matching for NIDS.

3 Technical Overview

We begin with a discussion of simple signatures illustrating how XFAs need much fewer states than DFAs, followed by an overview of the steps for compiling realistic signatures to XFAs suitable for NIDS use.

3.1 Reducing State Space with XFAs

Recognizing a signature set with n signatures of the form $. *S_i . *S'_i$, where all S_i and S'_i are distinct strings, leads to state space blowup with DFAs. Figure 1 shows an example for the case where $n = 2$, $S_1 = \text{ab}$, $S'_1 = \text{cd}$, $S_2 = \text{ef}$, and $S'_2 = \text{gh}$. In the general case, for each of the n signatures, the combined DFA must “remember” whether it already found the first string in the input so that it “knows” whether to accept if it sees the second string. For example, in Figure 1 the DFA is in state PV when neither **ab** nor **ef** has been observed. Similarly, it is in state RV when **ab** but not **ef** is seen, state PX when **ef** but not **ab** is seen, and state RX when both **ab** and **ef** have been seen. In general, to remember n independent bits of information, the DFA needs at least 2^n distinct states. An analysis of the generalized example shows that if the strings are of length l , then the actual number of states used by the combined DFA is $O(n!2^n)$.

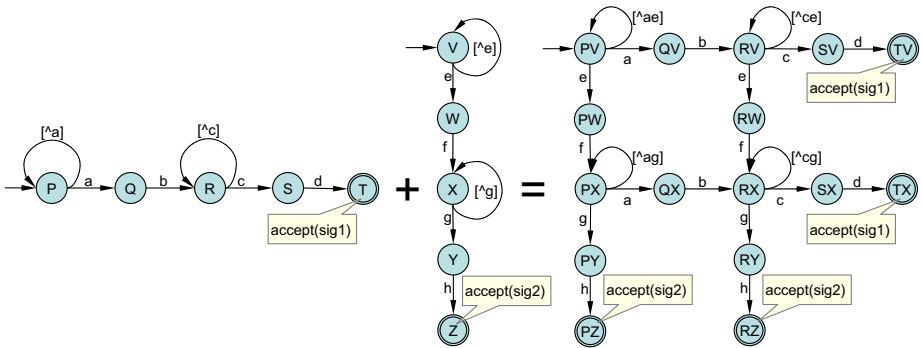


Fig. 1. The DFA recognizing both $. * \text{ab} . * \text{cd}$ and $. * \text{ef} . * \text{gh}$ has state space blowup. For simplicity, we do not show some less important transitions.

Figure 2 shows the same signatures as in Figure 1 when DFAs are replaced with XFAs. In this figure, the XFAs for $. * \text{ab} . * \text{cd}$ and $. * \text{ef} . * \text{gh}$ each use a single bit of scratch memory that is manipulated by instructions attached to specific edges (depicted in the figure as callout boxes). During matching, these instructions are “executed” each time the corresponding transition is followed. For each signature of the form $. *S_i . *S'_i$, as long as S_i does not overlap with S'_i , we can build XFAs like those in Figure 2 that uses a single bit of scratch memory.² This bit explicitly encodes whether S_i has appeared in the input so

² If S_i and S'_i overlap it is still possible to build an XFA recognizing the signature, but it will use more than one bit.

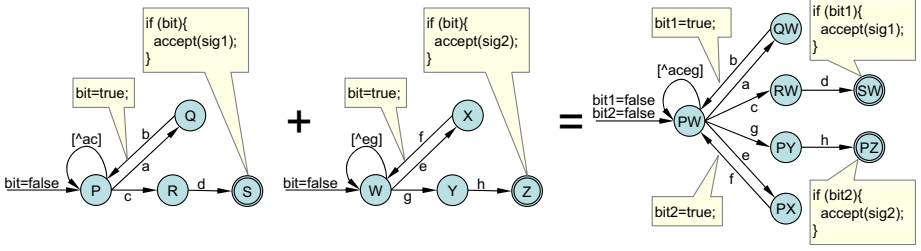


Fig. 2. The XFA recognizing both $.*ab.*cd$ and $.*ef.*gh$ without state space blowup. For simplicity, we do not show some less important transitions.

far, and the shape of the underlying automaton is very similar to that of the combined DFA recognizing $.*S_i$ and $.*S'_i$ independently. The combined XFA for the entire signature set uses n bits and $O(nl)$ states. Thus by adding n bits of scratch memory we obtain a combined XFA approximately 2^n times smaller than the combined DFA. The initialization time goes up from $O(1)$ to $O(n)$ and, assuming that the strings in the signatures are not suffixes of each other, only a small constant is added to the worst-case per byte processing cost as at most one bit is updated for any given byte from the input.

Note that the presence of scratch memory and use of instructions as defined for XFAs does not affect combination: the same process for combining DFAs is used for combining the underlying automata of XFAs. In reality, the combined XFA (Figure 2) is smaller than the combined DFA (Figure 1) because the automata structure in the source XFAs is different than for DFAs. When combined, these XFAs have benign interactions, just as with DFAs limited to string matching.

XFAs can provide large reductions in the number of states even when recognizing individual signatures. Figures 3 and 4 show the DFA and XFA, respectively, recognizing the language defined by $.\{n\}$ which consists of all strings of length n . Although no NIDS signatures have this exact form, signatures detecting buffer overflows use sequences of states similar to those in Figure 3 to count the

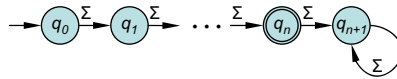


Fig. 3. DFA recognizing $.\{n\}$

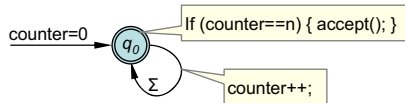


Fig. 4. XFA recognizing $.\{n\}$

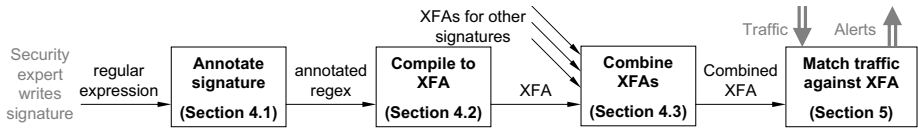


Fig. 5. Lifecycle of signatures in a NIDS using XFAs

number of characters that follow a given keyword. The minimal DFA for $\cdot\{n\}$ needs $n + 2$ states, whereas the XFA uses a single state and a counter. This counter is initialized to 0 and is incremented on every transition, signaling acceptance only when the value is n . Increment is defined so that once the counter reaches $n + 1$ it remains at $n + 1$. Thus the counter needs to take only $n + 2$ values, requiring only $k = \lceil \log_2(n + 2) \rceil$ bits of scratch memory. By adding these k bits we reduce the number of states by a factor of close to 2^k . Measuring run time in bit operations, the initialization cost and per-byte processing increase from $O(1)$ to $O(k)$. If we count instructions, a small constant is added to both initialization and per byte processing.

3.2 Using XFAs in a NIDS

Figure 5 depicts the steps involved in constructing XFAs and using them in a NIDS. Crafting NIDS signatures themselves is outside the scope of this paper since our proposal changes only the representation of signatures for matching, not the underlying semantics. Readers should refer to [26] for a description of combining XFAs for multiple signatures. Our Oakland 2008 and Sigcomm 2008 papers also contain a feasibility study that uses a signature set from the open-source Snort NIDS [20] to compare the performance of matching with an XFA against the performance of matching with DFAs.

4 Extended Finite State Automaton (XFA)

Definition 1. An *extended finite automaton (XFA)* is a 7-tuple $(Q, D, \Sigma, \delta, U, (q_0, d_0), F)$ where

- Q is a finite set of states, Σ is a finite set of input alphabets, and $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ is the transition relation,³
- D is a finite set of values in the data domain,
- $U : Q \times (\Sigma \cup \{\epsilon\}) \times Q \rightarrow 2^{D \times D}$ is the per transition *update relation* which defines how the data value is updated on every transition,⁴
- (q_0, d_0) is the *initial configuration* which consists of an initial state q_0 and an *initial data value* d_0 ,
- and $F \subseteq Q \times D$ is the set of accepting configurations.

³ We assume that ϵ is a special symbol not in Σ .

⁴ We assume that U is total (defined for all elements of the domain $Q \times (\Sigma \cup \{\epsilon\}) \times Q$). If $(q, a, q') \notin \delta$ then we define $U(q, a, q')$ to be the empty relation.

Notice that XFA adds two additional components (a data domain and an update relation) to the classical nondeterministic finite automaton (NFA). Moreover, XFAs change the initial state and the acceptance criteria to include the data domain.

A configuration is tuple (q, d) where $q \in Q$ and $d \in D$. There is a transition from (q, d) to (q', d') on an input symbol $a \in \Sigma$ (denoted by $(q, d) \xrightarrow{a} (q', d')$) iff $(q, a, q') \in \delta$ and $(d, d') \in U(q, a, q')$. A string $a_1 a_2 \cdots a_k$ is accepted by an XFA $X = (Q, D, \Sigma, \delta, U, (q_0, d_0), F)$ iff there is a sequence of transitions $(q_0, d_0) \xrightarrow{a_1} (q_1, d_1) \cdots (q_{k-1}, d_{k-1}) \xrightarrow{a_k} (q_k, d_k)$ such that $(q_k, d_k) \in F$ (recall that (q_0, d_0) is the initial configuration). The set of strings accepted by an XFA X is the language accepted by X (the language is denoted by $L(X)$). Since the data domain D is finite, $L(X)$ is regular.

An XFA $X = (Q, D, \Sigma, \delta, U_\delta, (q_0, d_0), F)$ is *state deterministic* if δ is a function from $Q \times \Sigma$ to Q . XFA X is *data deterministic* if for all $(q, a, q') \in Q \times \Sigma \times Q$, $U(q, a, q')$ is a function from D to D . A XFA X is a *deterministic* if it is both data and state deterministic.

5 Constructing XFAs

In classical automata theory (as discussed in [13]) the following steps are performed to convert a regular expression R into a DFA D :

- **Step 1:** Convert the regular expression R into a NFA M with ϵ transitions.
- **Step 2:** Convert NFA M into an NFA M' without ϵ -transitions.
- **Step 3:** Determinize a NFA M' into a DFA D .
- **Step 4:** Minimize the DFA D .

We need to provide algorithms corresponding to the four steps for XFAs.

5.1 Converting Regular Expressions to XFAs

We expand the grammar of regular expressions with additional operators. The set of *regular expressions with integer-range exponents* (*REIREs*) over the alphabet Σ (denoted by \mathcal{RE}_Σ) are defined by the following rules:

- $\emptyset \in \mathcal{RE}_\Sigma$.
- $\epsilon \in \mathcal{RE}_\Sigma$.
- For all $a \in \Sigma$, $\{a\} \in \mathcal{RE}_\Sigma$.
- If E_1 and E_2 are in \mathcal{RE}_Σ , then $E_1 + E_2$ is in \mathcal{RE}_Σ .
- If E_1 and E_2 are in \mathcal{RE}_Σ , then $E_1 \cdot E_2$ is in \mathcal{RE}_Σ .
- If $E \in \mathcal{RE}_\Sigma$, then E^* is in \mathcal{RE}_Σ .
- If E_1 and E_2 are in \mathcal{RE}_Σ , then $E_1 \# E_2$ is in \mathcal{RE}_Σ .
- If $E \in \mathcal{RE}_\Sigma$, n a non-negative integer, m a non-negative integer or ∞ , and $n \leq m$, then $E\{n, m\}$ is in \mathcal{RE}_Σ . Note that $E^* = E\{0, \infty\}$, but we defined an operator \star because it is more efficient to handle it as a separate case.
- If $E \in \mathcal{RE}_\Sigma$, then $(E) \in \mathcal{RE}_\Sigma$.

Given an $E \in \mathcal{RE}_\Sigma$, let $L(E) \subseteq \Sigma^*$ be the language corresponding to E . The language $L(E)$ is recursively defined as follows:

- $L(\emptyset)$ is equal to \emptyset .
- $L(\epsilon)$ is equal to $\{\epsilon\}$ (the set containing the empty string).
- For all $a \in \Sigma$, $L(\{a\})$ is defined as $\{a\}$.
- If $E = E_1 + E_2$, then $L(E)$ is $L(E_1) \cup L(E_2)$.
- If $E = E_1 \cdot E_2$, the $L(E)$ is define as follows:

$$\{x_1x_2 \mid \text{such that } x_1 \in L(E_1) \text{ and } x_2 \in L(E_2)\}$$

- If $E = E'^*$, then $L(E)$ is defined as follows:

$$\{x_1x_2 \cdots x_k \mid \text{for } 0 \leq i \leq k, x_i \in E'\}$$

- If $E = E_1 \sharp E_2$, the $L(E)$ is define as follows:

$$\{x_1x_2 \mid \text{such that } x_1 \in L(E_1) \text{ and } x_2 \in L(E_2)\}$$

- If $E = E'\{n, m\}$, then $L(E)$ is defined as follows:

$$\{x_1x_2 \cdots x_k \mid 0 \leq n \leq k \leq m \text{ for } 1 \leq i \leq k, x_i \in E'\}$$

- $L((E))$ is equal to $L(E)$

Except for the operators \sharp and $\{n, m\}$ the other operators are the same as that for classical regular expressions (see [13]). For all operators except \sharp and $\{n, m\}$ we know from classical automata theory how to construct NFAs, e.g. given NFAs for E_1 and E_2 , we know how to construct NFA for $E_1 \cup E_2$. These classical algorithms can be readily adapted for XFAs (as usual data domains have to be accounted for). Operators \sharp and $\{n, m\}$ have to be handled especially for XFA case (readers should consult our Oakland 2008 paper [26] for a detailed description of how these operators are handled). Figure 6 shows the XFA for $expr_1 \sharp expr_2$ in terms of the XFAs for $expr_1$ and $expr_2$. Note that a bit is used to ensure that XFAs for $expr_1 \sharp expr_2$ accepts the language corresponding to $expr_1 \cdot expr_2$. Figure 7 shows the XFA for $expr_1 \{m, n\}$ in terms of the XFA for $expr_1$.

5.2 Eliminating ϵ -Transitions

Eliminating ϵ -transitions for a NFA requires computing ϵ -reachability, *i.e.*, a state q' is ϵ -reachable from another state q if there is a path from q to q' only consisting of ϵ -transitions. The complication for XFAs is that we have to keep track of relations that update the data domain.

Consider an XFA $X = (Q, D, \Sigma, \delta, U, (q_0, d_0), F)$ with ϵ -transitions. First we extend the update relation U_δ to paths and sets of paths. Consider a path $\pi = q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{\epsilon} \cdots \xrightarrow{\epsilon} q_{k+1}$ from q_1 to q_k consisting only of ϵ -transitions. The update relation corresponding to the path π is $U(q_1, \epsilon, q_2) \circ U(q_2, \epsilon, q_3) \circ \cdots \circ U(q_k, \epsilon, q_{k+1})$

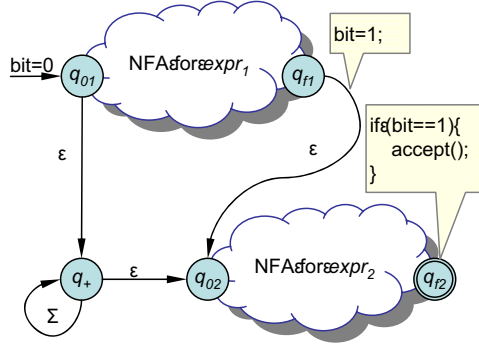


Fig. 6. Simplified NXFA construction step for parallel concatenation $expr_1 \# expr_2$

(\circ denotes relational composition).⁵ The update relation corresponding to a set of paths $\{\pi_1, \dots, \pi_k\}$ is $\bigcup_{i=1}^k U(\pi_i)$. We define ϵ -closure of a state q (denoted by $\epsilon\text{-CLOSURE}(q)$) as the following set of tuples:

(q', U') is in $\epsilon\text{-CLOSURE}(q)$ iff there exists a path from q to q' only consisting of ϵ -transitions (q' is ϵ -reachable from q) and $U' = \bigcup_{\pi \in \text{paths}(q, q')} U(\pi)$ (where $\text{paths}(q, q')$ is the set of ϵ -paths from q to q').

Even though the set $\text{paths}(q, q')$ can be infinite, $\epsilon\text{-CLOSURE}(q)$ can be computed using a standard worklist algorithm.⁶

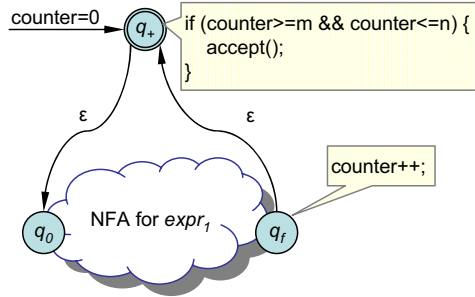


Fig. 7. Simplified NXFA construction step for $(expr_1)\{m, n\}$

Assume that we have computed $\epsilon\text{-CLOSURE}(q)$ for all states $q \in Q$. Define an XFA $X' = (Q, D, \Sigma, \delta', U', (q_0, d_0), F')$, where

⁵ Given two relations $U_1 \subseteq D \times D$ and $U_2 \subseteq D \times D$, the relational composition $U_1 \circ U_2$ is the relation which contains a tuple (d_1, d_2) iff there exists a d such that $(d_1, d) \in U_1$ and $(d, d_2) \in U_2$.

⁶ The standard worklist algorithm converges because D is finite.

- A transition $(q, a, q') \in \delta'$ ($a \in \Sigma$) iff
 - $(q, a, q') \in \delta$, or
 - there exists a state q_1 such that q_1 is ϵ -reachable from q and $(q_1, a, q') \in \delta$.
- $U'(q, a, q')$ is equal to the following relation:

$$U(q, a, q') \cup \bigcup_{(q_1, U_1) \in \epsilon\text{-CLOSURE}(q)} U_1 \circ U(q_1, a, q)$$

- If there is a configuration in F reachable from (q_0, d_0) , then $F' = F \cup \{q_0, d_0\}$; otherwise $F' = F$.

5.3 Determinizing a XFA

The classical algorithm for determinizing an NFA is based on subset construction (see [13]). We will now consider determinization of XFAs. In this section we will assume that a XFA does not have ϵ -transitions. Consider a XFA $X = (Q, D, \Sigma, \delta, U, (q_0, d_0), F)$ without ϵ -transitions. We will transform X to a state deterministic XFA $X' = (Q', D, \Sigma, \delta', U', (q'_0, d_0), F')$ where each component of X' is defined as follows:

- $Q' = 2^Q$.
- For all $a \in \Sigma$, a state q in $\delta'(\{q_1, \dots, q_m\}, a)$ iff there exists a transition $q_i \xrightarrow{a} q$ ($1 \leq i \leq m$).
- $U'(\{q_1, \dots, q_m\}, a, \{p_1, \dots, p_j\})$ is $\bigcup_{i=1}^m \bigcup_{j=1}^m U(q_i, a, p_j)$.
- $q'_0 = \{q_0\}$.
- $(\{q_1, \dots, q_m\}, d) \in F'$ iff there exists a $q_i \in \{q_1, \dots, q_m\}$ such that $(q_i, d) \in F$.

Unfortunately the XFA X' does not accept the same language as XFA X . Assume that in X' we create a state $\{q_1, q_2\}$ and there is transition on a from q_0 to q_1 and q_2 . In X' the update relation U associated with $(\{q_0\}, a, \{q_1, q_2\})$ is union of the update relations U_1 and U_2 associated with (q_0, a, q_1) and (q_0, a, q_2) . Assume that $(d_0, d_1) \in U_1$ and $(d_0, d_2) \in U_2$ but $(d_0, d_2) \notin U_1$ and $(d_0, d_1) \notin U_2$. Since $(d_0, d_1) \in U$ and $(d_0, d_2) \in U$, the configurations $(\{q_1, q_2\}, d_1)$ and $(\{q_1, q_2\}, d_2)$ are both reachable in X' from the initial state $\{\{q_0\}, d_0\}$. However, configurations (q_1, d_2) and (q_2, d_1) are not reachable from (q_0, d_0) in X . This phenomenon can lead to extra accepting paths in X' . This situation can be remedied if each state q has its own copy of the data domain to update.

Consider an XFA $X = (Q, D, \Sigma, \delta, U, (q_0, d_0), F)$. We construct an XFA $X_1 = (Q, D_1, \Sigma, \delta, U_1, (q_0, d'_0), F')$, where each component is defined as follows:

- $D_1 = Q \times D$.
- A tuple $((q, d), (q', d')) \in U_1(q, a, q')$ iff $(d, d') \in U_\delta(q, a, q')$.
- $d'_0 = (q_0, d_0)$.
- $(q, (q, d)) \in F'$ iff $(q, d) \in F$.

In order to determinize an XFA the following steps should be performed:

1. First convert X to X_1 by expanding the data domain D to $Q \times D$.
2. Convert X_1 to X' using the subset construction.

It is easy to see that X' is state deterministic. In order to create a data deterministic XFA extra steps need to be taken, which we will not describe in this paper.

Reachability analysis: Before we move to minimizing an XFA X we remove configurations (q, d) that are not reachable from the initial configuration (q_0, d_0) . We also remove configurations (q, d) that cannot reach any configuration in F .

5.4 Minimizing a XFA

Consider a DFA $M = (Q, \Sigma, \delta, q_0, F)$. Minimization of M finds the coarsest equivalence relation $R \subseteq Q \times Q$ that satisfies the following conditions:

- $(q_1, q_2) \in R$ implies that $q_1 \in F \leftrightarrow q_2 \in F$.
- $(q_1, q_2) \in R$ implies that for all $a \in \Sigma$, $(\delta(q_1, a), \delta(q_2, a)) \in R$.

Once the coarsest equivalence relation R is computed, all the states in Q in the same equivalence class are merged.

Note: Let \mathcal{R} be the set of all relations satisfying the abovementioned conditions. We say that $R_1 \in \mathcal{R}$ is coarser than $R_2 \in \mathcal{R}$ iff $R_2 \subseteq R_1$. Also note that if $R_1 \in \mathcal{R}$ and $R_2 \in \mathcal{R}$, then $R_1 \cup R_2$ is also in \mathcal{R} . Hence the coarsest relation satisfying the conditions given above is $\cup_{R \in \mathcal{R}} R$. Moreover, the coarsest relation can be computed using a greatest fixpoint computation.

Consider a deterministic XFA $X = (Q, D, \Sigma, \delta, U, (q_0, d_0), F)$. Assume that we find the coarsest equivalence relation $R \subseteq (Q \times D) \times (Q \times D)$ that satisfies the following conditions:

- $((q, d), (q', d')) \in R$ implies that $q = q'$ (configurations with different states are never equivalent).
- $((q, d), (q, d')) \in R$ implies that $(q, d) \in F$ iff $(q, d') \in F$.
- $((q, d), (q, d')) \in R$ implies that for all $a \in \Sigma$ $((q', U(q, a, q')(d)), (q', U(q, a, q')(d'))) \in R$ (where $q' = \delta(q, a)$).⁷

In this case for a state q if two data values d_1 and d_2 have the property that $((q, d_1), (q, d_2)) \in R$, then configurations (q, d_1) and (q, d_2) can be merged. Recall that during determinizing a XFA X , we expanded the data domain to $Q \times D$. The equivalence relation R allows us to merge data values for each state $q \in Q$.

Next we consider minimizing states of a deterministic XFA $X = (Q, D, \Sigma, \delta, U, (q_0, d_0), F)$. For each state $q \in Q$, $effects(q) = (U_q, p_q)$ is a tuple such that $U_q(a, q')$ is equal to $U(q, a, q')$ (U_q is a function from $\Sigma \times Q \rightarrow D^D$) and $p_q \subseteq D$ such that $d \in p_q$ implies that $(q, d) \in F$. Intuitively, $effects(q)$ capture the update function and accepting data values associated with state q . Assume that

⁷ Since X is deterministic, $U(q, a, q')$ is a function.

we find the coarsest equivalence relation $R \subseteq Q \times Q$ that satisfies the following conditions:

- $(q_1, q_2) \in R$ implies that $effects(q_1) = effects(q_2)$.
- $(q_1, q_2) \in R$ implies that for all $a \in \Sigma$, $(\delta(q_1, a), \delta(q_2, a)) \in R$.

Once we have computed the coarsest relation R satisfying the abovementioned conditions, two states in the same equivalence class can be merged into one.

6 Conclusion and Future Work

In this paper we described Extended Finite Automata (XFAs), which augment traditional finite state automata with a scratch memory that is manipulated by instructions attached to edges and states. We provide a formal definition for XFAs and present a technique for constructing them from regular expressions. Experimental results presented in [26] demonstrated using a set of HTTP signatures from Snort and observed that XFAs have matching speeds approaching DFAs yet memory requirements similar to NFAs. Compared to multiple DFA-based techniques, our tests used $10\times$ less memory *and* were $20\times$ faster.

The techniques and results we have presented here are preliminary in many respects and we are actively working to refine them. Some aspects of our construction procedure require some manual input, and some signatures require inordinately long construction times. In addition, there is still some missing functionality and inefficiencies in our interpreter and execution environment. We are investigating techniques for addressing these and other issues. Notwithstanding these open problems, we are hopeful that in the end XFAs will lead to better solutions for high speed signature matching.

Acknowledgements

This work is sponsored by NSF grants 0546585 and 0716538 and by a gift from the Cisco University Research Program Fund at Silicon Valley Community Foundation.

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Communications of the ACM (June 1975)
2. Alur, R.: Timed automata. In: Proceedings of the Int. Conf. on Computer Aided Verification, pp. 8–22 (1999)
3. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Communications of the ACM 20 (October 1977)
4. Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S.: Towards automatic generation of vulnerability-based signatures. In: IEEE Symposium on Security and Privacy, Oakland, California (May 2006)

5. Clark, C.R., Schimmel, D.E.: Scalable pattern matching for high-speed networks. In: IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 249–257 (April 2004)
6. Coit, C.J., Staniford, S., McAlerney, J.: Towards faster pattern matching for intrusion detection or exceeding the speed of Snort. In: 2nd DARPA Information Survivability Conference and Exposition (June 2001)
7. Crosby, S.: Denial of service through regular expressions. In: Usenix Security work in progress report (August 2003)
8. Eckmann, S.T., Vigna, G., Kemmerer, R.A.: STATL: An attack language for state-based intrusion detection. *Journal of Computer Security* 10(1/2), 71–104 (2002)
9. Fisk, M., Varghese, G.: Fast content-based packet handling for intrusion detection. TR CS2001-0670, UC San Diego (May 2001)
10. Fortnow, L.: Nondeterministic polynomial time versus nondeterministic logarithmic space: Time-space tradeoffs for satisfiability. In: Proceedings of Twelfth IEEE Conference on Computational Complexity (1997)
11. Handley, M., Paxson, V., Kreibich, C.: Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In: Usenix Security (August 2001)
12. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS), pp. 278–292 (1996)
13. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
14. Jordan, M.: Dealing with metamorphism. *Virus Bulletin Weekly* (2002)
15. Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P., Turner, J.: Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In: Proceedings of ACM SIGCOMM (September 2006)
16. Liu, R.-T., Huang, N.-F., Chen, C.-H., Kao, C.-N.: A fast string-matching algorithm for network processor-based intrusion detection system. *Transactions on Embedded Computing Sys.* 3(3), 614–633 (2004)
17. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In: ACM Conference on Computer and Communications Security (CCS) (2005)
18. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729. Springer, Heidelberg (2003)
19. Ptacek, T., Newsham, T.: Insertion, evasion and denial of service: Eluding network intrusion detection. In: Secure Networks, Inc. (January 1998)
20. Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proceedings of the 13th Systems Administration Conference, USENIX (1999)
21. Rubin, S., Jha, S., Miller, B.: Language-based generation and evaluation of NIDS signatures. In: IEEE Symposium on Security and Privacy (May 2005)
22. Rubin, S., Jha, S., Miller, B.P.: Protomatching network traffic for high throughput network intrusion detection. In: ACM Conference on Computer and Communications Security (CCS), pp. 47–58 (2006)
23. Sekar, R., Uppuluri, P.: Synthesizing fast intrusion prevention/detection systems from high-level specifications. In: Usenix Security (August 1999)
24. Shankar, U., Paxson, V.: Active mapping: Resisting NIDS evasion without altering traffic. In: IEEE Symposium on Security and Privacy (May 2003)
25. Sidhu, R., Prasanna, V.: Fast regular expression matching using FPGAs. In: Field-Programmable Custom Computing Machines (FCCM) (April 2001)
26. Smith, R., Estan, C., Jha, S.: Xfa: Faster signature matching with extended automata. In: IEEE Symposium on Security and Privacy (2008)

27. Smith, R., Estan, C., Jha, S., Kong, S.: Deflating the big bang: fast and scalable deep packet inspection with extended finite automata. In: SIGCOMM (2008)
28. Sommer, R., Paxson, V.: Enhancing byte-level network intrusion detection signatures with context. In: ACM Conference on Computer and Communications Security (CCS) (2003)
29. Sourdis, I., Pnevmatikatos, D.: Fast, large-scale string match for a 10gbps fpga-based network intrusion detection system. In: International Conference on Field Programmable Logic and Applications (September 2003)
30. Sourdis, I., Pnevmatikatos, D.: Pre-decoded CAMs for efficient and high-speed NIDS pattern matching. In: IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) (April 2004)
31. Tan, L., Sherwood, T.: A high throughput string matching architecture for intrusion detection and prevention. In: International Symposium on Computer Architecture (ISCA) (June 2005)
32. Wang, H.J., Guo, C., Simon, D., Zugenmaier, A.: Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In: Proceedings of the 2004 ACM SIGCOMM Conference (August 2004)
33. Yegneswaran, V., Giffin, J.T., Barford, P., Jha, S.: An architecture for generating semantics-aware signatures. In: 14th USENIX Security Symposium (August 2005)
34. Yu, F., Chen, Z., Diao, Y., Lakshman, T.V., Katz, R.H.: Fast and memory-efficient regular expression matching for deep packet inspection. In: Proceedings of Architectures for Networking and Communications Systems (ANCS), pp. 93–102 (2006)

Real-Time Alert Correlation with Type Graphs

Gianni Tedesco and Uwe Aickelin

School of Computer Science,
University of Nottingham,
Nottingham NG8 1BB,
United Kingdom

Abstract. The premise of automated alert correlation is to accept that false alerts from a low level intrusion detection system are inevitable and use attack models to explain the output in an understandable way. Several algorithms exist for this purpose which use attack graphs to model the ways in which attacks can be combined. These algorithms can be classified in to two broad categories namely scenario-graph approaches, which create an attack model starting from a vulnerability assessment and type-graph approaches which rely on an abstract model of the relations between attack types. Some research in to improving the efficiency of type-graph correlation has been carried out but this research has ignored the hypothesizing of missing alerts. Our work is to present a novel type-graph algorithm which unifies correlation and hypothesizing in to a single operation. Our experimental results indicate that the approach is extremely efficient in the face of intensive alerts and produces compact output graphs comparable to other techniques.

1 Introduction

The output of intrusion detection systems (IDS) is generally a time series of discrete events called “alerts” with each event describing, at a low level, features of the network traffic. These alert attributes typically include the endpoints and communication channels implicated in an alert and the type of alert. Arguably the most significant problem with analyzing IDS alerts is the high volume of false alarms. Even without false alarms IDS alerts require some interpretation. This is because attacks are often split in to several stages, each of which may generate many alerts.

This observation has lead to the proposal that alerts be automatically correlated using a model of attacks which encodes their prerequisites and consequences [10]. Typically these methods involve representing attack types as vertices in a directed acyclic graph which we shall call an “attack graph”. Edges in attack graphs represent the relationship between prerequisites and consequences of attacks. Intuitively speaking, a directed edge will connect attack A to attack B if A prepares for B .

Research has shown that such techniques are capable of:

1. Aggregating alerts which imply the same, or similar, consequences. An aggregated group of alerts is called a hyper-alert.

2. Ignoring extraneous alerts which do not correlate with anything.
3. Uncovering missing alerts in an alert stream and hypothesizing their attribute values where possible [8]. Hypotheses may optionally be compared against other evidence sources such as system logs [11].

These automated alert correlation techniques may be divided in to two categories based on the type of attack model which is encoded in the attack graph. We shall refer to the two categories as type-graph and scenario-graph algorithms. Scenario graph algorithms rely on a complete and correct vulnerability assessment to generate a graph of attack sequences specific to the protected network [6]. While this approach allows for real-time automated correlation it fails completely if network addresses are re-assigned or if the vulnerability assessment is erroneous. Conversely, type graph algorithms model only abstract attack types which allows for more robust correlation but with a higher computational cost. In [10] correlation is performed in batch mode only and in [15], where vulnerability assessment data is incorporated, a sliding correlation window is required to keep the problem manageable.

We assert that real-time correlation is desirable because it allows for timely automated responses. If the time lag between detection and response is too great then attacks such as rapidly spreading worms may become much more difficult to contain. Real-time operation also facilitates techniques such as [4] where correlation output is used to perform a targeted forensic analysis of network traffic for the purposes of discovering novel attacks and variations of known attacks.

Our work is motivated by the need for a correlation algorithm with both the flexibility of an abstract attack type-graph and similar performance characteristics to state of the art scenario graph algorithms. Specifically, we wish to avoid relying on prior knowledge of network topology and the distribution of vulnerabilities in the protected network. It is also desirable to avoid relying on a sliding correlation window which would allow “low and slow” attacks to become lost.

The aim of this paper is to develop an automated alert correlation algorithm using attack type graphs which is suitable for deployment in a real-time setting. A theoretical analysis of computational complexity will be provided. For verification the algorithm will be experimentally evaluated in terms of performance and accuracy.

Our proposed solution works by re-structuring the type graph correlation algorithm presented by Ning *et al.* such that it acts on individual alerts in sequence rather than all alerts in batch. The basic approach is to keep an internal database of hyper-alerts of each type and use in-memory indexes to efficiently find prerequisites of each new hyper-alert. The size of the in-memory database is minimized by eliminating redundant information which does not contribute to the correlation process. Hypothesizing of missing alerts is a recursive special-case of the correlation algorithm which can input hyper-alerts with wild-card attributes. The main contributions of this work are a type-graph correlation algorithm suitable for real-time use. The algorithm depends on a novel index structure and unifies the correlation and hypothesising steps in to a single algorithm.

This paper is structured as follows. First a brief discussion of related work is given in section 2. From here we present a description and formal definition of the problem in section 3. Building on this definition, a solution is presented in section 4 which solves the minimal IAC problem where there are no false negative alerts. This minimal algorithm is developed to the fuller solution presented and analyzed in section 5. Section 6 provides an empirical analysis of the algorithm. In the final section the results are discussed, conclusions drawn and future work proposed.

2 Related Work

Seminal works such as [13, 5] laid the groundwork for automated analysis of security related facts and events. These works proposed a formal theory of computer attacks by modeling the prerequisites and consequences of vulnerabilities in attack graphs and formal grammars respectively.

Wang *et al.* take a vulnerability-centric approach to alert correlation [6]. In this work an automated vulnerability analysis [12, 14] creates an attack graph consisting of two types of vertex, attacks and states. Only those attacks which have been found on the protected system are included. All attack vertices are bound with attribute values such as IP addresses and ports. The correlation algorithm works by performing a breadth first search on the attack graph. High performance is achieved by enumerating all possible fact assignments for every attack type and pre-computing an optimized graph structure for correlation. Another important concept in this work is “implicit correlation” whereby only the latest alert which satisfies an attack step is stored in memory. However, we have asserted that it is undesirable to assume that the defender can reliably know of all vulnerabilities on the network. Therefore our work uses an abstract attack-type model although we do use a similar hypothesizing technique and try to preserve the notion of implicit correlation as far as possible.

Ning *et al.* take a logical approach to modeling attack sequences for automated correlation[9, 10, 2]. The technique is intended to be applied in batch to an off-line database of collected hyper-alerts. The fundamental building block of the approach is the definition of a “hyper-alert type” which represents a type of attack and its prerequisites and consequences. Each hyper-alert type consists of a triple of fact names and prerequisites and conclusions which are predicate expressions with free variables bound from the fact names. If a predicate appears in the consequence of one hyper-alert type and the prerequisite of another then the former “may prepare for” the latter. The assignment of facts to any such shared predicate are used to calculate equality constraints between the two types.

A hyper-alert of a given type is simply a tuple of attribute values corresponding to the fact names for that type. Correlation is performed in batch on a set of hyper-alerts, each hyper-alert is considered a potential vertex in the correlation graph and if equality constraints are satisfied between other hyper-alerts then they are correlated by adding a directed edge between them provided that their timestamps show the correct temporal order.

Hypothesizing of missing alerts is treated in [8, 11]. The problem here is that when some steps in an attack have been missed by the underlying IDS then the resultant correlation graphs may be split and require additional processing to re-integrate them. The approach taken in their work involves four steps:

1. Subgraphs of the correlation graph are clustered according to the attribute values of their hyper alerts.
2. Once candidate subgraphs have been selected for integration, a special hyper-alert type-graph is consulted which has had indirect edges added to it. Pairs of subgraphs are then correlated using these new edges to define the indirectly-prepares-for relation.
3. When an indirect correlation occurs there are one or more paths in the type-graph connecting the two hyper-alert types. New hyper-alerts are created to connect the two correlation graphs and their attribute values inferred using the equality constraints in the graph.
4. Because the prior steps may generate many redundant hypotheses with equivalent fact values, a consolidation step reduces the size of the final correlation graph.

The work presented in this paper takes a different approach and simply relies on recursing backwards through the type-graph whenever a hyper-alert is input which has not had its prerequisites met by another hyper-alert in the system. Our method is at once more efficient and eliminates the consolidation step by terminating recursion as soon as a duplicate hypothesis is generated.

In [15] correlation and hypothesizing is performed, again, in batch mode. However in this case a state/event model is chosen so that evidence from complementary sources such as vulnerability analysis and raw audit logs. The attack model is converted in to a Bayesian network where prior probabilities are assigned manually by human experts. A sliding time window is used to limit memory usage and prevent a combinatorial explosion in run-time complexity associated with the Bayesian inference algorithm.

Our work is most similar to [7] in which in-memory indexes are used to significantly speed up correlation leaving the RDBMS just to store a log of hyper-alerts on disk. The most relevant contribution in their work is the proposal to index instances of predicates rather than hyper alerts. Their results indicate that the algorithm would be suitable for real-time operation but hypothesizing of missing alerts is not addressed and must presumably be performed as a post-processing step on the correlation graph. The work presented in this paper takes a different approach and instead indexes instances of the *PrepareFor* relation.

In summary, there are several automated correlation algorithms. Those which are suitable for real-time operation either rely on the defender being able to correctly and completely enumerate possible combinations of attacks on their protected network, or worse, rely on a sliding time window which opens up the correlator to “low and slow” or “alert injection” attacks. The abstract type-graph approach appears more promising and has been partly optimized for real-time deployment. Our work builds on prior techniques by using a novel indexing

structure and unifying the correlation and hypothesizing steps in to a single real-time algorithm.

3 Problem Definition

For the purposes of clarity the intrusion alert correlation (IAC) problem will be solved in two steps. Firstly the “minimal IAC problem” in which a totally accurate alert stream is input and no alerts are hypothesized and secondly; the “extended IAC problem” in which some alerts can be missing and the system must hypothesize alerts. The following problem definition is based on that proposed by Ning *et al.*[9, 10, 8, 11].

Definition 1. *An attack model consists of logical predicates, hyper-alert types and implication relations. A **hyper-alert type** T is a triple (fact, prerequisite, consequence) where fact is a set of attribute names associated with the type, prerequisite and consequence are sets of predicate expressions with free variables bound from fact. $Prereq(T)$ and $Conseq(T)$ denote the set of predicate expressions from the prerequisite and consequence elements of T respectively. For brevity we assume all implied expressions to be included in $Conseq(T)$. We shall refer to the set of all hyper-alert types in an attack model as τ .*

For the purpose of our examples we will assume that there are always 4 elements in *fact* (say, source address, source port, destination address, destination port).

Definition 2. *Given an ordered pair of hyper-alert types (A, B) then A **may prepare for** B if $Conseq(A)$ and $Prereq(B)$ share at least one predicate, with possibly different arguments.*

Definition 3. *Given an ordered pair of hyper-alert types (A, B) where A may prepare for B a set of **equality constraints** may be computed. Each such constraint is a set of logical conjunctions of equality comparisons between the attributes of the two types.*

Let the sequences u_1, u_2, \dots, u_n and v_1, v_2, \dots, v_n be distinct facts in type A and B respectively. Then each constraint takes the form:

$$u_1 = v_1 \wedge u_2 = v_2 \wedge \dots \wedge u_n = v_n$$

such that there exists $p(u_1, u_2, \dots, u_n) \in Conseq(A)$ and $p(v_1, v_2, \dots, v_n) \in Prereq(B)$ where p is the same predicate with possibly different fact assignments.

Note that the only substantial difference between our definition and that of Ning *et al.* is the restriction that any given fact may appear at most once in the arguments of a predicate. The purpose of this restriction will become clear in the following sections.

Definition 4. *Given an attack model, let us define an **attack-type graph** $TG = (V, E, C, T)$. Where (V, E) is a directed acyclic graph. T is a bijection of vertices on to hyper-alert types. An edge $e(v_0, v_1) \in E$ if and only if $T(v_0)$ may prepare for $T(v_1)$. C maps each edge to a set of constraints.*

Definition 5. A **hyper-alert** h is simply a tuple of attribute values. $Type(h)$ is a mapping on to the set of hyper-alert types. $Prereq(h)$ and $Conseq(h)$ denote the set of predicates from the prerequisite and consequence of the hyper-alert type with free variables bound from the attribute values of the hyper-alert. $Timestamp(h)$ denotes the timestamp of the hyper-alert. A **hyper-alert stream** is any time-ordered series of hyper-alerts.

Definition 6. A hyper-alert h of type A is said to **prepare for** a hyper-alert h' of type B if and only if $Type(h)$ may prepare for $Type(h')$ and at least one equality constraint evaluates to true when fact names have been substituted with actual values from the hyper alerts. Furthermore, since an event B can be the cause of an event C if and only if B occurs before C , an implicit time constraint ensures forward causality holds. In other words the directed edges in TG , like time, move from past to future.

Two hyper alerts are said to be correlated if and only if the former prepares for the latter. Since all that is required to correlate two hyper alerts is that any one of the constraints holds. We might say that each edge in TG is labeled with a predicate logical formula, consisting only of equality comparisons, in disjunctive normal form.

Definition 7. The output **correlation graph** CG is (V, E, H) where (V, E) is a DAG and H is a bijection of hyper-alerts to vertices and an edge $e(v_0, v_1) \in E$ if and only if $H(v_0)$ prepares for $H(v_1)$.

Definition 8. If a hyper-alert h exists where $Prereq(h)$ is non-empty and there does not exist a hyper-alert h' such that h' prepares for h then h is said to be “unexplained”.

An **unexplained alert** h may sometimes be explained by the construction of a sequence of hypothesized hyper alerts y_1, y_2, \dots, y_n such that y_n prepares for h , y_{n-1} prepares for y_n , ..., and a real (unhypothesised) hyper alert h' prepares for y_1 . There may be several alternative explanations for any such hyper-alert.

The **extended correlation graph** EG therefore consists of (V, E, H, Y) with the same definition as CG with the addition of Y , a mapping of vertices on to the set of hypothesised hyper-alerts which are required to explain any unexplained alerts in H . V is formed by the union of H and Y .

In summary our problem is to propose an algorithm which:

1. Is initialized with TG , and an empty CG .
2. At each time step:
 - (a) Input a hyper-alert.
 - (b) Construct a correct and complete CG as per definition 7 or, for the extended IAC problem, definition 8.

4 A Minimal Solution

The inner loop of our proposed algorithm consists of two steps. Firstly “searching for correlations” and secondly “marking of consequences”. When marking

consequences of a type T hyper-alert h we find each type T' such that T may prepare for T' . Then the equality constraints between the two types are used so as to index every possible combination of hyper-alert attributes for T' which should be considered prepared for by h . Each index entry created in this stage contains a pointer to h . Conversely when searching for correlations the indexes on type T are searched using the attributes of h . If an earlier hyper-alert h' has been input and marked its consequences it will be found during the searching for correlations stage if and only if h' prepares for h . The structure of our index is unique and, by indexing each attribute combination separately, the IAC is reduced to a sufficiently small constant number of search and insert operations on balanced binary trees[1] rather than multi-dimensional searches with wild-cards.

This approach exploits two properties of the structure of the problem. Firstly that time flows from past to future, meaning that prior alerts do not need to be checked and correlated twice. Secondly although the number of possible constraints on a given edge are exponentially related to the number of facts, in practice, the number of facts and therefore the maximum number of indexes required is small.

Lemma 1. *Given a pair of hyper-alert types (T_0, T_1) we take A and B to be their attribute sets. The sets of attributes are equipotent, each containing n elements. Each constraint may be represented as a set containing $0 \leq m \leq n$ ordered pairs of attributes (a, b) such that $a \in A$ and $b \in B$. No element of A may appear as a left component more than once, and no element of B may appear as a right component more than once since by definition 3 the problem is restricted to the simplified case in which each fact referred to in an equality constraint may make at most one appearance on each side of the equation.*

There are $P(n, m) \cdot C(n, m)$ ways to arrange m distinct pairs from n elements of A and n elements of B , where P and C are the permute and chose functions respectively. The number of possible equality constraints is therefore the sum of all constraints of each length m .

Proof. Our problem is to construct two sequences a_1, a_2, \dots, a_m and b_1, b_2, \dots, b_m where a_1 is paired with b_1 , a_2 is paired with b_2 , etc. We shall solve the problem in two separate steps. First we chose m elements of A and m elements of B and secondly we arrange the pairs. There are $C(n, m)^2$ ways to select a pair (A', B') where $A' \in$ the set of all m -combinations of elements in A and $B' \in$ the set of all m -combinations of elements in B . Now to pair them up we keep elements of A' in a fixed order and simply count the ways to permute the elements of B' . Since there are $m!$ ways to permute m attributes:

$$\begin{aligned}
 C(n, m)^2 \cdot m! &= \frac{n!}{m!(n-m)!} \cdot \frac{n!}{m!(n-m)!} \cdot m! \\
 &= \frac{n!}{m!(n-m)!} \cdot \frac{n!}{(n-m)!} \\
 &= C(n, m) \cdot P(n, m)
 \end{aligned}$$

□

Input: Hyper alert stream H , Hyper-alert types τ
Output: All pairs (h', h) such that h' prepares for h and both are in H
foreach $h \in H$ (*input in ascending time order*) **do**
 Let $T = \text{Type}(h)$;
 Let i be a index on $\text{ImplicitSet}(T, \tau)$;
 if $\text{Lookup}(i, h)$ **then**
 | Continue with next alert;
 end
 foreach *index i on $\text{IndexSet}(T, \tau)$* **do**
 Let the set of hyper-alerts $R = \text{Lookup}(i, h)$;
 foreach $h' \in R$ **do**
 | Add the pair (h', h) to Output;
 end
 end
 foreach *Type T' where T may prepare for T'* **do**
 foreach *Permutation p , index i on $\text{CorrelationSet}(T, T')$* **do**
 | Insert($i, \text{Permute}(h, p)$);
 end
 end
end

Algorithm 1. The minimal ATG algorithm

If we wish to count the maximum number of constraints when there is more than one type of attribute then we can re-use the formula above to count the ways of comparing the attributes of each type and take the product:

$$\prod_{i=1}^t \sum_{j=0}^{c_i} P(c_i, j) \cdot C(c_i, j) \quad (1)$$

Where t is the number of types, c_i is the number of attributes of the i^{th} type. Therefore, if we chose 4 facts: source and destination addresses and ports where addresses and ports are not comparable with each other. Then there are 49 possible constraints to an edge in TG . Since there are less combinations than permutations, the idea is to create an index for each of the 16 combinations of facts for each type. Permutations capture the possibly different orderings for the attributes in the equality constraints and will be used when inserting items in to the indexes.

Algorithm 1, requires several further definitions to determine which combinations of fields must be indexed for each type and how to evaluate what are the consequences for each hyper-alert so that they can be marked. A notion similar to implicit correlation in [6] is introduced. If two hyper-alerts have identical attribute values then they must also have identical consequences meaning that the correlation procedure is redundant the second time around. We define an implicit correlation so that all hyper-alerts of a given type are indexed based on the combination of fact values which are used in marking of consequences.

Definition 9. The *CorrelationSet* is a relation on a given pair of types (T, T') , such that $\text{CorrelationSet}(T, T')$ is a set of pairs of the form (a, b)

where a is a permutation of facts in T and b is an subset of facts in T' such that a and b are equipotent and there exists an equality constraint of the form $u_1 = v_1 \wedge u_2 = v_2 \wedge \dots \wedge u_n = v_n$ where sequence u_1, u_2, \dots, u_n is the elements of a arranged in to a fixed order and v_1, v_2, \dots, v_n is the sequence b .

Definition 10. The **Index Set** is a relation on a given type T and set of all types τ which returns subsets of facts in T which must be indexed. $IndexSet(T, \tau)$ returns every subset x of facts of T where there exists a T' such that T' may prepare for T and x is a right-component of $CorrelationSet(T', T)$.

Definition 11. The **Implicit Set** is a relation on a given type T and set of all types τ which returns a set of facts in T upon which future correlations may depend. $ImplicitSet(T, \tau)$ returns the union of every subset x of facts in T where there exists a T' such that T may prepare for T' and x is a left-component in an element of $CorrelationSet(T, T')$.

5 Hypothesising Missing Alerts

Algorithm 1 does not attempt to deal with missing alerts in the input alert stream. What should happen is that for any alert which arrives and is not explained by a prior alert then those alerts are hypothesized with appropriate fact values. This is done recursively until either a hyper-alert type with in-degree zero is found, no facts can be inferred for a hypothesis or until a real alert is found. Only if a real alert is found will the hypothesized sequence be entered in to the correlation graph. If no results are found in the “search for correlations” stage then the current alert is unexplained. Alerts are hypothesized with attributes satisfying each constraint on each incoming edge. Often times only a subset of the fact values may be inferred for a hypothesized alert as not all values have to be referred to in the equality constraints from the attack model.

This leads to a problem when we recurse more than one step. The recursion needs to terminate when a real hyper-alert may prepare for a hypothesized one. There is no guarantee that an index exists for the subset of fact values in the hypothesized alert. Our approach leads us to consider the hypothesizing problem as identical to the correlation problem, except that our hyper-alerts may contain a partial set of attribute values.

A pre-processing step is introduced in which an expanded version of the $IndexSet$ is calculated so that all such partial sets of attribute values are indexed. Also we introduce the relation $HypothesisSet(T, \tau)$ where T is a type and τ is the set of all hyper-alert types. This relations maps on to a set of 5-tuples with the components (t, i, p, m, o) :

1. t is a type which may prepare for T .
2. i is an element of the $IndexSet$ of t .
3. p is a permutation to apply to fact values of the current hyper-alert in order to query the index i of type t .
4. m is the combination of facts which appear in p .

5. o is the combination of facts that were originally required for the current constraint. In other words all facts mentioned on the right hand side of the equality comparisons for this constraint.

The hypothesizing algorithm then is a recursive procedure with two parameters the first of which is a *TG* vertex v' and the second is a hyper-alert h . The function returns *true* if a real hyper-alert was correlated or *false* otherwise. The procedure is that for each element in the *HypothesisSet* associated with v' :

1. Let f be the set of hypothesized fact attributes in h . Continue the loop if the union of f and o is not equal to m . This avoids generating unnecessary hypotheses based on a strict subset of the actually available fact attributes.
2. Let h' be a new hyper-alert. Use p to permute the facts in h and assign them to h' .
3. Create a key from h' which combines facts required for index i of t . Query i and if a result is found, correlate the result with h' and continue the loop.
4. Recurse to the vertex for type t passing hyper-alert h' . If the recursion returns true then correlate h' with h .

With this procedure hyper-alerts with identical attributes may be created in order to satisfy different paths through the attack graph even though they may eventually lead to the same place. Such alerts add nothing to the intelligibility of the result since one real alert could conceivably account for all such identical hypotheses.

We define two hyper-alerts as strategically indistinguishable provided that they are of the same type, have the same combination of facts assigned with the same values and appear before the hyper-alert they have been hypothesized to explain. Similarly to the implicit correlation step described in the previous section a hypothesized alert database is added to each vertex in the type-graph.

6 Empirical Results

To verify the theory the algorithm is implemented in C[3]. Trivial sub-graph elimination is implemented by keeping count of vertex degrees in *CG* as edges are added, only vertices with degree greater than zero are output. This small addition makes output graphs more manageable. The Lincoln Labs 1.0 dataset is used in the experiments for the purposes of generating results comparable to prior works. These data-sets include labeling data which allows for the construction of a perfectly accurate series of alerts. An attack model almost identical to that in [11] is used. The only difference is in fixing an error in the original in which UDP port-scans could be said to discover TCP services and *vice versa*, which is not the case. All experiments were run on a PC with 1.6GHz Intel Core 2 Duo CPU and 1GB RAM running a contemporary Linux distribution.

Two experiments are proposed: experiment #1 is designed to verify that the algorithm is suitable for application in a real-time correlation setting as intended. Experiment #2 is designed to qualitatively assess the hypothesizing algorithm when a random subset of relevant alerts have been deleted from a perfectly accurate alert stream.

6.1 Performance

The aim of this experiment is to test the suitability of our algorithm for real-time correlation. The method is to intersperse a true scenario consisting of 855 alerts within a large number of randomly generated alerts such that there are 1,000,000 alerts in total. No direct comparison with prior work is possible here since comparable algorithms are either not intended for real-time setting, do not perform the hypothesizing step or use a different attack model. Instead, the time taken for the software to perform the work will be recorded and divided by the number of alerts which will give us a correlation-rate. As long as the correlation-rate is higher than the rate at which we expect alerts to be produced by the underlying IDS then the algorithm ought to be suitable for real-time operation. The size of the output graphs is also recorded representing the bulk of the memory utilization of the program.

There are several parameters in this experiment. Firstly we will run the experiment with variations of the algorithm so that we can get an idea of the costs and benefits of each.

1. Algorithm 1. Minimal IAC problem.
 - (a) With implicit correlations disabled.
 - (b) With implicit correlations enabled.
2. Algorithm 2. Extended IAC problem.
 - (a) Without consolidating strategically indistinguishable hypotheses.
 - (b) Strategically indistinguishable hypotheses consolidated.

The question arises of how precisely to generate a large number of randomized false positive alerts. The attribute space is 96 bits in total, based on two 32 bit IP addresses, and two 16 bit port numbers. If all attributes are totally randomized the probability of false correlations being generated is exceedingly small. Conversely if we devise a non-random worst-case data-set in which false alarms are crafted specifically to generate correlations then we are venturing in to the area of specific attacks aimed at the correlator itself which is a problem beyond the scope of this paper.

The chosen solution is based on the observation that in a real-world setting the IDS is most often connected to a point in an IP network where it can observe all traffic entering or leaving that network. Therefore while one out of the source and destination addresses of a packet may be any of 2^{32} possible IP address values, the other side will be set to one of the addresses on the monitored network which will be a small subset of that address space. Traffic not conforming to these rules is taking place outside of the range of communications systems that the underlying IDS is placed to observe. Similarly, IP services tend to listen on well known ports, typically those under 1024.

Two randomization methods are chosen, one based on a class C IP network and the other on a class B network. These types of networks are defined as having 2^8 and 2^{16} addresses each. The algorithm for generating the data is:

1. Pick a totally random IP address and port number.
2. Pick a random IP address within the allowable range of our network class.
3. Pick a random 10 bit port number.
4. Toss a coin, if heads then the fully random IP is the source address, else it's the destination address.
5. Toss another coin, if heads then the fully random port number is the source address, else *vice versa*.

Five versions of the random data-set are created for each type of network, making ten data sets in total. Each of the four variations of the algorithm were run on each of the 10 data-sets making 40 runs in total. Each run is repeated three times and the mean CPU time taken as the final result. The variation in run time on the program on the same data-set turned out extremely low so, for the sake of concision, the individual run-timings are not presented here. The 885 real alerts from the LLDOS labeling data are interspersed randomly, but correctly ordered, within each dataset.

If we look at the final column of table 1 we can observe that the correlation rate is on the order of 100,000 alerts per second. This seems likely to be much faster than an IDS, certainly the majority of deployments in any case.

In table 2 the columns stand for the total number of vertices and edges in the output *CG* respectively. The number of false alerts seems rather alarming considering only 885 of them are part of our scenario. Although, bear in mind that our noise is distributed over only 20 alert types which are quite highly connected. Further we have opted to restrict alert values to “realistic” ranges. In practice a million alerts do not occur over a few seconds but perhaps days or weeks.

6.2 Quality of Output

The aim of this experiment is to take the same totally accurate data-set and remove random alerts and test the accuracy of hypothesizing by how accurate the the graphs are as an increasing number of alerts are missed. Unfortunately the number of ways of doing this with a data-set of of 855 alerts, such as ours, is astronomical and our sample sizes would have to be inappropriately large to gain

Table 1. CPU Times for Class B and Class C Respectively

Class	Exp.	Min. (s)	Max. (s)	Mean (s)	Mean Rate (a/s)
B	1(a)	7.47667	9.21	7.905	126,502
	1(b)	5.31	6.26333	5.675	176,221
	2(a)	6.55333	6.67667	6.599	151,541
	2(b)	6.45333	6.48	6.461	154,772
C	1(a)	7.0533	7.14667	7.088	141,088
	1(b)	6.79667	6.87	6.818	146,675
	2(a)	11.46	11.6033	11.52	86,380
	2(b)	10.9067	19.9233	12.43	80,440

Table 2. Output size for Class B and Class C Respectively

Exp.	Hyper-Alerts	Correlations	Hyper-Alerts	Correlations
	Class B		Class C	
1(a)	194,817	157,734	346,782	888,262
1(b)	182,727	148,457	129,220	641,115
2(a)	376,786	401,974	190,986	691,809
2(b)	299,395	302,553	190,417	691,112

results which can be interpreted with any confidence. From experience the algorithm is extremely robust either when all alerts of one or two types are removed or scores of alerts removed randomly. This intuition leads us on to an alternative experimental setup. There are only four types of alerts in the experimental data set. At least two types are required for there to be correlations and if all alerts are present then the output is ideal. We shall experiment with removing all 2 and 3 combinations of alert types and examining the false correlation rates which are calculated by hand in this case.

These experiments are run with Algorithm 2(b) only. To calculate false alert rates the output graphs are compared against the complete correlation graph which contains 58 hyper-alerts. A false negative is counted for every alert in the complete *CG* for which no hypothesis exists. Conversely a false positive is counted for every hypothesis which does not correspond to a hyper-alert in the complete *CG*. For labeling purposes alert types are named A, B, C and D, standing for *ping-sweep*, *sadmind-ping*, *sadmind-exploit* and *mstream-zombie* respectively.

The results in table 3 are difficult to analyze without taking a closer look at the output graphs produced. For attack sequences which are short in length, missing alerts can have a drastic effect on the false negative correlation rates. False positive hypotheses are a slightly less serious problem and in this case would be entirely eliminated with existing audit-record correlation techniques, as proposed in [15].

Table 3. Hypothesis Accuracy

Input Types	False Negatives	False Positives
ABD	3	12
BCD	32	0
ACD	26	0
ABC	14	0
AC	37	0
BD	35	12
CD	41	0
BC	44	0
AD	35	12
AB	20	0

7 Conclusions and Future Work

In this paper a real-time correlation algorithm using hyper-alert type graphs was proposed. Our general approach was to reduce the minimal IAC problem to a series of insertions to and removals from a balanced binary tree. We proceeded from there to approach the extended (hypothesizing) problem by re-phrasing the minimal problem such that we can recursively input hyper-alerts with unknown (or wild-card) attributes. It was shown that such algorithms are feasible provided a few conditions are met:

- The number of comparable facts in hyper-alerts is small.
- If hyper-alerts are to be hypothesized then type-graphs should be chosen carefully in order to prevent a exponential explosion in time complexity.

The algorithm was implemented and validated through a series of experiments which showed that a good implementation is suitable for real-time correlation even in cases where the IDS alert rate is alarmingly high. In these cases the size of the output graph becomes the overriding factor in determining the practical utility of the algorithm. It was also confirmed that picking the right aggregation function is invaluable in this respect by allowing many hyper-alerts to be merged in to a single logical unit. However it is not immediately clear how best to design these functions such as to minimize large output graphs to a satisfactory degree.

Although our approach does not require a vulnerability assessment it has been shown that it is possible to make use of such information if it is there [15]. It appears that our algorithm could be modified for similar purposes. The basic approach here would be to incorporate special constraints which depend on external evidence sources. These would be checked before correlating or hypothesizing an alert. However this leads to question of how to determine when to ignore false negative vulnerability assessments if a successful attack of the relevant type has been observed? This may also be a fruitful direction for investigation.

References

- [1] Bayer, R.: Symmetric Binary B-Trees: Data structure and maintenance algorithms. *Acta Inf.* 1, 290–306 (1972)
- [2] Xu, D., Ning, P.: Alert Correlation through Triggering Events and Common Resources. In: *Proc. 20th Annual Computer Security Applications Conference* (2004)
- [3] Tedesco, G.: ATG correlator source code and documentation (2008), <http://www.scaramanga.co.uk/atg/>
- [4] Tedesco, G., Twycross, J., Aickelin, U.: Integrating innate and adaptive immunity for intrusion detection. In: *Proc. International Conference on Artificial Immune Systems* (2006)
- [5] Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S.: Computer Attack Graph Generation Tool. In: *Proc. DARPA Information Survivability Conference & Exposition II* (2000)
- [6] Wang, L., Liu, A., Jajodia, S.: An Efficient, Unified Approach to Correlating, Hypothesizing, and Predicting Intrusion Alerts. In: *Proc. European Symposium on Computer Security* (2005)

- [7] Ning, P., Xu, D.: Adapting Query Optimization Techniques for Efficient Intrusion Alert Correlation. Technical Report TR-2002-14 NCSU Dept. of Computer Science (2002)
- [8] Ning, P., Xu, D.: Hypothesizing and Reasoning about Attacks Missed by Intrusion Detection Systems. *ACM Transactions on Information and System Security* 7(4), 591–627 (2004)
- [9] Ning, P., Cui, Y., Reeves, D.S.: Analyzing Intensive Intrusion Alerts Via Correlation. In: Wespi, A., Vigna, G., Deri, L. (eds.) RAID 2002. LNCS, vol. 2516. Springer, Heidelberg (2002)
- [10] Ning, P., Cui, Y., Reeves, D.S.: Constructing Attack Scenarios through Correlation of Intrusion Alerts. In: Proc. 9th ACM Conference on Computer & Communications Security, pp. 245–254 (2002)
- [11] Ning, P., Xu, D., Healy, C.G., Amant, R.S.: Building Attack Scenarios through Integration of Complementary Alert Correlation Methods. In: Proc. 11th Annual Network and Distributed System Security Symposium, pp. 97–111 (2004)
- [12] Deraison, R.: Nessus automated vulnerability scanner (2008), <http://www.nessus.org/>
- [13] Templeton, S.J., Levitt, K.: Requires/Provides Model for Computer Attacks. In: Proc. Workshop on New Security Paradigms (2000)
- [14] Noel, S., Jajodia, S., O’Berry, B.: Topological Analysis of Network Vulnerability. In: Managing Cyber Threats: Issues Approaches and Challenges (2005)
- [15] Zhai, Y., Ning, P., Iyer, P., Reeves, D.S.: Reasoning about Complementary Intrusion Evidence. In: Proc. 20th Annual Computer Security Applications Conference (2004)

Incorporation of Application Layer Protocol Syntax into Anomaly Detection

Patrick Düssel¹, Christian Gehl¹, Pavel Laskov^{1,2}, and Konrad Rieck¹

¹ Fraunhofer Institute FIRST

Intelligent Data Analysis, Berlin, Germany

² University of Tübingen

Wilhelm-Schickard-Institute for Computer Science, Tübingen, Germany

{duessel, gehl, laskov, rieck}@first.fraunhofer.de

Abstract. The syntax of application layer protocols carries valuable information for network intrusion detection. Hence, the majority of modern IDS perform some form of protocol analysis to refine their signatures with application layer context. Protocol analysis, however, has been mainly used for misuse detection, which limits its application for the detection of unknown and novel attacks. In this contribution we address the issue of incorporating application layer context into anomaly-based intrusion detection. We extend a payload-based anomaly detection method by incorporating structural information obtained from a protocol analyzer. The basis for our extension is computation of similarity between attributed tokens derived from a protocol grammar. The enhanced anomaly detection method is evaluated in experiments on detection of web attacks, yielding an improvement of detection accuracy of 49%. While byte-level anomaly detection is sufficient for detection of buffer overflow attacks, identification of recent attacks such as SQL and PHP code injection strongly depends on the availability of application layer context.

Keywords: Anomaly Detection, Protocol Analysis, Web Security.

1 Introduction

Analysis of application layer content of network traffic is getting increasingly important for protecting modern distributed systems against remote attacks. In many cases such systems must deal with untrusted communication parties, e.g. the majority of web-based applications. Application-specific attack can only be detected by monitoring the content of a respective application layer protocol.

Signature-based intrusion detection systems (IDS) possess a number of mechanisms for analyzing the application layer protocol content ranging from simple payload scanning for specific byte patterns, as in Snort [19], to protocol analysis coupled with a specialized language for writing signatures and policy scripts, as in Bro [15]. By understanding the protocol context of potential attack patterns, significant improvements in the detection accuracy of unknown application layer attacks can be achieved.

The main drawback of signature-based IDS is their dependence on the availability of appropriate exploit signatures. Rapid development of new exploits and their growing variability make keeping signatures up-to-date a daunting if not impossible task. This motivates investigation of alternative techniques, such as anomaly detection, that are capable to detect previously unknown attacks.

Incorporation of the protocol context into anomaly detection techniques is, however, a difficult task. Unlike a signature-based system which looks for a *specific pattern* within a *specific context*, an anomaly-based system must translate *general knowledge* about patterns and their context into a numeric measure of abnormality. The latter is usually measured by a distance from some typical “profile” of a normal event. Hence, incorporation of protocol syntax into the computation of distances between network events (e.g. packets, TCP connections etc.) is needed in order to give anomaly detection algorithm access to protocol context.

The idea behind the proposed method for syntactically aware comparison of network event is roughly the following. A protocol analyzer can transform a byte stream of each event into a structured representation, e.g. a sequence of token/attribute pairs. The tokens correspond to particular syntactic constructs of a protocol. The attributes are byte sequences attached to the syntactic elements of a protocol. Measuring similarity between sequences is well understood, for general object sequences [18], as well as byte streams of network events [16; 17; 23]. To compare sequences of token/attribute pairs we perform computation of sequential similarity at two levels: for sequences of tokens (with partial ordering) and byte sequence values of corresponding tokens. The resulting similarity measure, which we call the *attributed token kernel*, can be transformed into a Euclidean distance easily handled by most anomaly detection algorithms.

To illustrate effectiveness of the protocol syntax-aware anomaly detection, we apply the proposed method for detection of web application attacks. Such attacks, for example SQL injection, cross-site scripting (XSS) and other script injection attacks, are particularly difficult for detection due to (a) their variability, which makes development of signature a futile exercise, and (b) entanglement of attack vector within the protocol framework, which makes simple byte-level content analysis ineffective. Our experiments carried out on client-side HTTP traffic demonstrate a strong performance improvement for these kinds of attacks compared to byte-level analysis. The proposed method should be easily adaptable for other application layer protocols for which a protocol dissector is available.

2 Related Work

A large amount of previous work in the domain of network intrusion detection systems has focused on features derived from network and transport layer protocols. An example of such features can be found in the data mining approach of Lee and Stolfo [9], containing packet, connection and time window features derived from IP and TCP headers. The same work has pioneered the use of

“content” features that comprised selected application-level properties such as number shell prompts, number of failed login prompts, etc. deemed to be relevant for detection of specific attacks. Similar features comprising selected keywords from application layer protocols have been used by Mahoney and Chan for anomaly detection [12].

General content-based features using payload distribution of specific byte groups have been first proposed by Kruegel et al. [7] in the context of service-specific anomaly detection using separate normality models for different application layer protocols. Full distributions of byte values have been considered by Wang and Stolfo [23], extended to models of various languages that can be defined over byte sequences, e.g. n -grams [16; 22].

Incorporation of application-level protocol information into the detection process has been first realized in signature-based IDS. Robust and efficient protocol parsers have been developed for the Bro IDS [15]; however, until recently they were tightly coupled with Bro’s signature engine, which has prevented their use in other systems. The development of a stand-alone protocol parser binpac [14] has provided a possibility for combining protocol parsing with other detection techniques. Especially attractive features of binpac are incremental and bi-directional parsing as well as error recovery. These issues are treated in depth in the recent. Similar properties at a more abstract level are exhibited by the recent interpreted protocol analyzer GAPAL [1].

Combination of protocol parsing and anomaly detection still remains largely unexplored. By considering separate models corresponding to specific URI attributes in the HTTP protocol, Kruegel and Vigna [8] have developed a highly effective system for the detection of web attacks. The system combines models built for specific features, such as length and character distribution, defined for attributes of applications associated with particular URI paths. Ingham et al. [6] learn a generalized DFA representation of tokenized HTTP requests using delimiters defined by the protocol. The DFA inference and the n -grams defined over an alphabet of protocol tokens performed significantly better than other content-based methods in a recent empirical evaluation [5]. Our approach differs from the work of Ingham and Inoue in that our method explicitly operates on a two-tier representation – namely token/attribute pairs – obtained from a full protocol analyzer (binpac/Bro) which provides a more fine-grained view on HTTP traffic.

3 Methodology

A payload-based anomaly detection approach benefits from its ability to cope with unknown attacks. The architecture of our system which is specifically built for the requirements of anomaly detection at application layer is illustrated in Fig. 1. The following four stages outline the essential building blocks of our approach and will be explained in detail for the rest of this section.

1. **Protocol Analysis.** Inbound packets are captured from the network by Bro which provides robust TCP re-assembly and forwards incoming packets

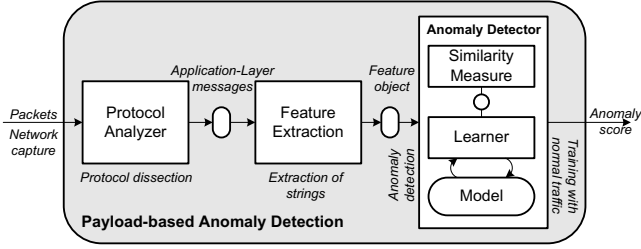


Fig. 1. Architecture of payload-based anomaly detection

to the *binpac* protocol analyzer. The latter extracts application-layer events at different levels of granularity, typical for common text protocols such as HTTP, FTP, SMTP or SIP. Initially, extraction starts at the level of request/response messages and can be further refined to specific protocol elements. A key benefit of using protocol dissectors as part of data pre-processing is the capability to incorporate expert knowledge into the feature extraction process. Details on protocol analysis can be found in Section 3.1.

2. **Feature Extraction.** Each parsed event is mapped into a feature vector which reflects essential characteristics. However, an event can be projected into byte-level or syntax-level feature spaces. Our approach allows to combine both. Details of the feature extraction process can be found in Section 3.2.
3. **Similarity Computation.** The similarity computation between strings is a crucial task for payload-based anomaly detection. Once a message is brought into a corresponding vectorial representation two events can be compared by computing their pairwise distance in a high-dimensional geometric space. We extend the common string similarity measures for token/attribute representations provided by a protocol parser, as explained in Section 3.4.
4. **Anomaly Detection.** In an initial training phase the anomaly detection algorithm learns a global model of "normality" which can be interpreted as a center of mass of a subset of training data. At detection time an incoming message is compared to a previously learned model and based on its distance an anomaly score is computed. The anomaly detection process is described in Section 3.3.

3.1 Protocol Analysis

Network protocols specify rules for syntax, semantics, and synchronization for bidirectional data communication between network endpoints. The protocol syntax is usually defined by an augmented Backus-Naur Form.

Our goal is to analyze network traffic based on the grammatical characteristics of an underlying protocol in order to detect network attacks. We perform the analysis at the granularity of protocol elements present in request/response messages. The HTTP protocol definition is a classical representative of such request/response protocols. Additionally, HTTP is the most frequently used protocol for web applications and so we limit our focus to this particular specification.

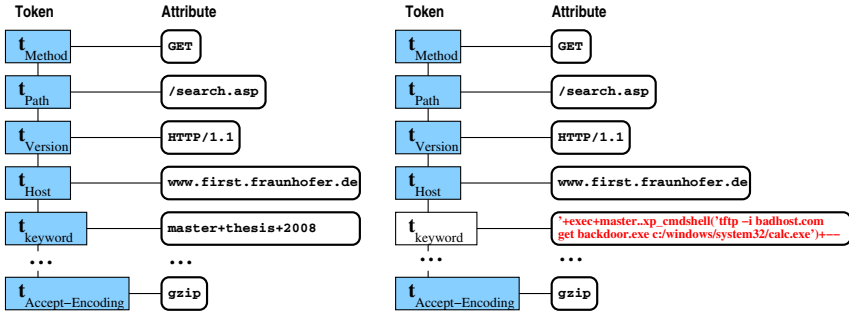


Fig. 2. Attributed token sequence of a benign and malicious HTTP request

Usually, a request is transmitted in a single TCP packet, although certain features of the TCP/IP (e.g. fragmentation) can make the process more complicated. An application protocol analyzer, e.g. binpac [14], allows one to transform the network event’s raw byte payload into a structured representation reflecting the syntactic aspects of an underlying application protocol. For example in the syntactic context of HTTP the sequence “Content-Length: 169” refers to a header “Content-Length” with attribute “169”.

We present two examples of HTTP connections to illustrate the effect of applying binpac’s grammar and parser to an application-level specification. The first example is a benign GET request containing common HTTP headers together with a CGI parameter.

```
GET /search.asp?keyword=master+thesis+learning HTTP/1.1\r\nHost: www.firs
t.fraunhofer.de\r\nUser-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.1; SV1)\r\nConnection: Keep-alive\r\nAccept-Encoding: gzip\r\n\r\n
```

The second example shows a SQL injection attack against a Microsoft SQL Server exploiting the vulnerable CGI parameter *keyword*.

```
GET /search.asp?keyword='+exec+master..xp_cmdshell('tftp -i badhost.com g
et backdoor.exe c:/windows/system32/calc.exe')+-- HTTP/1.1\r\nHost: www.f
```

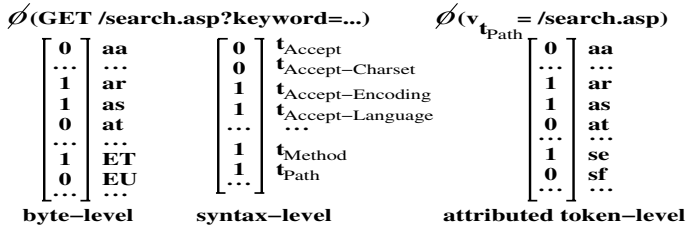


Fig. 3. Feature extraction for byte-level, syntax-level and attributed token-level

```
irst.fraunhofer.de\r\nUser-Agent: Mozilla/5.0\r\nConnection: Keep-alive\r\nAccept-Language: en-us,en\r\nAccept-Encoding: gzip\r\n\r\n
```

The token/attribute pairs generated by the protocol analyzer are shown in Fig. 2. One can clearly see that the difference between these two requests is given by the attributes attached to the token “keyword” of the parsed GET request.

3.2 Feature Extraction

Anomaly detection usually requires data to be in a vectorial representation. Therefore, the feature extraction process maps application layer messages, such as HTTP requests, into a feature space in which similarity between messages can be computed. Protocol dissection in the pre-processing stage allows to deploy feature extraction on two different levels explained in the following.

Byte-Level Features. An intuitive way to represent a message at byte level is to extract unique substrings by moving a sliding window of a particular length n over a message. The resulting set of feature strings are called n -grams. The first example in Fig. 3 shows the mapping of the benign HTTP request introduced in Section 3.1 into a binary 2-gram feature space.

Syntax-Level Features. Each message is transformed into a set of tokens. Although shown as a sequence of tokens in Fig. 2, these feature have only partial sequential order as the order of many (but not all) tokens in an HTTP request is not defined. An embedding of the benign HTTP request into a token feature space is exemplarily presented in Fig. 3.

With the extraction of byte-level features from token attributes a semantic notion is implicitly assigned to the corresponding protocol token. Note, that an explicit representation of application layer messages can easily become computational infeasible due to the combinatorial nature of byte sequences. Therefore, we use efficient data structures such as suffix trees or hash tables (“feature objects” in Fig. 1) that allow an implicit vectorial representation.

3.3 Anomaly Detection

Once, application layer messages are mapped into some feature space the problem of anomaly detection can be solved mathematically considering the geometric relationship between vectorial representations of messages. Although anomaly detection methods have been successfully applied to different problems in intrusion detection, e.g. identification of anomalous program behavior [e.g. 3; 4], anomalous packet headers [e.g. 11] or anomalous network payloads [e.g. 8; 16; 17; 22; 23], all methods share the same concept – *anomalies are deviations from a model of normality* – and differ in concrete notions of normality and deviation. For our purpose we use the one-class support vector machine (OC-SVM) proposed in [21] which fits a minimal enclosing hypersphere to the data which is characterized by a center c and a radius R as illustrated in Fig. 4.

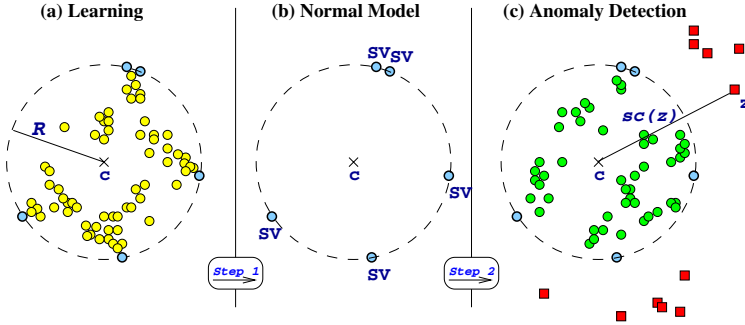


Fig. 4. Learning and anomaly detection using a one-class support vector machine

Mathematically, this can be formulated as a quadratic programming optimization problem:

$$\begin{aligned}
 \min_{\substack{R \in \mathbb{R} \\ \xi \in \mathbb{R}^n}} \quad & R^2 + C \sum_{i=1}^n \xi_i \\
 \text{subject to:} \quad & \|\phi(x_i) - c\|^2 \leq R^2 + \xi_i, \\
 & \xi_i \geq 0.
 \end{aligned} \tag{1}$$

By minimizing R^2 the volume of the hypersphere is minimized given the constraint that training objects are still contained in the sphere which can be expressed by the constraint in Eq.(1). A major benefit of this approach is the control of generalization ability of the algorithm [13], which enables one to cope with noise in the training data and thus dispense with laborious sanitization, as recently proposed by Cretu et al. [2]. By introducing slack variables ξ_i and penalizing the cost function we allow the constraint to be softened. The regularization parameter C controls the trade-off between radius and errors (number of training points that violate the constraint). The solution of the optimization problem shown in Eq. (1) yields two important facts:

1. The center $c = \sum_i \alpha_i \phi(x_i)$ of the sphere is a linear combination of data points, while α_i is a sparse vector that determines the contribution of the i -th data point to the center. A small number of training points having $\alpha_i > 0$ are called **support vectors** (SV) which define the model of normality as illustrated in Fig. 4.
2. The radius R which is explicitly given by the solution of the optimization problem in Eq. (1) refers to the distance from the center c of the sphere to the boundary (defined by the set of support vectors) and can be interpreted as a threshold for a decision function.

Finally, having determined a model of normality the anomaly score $sc(z)$ for a test object z can be defined as the distance from the center:

$$sc(z) = \|\phi(z) - c\|^2 = k(z, z) - 2 \sum_i \alpha_i k(z, x_i) + \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j), \tag{2}$$

where the similarity measure $k(x, y)$ between two points x and y refers to a *kernel* function which allows to compute similarity between two data points embedded in some geometric feature space \mathcal{F} . Given an arbitrary mapping $\phi : \mathcal{X} \mapsto \mathcal{F}$ of a data point $x \in \mathcal{X}$ a common similarity measure can be defined from the *dot product* in \mathcal{F} :

$$k(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_{i=1}^N \phi_i(x) \phi_i(y), \quad (3)$$

where $\phi_i(x)$ refers to the i -th dimension of a data point x mapped into \mathcal{F} .

3.4 Similarity Measures

Having defined a set of characteristic features we can develop similarity measures that, given two data points, returns a real number reflecting their similarity.

Spectrum Kernel. A natural way to define a kernel $k(s, u)$ between two application layer messages s and u is to consider n -grams that both messages have in common. Given the set \mathcal{A}^n of all possible strings of length n induced by an alphabet \mathcal{A} we can define a kernel function computing the dot product of both messages embedded into a $|\mathcal{A}|^n$ dimensional feature space.

$$k(s, u) = \langle \phi(s), \phi(u) \rangle = \sum_{w \in \mathcal{A}^n} \phi_w(s) \phi_w(u), \quad (4)$$

where $\phi_w(s)$ ¹ refers to a signum function that returns 1 if the w is contained in s and 0 otherwise. The kernel function $k(s, u)$ is referred to as *spectrum kernel* [10]. Using n -grams to characterize messages is intuitive but may result in a high-dimensional feature space. From an algorithmic point of view, it may seem that running a summation over all possible substrings $w \in \mathcal{A}^n$ can become computationally infeasible. Thus, special data structures such as tries, suffix trees or suffix arrays enable one to compute $k(s, u)$ in $O(|s| + |u|)$ time [20]. Interestingly, the Euclidean distance $d_{eucl}(s, u)$ which is of particular interest for anomaly detection can be easily derived from the above kernel formulation:

$$d_{eucl}(s, u) = \sqrt{k(s, s) + k(u, u) - 2k(s, u)}. \quad (5)$$

Attributed Token Kernel. In this section we address the problem of how to combine string similarity as defined in Section 3.4 and structural similarity of two application layer messages. Consider an alphabet $\mathcal{A} = t_1, t_2, \dots, t_z$ of tokens. Let $s = s_1, s_2, \dots, s_n$ and $u = u_1, u_2, \dots, u_m$, $s_i, u_j \in \mathcal{A}$, be the token sets of two application layer messages returned by a protocol analyzer. Let v_t^u denote an attribute attached to the token t found in a token set u . We can define a kernel function for attributes in the same way we have done it in Eq.(4):

$$k_t(s, u) = \begin{cases} k(v_t^s, v_t^u), & \text{if } t \text{ is found in both } s \text{ and } u \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

¹ Alternatively, the embedding function $\phi_w(s)$ may return the count or the term frequency of w in s .

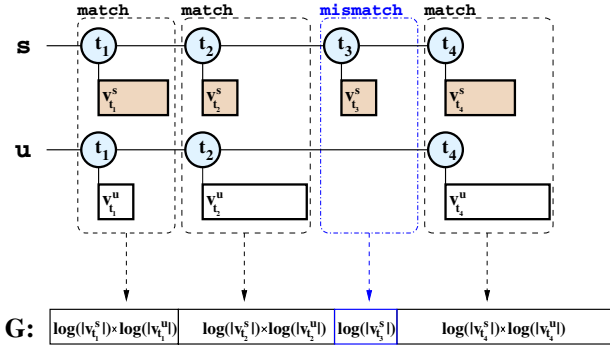


Fig. 5. Normalization of the similarity measure between attributed token sequences

Finally, we combine definitions (4) and (6) in the following way:

$$k(s, u) = \sum_{t \in s \cap u} \frac{\gamma(t)}{G} k_t(s, u), \quad (7)$$

where $s \cap u$ is an intersection of tokens in s and u , $\gamma(t)$ is a weight assigned to a particular token, and G is an overall normalization constant. The computation of the kernel function in Eq. (7) runs through all matching tokens and computes the similarity measure defined in Eq. (4) over associated attributes at byte level.

The weighting constant is defined as:

$$\gamma(t) = \log(|v_t^s|) \times \log(|v_t^u|),$$

and the normalization constant is defined as:

$$G = \sum_{t \in s \cup u} \gamma(t).$$

These constants are motivated by the need to normalize contributions from individual value sequences according to their lengths. The overall normalization constant also includes contributions from mismatching tokens. This allows the latter to indirectly influence the similarity measure by rescaling contributions from matching tokens; direct influence is not possible since it does not make any sense to compare value strings for mismatching tokens. For the rest of this paper we refer to the kernel function defined in Eq. (7) as *attributed token kernel*.

4 Experiments

We evaluate the impact of incorporation of protocol context into anomaly detection on two data sets containing HTTP traffic. Both data sets comprise exploits taken from the Metasploit framework² as well as from common security mailing lists and archives such as xssed.com, sla.ckers.org or Bugtraq.

² <http://www.metasploit.com/>

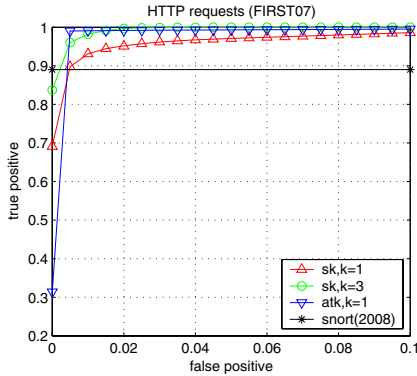
The first dataset (FIRST07) contains a sample of 16000 normal HTTP connections drawn from two months incoming traffic recorded at our institute’s web server. We mixed normal data with 42 attack instances of 14 different types, mostly **overflow-based attacks** (8 stack overflows, 4 heap overflows, 1 format string exploit, 1 web application attack). Except for using unsanitized data, this is a typical experimental protocol used in previous work, e.g. [5; 8; 23].

The second data set (NYT08) has been motivated by the observation that a traffic profile of a mostly static web site may be quite different from a profile of a site running web applications. In particular, a much more involved structure of URI and certain header fields can be expected in normal traffic, which may cause significantly higher false positive rates than the ones reported in evaluations on static normal traffic. Since we do not have access to a “pure” web application traffic, as e.g. the Google data used by Kruegel and Vigna [8], we have attempted to simulate such traffic using specially developed crawlers. Our request engine analyzes the structure and content of existing static traffic (in our case, the FIRST07 data) and generates valid HTTP requests using frequently observed protocol header elements while randomly visiting a target domain. We generated 15000 client-side connections accessing the *nytimes.com* news site and mixed the traffic with 17 instances of **web application attacks** (1 buffer overflow, 1 arbitrary command execution vulnerability, 3 Perl injection, 2 PHP include, 4 SQL injections and 6 XSS attacks). XSS attacks and SQL injections were launched against two prototypical CGI programs that were adapted to the structure of a “login”-site (8 CGI parameters) and a “search”-site (18 CGI parameters) found in the *nytimes.com* domain. In order to obtain a normal behavior for these sites we generated 1000 requests containing varying user names and passwords as well as search phrases and realistic parameter values for both prototypical “mirrors” of NYT sites. In contrast to previous work in which the structure of HTTP requests in exploits was used “as is”, we have also normalized the attacks by using the same typical headers injected by crawlers, in order to avoid attacks being flagged as anomalous purely because of programmatic but non-essential difference in their request structure.

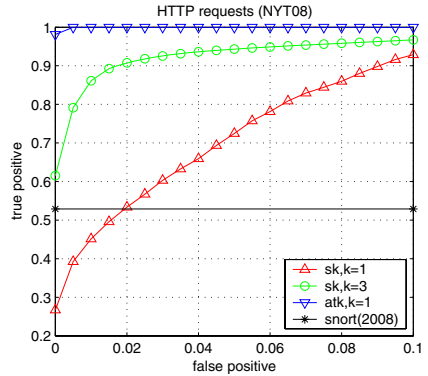
In our experiments, a model was trained using 500 requests taken from a normal pool of data and subsequently applied to 500 unknown requests taken from a distinct test set mixed with attacks. The detection accuracy was measured in terms of area under receiver operating characteristic curve ($ROC_{0,1}$) for false positive rates $\leq 10\%$. For statistical reasons experiments were repeated and the detection accuracy was averaged over 50 repetitions.

4.1 Detection Accuracy: Byte-Level Versus Attributed Token-Level

In the experiment on FIRST07 we investigate the detection of overflow-based attacks in HTTP requests. As shown in Fig. 6(a), the spectrum kernel **sk** on binary 3-grams attains a detection rate of 82% at 0% false positives anomaly, which is comparable to Snort. The only attack that repeatedly suffered a high false positive rate (1.5%-2%) is a *file inclusion* (“php_include”) which is the only non-buffer-overflow attack in this data set. Similar results have been obtained



(a) ROC curve of OC-SVM on the FIRST07 data set (overflow attacks)



(b) ROC curve of OC-SVM on the NYT08 data set (web application attacks)

Fig. 6. Detection accuracy on intrusion detection datasets FIRST07 and NYT08

for the attributed token kernel except for some initial false positives due to proxy requests containing anonymized header attributes.

In the experiment on NYT08 we investigate the detection of attacks that are bound to specific parameters of a CGI program. The results presented in Fig. 6(b) reveal that the detection of *web application attacks* using byte-level similarity over n -grams is much more difficult than for overflow-based attacks (Fig. 6(a)). The OC-SVM achieves its best detection rate of approximately 64% at zero percent false positives using a spectrum kernel (**sk**) over 3-grams and a binary feature embedding. Moreover, it can be observed that the accuracy of byte-level detection rises by increasing the size of n -grams. As illustrated in Fig. 6(b) significant improvements can be obtained by incorporating structure of application layer messages into payload-based anomaly detection as illustrated in Fig. 2. It turns out that anomaly detection using the attributed token kernel (**atk**) achieves a 97% detection rate without suffering any false positive.

In order to assess the computational effort of our method we measured the runtime for involved processing steps. Parsing and feature extraction require on average 1.2ms per request; average kernel computation and anomaly detection take 3.5ms per request. The overall processing time of 4.7ms per request yields a throughput of approximately 212 HTTP requests per second.

4.2 Visualization of Discriminative Features

In order to illustrate the increase of discriminative power gained through incorporation of application layer context into anomaly detection Fig. 4.2 displays 1-gram frequency differences between 1000 normal HTTP requests and two different attacks, a buffer overflow and a SQL injection. A frequency difference of 1 arises from bytes that only appear in normal data points whereas bytes that can be exclusively observed in the attack instance result in a frequency difference of -1. Bytes with a frequency difference close to zero do not provide

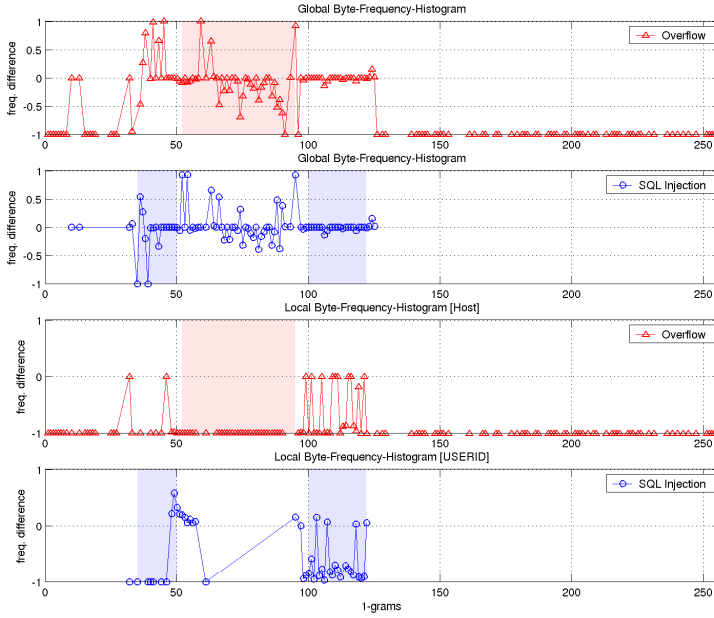


Fig. 7. Byte-frequency differences at request-level and token-level

discriminative information. The upper two charts display frequency differences for a buffer overflow attack (“edirectory_host_shikata_ga_nai”) and a SQL injection (“sql_union_injection”) that results from conventional byte-level analysis. The buffer overflow can be easily detected due to the presence of a large amount of non-printable characters (a large number of points at -1 for lower byte values and values above 128). On the other hand, the SQL injection contains mostly ASCII characters, which is reflected by close to zero frequency differences for most of the printable characters. As a consequence, the detection of this attack is relatively difficult. The lower two charts show the frequency differences for the same attacks within their appropriate application layer context. The frequency differences of the buffer overflow attack become even more obvious by examining the local byte distribution within the exploited request parameter “Host”. Similar clarification takes place for the SQL injection attack for which many previously normal bytes become clearly anomalous considering the CGI parameter “USERID”. This results in a major improvement of detection accuracy.

4.3 Comparison with Other Methods

To give a comparison to different intrusion detection techniques we examined a model-based anomaly detection method (**model-based AD**) proposed by Kruegel and Vigna [8] and the well-known signature IDS **Snort**³. The model-based detection method applies a number of different models to individual

³ VRT Certified Rules (registered user release) downloaded 07/15/2008.

Table 1. Comparison of fp-rates between model-based AD, Snort and OC-SVM

HTTP attacks (NYT08 dataset)	model-based AD			Snort	OC-SVM	
	ALM	ICD	Combined		sk	atk
edirectory_host_alpha_mixed	.0728	.0208	.0362	+	.0	.0
sql_l=1	.0112	.0379	.0214	—	.1798	.0002
sql_backdoor	.0	.0006	.0	+	.0004	.0
sql_fileloader	.0025	.0006	.0010	—	.0065	.0
sql_union_injection	.0	.0020	.0	—	.0047	.0
xss_alert	.0	.0	.0	+	.0388	.0
xss_dom_injection	.0	.0	.0	+	.0001	.0
xss_img_injection	.0	.0	.0	—	.0004	.0

parameters of HTTP queries. By learning various parameter models this method creates a "user profile" for each server-side program that is compared against incoming requests for a particular resource. To allow a fair comparison to our method we build models of not only CGI parameters in the URI, but also of header parameters and CGI parameters present in the request's body.

In our experiments the model-based AD comprises two models:

Attribute length model (ALM) estimates the attribute length distribution of CGI parameters and detects data points that significantly deviate from normal models.

Attribute character distribution model learns the *idealized character distribution* (ICD) of a given attribute. Using ICD instances are detected whose sorted frequencies differ from the previously learned frequency profile.

A comparison of false positive rates per attack class (percentage of false positives in test set given a detection of all instances from that attack class) is provided in Table 1. Anomaly detection using the attributed token kernel exhibits almost perfect accuracy of the 8 selected attacks, whereas both models of Kruegel and Vigna as well as their combination suffer from significant false positive rates for the first two attacks. Interestingly, Snort reported alarms for most of the cross site scripting and buffer overflow instances but failed to detect the majority of SQL injections which shows that specific exploits may require customization of signatures in order to provide a consistent protection.

5 Conclusions

In this paper, we have developed a general method for the incorporation of application layer protocol syntax into anomaly detection. The key instrument of our method is computation of similarity between token/attribute sequences that can be obtained from a protocol analyzer. A combined similarity measure is developed for such sequences which takes into account the syntactic context contained in tokens as well as the byte-level payload semantics.

The proposed method has proved to be especially useful for the detection of web application attacks. We have carried out experiments on realistic traffic

using the HTTP protocol analyzer developed in binpac. Although the additional effort of protocol analysis does not pay off for simple buffer overflow exploits, the detection rate for web application attacks has been boosted from 70% to 100% in the false positive rate interval of less than 0.14%. Surprisingly, our method has even outperformed a very effective model-based method of Kruegel and Vigna [8] that was specially designed for the detection of web application attacks.

Due to its generality, the proposed method can be used with any other application layer protocol for which a protocol analyzer is available. It can be deployed in a variety of distributed systems applications, especially the ones for which very few examples of potential exploits are currently known (e.g. IP multimedia infrastructure and SCADA systems).

Acknowledgements. This work was supported by the German Bundesministerium für Bildung und Forschung (BMBF) under the project ReMIND (FKZ 01-IS07007A).

References

- [1] Borisov, N., Brumley, D., Wang, H., Dunagan, J., Joshi, P., Guo, C.: Generic application-level protocol analyzer and its language. In: Proc. of Network and Distributed System Security Symposium (NDSS) (2007)
- [2] Cretu, G., Stavrou, A., Locasto, M., Stolfo, S., Keromytis, A.: Casting out demons: Sanitizing training data for anomaly sensors. In: *ieesp* (to appear, 2008)
- [3] Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T.: A sense of self for unix processes. In: Proc. of IEEE Symposium on Security and Privacy, Oakland, CA, USA, pp. 120–128 (1996)
- [4] Gao, D., Reiter, M., Song, D.: Behavioral distance measurement using hidden markov models. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 19–40. Springer, Heidelberg (2006)
- [5] Ingham, K.L., Inoue, H.: Comparing anomaly detection techniques for http. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 42–62. Springer, Heidelberg (2007)
- [6] Ingham, K.L., Somayaji, A., Burge, J., Forrest, S.: Learning dfa representations of http for protecting web applications. *Computer Networks* 51(5), 1239–1255 (2007)
- [7] Kruegel, C., Toth, T., Kirda, E.: Service specific anomaly detection for network intrusion detection. In: Proc. of ACM Symposium on Applied Computing, pp. 201–208 (2002)
- [8] Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: Proc. of 10th ACM Conf. on Computer and Communications Security, pp. 251–261 (2003)
- [9] Lee, W., Stolfo, S.: A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information Systems Security* 3, 227–261 (2000)
- [10] Leslie, C., Eskin, E., Noble, W.: The spectrum kernel: A string kernel for SVM protein classification. In: Proc. Pacific Symp. Biocomputing, pp. 564–575 (2002)
- [11] Mahoney, M., Chan, P.: PHAD: Packet header anomaly detection for identifying hostile network traffic. Technical Report CS-2001-2, Florida Institute of Technology (2001)

- [12] Mahoney, M., Chan, P.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 376–385 (2002)
- [13] Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K., Schölkopf, B.: An introduction to kernel-based learning algorithms. *IEEE Neural Networks* 12(2), 181–201 (2001)
- [14] Pang, R., Paxson, V., Sommer, R., Peterson, L.: binpac: a yacc for writing application protocol parsers. In: Proc. of ACM Internet Measurement Conference, pp. 289–300 (2006)
- [15] Paxson, V.: Bro: a system for detecting network intruders in real-time. In: Proc. of USENIX Security Symposium, pp. 31–51 (1998)
- [16] Rieck, K., Laskov, P.: Detecting unknown network attacks using language models. In: Büschkes, R., Laskov, P. (eds.) DIMVA 2006. LNCS, vol. 4064, pp. 74–90. Springer, Heidelberg (2006)
- [17] Rieck, K., Laskov, P.: Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology* 2(4), 243–256 (2007)
- [18] Rieck, K., Laskov, P.: Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research* 9, 23–48 (2008)
- [19] Roesch, M.: Snort: Lightweight intrusion detection for networks. In: Proc. of USENIX Large Installation System Administration Conference LISA, pp. 229–238 (1999)
- [20] Shawe-Taylor, J., Cristianini, N.: Kernel methods for pattern analysis. Cambridge University Press, Cambridge (2004)
- [21] Tax, D., Duin, R.: Data domain description by support vectors. In: Verleysen, M. (ed.) Proc. ESANN, Brussels, pp. 251–256. D. Facto Press (1999)
- [22] Wang, K., Parekh, J., Stolfo, S.: Anagram: A content anomaly detector resistant to mimicry attack. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 226–248. Springer, Heidelberg (2006)
- [23] Wang, K., Stolfo, S.: Anomalous payload-based network intrusion detection. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, pp. 203–222. Springer, Heidelberg (2004)

A Parallel Architecture for Stateful, High-Speed Intrusion Detection

Luca Foschini, Ashish V. Thapliyal, Lorenzo Cavallaro,
Christopher Kruegel, and Giovanni Vigna

Department of Computer Science
University of California, Santa Barbara
{foschini,ashish,sullivan,chris,vigna}@cs.ucsb.edu

Abstract. The increase in bandwidth over processing power has made stateful intrusion detection for high-speed networks more difficult, and, in certain cases, impossible. The problem of real-time stateful intrusion detection in high-speed networks cannot easily be solved by optimizing the packet matching algorithm utilized by a centralized process or by using custom-developed hardware. Instead, there is a need for a parallel approach that is able to decompose the problem into subproblems of manageable size. We present a novel parallel matching algorithm for the signature-based detection of network attacks. The algorithm is able to perform stateful signature matching and has been implemented only using off-the-shelf components. Our initial experiments confirm that, by making the rule matching process parallel, it is possible to achieve a scalable implementation of a stateful, network-based intrusion detection system.

1 Introduction

Intrusion detection is the process of analyzing a stream of events to identify the evidence of attacks. Intrusion detection can be applied to different domains and can be based on different techniques, which are usually characterized by the model used as the basis for detection. Systems used to specify what the “normal behavior” of an application is are usually called *anomaly-based*, because an attack will be detected as an event that does not fit the system’s idea of what is “normal.” On the other hand, systems used to specify what the manifestation of an attack is are known as *misuse-based*, and the models are often referred to as *signatures*.

Both anomaly-based and misuse-based systems have advantages and disadvantages. Anomaly-based systems are able to detect previously unknown attacks, but they traditionally produce more false positives (erroneous detections). Misuse-based systems, on the other hand, are usually more precise, but cannot detect attacks for which they do not have a description, and, therefore, they need continuous updating.

Because of their low false positive rate and their higher performance, misuse-based detection approaches are the basis for the majority of the existing

network-based intrusion detection systems. Unfortunately, as the speed of the network links increases, keeping up with the pace of events becomes a real challenge.

This problem has been addressed by current intrusion detection systems by minimizing the amount of state that is associated with the matching process. In particular, many systems operate on single packets (e.g., [12]) and do not provide support for complex, *stateful* signatures, where the evidence necessary to detect an attack is spread across multiple packets at different points in time.

A number of approaches have been proposed to address the problem of matching stateful signatures in high-speed networks. Kruegel et al. proposed a parallel architecture to partition the traffic into slices of manageable size, which are then examined by a set of intrusion detection sensors [10]. This slicing technique takes into account the signatures used by the intrusion detection sensors, so that all the evidence needed to detect an attack is guaranteed to appear in the slice associated with the sensor responsible for the detection of that particular attack. Therefore, no communication among the intrusion detection sensors is necessary.

Unfortunately, even though the system was able to improve the overall performance of the detection process, it was not able to effectively partition the traffic in the case of complex stateful signatures. This is because, in general, the correct partitioning of the traffic is known only at runtime, and, therefore, any approach that uses a *static* partitioning algorithm needs to over-approximate the event space associated with a signature, possibly resulting in a degenerate partitioning.

We propose a novel technique to perform parallel matching of intrusion detection signatures. Our technique is similar to the one proposed in [10], since it uses a partitioning mechanism to reduce the traffic on a high-speed link to slices of manageable size, and, in addition, it uses a number of parallel sensors. However, in our approach, the sensors are able to communicate through a high-speed, low-latency, dedicated control plane. By doing this, they can provide feedback to each other in order to synchronize their scanning processes. Therefore, they can detect attacks that would be missed if the traffic partitions were analyzed separately as it happens in [10]. In addition, our system does not rely on any predetermined partitioning of the traffic and can optimally distribute the load over the available sensors.

In this paper, we make the following contributions:

- We introduce a novel architecture for the parallel matching of stateful network-based signatures.
- We present a novel algorithm that allows for the detection of complex, stateful attacks in a parallel fashion.
- We provide a precise characterization of the bottlenecks that are inherent to the parallel matching of stateful signatures in the most general case.
- We describe a prototype implementation of our system and we show that the proposed architecture allows for the parallel matching of network signatures.

The rest of this paper is structured as follows. In § 2, we present our rule matching model. Then, in § 3, we present the architecture of our parallel intrusion detection

system. In § 4, we describe an algorithm for parallel rule matching that we implemented on the described architecture and in § 5 we show experimental results. Then, § 6 discusses related work in the field of high-speed intrusion detection and, finally, § 7 concludes.

2 Model

In general, an intrusion detection system scans a stream of events and detects attacks using signatures. Rules are matched against the stream of events by a rule matching system to identify signatures of attacks.

Usually, a rule r is described as composed of a predicate P , possibly a state S (in this case, the rule is said to be “stateful”), and an action A [5,17]. The predicate P describes the constraints on the values of the event and attributes of S . If the predicate evaluates to true for an event, then the action is triggered. The action A consists in modifying the rule state or raising an alert (that is, generating a new event for further processing). The rule maintains the state S in order to keep track of the steps of an attack. For example, a counter might be the state used by a rule to keep track of the number of TCP connection attempts to a host. This information can be used to detect port scans. A number of architectures that implement a rule matching system as described above are publicly available [5,14,15]. These architectures are, in general, centralized, i.e., a single processing node deals with all the events.

A parallel architecture for intrusion detection should help lower the processing power demand and the memory footprint of packet analysis by distributing the network traffic to be analyzed among different sensors. We define a parallel model for intrusion detection by expanding the aforementioned event model, with some assumptions:

1. The network traffic is split and sent to one or more *sensors*.
2. Each sensor tries to match one or more signatures against the traffic it receives.
3. The system does not rely on any predefined mapping between packets and sensors, i.e., each packet could be sent to any sensor. This is a more general approach than the one described by Kruegel et al. in [10], where the traffic slicer enforced a mapping between packet features and sensors based on the analysis of the signatures’ event spaces. Here, we take into consideration the most general case, that is, for any pair of packets, we have no *a priori* knowledge that they both belong to the same event space.

It is easy to see that, given these assumptions, stateful rules are difficult to implement in a parallel rule matcher. The reason is that, without any pre-existent mapping between packet features and sensors, there is no guarantee that two packets that are required to match a signature are processed by the same sensor.

A trivial solution to overcome this problem would be to replicate the traffic so that each packet is sent to every sensor. Unfortunately, this solution would render the parallel approach totally ineffective. A better approach is to minimize

the amount of traffic to be scanned by processing each packet only once, which, in turn, entails that every sensor has to maintain the same rules with the same state loaded. Therefore, our parallel machine, by design, requires no replication of the incoming traffic, but a complete replication in the matcher state, which has to be the same for all the sensors at any time. To refine this observation, we express the predicate evaluation of a stateful rule in the following form:

Definition 1. *Predicate evaluation of a stateful rule:*

$$S_{R,t+1} = F(I_t, S_{R,t}),$$

where I_t is the input at time t , R is a rule, and $S_{R,t}$ is the state of the rule at time t .

If one sees the rule matching process as a finite state machine (FSM), F is nothing more than its *transition function*. The computationally expensive parts of the whole process are hidden in the formalism above and consist, most notably, of: (a) the evaluation of the input I_t , i.e., the packet scanning, which will possibly trigger a state transition, and (b) the corresponding state update that a transition might imply.

Evaluating the predicate is part of the packet scanning process, and it can be performed in parallel with no additional effort¹, since each sensor evaluates the same predicate on different packets. On the other hand, the state update is an operation that we aim to avoid in our system, because it needs to be repeated on all the sensors, in order to have the state of the parallel machine to evolve simultaneously.

These considerations are not very encouraging when building a parallel system. Amdahl's law [1] here fits perfectly: the time needed by the state update has to be spent by all the sensors for each packet that modifies the state, even though the packet matching occurs on only one sensor, and, therefore, cannot be parallelized. Simply put, during the state update triggered by a packet matching a rule, all the sensors do the same operation, behaving as if they were a single machine and without exploiting any parallelization. Therefore, it is already clear that the bottleneck of an architecture that has to address a general, parallel stateful signature matching system lies in updating the state of the rule(s) when a match occurs.

Note that the above considerations hold only for the scanned packets that match a rule and, thus, update the rule's state. However, not all the packets will match (hopefully, very few will). Therefore, we can significantly benefit from parallelizing the packet scanning process.

Intuitively, we need to make the update of the state of a rule as fast as possible. In order to do that, we can partition the rule's state. More precisely, we split the rule's state in two parts: a *scanning state*, or predicate state, and a *working state*. The *working state* does not modify the scanning state and thus does not affect the matching capability of a sensor. Only the predicate state needed for the evaluation has to be updated in the aforementioned non-scalable fashion.

¹ These kind of problems are also known as “embarrassingly parallel.”

The actual working state can, on the other hand, live on a separate centralized machine, which we call “control node,” and it can be updated asynchronously (in § 3, we will describe the system architecture in more detail). Ideally, we would like the scanning state to be as small as possible. More formally, we can functionally decompose a rule as below:

Definition 2. *State transition and output of a stateful sensor:*

$$S_{P,t+1} = F_{SP}(I_t, S_{P,t}); \quad O_{P,t+1} = F_{OP}(I_t, S_{P,t})$$

Definition 3. *State transition and output of a control node:*

$$S_{W,t+1} = F_{SW}(O_{P,t}, S_{W,t}); \quad O_{W,t+1} = F_{OW}(S_{W,t})$$

In the definition above, S_P is the *scanning (or predicate) state* and S_W is the *working state*, while O_P and O_W are, respectively, the output of the sensors (going to the control node) and the output of the control node, which could be an alert. F_S are the state transition functions, which map states and inputs to new states, and F_O are the output functions, which map states, or states and input, to output.

Def. 2 describes the behavior of a generic sensor in terms of a *Mealy Machine*, while Def. 3 describes the control node in terms of a *Moore Machine*. The fundamental point is that the control node can process the output from the sensors in an environment that is completely decoupled from the sensors.

Following the considerations above, we want to split the state management so that the state update process on the sensors is minimized and does not require any feedback from the control node to the sensors. For example, a rule that detects a port scan attack can be implemented as a TCP stream-based predicate on the sensors. The stream-based predicate and the predicate state take care of filtering out packets that are part of a flow (thus, maintaining some predicate state that mimics the TCP state machine for each stream and filters out packets when they are found to belong to a legitimate connection). The detection part (working state) on the control node takes care of keeping in memory and updating all the data structures needed for the detection of the scan, such as the list of targets and scanners, counters and timers, or, to detect coordinated port scans, a whole implicit graph representation of inter-host connectivity as described in [9], on which to run periodically complex set-covering algorithms.

3 Architecture

The proposed system uses a parallel architecture as shown in Figure 1. In our setup, packets are obtained from a tap (T) into the high-speed network link(s) that copies the packets and sends them to a packet numbering unit (PNU). The PNU, in turn, tags the packets with a logical timestamp (e.g., generated using a counter starting from zero) and forwards them to the splitter. The splitter sends the traffic in a round-robin fashion to N sensor nodes in charge of performing the detection.

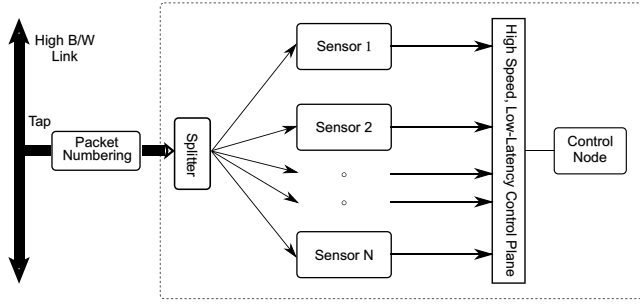


Fig. 1. The architecture of the parallel rule matcher

The sensor nodes are managed by a control node that maintains the working state associated with the attack instances, as described in the previous section. The sensor nodes forward to the control node any packet² that is tagged as suspicious, i.e., that it is possibly part of a stateful attack.

The control node updates its state (the working state) according to the messages received from the sensors. In addition, the sensors are able to talk to each other through a low-latency broadcast channel. As a result, when one of the sensors matches a rule, the others can update their predicate state (once again, the same for all sensors) consistently.

In order for the parallel matcher to emulate a serial one, each sensor is provided with a buffer that we call *sensor match buffer* (SMB), which is utilized to store already-scanned packets. This buffer is required in case an earlier packet was matched by another node, which triggered a sensor state change. In § 4, an algorithm will be described that leverages the SMB on each sensor in order to make the parallel rule matcher behave like a serial one.

In this architecture, the traffic is split in a round-robin fashion, and, therefore, each node receives $1/Nth$ of the total traffic, without incurring any potential load-balancing problems that instead would be caused by alternative traffic partitioning techniques employed in other frameworks, such as [2,10]. Moreover, since there are no constraints on the mapping between sensors and traffic, if a sensor is slower than the others and is overloaded, the excess traffic can be seamlessly diverted to other sensors in order to keep the load constant, without using any complicated load-balancing algorithm.

4 Algorithm and Protocol

We present a general algorithm, called “simple-sequencer,” that we use to perform the parallel update of the state transition function of the predicate state as described in Def. 2. The algorithm is implemented on the sensor nodes and

² More precisely, only the relevant attributes of the packet such as, for example, source or destination IP, are forwarded to the other sensors and the control node.

is used to keep synchronized their predicate state. The simple-sequencer parallel algorithm behaves like a serial algorithm performing the same task, as far as correctness is concerned, if the predicate state transition function of Def. 2 belongs to a class of *strict-sequence* state machine that we define below. The control node implements a serial rule-matching algorithm corresponding to the working state update function in Def. 3 and needs only to perform reordering of out-of-order packets sent by the sensors.

Definition 4. *Strict-sequence state machine:*

A strict-sequence state machine recognizes events that happen in a sequential fashion, i.e., any event in the sequence can be followed by one and only one different event in the event stream.

According to Kumar et al. [11], the majority of known intrusion patterns can be formulated as a set of events that happen in strict sequence. Nevertheless, the ability of the algorithm to match events in strict sequence can also be leveraged to recognize patterns not directly implying an attack. For instance, the simple-sequencer algorithm allows to identify the beginning of a TCP connection, described by the sequence (SYN, SYN-ACK, ACK) in this particular order. Even though the starting of a TCP connection is not a manifestation of an attack itself, it could help, for instance, filter out packets belonging to a legitimate TCP connection in a signature, which would reduce the rate of the traffic processed by the parallel matcher.

Note that the strict-sequence constraint above applies only to the portion of rule implemented on the sensors. The control node can evaluate more complex functions (order-variant, with partial order constraints, etc.) on the output of the sensor nodes, since it does not operate in parallel and leverages a centralized state.

4.1 Simple-Sequencer Matching Algorithm

Overview of the Algorithm. From now on, we refer to packets as being “earlier” or “later,” using the packet ID as the ordering parameter; with smaller corresponding to earlier and larger corresponding to later. In addition, when we say “packet x ” we mean “packet with packet ID x ”. We also say that a packet “matches” the rule when it causes a state update on the associated FSM defined in Def. 2.

The intuition behind the inner works of the simple-sequencer algorithm is the following: if a packet causes a state change on a sensor, then any other packet following it in the input has to be scanned against the new state. This means that no packets can be discarded from the sensors’ SMBs as long as they are needed.

The crux of the algorithm lies in the logic used to prune the sensors’ SMBs, which, otherwise, would grow unbounded. The algorithm guarantees that no packet that could be needed at some sensor is ever discarded. This condition is enforced in two ways: (i) no packet later than the earliest packet currently scanned (i.e., the packet scanned by the slowest sensor) is discarded; (ii) if a

packet caused a match and a state update on all the sensors, it is retained until all the matches triggered by that state update are completed.

The need for (ii) is clarified by an example. Suppose that the slowest sensor is scanning packet x and that packet x causes a match. At this point, we say that packet x has a pending match. The match is communicated to the other sensors, which will update their state accordingly. The sensors will rescan any packet later than x . If, during this operation the packet $x + 1$ causes a new match on any sensor, then all the packets later than $x + 1$ must be retained on the sensors. But in the meantime, the slowest sensor could have moved far beyond packet $x + 2$, thus discarding all the earlier packets (and, $x + 2$ itself) by (i). Therefore, packet $x + 2$ needs to be retained in the slowest sensor's SMB by additional logic, as described by (ii).

A match remains pending until it has been acknowledged by all the sensors. A sensor acknowledges a match when all the matches triggered by this match have been acknowledged, or, no new matches have been triggered. Packet $x + 2$ will become available to be discarded only when no packets earlier than it have pending matches.

We assume that all messages between a pair of sensors are transported using a first-in-first-out messaging system. Each sensor i keeps the following variables:

- *lastInspected_i*: The packet ID of the last packet inspected by sensor i .
- *pendingMatches_i*: A priority queue arranged in ascending order of the value of packet ID k_i . This queue contains the matches that are pending, i.e., not yet acknowledged by the other sensors.
- *earliestNeeded_i*: The head of the *pendingMatches_i*, i.e., the packet ID of the earliest packet that needs to be retained at sensor i .
- *SMB_i*: The buffer in which packets are stored, ordered by packet ID. Packets stored in this buffer include those yet to be scanned, as well as those already scanned but not yet discarded, i.e., all the packets later than *earliestNeeded_i*.

In addition to the previous per-sensor variables, we define *discardPtr* to be the minimum of *earliestNeeded_j* over all the sensors j .

Now, we are ready to specify the algorithm, and we do so for one sensor i . We rely on the following primitives:

BroadcastMatch($\langle Match(P_k, r_l, i) \rangle$): If the packet matches a rule r_l then broadcast $\langle Match(P_k, r_l, i) \rangle$ to all nodes. Add $\langle k, triggerMsg, timestamp \rangle$ to the *pendingMatches_i* priority queue. The parameter *triggerMsg* may be *null* if no other match triggered this match. P_k is the packet that matched³.

ProcessMatch($\langle Match(P_k, r_l, j) \rangle$): A rule match $\langle Match(P_k, r_l, j) \rangle$ is received by sensor i . Update rule r_l according to the action specified by the rule and the data present in packet P_k , then scan the packets up to and

³ Of course, in a real-world deployment, only the relevant information needed by other sensors will be broadcast.

```

// Task 1: new packets are enqueued in the sensor's SMB
foreach new packet  $P_k$  available from network do
  | Insert  $P_k$  at the end of buffer  $SMB_i$ ;
end
// Task 2: packet inspection and communication with other
  sensors
foreach packet  $P_k$  not yet scanned in  $SMB_i$  do
  | if a Match message is available wrt packet  $P_h$  and rule  $r_l$  for sensor  $j$ 
  |   then
  |     | ProcessMatch(< Match( $P_h, r_l, j$ ) >);
  |   else
  |     | if a MatchAck message is available wrt packet  $P_h$  and rule  $r_l$  for
  |       sensor  $j$  then
  |         | ProcessMatchAck(< Match( $P_h, r_l, j$ ) >);
  |       end
  |   end
  |   Scan  $P_k$  against each rule;
  |   Set lastInspected $_i$  to  $k$ ;
  |   if a match with rule  $r_l$  occurred then
  |     | BroadcastMatch(< Match( $P_k, r_l, i$ ) >)
  |   end
end
// Task 3: synchronize sensors' buffers
Every  $T$  packets scanned, run the bully algorithm to compute discardPtr;

```

Fig. 2. The simple-sequencer rule matching algorithm

including *lastInspected $_i$* in buffer SMB_i that are after packet k for a match against the updated rule. For each match that occurs at packet h , for rule m , call BroadcastMatch(< Match(P_h, r_m, i) >).

ProcessMatchAck(< MatchAck(P_k, r_l, j) >): Mark in *pendingMatches $_i$* that sensor j has acknowledged < $k, triggerMsg = < Match(P_h, r_m) >, timestamp >$. If all the nodes have acknowledged this match, remove < $k, triggerMsg = < Match(P_h, r_m, j) >, timestamp >$ from the priority queue *pendingMatches $_i$* and send a match acknowledgment < MatchAck(P_h, r_m, i) > to node j .

The algorithm, shown in Figure 2 for the i -th sensor, is composed of three tasks, which can be performed concurrently.

The bully algorithm [8] is utilized to compute the *discardPtr*. All the sensors can discard the packets from their SMBs periodically (an appropriate timer can be utilized to trigger discards). More precisely, the sensors can discard packets earlier of the last *discardPtr* seen. This deferred deletion ensures that the transitories associated with the contention have vanished, so that there is not any packet earlier than *discardPtr*, which is needed.

Properties of the Simple-Sequencer Algorithm. The simple-sequencer algorithm has an interesting property, i.e., it behaves as a serial algorithm in matching strict-sequence FSMs as described in Def.4. This property is important because it guarantees the matching of strict-sequence FSMs avoiding any race condition among the sensors' state.

Due to space constraints, a more thorough description of the simple sequencer algorithm with exemplifications can be found in [7]. Also, for the same reasons, an analysis of the scaling properties of the architecture when running the simple-sequencer algorithm is omitted and reported in [6], Section 4.4.

4.2 Control Node

The simple-sequencer algorithm works on the sensors by implementing the predicate state transition function of Def. 2. This function is constrained to belong to the class of strict-sequence FSM as defined in Def. 4. Nevertheless, the Moore machine of Def. 3 implemented on the control node and used to update the working state does not have any limitation. Therefore, any kind of rule, as decomposed in Def. 2 and Def. 3, can be implemented on the parallel architecture made of sensors and control node, as long as the rule is decomposed in such a way that guarantees that its predicate part is an FSM that abides to Def. 4.

Going back to the port scan detection example of § 2, now we can see how this signature can be detected by our parallel architecture. The sensor nodes are responsible of tracking established TCP connections and filter out packets belonging to a legitimate connection. Tracking the beginning of a TCP connection means identifying the three-way handshake between the involved hosts. The (SYN, SYN-ACK, ACK) pattern can be matched by a strict-sequence rule implemented on the sensors. Once the sensors identify a connection they start discarding packets belonging to it. Other packets are instead forwarded to the control node, which performs a thorough port scan detection on them. In this way, the joint action of sensors and control node can detect a signature that cannot be described by a strict-sequence FSM itself. We will review this setup more in depth in § 5.

We stress that the control node, being a serial component, simply processes packets sent to it by the sensors. However, sensors can still send to the control node packets out of order. The reordering is caused by the different relative instantaneous speeds at which sensors operate. To prevent the control node to analyze packets sent by the sensors in the wrong order, we use a simple reordering buffer.

The buffer must be large enough to accommodate as many packets as can fit the gap between the fastest and the slowest sensor. In the same fashion, the delay before processing packets must be large enough to allow late packets to come in and be put in the right order.

Different from the SMBs, where the correct size of the buffers were enforced as a byproduct of the simple-sequencer algorithm, the size of the control node front buffer cannot be easily predicted. We will therefore set the size of the buffer and the delay before processing to conservative values, which will be validated in § 5.

5 Experimental Validation

To test our model, following the model and algorithm presented in § 2 and § 4, we implemented a parallel architecture to detect port scans.

Our implementation of the sensors and the control node relies on Snort version 2.6 [15]. The Snort running on the sensors has two preprocessors loaded, namely `stream4_reassemble` and `sfportscan`, which have been modified to update their state synchronously, as described in the algorithm reported in Fig. 2. More precisely, the `stream4_reassemble` preprocessors have been enabled to communicate with each other to synchronize the state of the observed TCP connections.

The control node runs Snort 2.6 with the `sfportscan` preprocessor loaded and has a packet reordering buffer. The packet reordering buffer is used to reorder possibly out-of-order packets from the sensors. The sensors use the `stream4_reassemble` to check if a packet belongs to an established TCP connection. When this is not the case, the packet is forwarded to the control node that uses the `sfportscan` preprocessor to match it against the port scan signature.

We expect our parallel architecture to be able to detect port scans as a single-instance vanilla Snort with `sfportscan` preprocessor loaded would do. In addition, we expect the parallel machine to be able to keep up with a higher throughput than what a single instance detector can cope with.

At first, we have performed a functionality test in an emulated environment based on tap interfaces and the Virtual Distributed Ethernet [3,4]. A complete description of the emulated environment cannot be reported here due to lack of space and can be found in [6], § 6.1.

We have also set up a real-world network testbed to evaluate our algorithm, protocol, and architecture. From the network point of view, the system was built following the architecture described in § 3. More precisely, the system is composed of two networks:

The distribution network, through which packets are sent from the splitter to the sensors. This has been implemented as an Ethernet network with no IP-level processing. The splitter has a high throughput network interface connected to a switch, configured with static routes from the interface to which the splitter is connected towards the outgoing interfaces to the sensors.

The control network, through which the sensors exchange control packets among each other and with the control node. The sensors and the control node are connected through a hub, since there are no high throughput requirements for control traffic, and only low latency is required.

We analyzed the behavior of the sensors by following the journey of a packet captured from the live interface by a sensor.

1. When a new packet is captured on the live interface, it is passed to the Snort decoding engine.
2. Before sending a packet to the preprocessors, each sensor listens (in non-blocking mode) for incoming packets on the control interface and processes them, if any.

3. Packets received from the live interface are sent to the **stream4_reassemble** preprocessor. If the packet changes the state of the reassembly machine (i.e., initiates, tears down, or modifies the state of some connections), which in this case represents the scanning state described in our theoretical framework, then it is broadcast to the other sensors as a *Match* beacon.
4. Packets received from the control interface (coming from other sensors) are stored into the SMB and will be used to re-create the same state changes on the **stream4_reassemble** preprocessor.
5. Packets coming from the live interface and sent through the stream reassembly preprocessor are then handed out to the port scan preprocessor. At the beginning of the process, some tests are performed on the packet in order to exclude benign packets. One of these tests consists in checking if the packet is part of a legitimate connection. If this is the case, the packet surely cannot be part of a scan, and, thus, no further checks are performed on it.

Here is where the scanning state of the **stream4_reassemble** preprocessor is exploited. Packets that are not tagged as benign at this stage do not go through a complete analysis against port scan detection heuristics, but are simply forwarded to the control node, which takes care of them. Therefore, sensors do not need to maintain any state associated with the port scan preprocessor, such as scanners and scanned hosts lists. Moreover, only a small fraction of the input traffic will reach the control node.

6. The control node puts the packets received from the sensors in the reordering buffer and, after a delay of 0.2 seconds, scans them and identifies scan attacks. In our experiments we have seen that a delay of 0.2 second was sufficient to allow the control node for processing all the packets in order.

It is easy to recognize that the aforementioned setup is a particular case of the general architectural model described in § 2. The preprocessor **stream4_reassembly** on the sensors maintains the predicate state and uses the simple-sequencer algorithm to keep the state of TCP connections synchronized among sensors. The **sfportscan** preprocessor loaded on the sensors performs the rule predicate evaluation to filter out benign traffic. The working state is only kept in the **sfportscan** preprocessor on the control node in the form of all the relevant structures maintained by the preprocessor itself, such as the scanner hosts and the scanned hosts lists.

5.1 Evaluation Dataset

To evaluate our system, we have crafted an artificial dataset by merging attack-free traffic collected on a real-world network at UCSB with a dump of a port scans performed against a host inside the home network. The scans were performed using the **nmap** tool. The attack-free traffic contains 3.46 million packets, and includes a mix of web traffic, mail traffic, IRC traffic, and other common protocols.

We rewrote, by means of the **tcprewrite** tool [20], the MAC address of every frame in order to induct a round-robin scattering pattern on the splitter.

5.2 Hardware and Software Configuration

We set up four Linux sensors running Snort 2.6. Each sensor is equipped with four Intel Xeon CPUs X3220 at 2.40GHz and 4GB of RAM.

The control node is deployed on another Linux host with the same hardware characteristics of the sensors. The control node runs a version of Snort with the reordering packet buffer mentioned before and a vanilla version of the `sfportscan` preprocessor. Another Linux box is used only to inject the traffic into the testbed using a Gigabit Ethernet card connected to a PCI-Express bus.

Each sensor box mounts two Gigabit Ethernet network cards, one to receive the traffic from the splitter, and another one to communicate with the other sensors and the control node. Sensors are connected to the splitter box through a Cisco Catalyst 3500 XL switch configured with static routes to the sensors capture interfaces. The switch has a 1000-BaseT GBIC module that receives the traffic from the splitter output interface and is connected to sensors through FastEthernet ports. The sensors' control interfaces are connected to each other through an ordinary FastEthernet hub.

5.3 Experiments

We performed several experiments to validate our model. We compared the parallel rule matcher setup with a single instance of Snort with the `stream4_reassemble` and `sfportscan` preprocessor loaded. The Snort community ruleset [19] (version "CURRENT" as of July 2008) was loaded on both the parallel sensors and the single instance. We note that none of the rules loaded on the sensors were extended to work correctly on the parallel machine. The rules were used only to represent a realistic per-packet workload in a real-world intrusion detection system.

We expected that, at a certain traffic rate, the single instance Snort would not be able to keep up with the traffic, and, therefore, would miss the port scan attack, while the parallel version of the sensor would be able to catch it. Each experiment has been repeated 10 times and the results reported are the average of the outcomes. More precisely, the setups compared are as follows.

Single Snort: In this setup, the traffic has been replayed via a cross cable to a single box mounting a Gigabit Ethernet interface.

Parallel architecture with four sensors: In this setup, we implemented the complete parallel architecture with four sensors and a control node.

Parallel architecture with two and three sensors: In this setup, we changed the number of sensor utilized to study the scalability of our approach.

Results are reported in Figure 3. For each setup, the percentage of attacks detected versus the input traffic throughput is plotted. The percentage of detected attacks is computed as the average number of port scan packets detected with respect to the total present in the trace. It can be seen how the four-sensors Snort outperforms the single instance starting from a traffic throughput of 190Mb/s.

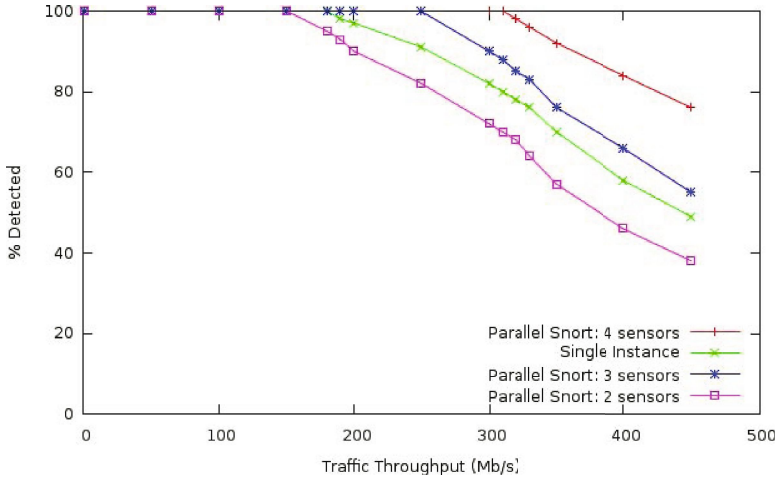


Fig. 3. % of correct detection when varying the number of sensors and the traffic speed

At this speed, the single instance of Snort starts discarding packets. The percentage of discarded packets grows almost linearly as the throughput of the traffic increases. On the other hand, the parallel Snort starts discarding packets at around 320Mb/s. This happens because we saturated the aggregated capacity of the four outbound ports of the Catalyst switch going to the sensors. In fact, we found that none of the four Snort sensors employed dropped any packet during their analysis; instead, packets were dropped by the switch. We expect our parallel architecture to be able to achieve higher throughputs, but we were not able to prove our expectation due to the physical limitations of the hardware employed. The same problem of capacity saturation was also encountered with the parallel architecture using two and three sensors, as can be seen in Figure 3. For two sensors, in particular, the performance in detection is worse than using a single Snort. This is easily explained by the fact that the single Snort was tested using a crossover connection between the splitter and the instance, therefore bypassing, the FastEthernet switch.

CPU Usage. We measured the time spent by the parallel-enabled Snort in the portion of code implementing the communication and parallel synchronization logic. The RDTSC instruction was used to achieve a precise time measure. RDTSC returns time in arbitrary units (count of ticks from processor reset). Therefore, it can be used only to get relative measurements. From our experiment, we have measured that the average time spent by the sensors in the additional logic (at the highest possible speed) is about 5.3% of the total time spent in the Snort engine. The CPU usage measurements were performed without the community ruleset used for the throughput experiments mentioned above, which is the worst case. In fact, if the ruleset had been loaded, the fraction of time spent in the parallel logic would have dropped even more, as it is independent of the loaded rules.

Latency and Cost of Communication. Another important factor to take into consideration is the latency introduced by the SMBs. In our experimental setup, we chose to limit the length of the SMBs to 1,000 packets. Therefore, this number determines the upper bound of the latency that a traffic packet can experience in its journey to the control node. With four sensors scanning the traffic at 100Mb/s, the maximum latency experienced would be ~ 0.05 seconds, which is negligible. Moreover, the latency decreases with the increasing of throughput, becoming less significant at higher throughputs.

As far as the communication cost introduced by the communication among sensors and control node is concerned, we report that only 58K packets out of 3.46 million were forwarded to the control node when the setup with four sensors was employed at 100Mb/s (no packet loss). 17K packets were exchanged among the sensors to update the predicate state. Most of the rest of the communication cost of the parallel machine is spent for the buffer synchronization protocol. More precisely, 127K packets were exchanged among sensors as part of the bully algorithm. We note that, since we chose to limit the SMB length to 1,000 packets, we could have disabled the buffer synchronization mechanism. However, we decided to leave it active to determine experimentally the cost of the number of packets exchanged for buffer synchronization purposes. We also note that the total cost spent in communication among the sensors and the control node accounts for 203K packets, out of the 3.46 million present in the traffic. This means that only a small fraction of the input traffic ($\sim 6\%$) reached the control node and the majority of packets were filtered out by the sensors.

6 Related Work

As we mentioned in the introduction, Kruegel et al. proposed a distributed architecture to partition the traffic into slices of manageable size, which were then examined by dedicated intrusion detection sensors [10]. Our technique is different because the sensors can communicate with each other in order to keep a synchronized version of the matching state, which, in turn, allows the packets to be sent independently to any sensors.

Sekar et al. [16] describe an approach to perform high-performance analysis of network data. They propose a domain-specific language for capturing patterns of both normal and abnormal packet sequences (therefore, their system encompasses both misuse and anomaly detection). Moreover, they developed an efficient implementation of data aggregation and pattern matching operations. A key feature of their implementation is that the pattern-matching time is insensitive to the number of rules, thus making the approach scalable to large rule sets. The shortcomings in their work are that the processing of the packets is done on a central sensor, and, therefore needs to be able to keep up with the fast pace of packets on high-speed networks.

Building up from the foundations laid by Sekar et al., but using a more general approach, Meier et al. [13] propose a method for system optimization in order to detect complex attacks. Their method reduces the analysis run time that focuses on complex, stateful, signatures.

Sommer et al. [18] propose a way to exchange state among sensors by serializing it into a file and exchanging it between sensors. Their approach is focused on providing a framework to transfer fine-grained state among different sensors in order to enhance the IDS with a number of features. Although the authors provide a way to exchange fine-grained state among sensors, no emphasis is put on performance.

However, the mechanisms introduced in [18] were leveraged by Vallentin et al. to build a cluster-based NIDS [21]. This design has a number of similarities with our architecture, as it includes a component that splits the traffic across sensors, which, in turn, exchange information to detect attacks whose evidence span multiple slices. However, the focus of the work of Vallentin et al. is on the engineering challenges associated with the creation of a high-performance, cluster-based NIDS, while the focus of the research we described in this paper is on the modeling and analysis of the general problem of performing parallel detection of stateful signatures. For example, a substantial part of the complexity of our algorithm is due to the analysis of previously processed packets when a new rule triggers a change in the replicated state. This problem is simply avoided by Vallentin et al. by partitioning the traffic according to source/destination pairs, using loose synchronization, and by limiting the possibility of race conditions by means of signature-specific techniques. Even though the two approaches have different foci, many of the lessons learned by implementing each approach can be used as a basis to improve the respective designs.

Other approaches, such as the ones from Colajanni et al. [2] and Xinidis et al. [22] propose optimization based on load-balancing and early filtering to reduce the load of each sensor. However, their work does not focus on the design of a truly parallel matching algorithm.

7 Conclusions

The speed of networking technologies has increased faster than the speed of processors, and, therefore, centralized solutions to the network intrusion detection problems are not scalable. The problem of detecting attacks in high-speed environment is made more difficult by the stateful nature of complex attacks.

In this paper, we have presented a novel approach to the parallel matching of stateful signatures in network-based intrusion detection systems. Our approach is based on a multi-sensor architecture and a parallel algorithm that allow for the efficient matching of multi-step signatures. We have analyzed the feasibility and described a proof-of-concept implementation of a parallel, stateful intrusion detection system for high-speed networks.

The architecture has been deployed on a real network using four Linux boxes as sensors and it has been tested for port scan detection in various configurations. The result of the evaluation have confirmed the validity of the theoretical model, since the parallel rule matcher, composed of four sensors, has successfully outperformed a single sensor instance, which we used as a baseline for comparison, performing the same kind of detection on the same dataset.

References

1. Amdahl, G.: Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In: Proceedings of the AFIPS Conference (1967)
2. Colajanni, M., Marchetti, M.: A parallel architecture for stateful intrusion detection in high traffic networks (September 2006)
3. Davoli, R.: Vde: Virtual distributed ethernet. Technical report (2004)
4. Davoli, R.: Vde: Virtual distributed ethernet. In: TRIDENTCOM 2005: Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COmmunities, Washington, DC, USA, pp. 213–220. IEEE Computer Society, Los Alamitos (2005)
5. Eckmann, S., Vigna, G., Kemmerer, R.: STATL: An Attack Language for State-based Intrusion Detection. In: Proceedings of the ACM Workshop on Intrusion Detection Systems, Athens, Greece (November 2000)
6. Foschini, L.: A formalization and analysis of high-speed stateful signature matching for intrusion detection (2007)
7. Foschini, L., Thapliyal, A.V., Cavallaro, L., Kruegel, C., Vigna, G.: A Parallel Architecture for Stateful, High-Speed Intrusion Detection. Technical report (2008)
8. Garcia-Molina, H.: Elections in a Distributed Computing System. IEEE Transactions on Computers (1982)
9. Gates, C.: Co-ordinated Port Scans: A Model, A Detector and An Evaluation Methodology. PhD thesis, Dalhousie University, Halifax, Nova Scotia (February 2006)
10. Kruegel, C., Valeur, F., Vigna, G., Kemmerer, R.A.: Stateful Intrusion Detection for High-Speed Networks. In: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, May 2002, pp. 285–293. IEEE Press, Los Alamitos (2002)
11. Kumar, S., Spafford, E.H.: A Pattern Matching Model for Misuse Intrusion Detection. In: Proceedings of the 17th National Computer Security Conference, pp. 11–21 (1994)
12. Lu, H., Zheng, K., Liu, B., Zhang, X., Liu, Y.: A Memory-Efficient Parallel String Matching Architecture for High-Speed Intrusion Detection. IEEE Journal on Selected Areas in Communication 24(10) (October 2006)
13. Meier, M., Schmerl, S., Koenig, H.: Improving the Efficiency of Misuse Detection. In: Proceedings of RAID (2005)
14. Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time. In: 7th Usenix Security Symposium (1998)
15. Roesch, M.: Snort - Lightweight Intrusion Detection for Networks. In: Proceedings of the Large Installation System Administration Conference (LISA), Seattle, WA (November 1999)
16. Sekar, R., Guang, V., Verma, S., Shanbhag, T.: A High-performance Network Intrusion Detection System. In: Proceedings of the 6th ACM Conference on Computer and Communications Security (November 1999)
17. Snort - The Open Source Network Intrusion Detection System (2004), <http://www.snort.org>
18. Sommer, R., Paxson, V.: Exploiting Independent State For Network Intrusion Detection. In: Proceedings of ACSAC (2005)
19. The open source community. Snort Community ruleset
20. Turner, A.: tcprewrite trac page, <http://tcpreplay.synfin.net/trac/wiki/tcprewrite>

21. Vallentin, M., Sommer, R., Lee, J., Leres, C., Paxson, V., Tierney, B.: The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637. Springer, Heidelberg (2007)
22. Xinidis, K., Charitakis, I., Antonatos, S., Anagnostakis, K., Markatos, E.: An active splitter architecture for intrusion detection and prevention. IEEE TDSC 3(1), 31 (2006)

Indexing Multimodal Biometric Databases Using Kd-Tree with Feature Level Fusion

Umarani Jayaraman, Surya Prakash, and Phalguni Gupta

Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur, Kanpur-208 016, India
{umarani,psurya,pg}@cse.iitk.ac.in

Abstract. This paper proposes an efficient indexing technique that can be used in an identification system with large multimodal biometric databases. The proposed technique is based on Kd-tree with feature level fusion which uses the multi-dimensional feature vector. A multi dimensional feature vector of each trait is first normalized and then, it is projected to a lower dimensional feature space. The reduced dimensional feature vectors are fused at feature level and the fused feature vectors are used to index the database by forming Kd-tree. The proposed method reduces the data retrieval time along with possible error rates. The system is tested on multimodal databases (feature level fusion of ear, face, iris and signature) consists of 5400 images of 150 subjects (*i.e.* 9 images per subject per trait). Out of the 9, 8 images are used for training and 1 is used for testing. The performance of the proposed indexing technique has been compared with indexing based on score level fusion. It is found that proposed technique based on feature level fusion performs better than score level fusion.

Keywords: indexing, feature level fusion, Kd-tree, multi-dimensional data structure.

1 Introduction

Biometric system provides an automated method to verify or to identify an individual based on unique behavioral or physiological characteristics. Such a system consists of biometric readers, or sensors; feature extractor to compute salient attributes from the input template; and feature matcher for comparing two sets of biometric features. An authentication system consists of two subsystems; one for enrollment while other for authentication. During enrollment, biometric measurements are captured from a subject and relevant information is stored in the database. The task of the authentication module is to recognize a subject as a later stage, and is either identification of one person among many, or verification that a person's biometric matches a claimed identity [1]. Let T_i be the feature vector of i^{th} template in the database of d dimension and is defined as follows:

$$T_i = [f_1, \dots, f_d] \quad (1)$$

If the query image Q with feature vector of d dimension is defined as $Q = [q_1, \dots, q_d]$, then $\forall j$, q_j may not be same as f_j , where f_j and q_j are the j^{th} feature values of T_i and Q respectively. For a given query template Q , the problem of identification system is to find the n nearest neighbors in the database consisting of N templates. To make the system more powerful and fast, a feature matcher should search for the templates with some pruning technique. Traditional databases index the records in an alphabetical or numeric order for efficient retrieval. In biometric database, there is no natural order by which one can keep the biometric templates. Hence there is a need for good indexing technique to make the search process more efficient. Since the feature vector generated for every templates are known a priori, these vectors can be arranged in such a way that an efficient searching algorithm can be used.

In the literature, there are few techniques available to index the biometric templates. These techniques either cluster the multi-dimensional templates and uses cluster information for searching or map the multi-dimensional feature vector to a scalar value [2] and use an existing data structure to index it. The study of reducing the search space of biometric databases is already made by Center for Unified Biometrics and Sensors (CUBS) group. Their effort is based on the binning/clustering technique. The effort of binning was performed in [3] and it was demonstrated that the search space could be reduced to approximately 5% of the original database. The data was clustered using K-means clustering algorithm. The test template is then associated with the clusters it is closest to, with the new search space being the templates within these closest clusters. However, the binning approach needs re-partition the entire database on addition of new samples to it. Thus binning/clustering systems are useful only in cases of static databases. In [2], CUBS group made another effort to index biometrics database using pyramid technique. Pyramid technique partitions a d -dimensional universe into $2d$ pyramids that meet at the center of the universe. Then, every pyramid is divided into several slices parallel to the basis of the pyramid. The d -dimensional vectors representing points in space are approximated by one-dimensional quantities, called pyramid values, which are indexed by a B+ tree. However, the pyramid technique is ineffective in situations when the data points fall on or near the boundaries of the original space. Another problem in this technique is the increased storage overhead due to the need to keep the index of both the pyramid values and the original points. Kd-tree stores actual data (original points) based on the i^{th} discriminator as a key value in a node, and hence there is no storage overhead as like pyramid technique. In addition since Kd-tree is based on space partitioning indexing technique, there is no overlapping between nodes as like bounding region based indexing techniques such as R-tree, R*-tree, M-tree and X-tree [4,5]. This paper proposes an efficient indexing technique which reduces the search space of multimodal biometric databases for any query template using Kd-tree with feature level fusion.

Section 2 presents dimension reduction technique and Kd-tree which serve as the foundation for further discussion. Section 3 describes the feature extraction algorithms for Iris, Signature, Ear and Face biometric traits. In Section 4 the

indexing technique has been proposed using Kd-tree with feature level fusion for the multimodal identification system. Results are analysed in the section 5. Conclusion is given in Section 6.

2 Preliminaries

This section discusses some of the basic techniques which are required in developing the proposed indexing technique. Section 2.1 describes dimension reduction technique that has been used to reduce the dimension of the feature vectors. Section 2.2 describes the Kd-tree data structure which is used to index the databases by forming Kd-tree.

2.1 Dimension Reduction Technique

Suppose, we have a dataset T consisting of $\{t_i\}_{i=1}^N$ training samples. Each sample is described by a set of features F ($d = |F|$), so there are N samples described by d dimensional feature vector each. This can be represented by feature object matrix $T_{d \times N}$ where each column represents a sample. The objective of dimension reduction is to reduce the data into another set of features F' where $k = |F'|$ and $k < d$; $T_{d \times N}$ is reduced to $T'_{k \times N}$. Typically, this is a linear transformation $T' = WT$ that reduces dataset T to T' in k dimension, where $W = W_{k \times d}$ is the transformation technique. Principal Component Analysis (PCA) which is the dominant dimension reduction technique, transforms the data into a reduced space that captures most of the variance in the data. PCA reduces dimension of a dataset by retaining those characteristics of the dataset that contribute most to its variance. It can be done by keeping lower-order principal components and ignoring higher-order ones. Such low-order components often contain the most important aspects of the data. PCA can be mathematically defined [6] as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie

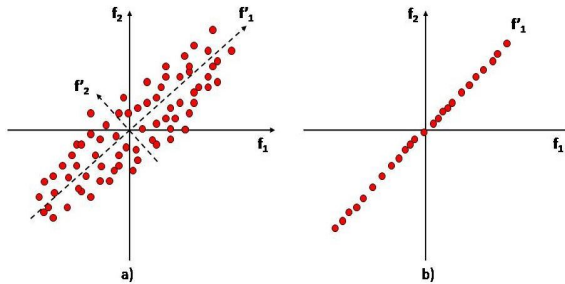


Fig. 1. PCA in 2 – D , the variance in the f'_1 direction is maximum. a) Solid lines: The original basis; Dashed lines: The PCA basis; b) The projection (1D reconstruction) of the data using the first principal component.

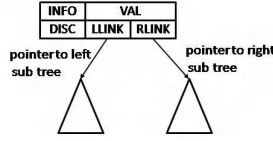


Fig. 2. Structure of Kd-tree node

on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. PCA is theoretically an optimum transformation for a given data in least square terms. Dimensionality reduction in PCA is accomplished by computing a set of eigenvalues of total scatter matrix S_t of the data samples defined as:

$$S_t = \sum_{i=1}^N (t_i - m)(t_i - m)^T \quad (2)$$

where m is the mean value of the sample set T . For dimensionality reduction, k (where $k < d$) eigenvectors $U = [u_1, \dots, u_k]$ corresponding to first k largest eigenvalues of S_t are selected. Reduced dimension training samples $T' = [t'_1, \dots, t'_n]$ can be obtained by the transformation $T' = U^T T$. Now, when a probe template t_p is presented for identification/verification, it is projected on U to obtain a reduced dimension vector $t'_p = U^T t_p$. Geometric interpretation of PCA is shown in Fig. 1. To see how the data is spread, we encapsulate the dataset inside an ellipse and take a look at the major and minor axes that form the vectors f'_1 and f'_2 . These are the principal component axes *i.e.* the base vectors that are ordered by the variance of the data. PCA finds these vectors and gives a $[f'_1, f'_2] \rightarrow [f'_1]$ transformation. While this example is for $2 - D$, PCA works for multi-dimensional data, and it is generally used with high dimensionality problems.

2.2 Kd-Tree Data Structure

The proposed indexing technique is based on the Kd-tree data structure [7,4,8]. This section discusses the salient features of Kd-tree. It is a binary tree that represents a hierarchical subdivision of space using splitting planes that are orthogonal to the coordinates axes. Kd-tree is a space-partitioning data structure for organizing points in a k dimensional space. Any application in which features are generated as multi-dimensional is a potential application for Kd-tree. Structure of a node in Kd-tree is given in Fig. 2. Each node in Kd-tree consists of five fields. Node contains two pointers known as *LLINK* and *RLINK*, which pointing to left subtree and right subtree respectively, if exists. Otherwise, it points to *null*. The field *VAL* is an array of length k containing real feature vector. The *INFO* field contains descriptive information about the node. The *DISC* field is a discriminator, which is an integer between 1 and k , both inclusive. In general, for any node P in the Kd-tree, let i be $DISC(P)$ and is defined

as $level(P) \bmod k$. Then for any node L in $LLINK(P)$, $L.VAL[i] < P.VAL[i]$; likewise, for any node R in $RLINK(P)$, $R.VAL[i] \geq P.VAL[i]$. All nodes on any given levels of the tree have the same discriminator. The root node has discriminator 1, and its two sons have discriminator 2, and so on to the k^{th} level on which the discriminator is k . Again the $(k+1)^{th}$ level has discriminator 1, and the cycle repeats; In general, next discriminator denoted as $NEXTDISC$, is a function defined as

$$NEXTDISC(i) = (i + 1) \bmod k \quad (3)$$

Number of nodes in the Kd-tree are same as the number of templates in the input file to be inserted in the tree. As it is mentioned already that k is the dimensionality of the template.

In order to insert a node P having the data into the Kd-tree, it starts searching from the root of the Kd-tree and finds its appropriate position where the node can be inserted. Bentley [7] shows that the average cost of inserting and searching a node in Kd-tree consisting of N nodes is $O(\log_2 N)$.

Further, in order to perform a range search [9] for a given query template Q with a distance r , it determines all templates T having euclidean distance from Q less than or equal to r . The average cost to perform a range search in Kd-tree consisting of N nodes is $O(k.N^{1-1/k})$ [7].

3 Feature Extraction

A template in the proposed system is represented by four real feature vectors: iris feature vector T_I , signature feature vector T_S , ear feature vector T_E and face feature vector T_F . The signature feature vector is obtained using parameter extraction algorithm [10,11], while other feature vectors are extracted using discrete haar wavelet transform. In this section, the process of feature extraction for each trait has been discussed.

3.1 Iris Feature Extraction

Features are extracted from the iris image using localization of inner and outer iris boundaries [12,13]. In the proposed strategy, the inner iris boundary is localized on the iris image using circular hough transformation [14,15]. Once the inner iris boundary (which is also the boundary of the pupil) is obtained, outer iris is determined using intensity variation approach [16]. The annular portion of iris after localization is transformed into rectangular block to take into consideration the possibility of pupil dilation. This transformed block is used for feature extraction using Discrete Haar Wavelet Transform (DHWT). Haar wavelet operates on data by calculating the sums and differences of adjacent values. It operates first on adjacent horizontal values and then on adjacent vertical values. The decomposition is applied up to four levels on transformed rectangular iris block as shown in Fig. 3. A d dimensional real feature vector T_I is obtained from the fourth level decomposition and is given by

$$T_I = [i_1, \dots, i_{d_I}] \quad (4)$$

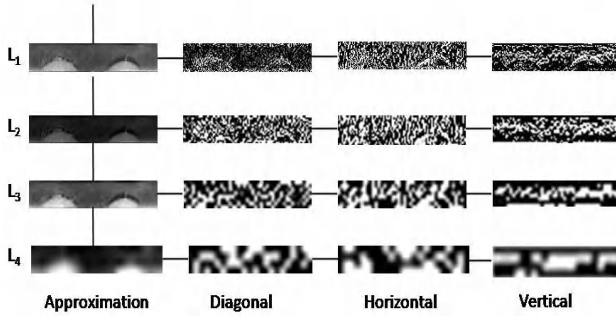


Fig. 3. Four levels discrete haar wavelet transform on iris strip

3.2 Signature Feature Extraction

The signature image is scanned and cropped [approximately 300×150 pixels] in the required format. The image is first passed to the noise removal module which returns a noise free image in the form of a $2 - D$ integer array of same size as the input image. The image extraction module is called to extract the binary, high pressure region (HPR), and thinned images from the array.

Features can be classified into two types: global and local features, where global features are characteristics that identify or describe the signature as a whole and local features are confined to a limited portion (e.g. a grid) of the signature [10,11]. Examples of global features include width and height of individual signature components (as shown in Fig. 4.), width to height ratio, total area of black pixels in the binary and high pressure region (HPR) images, horizontal and vertical projections of signature images, baseline, baseline shift, relative position of global baseline and center of gravity with respect to width of the signature, number of cross and edge points, slant, run lengths etc. These parameters are extracted from a signature template and are stored as feature vector of d dimension as follows:

$$T_S = [s_1, \dots, s_{d_S}] \quad (5)$$

3.3 Ear Feature Extraction

The given input side face image is first resized to 240×340 pixels, and the ear part is cropped from the side face image. Discrete haar wavelet decomposition

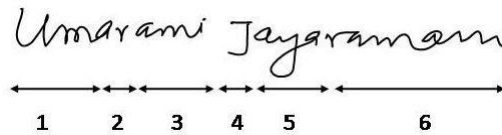


Fig. 4. Different components of the signature image

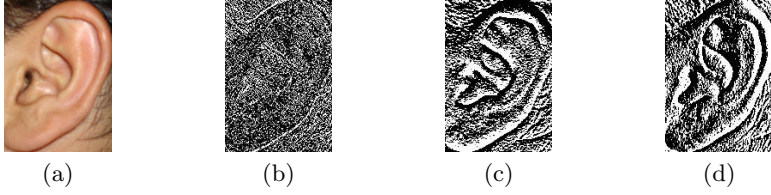


Fig. 5. Two levels of haar wavelet transform on ear image (a) Input ear image, (b) Diagonal coefficient, (c) Horizontal and (d) Vertical

is applied to the cropped ear image to extract the wavelet Approximation (CA), Diagonal (CD), Horizontal (CH) and Vertical (CV) coefficients. Fig. 5. shows an example for the two levels of the haar wavelet decomposition applied on the cropped input ear image. The decomposition is applied up to the fifth level on cropped ear image and a d dimensional real feature vector T_E is obtained from the fifth level decomposition is stored as

$$T_E = [e_1, \dots, e_{d_E}] \quad (6)$$

3.4 Face Feature Extraction

The given input face image is resized to 640×480 pixels and face part of the image is cropped leaving the background (detected face image of size 120×250 pixels). Haar wavelet is used for extracting the features from a detected face image. The detected input image is decomposed up to the required level giving an Approximation (CA), Vertical (CV), Horizontal (CH) and Diagonal (CD) coefficients using the haar wavelet transformation. Fig. 6. shows an example for the two levels of haar wavelet decomposition is applied on the detected face image and the corresponding Diagonal, Horizontal and Vertical coefficients. A d dimensional real feature vector T_F is obtained from the fifth level decomposition and is given by

$$T_F = [f_1, \dots, f_{d_F}] \quad (7)$$



Fig. 6. Two levels of haar wavelet transform on face image (a) Input face image, (b) Diagonal coefficient, (c) Horizontal and (d) Vertical

4 The Proposed Indexing Technique

The proposed method is based on Kd-tree with feature level fusion. Feature level fusion [17] refers to combining different features sets extracted from multiple biometric sources. When the feature sets are homogeneous (e.g. multiple measurements of a person's signature), a single resultant feature vector can be calculated as a weighted average of the individual feature vector. When the feature sets are non-homogeneous (e.g. features of different biometric modalities like face and signature), we can concatenate them to form a single feature vector. The following subsection explains the concatenation of non-homogeneous feature vectors (ear, face, iris, signature) and Fig. 7. shows the overview of the proposed indexing technique.

4.1 Feature Normalization

The individual feature values of vectors $X = [x_1, x_2, \dots, x_m]$ and $Y = [y_1, y_2, \dots, y_n]$ may exhibit significant difference in range as well as form (i.e. distribution). So there is a need of feature level normalization to modify the location (mean) and scale (variance) of the features values via a transformation function in order to map them into a common domain. Adopting an appropriate normalization scheme also helps to address the problem of outliers in feature values. Let x and x' denote a feature vectors before and after normalization, respectively. The min-max technique of normalization can be used to compute x' as:

$$x' = \frac{x - \min(F_x)}{\max(F_x) - \min(F_x)} \quad (8)$$

where F_x is the function which generates x , and $\min(F_x)$ and $\max(F_x)$ represent the minimum and maximum values of F_x for all possible x values, respectively. The

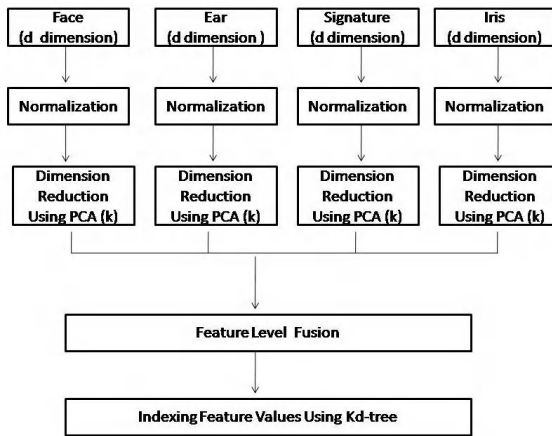


Fig. 7. Overview of proposed indexing technique

min-max technique is effective when the minimum and the maximum values of the component feature values are known beforehand. Normalizing the feature values results in modified feature vectors $X' = [x'_1, x'_2, \dots, x'_m]$ and $Y' = [y'_1, y'_2, \dots, y'_n]$. In the proposed technique, feature values for all traits are normalized between 0 and 1 using min-max technique.

4.2 Dimension Reduction Using PCA

The given real four feature vectors T_I, T_S, T_E and T_F of d dimension are reduced to k dimension using the PCA. The features extracted from four traits are of very high dimensions. Since all the features generated are not equally important, proposed technique uses PCA to reduce the dimensions which represents the compact information. In the proposed technique, the four traits features of ear, face, iris and signature are reduced from $e_{d_E}, f_{d_F}, i_{d_I}$ and s_{d_S} to $e_{k_E}, f_{k_F}, i_{k_I}$ and s_{k_S} dimensions respectively.

4.3 Feature Level Fusion

The reduced k dimension features are fused to yield a new feature vector T_N to represent an individual. The vector T_N of dimensionality $4k$, ($4k \leq (e_{d_E} + f_{d_F} + i_{d_I} + s_{d_S})$) can be generated by augmenting vectors T_I, T_S, T_E and T_F of iris, signature, ear and face respectively.

4.4 Indexing the Feature Vectors

The fused feature values are indexed using the Kd-tree. Kd-tree has been proposed for efficient storage and retrieval of records having multi-dimensional feature vector. Kd-tree is an appropriate data structure for biometric identification system particularly in the analysis of execution of range search algorithm. The proposed system decreases the search time as the Kd-tree is supporting the range search with good pruning.

4.5 The Technique

Indexing the database implies a logical partitioning of the data space. Based on the above steps, the algorithm for proposed indexing technique is given in Algorithm 1. In this technique, reduction makes use of Kd-tree structure for

Algorithm 1. Proposed Indexing Technique

- 1: Normalize the feature vectors of each trait
 - 2: Reduce the feature vectors using PCA.
 - 3: Fuse the feature vectors at feature level.
 - 4: Form the Kd-tree with the fused feature vectors for indexing.
 - 5: Invoke this indexing technique through multimodal identification system.
-

organizing the data in such away that each node can store one biometric template. In case of range search, only a fraction of the database lying within the range of the indices of the test template would be the search of the entire database.

5 Experimental Results

In order to demonstrate the indexing using Kd-tree for multimodal identification, data for four traits (iris, signature, ear and face) are obtained from the database available at Indian Institute of Technology, Kanpur (IITK). In the following subsections, first we briefly discuss the databases that we have used, and then present our results of indexing on multimodal identification using Kd-tree with feature level fusion and score level fusion.

Iris Database: IITK iris image database is used for the iris data. The database comprises of 1350 iris images of 150 subjects (9 images per subject) from their left eye. The images are JPEG in gray scale and are resized with 450×350 pixels resolution. Out of the 9, 8 images are used for training and 1 image is used for testing.

Signature Database: IITK signature database is used for signature data. The database consists of 1350 signature images of 150 subjects (9 images per subject). These images are in JPEG format. In the predefined sheet, users are asked to sign their signatures in the 9 boxes provided in the sheet. The sheet is scanned at 200 dpi as a gray scale image. Size of the scanned image is around 1700×2300 pixels and the size of one box is 300×150 pixels. Out of the 9, 8 images are used for training and 1 image is used for testing. A sample scanned signature sheet is shown in Figure 8.

Ear Database: Ear database consists of 1350 side images of human faces from IITK with the resolution of 240×340 pixels. The images are taken from 150 subjects (9 images per subject). These images are captured using a digital camera from a distance of 0.5 to 1 meter. Ears are cropped from these input images and used for the feature extraction.

Face Database: Face database consists of 1350 images (9 images per subject) of 150 subjects from IITK. These images are captured using a digital camera from a distance of 0.5 to 1 meter. The face is localized (120×250 pixels) from the given input image by clicking three points. Two points on left and right eyes

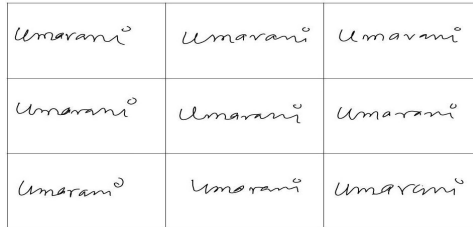


Fig. 8. Scanned signature sheet

and third point on lower lip portion. The localized face (detected face) image is used for the feature extraction.

In our experiment the dimension of feature vectors for each of the traits are found to be 64 for ear and face, 88 for the iris and 27 for the signature respectively.

5.1 Score Level Fusion Indexing Technique

In the score level fusion technique, we index the data of iris, ear, face and signature separately using four Kd-trees, one for each trait. For a given query template, identification is performed using Kd-trees of individual traits (*i.e* iris, ear, face and signature) and the results of the individual traits are fused to get the top matches. In feature level fusion there is only one Kd-tree is formed. For a given query template, identification is performed using the Kd-tree formed to get the top matches.

The experiment proceeds as follows. In our experiment, 64 dimension of face and ear as well as 88 dimension of iris, 27 dimension of signature feature values are reduced to 10 dimensions correspondingly (Top 10 eigen values are considered). This reduced feature vectors are used to index the database by forming four Kd-tree one for each trait. It takes the root of a Kd-tree and query template Q of k dimensions $[q_1, \dots, q_k]$ as input and retrieves all sets of templates T with k dimension $[t_1, \dots, t_k]$ that lies within distance r from Q . Proximity is quantified between them using Euclidean distance. The indexing technique built with Kd-tree for individual traits are invoked (ear, face, iris and signature). The distance r of each tree has to vary to get sets of matched *IDs*. Fix distance r as an optimum, where we get the minimum *FRR* in the multimodal identification system. As a result, we get four separate sets of matched *IDs* for ear, face, iris and signature respectively. These matched *IDs* are fused using weighted sum rule to declare the top matches which are the nearest matched *IDs*. Matching score M for an *ID* x in weighted sum rule is defined as follows:

$$M = w_E \times occ_E(x) + w_F \times occ_F(x) + w_I \times occ_I(x) + w_S \times occ_S(x) \quad (9)$$

where $occ_E(x)$, $occ_F(x)$, $occ_I(x)$, $occ_S(x)$ are the occurrences of *ID* x in ear, face, iris and signature result dataset respectively and w_E , w_F , w_I and w_S are respective assigned weights such that $w_E + w_F + w_I + w_S = 1$. The weights w_E , w_F , w_I and w_S are modeled using the accuracy of the verification system when only individual trait is considered and be defined as follows:

$$Acc_T = Acc_E + Acc_F + Acc_I + Acc_S \quad (10)$$

$$w_E = \frac{Acc_E}{Acc_T}, w_F = \frac{Acc_F}{Acc_T}, w_I = \frac{Acc_I}{Acc_T}, w_S = \frac{Acc_S}{Acc_T} \quad (11)$$

where $Acc_E = 96.0$, $Acc_F = 96.2$, $Acc_I = 95.5$ and $Acc_S = 86.6$ are various accuracies through our experiments we have got when they are individually used in the verification system. The matching score M is calculated for each of the matched *IDs* for an *ID* x . This matching scores are sorted to declare the top matches which are the nearest matched *IDs* for an *ID* x . In this experiment,

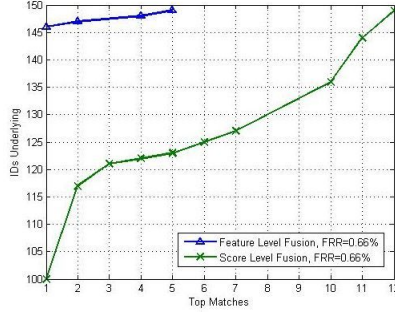


Fig. 9. Top matches vs number of IDs underlying

distance of $r_e = 17$, $r_f = 18$, $r_i = 1200$, and $r_s = 30$ the minimum $FRR=0.66\%$ is achieved, where r_e , r_f , r_i and r_s are distance of ear, face, iris, and signature respectively.

5.2 Feature Level Fusion Indexing Technique

In case of feature level fusion the fused feature vectors (ear, face, iris and signature) T_N of dimensionality $4k$ is used. In our experiment, 64 dimension of face and ear as well as 88 dimension of iris, 27 dimension of signature feature vectors are reduced to 10 dimensions correspondingly (Top 10 eigen values are considered). These reduced 10 dimensions from each of the traits are fused together, as a result we get 40 dimension fused feature vectors (T_N). This 40 dimensional feature vectors are used to index the database by forming Kd-tree. The process of identification built with Kd-tree is invoked. It takes the root of a Kd-tree and query template Q of k dimension $[q_1, \dots, q_k]$ as input and retrieves all sets of templates T with k dimensions $[t_1, \dots, t_k]$ that lies within distance r from Q . Proximity is quantified between them using Euclidean distance. As a result, we get the set of matched IDs then we sort them based on the occurrence and declare the top matched IDs . In this experiment, the distance of $r = 0.67$ the minimum $FRR = 0.66\%$ is achieved.

5.3 Comparison of Feature Level Fusion and Score Level Fusion

In score level fusion there are four separate Kd-tree is formed, one for each trait, while only one Kd-tree is formed in feature level fusion. Hence the search process is more efficient in case of feature level fusion. Fig.9 shows the top matches against IDs underlying for the 150 query templates for both score level fusion and feature level fusion. Indexing based on feature level fusion performs better than the score level fusion. Out of the 150 query template, 146 IDs fall in the first match and 147 IDs fall in the top second match and 149 IDs fall in the top 5 matches with 0.66% FRR . In score level fusion, out of the 150 IDs , 100 IDs fall in the first match, 117 IDs fall in the top second match and 149 IDs fall in the top 12 matches with 0.66% FRR .

6 Conclusion

This paper has proposed an efficient indexing technique using Kd-tree with feature level fusion to reduce the search space of large multimodal biometric databases. Kd-tree has been used for efficient storage and retrieval of records having multi-dimensional feature vector. Kd-tree is shown as appropriate data structure for biometric identification system particularly in the analysis of execution of range search algorithm. The proposed technique decreases the search time as only one Kd-tree is formed which supports range search with good pruning. The performance of the proposed indexing technique (Kd-tree with feature level fusion) has been compared with indexing based on Kd-tree with score level fusion. It is found that feature level fusion performs better than score level fusion and to get top matches for any query template.

Acknowledgment

The authors would like to thank Pradeep Nayak, Gaurav Gadhoke and Anuj Kumar Kaushal who have helped in collecting the data, and reviewers for their valuable comments and suggestions.

References

1. Jain, A.K., Flynn, P., Ross, A.: Handbook of Biometrics. Springer, Heidelberg (2008)
2. Mhatre, A., Chikkerur, S., Govindaraju, V.: Indexing Biometric Databases using Pyramid Technique. LNCS, pp. 841–849. Springer, Heidelberg (2005)
3. Mhatre, A., Palla, S., Chikkerur, S., Govindaraju, V.: Efficient search and retrieval in biometric databases. SPIE Defense and Security (2001)
4. Samet, H.: The Design and Analysis of Spatial Data Structures. Addison-Wesley, Reading (1990)
5. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: ACM SIGMOD Int. Conference on Management of Data, pp. 47–57 (1984)
6. Jolliffe, I.T.: Principal Component Analysis, 2nd edn. Springer, Heidelberg (2002)
7. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM 18(9), 509–517 (1975)
8. Bentley, J.L.: K-d trees for semidynamic point sets. In: Proc. 6th Annual Symposium on Computational Geometry, pp. 187–197 (1990)
9. Bentley, J.L., Friedman, J.H.: Data structure for range searching. ACM Computing Surveys 11, 397–409 (1979)
10. Kaewkongka, T., Chamnongthai, K., Thipakorn, B.: Off-line Signature Recognition using Parameterized Hough Transform. In: Proc. of Int. Symposium on Signal Processing and Its Applications, vol. 1, pp. 451–454 (1999)
11. Jin, A.B., Ooi, O.T.S., Teoh, S.Y.: Offline Signature Verification through Biometric Strengthening. In: IEEE Workshop on Automatic Identification Advanced Technologies, pp. 226–231 (2007)
12. Daugman, J.: How iris recognition works. IEEE Transactions on Circuits and Systems for Video Technology 14(1), 21–30 (2004)

13. Wildes, R.P.: Iris Recognition: An Emerging Biometric Technology. *Proc. of the IEEE* 85(9), 1348–1363 (1997)
14. Chen, T.-C., Chung, K.-L.: An Efficient Randomized Algorithm for Detecting Circles. *Computer Vision and Image Understanding* 83(2), 172–191 (2001)
15. He, X., Shi, P.: A novel iris segmentation method for hand-held capture device. *LNCS*, pp. 479–485. Springer, Heidelberg (2005)
16. Ma, L., Tan, T.N., Wang, Y.H., Zhang, D.: Local intensity variation analysis for iris recognition. *Pattern Recognition* 37(6), 1287–1298 (2004)
17. Feature Level Fusion Using Hand and Face Biometrics. In: *SPIE conference on Biometric Technology for Human Identification II*. vol. 5779 (2005)

Audio Watermarking Based on Quantization in Wavelet Domain

Vivekananda Bhat K., Indranil Sengupta, and Abhijit Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur 721 302, India
{vbk, isg, abhij}@cse.iitkgp.ernet.in

Abstract. A robust and oblivious audio watermarking based on quantization of wavelet coefficients is proposed in this paper. Watermark data is embedded by quantizing large wavelet coefficient in absolute value of high frequency detail sub-band at the third level wavelet transform. The watermark can be extracted without using the host signal. Experimental results show that the proposed method has good imperceptibility and robustness under common signal processing attacks such as additive noise, low-pass filtering, re-sampling, re-quantization, and MP3 compression. Moreover it is also robust against desynchronization attacks such as random cropping and jittering. Performance of the proposed scheme is better than audio watermarking scheme based on mean quantization.

Keywords: Audio watermarking, quantization, discrete wavelet transform (DWT).

1 Introduction

With the rapid development of the Internet and multimedia technologies in the last decade, the copyright protection of digital media is becoming increasingly important and challenging. Digital watermarking [1] has been proposed as a tool to a number of media security problems. The purpose of audio watermarking is to supply some additional information about the audio by hiding watermark data into it. This watermark data may be used for various applications such as authentication, copyright protection, proof of ownership, *etc.* Research in the area of digital watermarking has focused primarily on the design of robust techniques for Digital Rights Management (DRM).

Audio watermarking technique should exhibit some desirable properties [1]. Imperceptibility and robustness are two fundamental properties of audio watermarking schemes. The watermark should not be visible to the viewer and the watermarking process should not introduce any perceptible artifacts into the original audio signal. In other words, watermark data should be embedded imperceptibly into digital audio media. Also, the watermark should survive after various intentional and unintentional attacks. These attacks may include additive noise, re-sampling, low-pass filtering, re-quantization, MP3 compression, cropping, jittering and any other attacks that remove the watermark or confuse

watermark reading system [2]. A trade-off should be maintained between these two conflicting properties.

In general, the audio watermarking techniques can be classified into different ways. The easier watermarking techniques were almost time domain approaches. The simplest method is based on embedding the watermark in the least significant bits (LSB's) of audio samples. Probably the most famous time domain technique proposed in [3] is based on human auditory system (HAS). However, time domain techniques are not resistant enough to MP3 compression and other signal processing attacks. For example, a simple low-pass filtering may eliminate the watermark. Transform domain watermarking [1] schemes are those based on the fast Fourier transform (FFT), and DWT, typically provide higher audio fidelity and are much robust to audio manipulations. A very few quantization based audio watermarking schemes have been proposed in the literature. Wang *et al.* [4] proposed blind audio watermarking technique based on mean quantization. Low frequency wavelet coefficients are selected to embed watermark data. From the experimental results it follows that this method is robust to common signal processing attacks. But the robustness against desynchronization attacks is not discussed in this scheme. Kalantari *et al.* [5] presented a oblivious scheme based on mean quantization in wavelet domain. The watermark data is embedded by quantizing the means of two selected wavelet sub-band. The robustness of this scheme is not discussed against the desynchronization attacks. A novel quantization based audio watermarking algorithm using discrete cosine transform is discussed in [6]. Experimental results shows that, this method has compromised audibility and robustness in better manner. Ronghui *et al.* [7] gave an semi-fragile quantization based audio watermarking scheme in wavelet domain. The watermark is embedded by changing the value of selected coefficients using quantization. This scheme has limited robustness against common signal processing. A fragile audio watermarking using adaptive wavelet packets based on quantization is reported in [8]. We present an audio watermarking algorithm in the wavelet domain based on quantization. The motivation of our work is based on the idea of image watermarking technique proposed in [9]. The important features of the proposed algorithm are: (i) The binary watermark is encrypted using Arnold transform. (ii) Watermark is embedded using quantization of wavelet coefficients. (iii) Watermark extraction is blind without using original audio signal. (iv) The proposed algorithm has better robustness compared to the scheme [4] against common signal processing and desynchronization attacks. The rest of this paper is organized as follows: A detail overview of the proposed scheme are given in section 2. The experimental results of this scheme are presented and discussed in section 3. Finally, section 4 concludes the paper.

2 Quantization Based Watermarking Scheme

The basic idea in the DWT for a one dimensional signal is the following. A signal is split into two parts, usually high frequency detail sub-band and low frequency approximate sub-band using wavelet filter. The low frequency part is split again

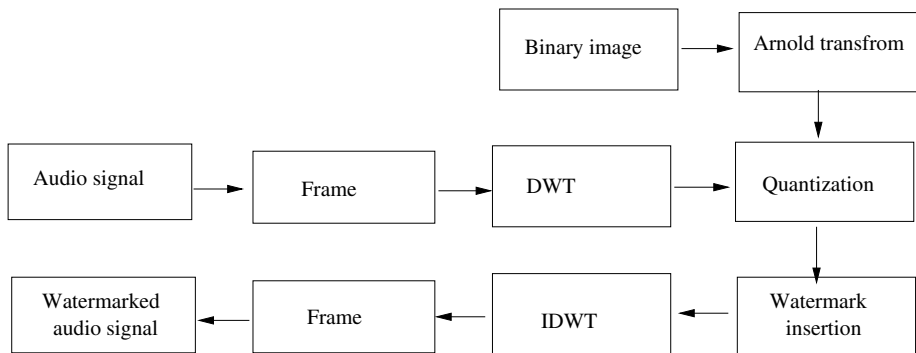


Fig. 1. Watermark embedding scheme

into two parts of high and low frequencies. This process is repeated finite number of times. The original signal is restored back similarly using inverse DWT (IDWT). The Arnold transform [10] is used for watermark permutation to ensure security and robustness of the proposed scheme. The watermark is embedded by quantizing the largest wavelet coefficient in absolute value of the high frequency detail sub-band. The detail steps involved in the watermark embedding and extraction are outlined below.

2.1 Embedding Scheme

The block diagram of the watermark embedding algorithm is shown in Fig. 1. The watermark embedding process involves the following steps:

Step 1: The watermark data P is a binary image of size $N \times N$ and is given by $P = \{p(n_1, n_2) : 1 \leq n_1 \leq N, 1 \leq n_2 \leq N, p(n_1, n_2) \in \{0, 1\}\}$.

Step 2: The watermark is pre-processed before embedding into the host signal. All the elements P will be scrambled by applying Arnold transform. Due to the periodicity of the Arnold transform, the watermark can be recovered easily after transformation. Let $(n_1, n_2)^T$ is the coordinate of the watermark image's pixel, then $(n'_1, n'_2)^T$ is the coordinate after the transform. Arnold transform can be expressed as

$$\begin{bmatrix} n'_1 \\ n'_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \pmod{N} \quad (1)$$

We should convert the two-dimensional scrambled watermark into the one-dimensional sequence in order to embed it into the audio signal. The corresponding one-dimensional sequence is given by: $W = \{w(k) = p(n'_1, n'_2) : 1 \leq n'_1 \leq N, 1 \leq n'_2 \leq N, k = (n'_1 - 1) \times N + n'_2, 1 \leq k \leq N \times N\}$.

Step 3: The original audio is first divided into non-overlapping frames, with a frame size of 2048 samples. The number of frames is equal to the size of the watermark data.

Step 4: The audio frame is decomposed into 3-level wavelet transform using 4-coefficient Daubechies wavelet (db4) filter. The wavelet coefficients after decomposition are given by $cA3, cD3, cD2$, and $cD1$, where $cA3$ is the low frequency approximate coefficients of the audio and $cD3, cD2, cD1$ are the high frequency detail coefficients of the audio.

Step 5: Select the largest coefficient $cD3_{max}(m)$ in absolute value of $cD3$ to embed watermark using quantization.

Step 6: Quantize the largest coefficient as follows. The following quantization function Q is used during the embedding and extraction process.

$$Q(cD3_{max}(m)) = \begin{cases} 0 & \text{if } \lfloor \frac{cD3_{max}(m)}{\Delta 2^3} \rfloor \text{ is even} \\ 1 & \text{if } \lfloor \frac{cD3_{max}(m)}{\Delta 2^3} \rfloor \text{ is odd} \end{cases} \quad (2)$$

where Δ is a user defined positive real number called quantization parameter and $\lfloor \cdot \rfloor$ is the floor integer function.

If $Q(cD3_{max}(m)) = w(k)$, then no change is made to the coefficient. If $Q(cD3_{max}(m)) \neq w(k)$ then

$$cD3_{max}(m) = \begin{cases} cD3_{max}(m) - \Delta 2^3 & \text{if } cD3_{max}(m) > 0 \\ cD3_{max}(m) + \Delta 2^3 & \text{if } cD3_{max}(m) \leq 0 \end{cases} \quad (3)$$

Step 7: Inverse DWT is applied to the modified wavelet coefficients to get the watermarked audio signal.

2.2 Extraction Scheme

The watermark can be extracted without using original audio signal. The extraction algorithm is given below:

Step 1: The input watermarked audio signal is segmented into non-overlapping frames, with a frame size of 2048 samples.

Step 2: The frame is transformed into 3-level wavelet transform using db4 filter.

Step 3: Select the largest coefficient $cD3'_{max}(m)$ in absolute value of 3^{rd} level high frequency detail coefficient $cD3'$.

Step 4: The extracted watermark is given by, $w'(k) = Q(cD3'_{max}(m))$.

Step 5: The extracted watermark is de-scrambled using inverse Arnold transform to obtain the original binary watermark image.

3 Experimental Results and Comparison

Audio files used in the experiment are 16 bit mono audio signal in WAVE format sampled at 44100 Hz sampling rate. A plot of a short portion of the jazz audio signal and its watermarked version is shown in Fig. 2. The embedded watermark is a Crown binary logo image of size 36×36 shown in Fig. 3. The binary

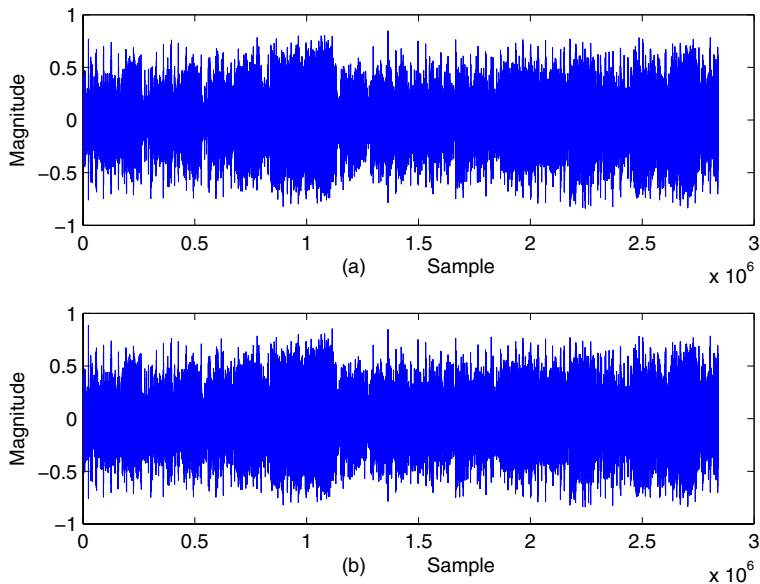


Fig. 2. (a) Jazz audio signal (b) Watermarked jazz audio signal



Fig. 3. Binary watermark

watermark is encrypted using Arnold transform. The signal to noise ratio (SNR) for evaluating the quality of watermarked signal is given by the equation:

$$SNR = 10 \log_{10} \frac{\sum_{a=1}^M Z^2(a)}{\sum_{a=1}^M [Z(a) - Z^*(a)]^2} \quad (4)$$

where Z and Z^* are original audio signal and watermarked audio signal respectively.

The SNR of all selected audio files are above 20 dB. This ensures fact that watermarked audio signal is quite similar to original audio signal. Which means our watermarking method has good imperceptibility. We know that there is a trade-off between imperceptibility of the watermark with different value of the quantization parameter Δ . A large value of the parameter makes the watermark robust, but it will destroy quality of original signal. A small value of the parameter allows us to achieve good imperceptibility, but it is fragile to attacks. The following attacks were performed to test the robustness and effectiveness of our scheme.











Watermark					
NC	1	0.9932	0.9847	0.9729	0.9050
Watermark					
NC	0.8143	0.8116	0.8062	0.7899	0.7831

Fig. 4. Extracted watermark with various NC

- (i) Additive white Gaussian noise (AWGN): White Gaussian noise is added so that the resulting signal has a SNR of 30 dB.
- (ii) Re-sampling: The watermarked signal originally sampled at 44.1 kHz is re-sampled at 22.05 kHz, and then restored by sampling again at 44.1 kHz.
- (iii) Low-pass filtering: The low-pass filter used here is a second order Butterworth filter with cut-off frequency 11025 Hz.
- (iv) Re-quantization: The 16-bit watermarked audio signals have been re-quantized down to 8 bits/sample and back to 16 bits/sample.
- (v) MP3 Compression: The MPEG-1 layer 3 compression with 64 kbps is applied.
- (vi) Cropping: Segments of 500 samples were randomly removed and replaced with segments of the signal attacked with filtering and additive noise.
- (vii) Jittering: Jittering is an evenly performed form of random cropping. We removed one sample out of every 2000 samples in our jittering experiment.

The similarity measurement between extracted watermark and original watermark can be evaluated using normalized correlation (NC), which is defined as follows:

$$NC(Y, Y^*) = \frac{\sum_{i=1}^N \sum_{j=1}^N Y(i, j) Y^*(i, j)}{\sqrt{\sum_{i=1}^N \sum_{j=1}^N Y(i, j)^2} \sqrt{\sum_{i=1}^N \sum_{j=1}^N Y^*(i, j)^2}} \quad (5)$$

where Y and Y^* are original and extracted watermarks respectively, i and j are indexes of the binary watermark image.

The bit error rate (BER) is used to find the percentage of the extracted watermark, it is defined by:

$$BER = \frac{B}{N \times N} \times 100\% \quad (6)$$

where B is the number of erroneously detected bits.

Table 1. NC values and BER of extracted watermark from different types of attacks

Audio file	Type of attack	Proposed scheme NC	Scheme [4] NC	Proposed scheme BER (%)	Scheme [4] BER(%)
Classical	Attack free	1	1	0	0
	AWGN	1	0.9729	0	4
	Re-sampling	1	0.9958	0	1
	Low-pass filtering	1	0.9932	0	1
	Re-quantization	1	0.9847	0	2
	MP3 64 kbps	0.9050	0.8909	13	14
	Cropping	1	0.9889	0	2
	Jittering	1	0.9905	0	1
Country	Attack free	1	1	0	0
	AWGN	1	0.9916	0	1
	Re-sampling	1	0.9905	0	1
	Low-pass filtering	1	0.9858	0	2
	Re-quantization	1	0.9852	0	2
	MP3 64 kbps	0.8116	0.7831	26	29
	Cropping	1	0.9905	0	1
	Jittering	1	0.9968	0	0
Jazz	Attack free	1	1	0	0
	AWGN	1	0.9789	0	3
	Re-sampling	1	1	0	0
	Low-pass filtering	1	0.9958	0	1
	Re-quantization	1	0.9937	0	1
	MP3 64 kbps	0.8143	0.7899	26	28
	Cropping	1	0.9963	0	1
	Jittering	1	0.9958	0	1
Pop	Attack free	1	1	0	0
	AWGN	1	0.9920	0	1
	Re-sampling	1	0.9942	0	1
	Low-pass filtering	1	0.9911	0	1
	Re-quantization	1	0.9764	0	4
	MP3 64 kbps	0.8062	0.6324	26	46
	Cropping	1	0.9905	0	1
	Jittering	1	0.9968	0	0

The Fig. 4. shows the some of the extracted watermarks after the above attacks with different NC values. Performance comparison of our method with the audio watermarking scheme based on the mean quantization [4] for the above attacks is summarized in the Table 1. Moreover our scheme has high NC and low BER as compared to the scheme [4] for the above mentioned attacks, which clearly shows the efficiency of our approach. The proposed scheme has NC above 0.8 for all the above attacks, that ensures extracted watermark is recognizable and acceptable. For example, the NC value of our scheme under MP3 compression attack at 64 kbps for pop audio file is 0.8062, where as the NC value for the scheme [4] is 0.6324 under the same attack. This result shows that our scheme

is better than the scheme [4]. Also for the desynchronization attacks such as random cropping and jittering the results of our approach is better than [4]. We observe that the performance is mainly depends on the audio files and watermark logo image used in the experiment. Audio files such as classical, country, jazz, and pop are good host signals for our audio watermarking scheme. These results prove that the proposed method has superior performance.

4 Conclusions

A blind audio watermarking technique resistant to signal processing and desynchronization attacks was presented in this paper. Arnold transform is used to ensure the security and robustness of the watermark data. This technique yields impressive results both in terms of image quality and robustness against various signal processing attacks. Watermarked audio signal maintains more than 20 dB SNR, which assures the imperceptibility of our scheme. From the robustness tests, we can see that the algorithm is robust to common signal processing and desynchronization attacks. Experimental results shows that performance of the proposed scheme is much better than audio watermarking scheme based on mean quantization.

References

1. Cox, I., Miller, M., Bloom, J., Fridrich, J., Kalke, T.: Digital watermarking and steganography, 2nd edn. Morgan Kaufmann, San Francisco (2007)
2. Cvejic, N., Seppanen, T.: Digital audio watermarking techniques and technologies. Information Science Reference, USA (August 2007)
3. Basia, P., Pitas, I., Nikolaidis, N.: Robust audio watermarking in the time domain. *IEEE Transactions on Multimedia* 3(2), 232–241 (2001)
4. Lanxun, W., Chao, Y., Jiao, P.: An audio watermark embedding algorithm based on mean-quantization in wavelet domain. In: 8th International Conference on Electronic Measurement and Instruments, pp. 2-423–2-425 (2007)
5. Kalantari, N., Ahadi, S., Kashi, A.: A robust audio watermarking scheme using mean quantization in the wavelet transform domain. In: *IEEE International Symposium on Signal Processing and Information Technology*, pp. 198–201 (2007)
6. Zhou, Z., Zhou, L.: A novel algorithm for robust audio watermarking based on quantification DCT domain. In: *Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP 2007)*, vol. 1, pp. 441–444 (2007)
7. Tu, R., Zhao, J.: A novel semi-fragile audio watermarking scheme. In: *Proceedings of the 2nd IEEE International Workshop on Haptic, Audio and Visual Environments and Their Applications (HAVE 2003)*, pp. 89–94 (2003)
8. Quan, X., Zhang, H.: Perceptual criterion based fragile audio watermarking using adaptive wavelet packets. In: *Proceedings of the 17th International Conference on Pattern Recognition (ICPR 2004)*, vol. 2, pp. 867–870 (2004)
9. Kundur, D., Hatzinakos, D.: Digital watermarking for telltale tamper proofing and authentication. *Proceedings of the IEEE* 87(7), 1167–1180 (1999)
10. Voyatzis, G., Pitas, I.: Applications of toral automorphisms in image watermarking. In: *Proceedings of International Conference on Image Processing*, vol. 1, pp. 237–240 (1996)

Overwriting Hard Drive Data: The Great Wiping Controversy

Craig Wright¹, Dave Kleiman², and Shyaam Sundhar R.S.³

¹ BDO Kendalls, Sydney, Australia

Craig.Wright@bdo.com.au

² ComputerForensicExaminer.com, Florida, US

dave@davekleiman.com

³ Symantec, USA

shyaam@gmail.com

Abstract. Often we hear controversial opinions in digital forensics on the required or desired number of passes to utilize for properly overwriting, sometimes referred to as wiping or erasing, a modern hard drive. The controversy has caused much misconception, with persons commonly quoting that data can be recovered if it has only been overwritten once or twice. Moreover, referencing that it actually takes up to ten, and even as many as 35 (referred to as the Gutmann scheme because of the 1996 Secure Deletion of Data from Magnetic and Solid-State Memory published paper by Peter Gutmann) passes to securely overwrite the previous data. One of the chief controversies is that if a head positioning system is not exact enough, new data written to a drive may not be written back to the precise location of the original data. We demonstrate that the controversy surrounding this topic is unfounded.

Keywords: Digital Forensics, Data Wipe, Secure Wipe, Format.

1 Introduction

Often we hear controversial opinions on the required or desired number of passes to utilize for properly overwriting, sometimes referred to as wiping or erasing, a modern hard drive. The controversy has caused much misconception, with persons commonly quoting that data can be recovered if it has only been overwritten once or twice. Moreover, referencing that it actually takes up to ten, and even as many as 35 (referred to as the Gutmann scheme because of the 1996 Secure Deletion of Data from Magnetic and Solid-State Memory published paper by Peter Gutmann, [12]) passes to securely overwrite the previous data.

One of the chief controversies is that if a head positioning system is not exact enough, new data written to a drive may not be written back to the precise location of the original data. This track misalignment is argued to make possible the process of identifying traces of data from earlier magnetic patterns alongside the current track.

This was the case with high capacity floppy diskette drives, which have a rudimentary position mechanism. This was at the bit level and testing did not consider the accumulated error.

The basis of this belief is a presupposition is that when a one (1) is written to disk the actual effect is closer to obtaining a 0.95 when a zero (0) is overwritten with one (1), and a 1.05 when one (1) is overwritten with one (1). This we can show is false and that in fact, there is a distribution based on the density plots that supports the contention that the differential in write patterns is too great to allow for the recovery of overwritten data.

The argument arises from the statement that “each track contains an image of everything ever written to it, but that the contribution from each “layer” gets progressively smaller the further back it was made”. This is a misunderstanding of the physics of drive functions and magneto-resonance. There is in fact no time component and the image is not layered. It is rather a density plot.

This is of prime importance to forensic analysts and security personal. The time needed to run a single wipe of a hard drive is economically expensive. The requirements to have up to 35 wipes [12] of a hard drive before disposal become all the more costly when considering large organisations with tens of thousands of hosts. With a single wipe process taking up to a day to run per host through software and around an hour with dedicated equipment, the use of multiple wipes has created a situation where many organisations ignore the issue all together – resulting in data leaks and loss.

The inability to recover data forensically following a single wipe makes the use of data wiping more feasible. As forensic and information security professionals face these issues on a daily basis, the knowledge that a single wipe is sufficient to remove trace data and stop forensic recovery will remove a great deal of uncertainty from the industry and allow practitioners to focus on the real issues.

1.1 What Is Magnetic Force Microscopy¹

Magnetic force microscopy (MFM) images the spatial variation of magnetic forces on a sample surface. The tip of the microscope is coated with a ferromagnetic thin film. The system operates in non-contact mode, detecting changes in the resonant frequency of the cantilever induced by the magnetic field's dependence on tip-to-sample separation. A MFM can be used to image naturally occurring and deliberately written domain structures in magnetic materials. This allows the device to create a field density map of the device.

1.2 MFM Imagery of Overwritten Hard Disk Tracks

The magnetic field topography (Fig. 2A below) was imaged with an MFM to measure the magnetic force density. This image was captured using the MFM in Lift Mode (lift height 35 nm). This results in the mapping of the shift in the cantilever resonant frequency.

¹ The MFM senses the stray magnetic field above the surface of a sample. A magnetic tip is brought into close proximity with the surface and a small cantilever is used to detect the force between the tip and the sample. The tip is scanned over the surface to reveal the magnetic domain structure of the sample at up to 50 nm resolution.

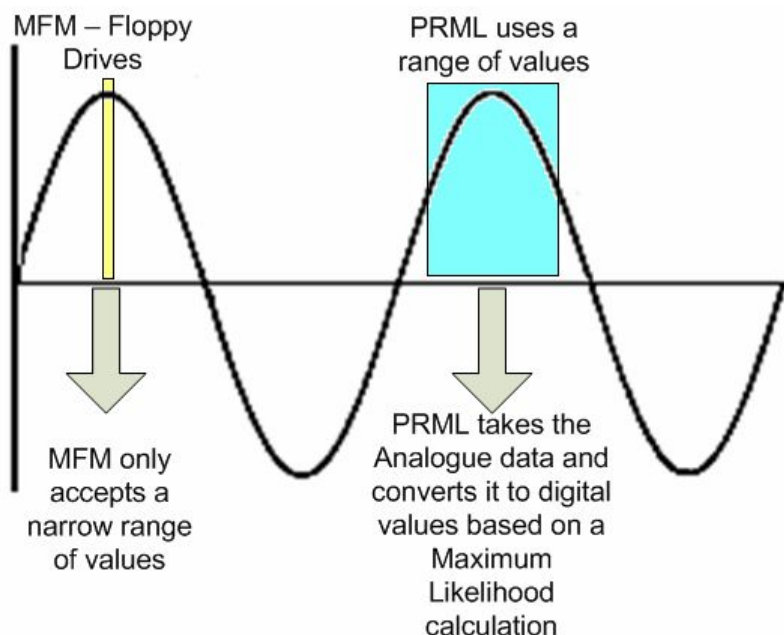


Fig. 1. The concepts of how Partial Response Maximum Likelihood (PRML) (a method for converting the weak analog signal from the head of a magnetic disk or tape drive into a digital signal) (and newer Extended Partial Response Maximum Likelihood (EPRML) drive) encoding is implemented on a hard drive. The MFM reads the unprocessed analog value. Complex statistical digital processing algorithms are used to determine the “maximum likelihood” value associated with the individual reads.

The acquisition time for 1 byte is about 4 minutes (this would improve with newer machines). The image displays the:

- track width and skew,
- transition irregularities, and
- the difference between written and overwritten areas of the drive.

Because of the misconception, created by much older technologies (such as floppy drives) with far lower densities, many believe that the use of an electron microscope will allow for the recovery of forensically usable data. The fact is, with modern drives (even going as far back as 1990) that this entire process is mostly a guessing game that fails significantly when tested. Older technologies used a different method of reading and interpreting bits than modern hard called *peak detection*. This method is satisfactory while the peaks in magnetic flux sufficiently exceed the background signal noise. With the increase in the write density of hard drives (Fig. 3), encoding schemes based on peak detection (such as Modified Frequency Modulation or MFM) that are still used with floppy disks have been replaced in hard drive technologies. The encoding of hard disks is provided using PRML and EPRML encoding technologies that have allowed the write density on the hard disk to be increased by a full 30-40% over that granted by standard peak detection encoding.

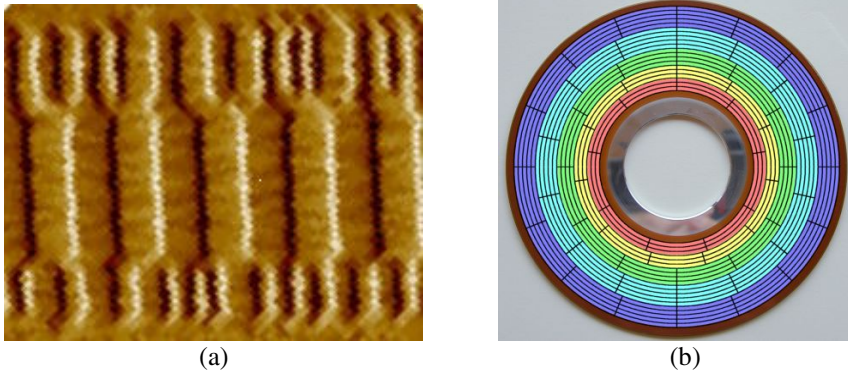


Fig. 2. (a). This image was captured and reconstructed at a 25- μm scan from an Atomic Force Microscope [15]. The image displays the residual from overwrites and alignment. (b). This image from PCGuide.com displays the configuration of a 20 track hard drive. These tracks are separated into five zones which are displayed in a separate color as follows: 5 x 16 sector tracks in the blue zone, 5 x 14 sector tracks in the cyan zone, 4 x 12 sector tracks in the green zone, 3 x 11 sectors tracks in the yellow zone, and 3 x 9 sector tracks in the red.

Additionally, hard disk drives use zoned bit recording (Fig 2b) which differs from floppy drives and similar technologies. Older technologies (including floppy disks) used a single zone with a write density that is several orders of magnitude larger than that used with hard disks. We have not tested recovery from a floppy disk using these methods, but it would be expected that the recovery rate would be significantly greater than with respect of that of a hard disk platter - although still stochastically distributed.

The fact is many people believe that this is a physical impression in the drive that can belie the age of the impression. This misconception is commonly held as to the process used to measure the magnetic field strength. Using the MFM in Tapping Mode², we get a topography image that represents the physical surface of the drive platter.

The magnetic flux density follows a function known as the hysteresis loop. The magnetic flux levels written to the hard drive platter vary in a stochastic manner with variations in the magnetic flux related to head positioning, temperature and random error. The surfaces of the drive platters can have differing temperatures at different points and may vary from the read/write head. This results in differences in the expansion and contraction rates across the drive platters. This differential can result in misalignments. Thermal recalibration is used on modern drives to minimize this variance, but this is still results in an analogue pattern of magnetic flux density.

One of ways used to minimize the resultant error has come through the introduction of more advanced encoding schemes (such as PRML mentioned previously). Rather than relying on differentiating the individual peaks at digital maxima,

² Tapping mode can also be called Dynamic Force mode, intermittent contact mode, non-contact mode, wave mode, and acoustic AC mode by various microscope vendors. When operating in tapping mode the cantilever is driven to oscillate up and down at near its resonance frequency by a small piezoelectric element.

magnetic flux reversals are measured by hard drive heads and processed using an encoding process (PRML or EPRML) that is based on determining maximum likelihood for the flux value using a digital signal sampling process. Complex statistically based detection algorithms are employed to process the analog data stream as it is read the disk. This is the "partial response" component. This stochastic distribution of data not only varies on each read, but also over time and with temperature differentials. This issue has only grown as drive densities increase.

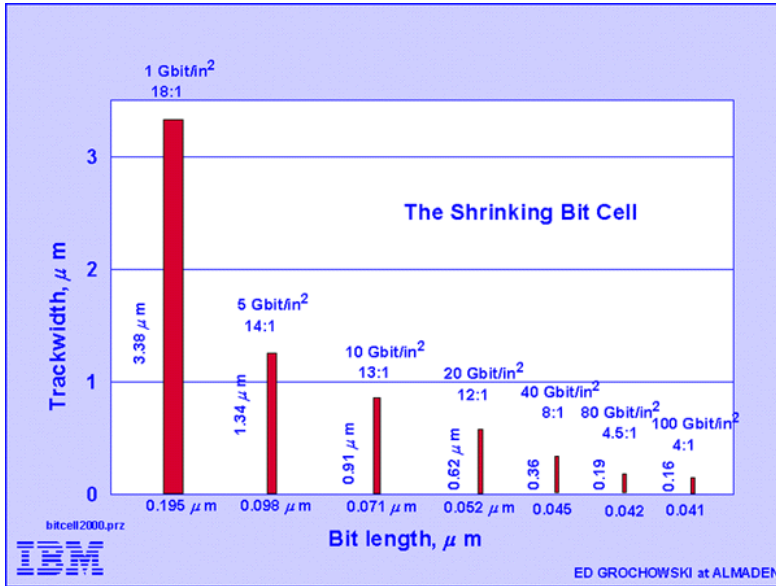


Fig. 3. This graph from IBM demonstrates how the bit size used with modern hard drives is shrinking. This has resulted in a dramatic increase in the density of hard disks which has resulted in the error rate from movement and temperature remaining an issue even with the improvements in compensating technologies.

A Track is a concentric set of magnetic bits on the disk. Each track is commonly divided into 512 bytes sectors. The drive sector is the part of each track defined with magnetic marking and an ID number. Sectors have a sector header and an error correction code (ECC).

A Cylinder is a group of tracks with the same radius.

Data addressing occurs within the two methods for data addressing:

- CHS (cylinder-head-sector) and
- LBA (logical block address).

The issue from Guttmann's paper [12] is that we can recover data with foreknowledge of the previous values, but not with any level of accuracy. The issues with this are twofold. First, to have any chance of recovery it is necessary to have perfect

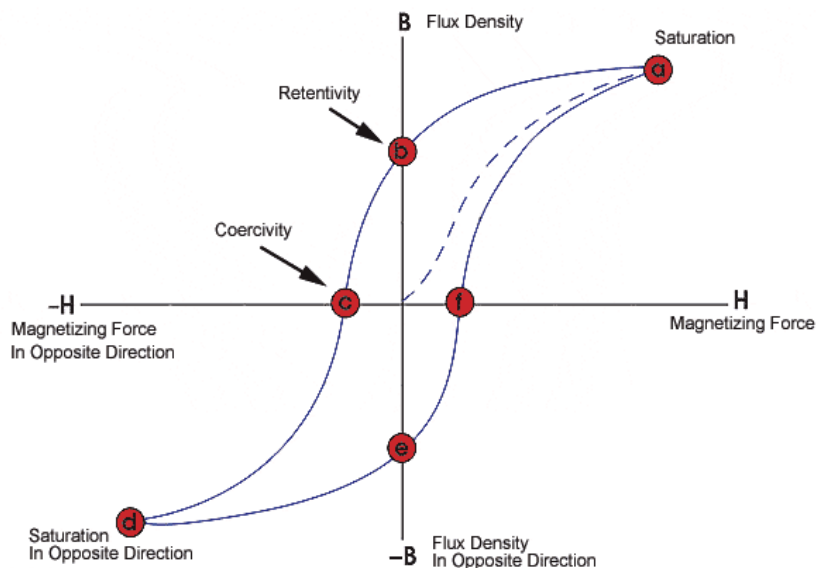


Fig. 4. The hysteresis loop³ demonstrates the relationship between the induced magnetic flux density (B) and the magnetizing force (H). It is often referred to as the B-H loop. This function varies with a number of prevalent conditions including temperature.

knowledge of what was previously written to the drive. This situation most often never occurs in a digital forensic investigation. In fact, if a perfect copy of the data existed, there would be no reason to recover data from the wiped drive. Next, the level of recovery when presented with a perfect image is too low to be of use even on a low density pristine drive (which does not exist in any actual environment). Carroll and Pecora (1993a, 1993b) demonstrated this effect and how stochastic noise results in a level of controlled chaos. The Guttman preposition [12] is true based on a Bayesian a-prior model assuming that we have the original data and the pattern from the overwrite, but of course this defeats the purpose of the recovery process and as noted is still not sufficiently accurate to be of any use. Stating that we can recover data with a high level of accuracy, given that we have the original data, is a tautology, and there would be no reason to do the recovery.

The previously mentioned paper uses the determination that the magnetic field strength is larger or smaller than that which would be expected from a write suggests the prior overwritten value. This is that a factored magnetic field strength of 0.90 or 1.10 (where 1.0 is a “clean” write with no prior information) would represent the previous information written to the drive that has been overwritten. This is postulated to be a means through which the use of an electron microscope could be deployed to recover data from a drive that has been wiped. The problem with this theory is that there are both small write errors on an unwritten sector and remnant magnetic field densities from prior use of the drive sector.

³ Image sourced from Iowa’s State University Center for Nondestructive Evaluation NDT (Non Destructive Testing).

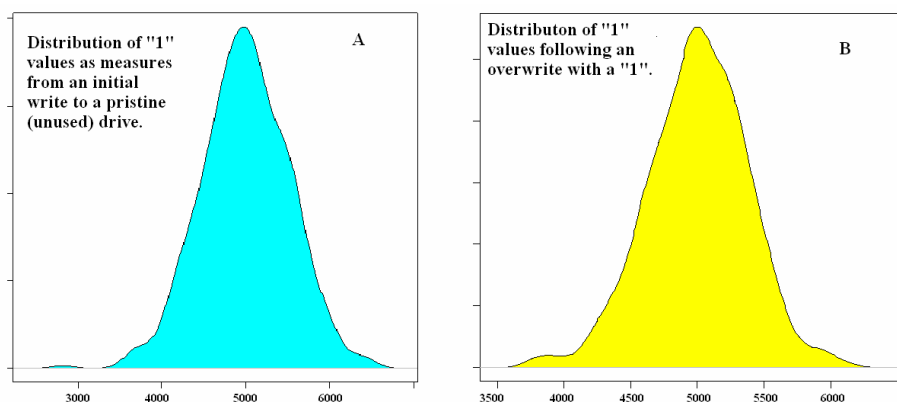


Fig. 5. This example displays the experimentally derived magnetic field density functions for hard drive rewrites where “A” displays the measured distribution of binary “1” values on initial copy. “B” displays the distribution of values associated with a binary “1” value following an overwrite with another binary “1”.

Magnetic signatures are not time-stamped, accordingly there is no “unerase” capability [15]. Figure 4 displays the B-H loop for magnetic flux. Starting at a zero flux density for a drive platter that has not been previously magnetized, the induced flux density created when the drive head writes to the platter follows the dashed line (displayed in Fig. 4) as the magnetizing force is increased. Due to a combination of power constraints, timing issues and write density, modern hard drives do not saturate the magnetic flux on the drive to point “a”. Rather, they use sophisticated statistical measures (PRML and EPRML) to determine the maximum likelihood of the value stored on the drive. In demagnetizing a drive (reducing H to zero) the curve moves from point “a” to point “b” on Figure 4. Some residue from the prior magnetic flux remains in the material even though the magnetizing force is zero. This phenomenon is known as remanence. The retentivity of disk platter will not reach the maxima (defined by points “b” and “d” in figure 4) as the drive heads do not reach saturation. Further, fluctuations in temperature, movement and prior writes influence the permeability⁴ of the platter. Jiles [21] notes that in the event that the temperature of a drive platter is increased from, 20 to 80 centigrade then a typical ferrite can become subject to a 25% reduction in the in permeability of the platter.

Consequently, the B-H curve does not go back to the originating point when the magnetic flux is rewritten and the B-H curve will vary with use due to temperature fluctuations. On subsequent writes, the hysteresis curve follows a separate path from position “f” in Figure 4. As drive heads do not cause the hard drive platter to reach the saturation point, the resultant B-H loop will vary on each write.

⁴ Permeability is a material property that is used to measure how much effort is required to induce a magnetic flux within a material. Permeability is defined the ratio of the flux density to the magnetizing force. This may be displayed with the formula: $\mu = B/H$ (where μ is the permeability, B is the flux density and H is the magnetizing force).

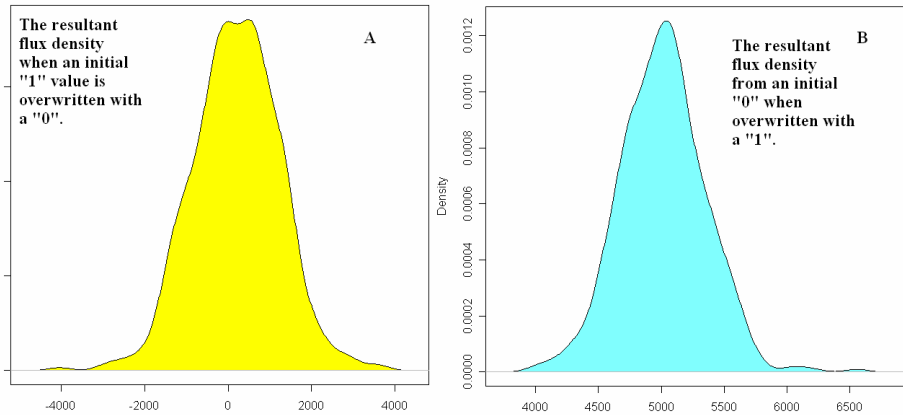


Fig. 6 (A and B). This example displays the magnetic field density functions that were experimentally obtained following a rewrite (wipe) of the prior binary unit on the hard drive. Plot “A” displays the density distribution associated with “1” values following an overwrite with a “0”. Plot “B” displays the density function for an initial “0” value that has been overwritten with a “1”.

A common misconception concerning the writing of data to a hard drive arises as many people believe that a digital write is a digital operation. As was demonstrated above, this is a fallacy, drive writes are analogue with a probabilistic output [6], [8], [10]. It is unlikely that an individual write will be a digital +1.00000 (1). Rather - there is a set range, a normative confidence interval that the bit will be in [15].

What this means is that there is generally a 95% likelihood that the +1 will exist in the range of (0.95, 1.05) there is then a 99% likelihood that it will exist in the range (0.90, 1.10) for instance. This leaves a negligible probability (1 bit in every 100,000 billion or so) that the actual potential will be less than 60% of the full +1 value. This error is the non-recoverable error rating for a drive using a single pass wipe [19].

As a result, there is no difference to the drive of a 0.90 or 1.10 factor of the magnetic potential. What this means is that due to temperature fluctuations, humidity, etc the value will most likely vary on *each* and *every* pass of a write. The distributions of these reads are displayed as histograms in Fig. 5. The distribution is marginally different to the original, but we cannot predict values. From Fig. 6 it is simple to see that even with the prior data from the initial write we gain little benefit. These images display the differences in the voltage readings of the drives (which are determined through the magnetic field strength). Clearly, some values that are more distantly distributed than would be expected in the differenced results (Fig. 6 B) with voltage values that are significantly greater than are expected. The problem is that the number of such readings is far lower than the numbers that result through sheer probability alone.

Resultantly, there is no way to even determine if a “1.06” is due to a prior write or a temperature fluctuation. Over time, the issue of magnetic decay would also come into play. The magnetic flux on a drive decays slowly over time. This further skews the results and raises the level of uncertainty of data recovery.

Consequently, we can categorically state that there is a minimal (less than a 0.01% chance) of recovering any data on a NEW and unused drive that has a single raw wipe pass (not even a low-level format). In the cases where a drive has been used (even being formatted for use) it is not possible to recover the information – there is a small chance of bit recovery, but the odds of obtaining a whole word are small.

The improvement in technology with electron microscopes will do little to change these results. The error from microscope readings was minimal compared to the drive error and as such, the issue is based on drive head alignment and not the method used for testing.

1.3 Read Error Severities and Error Management Logic

A sequence of intricate procedures are performed by the hard drive controller in order to minimise the errors that occur when either writing data to or reading data for a drive. These processes vary with each hard drive producer implementing their own process. Some of the most common error management processes have been listed below.

ECC Error Detection: A drive sector is read by the head. An error detection algorithm is used to determine the likelihood of a read error. In the event that an error state is considered to be unlikely, the sector is processed and the read operation is considered as having been concluded successfully.

ECC Error Correction: The controller uses the ECC codes that it has interpreted for the sector in order to try and correct the error. A read error can be corrected very quickly at this level and is usually deemed to be an "automatic correction".

Automatic Retry: The next phase involves waiting until the drive platter has completed a full spin before attempting to read the data again. Stray magnetic field variances are a common occurrence leading to drive read error. These fluctuations may result due to sudden movement and temperature variations. If the error is corrected following a retry, most drives will judge the error condition to be "corrected after retry".

Advanced Error Correction: Many drives will, on subsequent retries after the first, invoke more advanced error correction algorithms that are slower and more complex than the regular correction protocols, but have an increased chance of success. These errors are "recovered after multiple reads" or "recovered after advanced correction".

Failure: In the event that the drive is incapable of reading the sector, a signal is sent to the drive controller noting a read error. This type of failure is an unrecoverable read error.

Modern encoding schemes (PRML and EPRML) have a wide tolerance range allowing the analogue values that the drive head reads from and writes to a hard disk to vary significantly without loss of data integrity. Consequently, the determination of a prior write value is also a stochastic process.

2 Data and Method

In order to completely validate all possible scenarios, a total of 15 data types were used in 2 categories. Category A divided the experiment into testing the raw drive (this is a pristine drive that has never been used), formatted drive (a single format was completed in Windows using NTFS with the standard sector sizes) and a simulated used drive (a new drive was overwritten 32 times with random data from /dev/random on a Linux host before being overwritten with all 0's to clear any residual data).

The experiment was also divided into a second category in order to test a number of write patterns. Category B consisted of the write pattern used both for the initial write and for the subsequent overwrites. This category consisted of 5 dimensions:

- all 0's,
- all 1's,
- a "01010101 pattern,
- a "00110011" pattern, and
- a "00001111" pattern.

The Linux utility "dd" was used to write these patterns with a default block size of 512 (bs=512). A selection of 17 models of hard drive where tested (from an older Quantum 1 GB drive to current drives dated to 2006). The data patterns where written to each drive in all possible combinations.

1. The data write was a 1 kb file (1024 bits).
2. Both drive skew and the bit was read.
3. The process was repeated 5 times for an analysis of 76,800 data points.

Table 1. Table of Probability Distributions for the older model drives. Note that a "used" drive has only a marginally better chance of any recovery than tossing a coin. The Pristine drive is the optimal case based on an early Seagate 1Gb drive.

Probability of recovery	Pristine drive	Used Drive (ideal)
1 bit	0.92	0.56
2 bit	0.8464	0.3136
4 bit	0.71639296	0.098345
8 bits ⁵	0.51321887	0.009672
16 bits	0.26339361	9.35E-05
32 bits	0.06937619	8.75E-09
64 bits	0.00481306	7.66E-17
128 bits	2.3166E-05	5.86E-33
256 bits	5.3664E-10	3.44E-65
512 bits	2.8798E-19	1.2E-129
1024 bits	8.2934E-38	1.4E-258

⁵ This represents one (1) ASCII character.

The likelihood calculations were completed for each of the 76,800 points with the distributions being analyzed for distribution density and distance. This calculation was based on the Bayesian likelihood where the prior distribution was known. As has been noted, in real forensic engagements, the prior distribution is unknown. This presents this method with an advantage to recovering the data that would not be found when conducting a forensic examination and recovery of a drive.

Even on a single write, the overlap at best gives a probability of just over 50% of choosing a prior bit (the best read being a little over 56%). This caused the issue to arise, that there is no way to determine if the bit was correctly chosen or not. Therefore, there is a chance of correctly choosing any bit in a selected byte (8-bits) – but this equates a probability around 0.9% (or less) with a small confidence interval either side for error.

Resultantly, if there is less than a 1% chance of determining each character to be recovered correctly, the chance of a complete 5-character word being recovered drops exponentially to 8.463E-11 (or less on a used drive and who uses a new raw drive format). This results in a probability of less than 1 chance in 10^{Exp50} of recovering any useful data. So close to zero for all intents and definitely not within the realm of use for forensic presentation to a court.

Table 1 below, shows the mapped out results of probable recovery with a pristine drive of a similar make and model⁶ to that which would have been used in the paper by Dr. Gutmann. This drive had never been used and was had raw data written to it for the first time in this test. The other drive was a newer drive⁷ that has been used (I used this for my daily operations for 6 months) prior to the wiping procedure. A total of 17 variety of drives dated from 1994 to 2006 of both the SCSI and IDE category where tested for this process. A total of 56 drives where tested. On average only one (1) drive in four⁸ (4) was found to function when the platter had been returned after an initial reading with the MFM.

3 Data Relationships

The only discernable relationship of note is between an initial write of a “1” on a pristine drive that is overwritten with a “0”. This is a function of the drive write head and has no correlation to data recovery, so this is a just point of interest and noting to aid in data extraction from a forensic perspective. All other combinations of wipes displayed comparative distributions of data that where suggestive of random white noise.

3.1 Distributions of Data

The tables used in this section display the probabilities of recovery for each of the drives tested. Although the chances of recovering any single bit from a drive are relatively good, the aim in any forensic engagement is to recover usable data that can be presented in court.

⁶ SEAGATE: ST51080N MEDAL.1080 1080MB 3.5"/SL SCSI2 FAST.

⁷ Western Digital WD1200JS.

⁸ 23.5% of drives where able to be used for an overwrite following an initial MFM scan.

These tests were run as a series of 4 tests on each of 17 types of drives. The reported (Table 1) recovery rate of 92% this was the optimal rate (which was itself stochastically distributed). The results were distributed over a wide range of values with the use of the drive impacting on the capacity to recover data.

This clearly shows that any data recovery is minimal and that no forensically sound recovery is possible. The recovery of a single 32 bit value (such as an IP address) is highly unlikely. It has been stated⁹, that the smallest fragment of usable digital forensic evidence is a 32 bit field (the IP address). To be used in a Civil court case, the evidence needs to be subjected to the balance of probability (usually 51%). In a criminal matter, the preponderance is set at between 95% and 99% to account for all reasonable doubt. The rate at which evidence may be recovered using this technique is too low to be useful. In fact, with the optimal recovery under 7% for a single IP address on an older drive. This is an event that cannot occur outside the lab.

The bit-by-bit chance of recovery lies between 0.92 (+/- 0.15)¹⁰ and 0.54 (+/- 0.16)¹¹. We have used the higher probability in the calculations to add an additional level of confidence in our conclusions. This demonstrates that the chances of recovering a single 8-bit character on the pristine drive are 51.32%. The recovery rate of a 32-bit word is 0.06937619 (just under 7%). As such, the chances of finding a single 4 letter word correctly from a Microsoft Word document file is 2.3166E-05 (0.00002317%)

Table 2 below is a table that further illustrates the wiping fallacy. We tested this by completing a single pass wipe, to simulate minimal use we repeated the process.

Once again, we can see the data recovery is minimal.

Table 2. Table of Probability Distributions for the “new” model drives

Probability of re-covery	Pristine drive (plus 1 wipe)	Pristine drive (plus 3 wipe)
1 bit	0.87	0.64
2 bit	0.7569	0.4096
4 bit	0.57289761	0.16777216
8 bits	0.328211672	0.028147498
16 bits	0.107722901	0.000792282
32 bits	0.011604223	6.2771E-07
64 bits	0.000134658	3.9402E-13
128 bits	1.81328E-08	1.55252E-25
256 bits	3.28798E-16	2.41031E-50
512 bits	1.08108E-31	5.8096E-100
1024 bits	1.16873E-62	3.3752E-199

The standard daily use of the drive makes recovery even more difficult, without even considering a wipe, just *prior* use. In this case, the 3 former wipes are used to simulate use (though minimal and real use is far more intensive). The chances of

⁹ Rob Lee, SANS Forensics 508.
¹⁰ For the optimal recovery on an old drive.
¹¹ On a used “new” drive.

recovering a single 8-bit word (a single character) are 0.0281475 (or 2.8%) – which is actually lower than randomly selecting the character.

The calculated probability of recovering data from any used drive that uses a newer encoding scheme (EPRML) and high density was indistinguishable from a random guess. When recovering data from the 2006 model drive, the best determination of the prior write value was 49.18% (+/- 0.11)¹² from the “all 0’s” pattern when overwritten with the “all 1’s” pattern. The other overwrite patterns actually produced results as low as 36.08% (+/- 0.24). Being that the distribution is based on a binomial choice, the chance of guessing the prior value is 50%. In many instances, using a MFM to determine the prior value written to the hard drive was less successful than a simple coin toss.

3.2 Distribution of Recovered Data

The following is a retrieval pattern from the drive. Where the 8-bit word is correctly read, a “1” is listed. Where the value did not match the correct pattern that was written to the drive, a “0” is displayed.

```
[1] 0 0 1 0 1 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1
[48] 0 1 0 1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 0 1 0 0 0
[95] 0 1 1 0 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 0 0 1 1
[142] 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 1 1 1 0 0 0 1 1 0 0 0 1 0 0
[189] 1 0 1 1 0 1 0 1 0 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 1 0 1
[236] 0 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1
[283] 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0
[330] 0 0 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 0 0 1 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0
[377] 1 1 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 0 1
[424] 1 1 0 0 1 1 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0 1 1 0 0 0 1
[471] 1 1 1 1 0 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 0 0 0 0
[518] 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0
[565] 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 1 0 0 1 0 0 0
[612] 1 1 0 1 1 1 1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0
[659] 1 1 0 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1 1 1 0 0 0 1 1
[706] 1 0 0 1 1 1 0 1 0 1 0 1 0 1 1 0 1 0 0 0 0 0 1 1 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 1
[753] 1 0 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0
[800] 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 0 1 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0
[847] 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0
[894] 1 1 1 1 0 1 0 1 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 1 1 1 0 1 0 1 0 0 0 1 0 0 1 1
[941] 1 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0 1 0 0 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0
[988] 0 0 1 0 1 1 1 1 0 1 0 0 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 1 0
```

As an example, the following is the start of the paper by Peter Gutmann [12], first displayed accurately, and next at an optimal retrieval level.

3.2.1 Correct Display

Secure deletion of data - Peter Gutmann - 1996

Abstract

With the use of increasingly sophisticated encryption systems, an attacker wishing to gain access to sensitive data is forced to look elsewhere for information. One avenue of attack is the recovery of supposedly erased data from magnetic media or random-access memory.

3.2.2 Display from Recovery (Optimal)

%o

'cKræ}d8CEti²n•of0daÊI0Ptr0G\$ŧWÇîî_!4ÁIu960eb8tÈñutW00000Dç•Ã#Ì0
Hf\$00;000%£z0N0ã0000á0áä<it\tpÛ0u³e•Fjª™%leàsingTyøtopÚ”È:i†aze0

¹² This is reported at a 99% confidence level.

®Mcryption0sîÛtems?DKtA""cĐİ0+ϕsinCE0toK-ai2z÷c(ns~0tî0;e
½iti)e""daÆa>s0foôce,ÑtÔİl2o-
iell¶~\$eöe>Ÿr""inf¬rm%äion.0OnRiavem>egoN0-`iRÁ"li
läßh±0"eÛoie=y0Cz-
su•`s/lÛf'era`Jd0dataF`ro>•magne³;&£õãÈáã~or*r*endoª-Qcc«ÇŸ0mà
@ryl000000000000000000

Although on the perfect drive some words could be recovered, there is little of forensic value.

3.2.3 Display from Recovery (Expected)

ĵÄuÛtPdM@""iFnFã:à•ÅÖİ³/4`L`¿ôPÛ!#`-xL^ÛÆ!mC
2`³,„,‡·}NŽýñêZØ^·l©pì®·äÖEvy¿^æ°0Tİ[°HYBš,ð
7zôl»dëÖ/""[ýÁ†,kR¿xt,÷Í2\$Iã""·ÑU%TóÁ`ØoxÈ\$
Wt^TMoES²Æ,Ê°ñ ÒeS» eüB®Èk·YrÍÈ¶=İİSÃ;öp¥D
ôÈŽ"lûÚA6,æ÷U•\$µM¢;Ôæe•İİMÀùæç]#•Q
—————Á¹Û""—OX“h
ÍýİÉûÈ Ã""W\$5Ã=rB+5•ö-GßÛü9iõNë-β`Ya“-i%×Ó¿Ô[Māü
·†Î,f,...[Ä,KDnFJ·×ÅÆ¿êüd¬sPÖi8`v0æ#!)YĐúÆ©
k-«HÄ^ø\$°•Ø°İm/Wic@Û»l"„zbíp000000000000000000

On the drive that had been wiped 3 times (prior) to the data being written and then added, the results are worse. What needs to be noted is that small errors in the calculations lead to wide discrepancies in the data that is recovered. Further, it needs to be noted that any drive recovered is not likely to be in a pristine state. The daily use of a drive reduces the chances of recovery to a level that is truly insignificant.

4 Conclusion

The purpose of this paper was a categorical settlement to the controversy surrounding the misconceptions involving the belief that data can be recovered following a wipe procedure. This study has demonstrated that correctly wiped data cannot reasonably be retrieved even if it is of a small size or found only over small parts of the hard drive. Not even with the use of a MFM or other known methods. The belief that a tool can be developed to retrieve gigabytes or terabytes of information from a wiped drive is in error.

Although there is a good chance of recovery for any individual bit from a drive, the chances of recovery of any amount of data from a drive using an electron microscope are negligible. Even speculating on the possible recovery of an old drive, there is no likelihood that any data would be recoverable from the drive. The forensic recovery of data using electron microscopy is infeasible. This was true both on old drives and has become more difficult over time. Further, there is a need for the data to have been written and then wiped on a raw unused drive for there to be any hope of any level of recovery even at the bit level, which does not reflect real situations. It is unlikely that a recovered drive will have not been used for a period of time and the interaction of defragmentation, file copies and general use that overwrites data areas negates any chance of data recovery. The fallacy that data can be forensically recovered using an electron microscope or related means needs to be put to rest.

References

1. Abramowitz, M., Stegun, I.A.: *Handbook of Mathematical Functions*. Dover, New York (1965)
2. Amit, D.J.: *Field Theory*. In: *The Renormalization Group and Critical Phenomena*. World Scientific, Singapore (1984)
3. Braun, H.B.: Fluctuations and instabilities of ferromagnetic domain-wall pairs in an external magnetic field. *Phys. Rev. B* 50, 16485–16500 (1994)
4. Brown, G., Novotny, M.A., Rikvold, P.A.: Thermal magnetization reversal in arrays of nanoparticles. *J. Appl. Phys.* 89, 7588–7590 (2001)
5. Bulsara, A., Chillemi, S., Kiss, L., McClintock, P.V.E., Mannella, R., Marchesoni, F., Nicolis, G., Wiesenfeld, K. (eds.): *International Workshop on Fluctuations in Physics and Biology: Stochastic Resonance, Signal Processing and Related Phenomena*, p. 653. *Nuovo Cimento* 17D (1995)
6. Carroll, T.L., Pecora, L.M.: *Phys. Rev. Lett.* 70, 576 (1993a)
7. Carroll, T.L., Pecora, L.M.: *Phys. Rev. E* 47, 3941 (1993b)
8. Gomez, R., Adly, A., Mayergoyz, I., Burke, E.: Magnetic Force Scanning Tunnelling Microscope Imaging of Overwritten Data. *IEEE Transactions on Magnetics* 28(5), 3141 (1992)
9. Gammaitoni, L., Hänggi, P., Jung, P., Marchesoni, F.: Stochastic resonance. *Reviews of Modern Physics* 70(1) (January 1998)
10. Gomez, R., Burke, E., Adly, A., Mayergoyz, I., Gorczyca, J.: Microscopic Investigations of Overwritten Data. *Journal of Applied Physics* 73(10), 6001 (1993)
11. Grinstein, G., Koch, R.H.: Switching probabilities for single-domain magnetic particles. *Phys. Rev. B* 71, 184427 (2005)
12. Gutmann, P.: Secure Deletion of Data from Magnetic and Solid-State Memory. In: *Proceedings of the Sixth USENIX Security Symposium*, San Jose, CA, July 22–25, pp. 77–90 (1996), http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html
13. Hänggi, P., Bartschek, R.: In: Parisi, J., Müller, S.C., Zimmermann, W. (eds.) *Nonlinear Physics of Complex Systems: Current Status and Future Trends*. Lecture Note in Physics, vol. 476, p. 294. Springer, Berlin (1991)
14. Liu, D.: *Topics in the Analysis and Computation of Stochastic Differential Equations*, Ph. D. thesis, Princeton University (2003)
15. Mayergoyz, I.D., Tse, C., Krafft, C., Gomez, R.D.: Spin-stand imaging of overwritten data and its comparison with magnetic force microscopy. *Journal Of Applied Physics* 89(11) (2001)
16. Moss, F.: In: Weiss, G.H. (ed.) *Contemporary Problems in Statistical Physics*, pp. 205–253. SIAM, Philadelphia (1994)
17. Ren, W.E., Vanden-Eijnden, E.: Energy landscape and thermally activated switching of submicron-size ferromagnetic elements. *J. Appl. Phys.* 93, 2275–2282 (2003)
18. Reznikoff, M.G.: *Rare Events in Finite and Infinite Dimensions*, Ph. D. thesis, New York University (2004)
19. Rugar, D.H.M., Guenther, P., Lambert, S., Stern, J., McFadyen, I., Yogi, T.: Magnetic Force Microscopy: General Principles and Application to Longitudinal Recording Media. *Journal of Applied Physics* 68(3), 1169 (1990)
20. Tesla, N.: *The Great Radio Controversy*, http://en.wikipedia.org/wiki/Invention_of_radio
21. Jiles, David: *Introduction to magnetism and magnetic materials*, 2nd edn. Chapman & Hall, Boca Raton (1998)

Optimizing the Block Cipher and Modes of Operations Overhead at the Link Layer Security Framework in the Wireless Sensor Networks

Devesh Jinwala¹, Dhiren Patel¹, and Kankar Dasgupta²

¹ Department of Computer Science & Engineering, S.V. National Institute of Technology,
Ichchhanath, Surat – 395007, India
{dcj,dhiren}@svnit.ac.in

² Space Applications Centre, Indian Space Research Organization, Gulbai Tekra,
Jodhpur Village Road, Ambavadi PO, Ahmedabad, India
ksdasgupta@hotmail.com

Abstract. Due to the resource constrained environments under which the Wireless Sensor Networks (WSNs) operate, the security protocols and algorithms should be so designed to be used in WSNs, as to yield the optimum performance. The efficiency of the block cipher and the mode under which it operates are vital factors in leveraging the performance of any security protocol. In this paper, therefore, we focus on the issue of optimizing the security vs. performance tradeoff at the link layer framework in WSNs. We evaluate the resource requirements of the block ciphers characteristically distinct from each other, in conventional modes and the Authenticated Encryption (AE) modes. We use the Skipjack cipher in CBC mode, as the basis for comparison. To the best of our knowledge, ours is the first experimental evaluation of the AES cipher, the XXTEA cipher, the OCB mode and the CCM mode at the link layer security architecture for WSNs.

Keywords: Wireless Sensor Networks, Link Layer Security, Block Ciphers, Authentication, Encryption.

1 Introduction

The security protocols in typical wireless sensor networks have to be so designed, as to yield the optimum performance, while ensuring the essential communications security. In the recent years, there has been considerable improvement in the available resources viz. computational, memory, bandwidth and the energy availability of the wireless sensor nodes [1]. This technological boost is compounded with the availability of the IEEE 802.15.4 compliant radio transceiver chips with the security support built-in e.g. CC2420 [2][3]. But, in spite of the increase in the availability of the resources (a) the overhead due to the use of the security protocols in WSNs, is still substantial and (b) the use of security-enabled transceiver chips does not allow transparent enablement of the security support for the existing WSN applications (i.e. the application programs are to be re-written using the appropriate API). Hence, the security issues in (one-time-deployed) pervasive WSN applications require careful investigation.

The WSNs follow the data-centric multi-hop communication, to support the processing of the data on-the-fly, while being transmitted to the base station. The advantage is the reduced overall overhead in communication [4]. As a result, in the WSN applications based on such dynamic in-network processing of the data, the conventional end-to-end security mechanisms become non-feasible [5]. Hence, the applicability of the standard end-to-end security protocols like SSH, SSL [6] or IPSec [7] is questionable. Therefore, the security support in WSNs should be devised at the link layer, while assuring reasonable resource overhead.

There are a number of research attempts that aim to do so. The notable ones are TinySec [5], SenSec[8] and MiniSec[9]. These link layer security protocols have an open-ended design, so as to enable the use of any block cipher with appropriate mode of operation.

On the other hand, the range of applications for which the WSNs can be used, is very wide. Hence, it is necessary to optimize the *security-levels-desired* vs. the *resource-consumption* trade-off. One way to do so is to ensure that the link layer security protocol employed is configurable, with respect to (a) the actual cipher and the mode of operation employed and, (b) the security attributes desired i.e. encryption, message authentication or replay protection. As for example, with the combined Authenticated-Encryption technique that the Output Codebook Mode (OCB) [10] and the Counter with Cipher Block Chaining MAC (CCM) Mode [11] follow, the applications demanding both the message confidentiality as well as data integrity will be more efficient to implement as compared to the conventional modes. The conventional modes like CBC [12] do not support message integrity check, at all. The message integrity check mechanism like CBC-MAC [13] is required to be employed separately, that use one more block cipher call. At the same time, it is to be emphasized that for the applications demanding only message integrity, the same latter mode can be feasibly employed to an advantage; because the additional CPU cycles in encryption (that are not required for such applications) will not be used in conventional modes.

We believe that the efficiency of the block cipher is one of the important factors in leveraging the performance of a link layer protocol, when aiming the desired degree of security. Therefore, even though the Skipjack [14] (80-bit cipher key with 64-bit block-size) is the default block cipher used by TinySec, SenSec and MiniSec; we have attempted to carefully investigate the feasibility of applying the

- Advanced Encryption Standard (AES) block cipher Rijndael (128-bit cipher key with 128-bit block-size) [15] and
- light-weight cipher Corrected Block Tiny Encryption Algorithm (XXTEA) (128-bit cipher key with 64-bit block-size) [16]
- OCB and CCM modes as against the CBC mode as the desired block cipher mode.

In this paper, therefore, we present our experimental results in the evaluation of the block ciphers, as well as the authenticated encryption modes of operation. We use the Skipjack in CBC mode (the default configuration employed by all the existing link layer security frameworks) as the baseline, for comparing our evaluation.

To the best of our knowledge, ours is the first attempt in implementing and benchmarking the storage requirements of the XXTEA and the AES ciphers in the CBC, OCB and CCM modes, in the link layer security framework in WSNs.

We emphasize that the actual cipher to use and the specific mode of operation to be employed, must be arrived at only after looking at the specific security demands of the application under consideration – rather than by following any ad-hoc demands.

Therefore, for the resource constrained WSN environments, we believe our implementation and evaluation exercise will be useful, in proposing and justifying configurable link layer security architecture. Such architecture implemented in software, will have the advantage of enabling transparent & seamless integration of security support for the existing WSN applications.

The rest of the paper is organized as follows: in section 2, we present the necessary background on link layer protocols and an overview of the related work in the area. In section 3, we briefly describe the characteristics of the ciphers and the modes of operation used by us. In section 4, we describe our methodology and the experimental setup. In section 5, we present the significance of the results obtained, whereas we conclude in section 6 with the future work aimed.

2 Background and the Related Work

In this section, we first discuss the existing link layer security architectures. Then, we elaborate on the related work in the area and justify that our attempt here, is indeed unique and significant.

2.1 Existing Link Layer Security Architectures

TinySec proposed in [5] is designed for the Berkeley Mica Motes. TinySec employs link layer encryption with Skipjack and RC5 [17] as the default ciphers in CBC mode and CBC-MAC as the authentication mechanism.

TinySec realizes the goals of ensuring (a) security at reasonable performance overhead (b) providing minimally configurable security in terms of selection of desired security attributes (c) seamless integration of security support for the existing WSN applications.

Tieyan Li *et al* [6] propose an alternate link layer security architecture viz. SenSec that draws upon its basic design from TinySec, but offers encryption as well as authentication by default.

Luk Mark *et al* [7] propose another alternate architecture viz. MiniSec that is designed for the Telos motes [18]. MiniSec offers all the basic desired link layer security properties viz. data encryption, message integrity and replay protection. MiniSec employs the OCB mode of operation, but the authors do not attempt at any sort of evaluation of this mode against conventional modes. Neither do the authors attempt at evaluating the XXTEA and AES ciphers, as we do here.

The IEEE 802.15.4 specification specifies a new class of wireless radios and protocols targeted at low power devices and sensor nodes [2]. ZigBee protocol conforms to the IEEE 802.15.4 standard [19]. ZigBee is a specification, targeted at RF applications that require a low data rate, long battery life. As per the IEEE 802.15.4 specification, the block cipher employed is the 128-bit key-sized AES Rijndael in the CCM mode. But we argue (and indeed our performance results testify) that XXTEA (a lightweight cipher with minimal key setup operations) can indeed be a better choice.

This is so especially for those environments where the available storage is limited, but still the demand is for high security (with the support for 128-bit key).

2.2 Existing Evaluations of the Block Ciphers and Modes

In this section, we discuss other attempts at evaluating the block ciphers and their modes of operation and emphasize the distinction of our work, here.

In general, the block ciphers used for evaluation in WSN environments are viz. RC5 [16], Skipjack [14], Rijndael [15], Twofish [20], KASUMI [21], Camellia [22], TEA [23].

There have been many benchmarks and evaluation of the block ciphers for the WSNs as surveyed here. But none of them focus specifically on the security at the link layer framework.

Law *et al* in [24], present a detailed evaluation of the block ciphers viz. Skipjack, RC5, RC6, MISTY1, Rijndael, Twofish, KASUMI, and Camellia. The evaluation is based on security properties, storage and energy efficiency of the ciphers. The results prescribe Skipjack (low security at low memory), MISTY1 (higher security at low memory) and Rijndael (highest speed but higher memory) as the most suitable ciphers. However, (a) this work does not consider the OCB & the CCM block cipher modes of operations (b) as against the recommendation of these results, RC5 has been reported to be having higher speed than AES in [25] (c) the evaluation of the ciphers in [24], is not done within any link layer architecture (d) no attempt has been made to optimize the cipher code, employing only the openssl [26] versions of the ciphers.

In [27], Großshädl *et al* attempt at energy evaluation of the software implementations of the block ciphers. The authors have considered the ciphers RC6 [28], Rijndael, Serpent [29], Twofish [20] and XTEA [30]. The evaluation is done by simulation on the StrongARM SA-1100 processor that is used principally in embedded systems like cell phones and PDAs. However, this evaluation does not consider (a) the overhead due to the operating system support or due to the link layer security protocol (b) the actual deployment of the code on the sensor nodes or any typical WSN platform [31].

In [32], Guimarães Germano *et al* discuss another attempt at evaluating the security mechanisms in WSNs. The authors carry out a number of measurements like the impact of cipher on - the packet overhead, the energy consumption, the CPU and memory usage, the network throughput and the network latency. The authors evaluate the ciphers viz. TEA, Skipjack and RC5 but do not consider XXTEA and AES.

In [33], Luo Xiaohua *et al* evaluate the performance of ciphers viz. SEAL [34], RC4 [35], RC5, TEA by implementation on the Mica2 motes. The evaluation makes a surprising claim that RC5 is not suited for the WSNs.

In [36], Ganesan Prasanth *et al* attempt on analyzing and modeling the encryption overhead by estimating the execution time and memory occupancy for the encryption as well as message digest algorithms viz. RC4, IDEA[37], RC5, MD5[38], and SHA1[39] on various hardware platforms. Thus, the algorithms like the AES Rijndael, XXTEA, Skipjack are not considered, which we do here.

Thus, none of these evaluations consider the evaluation of (a) Authenticated Encryption (AE) modes of operations (b) the corrected Block TEA (XXTEA) cipher and (c) the AES Rijndael cipher on link layer architecture, as we attempt to do, here.

3 The Block Ciphers and the Modes Examined

We have selected the Rijndael and XXTEA ciphers for evaluating their performance against the TinySec default cipher Skipjack operating in the CBC mode of execution.

Skipjack cipher uses 80-bit key and 64-bit block-size in 32 rounds of an unbalanced Feistel network. The best cryptanalytic attack on Skipjack is reported on 31 of the 32 rounds of the cipher, employing differential cryptanalysis [40].

We believe that the size of the cipher key is an indicative measure of the strength of the *computational security* of the cipher. At the minimum, the cipher key size must be enough, so as to prevent the brute force attack against the cipher. With the rapid advancement in technology, the conventional key size of 80-bits is no longer sufficient. As per the claims of RSA Security Labs, 80-bit keys would become *crackable* by 2010 [41]. Hence, it is essential to move towards ciphers with 128-bit cipher key sizes.

Our selection of the XXTEA cipher is based on it being a 128-bit key size cipher. It is a simple lightweight cipher, proposed by David Wheeler and Roger Needham in 1998 [16]. The cipher was proposed to improve upon its predecessor XTEA [30]. XXTEA is an unbalanced Feistel network cipher with at least 64-bit block-size, employing 32 cycles. Because of its simplicity in design, we believe XXTEA is appropriate cipher for the resource constrained WSN environments.

Rijndael is a block cipher with variable 128/192/256-bit key-size, the variable 128/192/256-bit block-size and variable 10/12/14 rounds. We have selected Rijndael in the configuration of 128/128/10.

Finally, we have selected the OCB and the CCM modes because these are the AE modes of operation. The OCB mode was first proposed by P. Rogaway et al in [10] whereas the CCM mode was first proposed by R. Housley et al in [11].

The principal advantage of AE schemes is the overall lower computational costs, because of the integration of the computation of the MAC into the block cipher call. Thus, the AE schemes avoid the need to use two different block cipher calls viz. a computation of MAC for authentication and a block cipher encryption for confidentiality. The CCM mode requires only two block cipher calls per each block of encrypted and authenticated message and one call per each block of associated authenticated data. Similarly OCB mode requires only four block cipher calls.

Albeit for this gain in performance, it is emphasized that the AE modes are useful only when an application demands confidentiality as well as message integrity both. The applications requiring only integrity of the message, the AE modes are not useful.

4 Experimental Setup and Methodology of Evaluation

In this section we present the tools and the platform used by us for the experiments conducted as well as the methodology adopted and the test application used to do so.

4.1 Platforms and Tools Used

We have used the TinySec in the TinyOS 1.1x operating environment [42] with the nesC [43] as the language of implementation. We have also employed TOSSIM simulator [44] for simulation and then deploy the code on the Mica2 sensor nodes.

We have used Avrora simulator [45] to carry out the evaluation of the CPU cycles, throughput and energy consumed by the cipher. Avrora is an emulator implemented in Java that runs the runs actual Mica code, while emulating each WSN node as its own thread. Avrora runs code in an instruction-by-instruction fashion.

Thus our evaluation is based on a three-step approach: (a) first, we simulated the performance of the ciphers and modes implemented in nesC. The nesC compiler itself gives as output, the RAM and ROM requirements of the application under consideration (b) next, we determine the throughput in bits/sec and the energy consumed using the Avrora simulator (c) third, we deploy the application under consideration on the Mica2 motes Atmega128L processor at 7.3728 MHz with 128 KB Flash and 4 KB of Data memory and Chipcon CC1000 radio.

One question which arises here is: why did, we employ Mica2 motes, when resource enriched *next generation* motes like Intel iMote [46] and Crossbow Iris motes [47] are available, today. We believe that (a) our evaluation that is carried out on more stringent environment of Mica2 motes, can always be true in more resource-rich environments (b) the deployment of the type of motes to be used for various applications, varies with the level at which it is deployed. Hence, for mass deployment in unattended environments, the cheaper Mica2 motes can indeed prove to be more economical as compared to the expensive next generation motes.

4.2 The Test Application

We employ a simple application which comes bundled with the TinyOS for the evaluation. The pseudocode of the application is as shown below whereas the call-graph of the application generated with the compiler is shown in Fig. 1.

```
Algorithm: TestTinySec
1. initialize counter to 0
2. while (Timer fires){
    increment the counter;
    if (Send(Data Packet)) then LED = green;
    else
        if(Receive(Data Packet)) then LED = red;
}
```

As can be observed, the application under consideration, implements a counter (which is part of the data packet sent/received over the radio) that is incremented on firing of a timer.

The counter value is periodically modified by the component Counter. It is further passed by TestTinySecM (the main module of the application) through the SendMsg interface of TinyOS, for onward transmission over the radio, to the component SecureGenericComm of TinyOS. Also, when the message is sent, the Leds interface is used to toggle the LED on the mote. When the message is transmitted by a mote, the LED is turned green whereas, when the message is received by a mote, the LED is turned red. The entire communication takes place with the security attributes enabled. In Fig. 2, we show the partial call-graph depicting the security components of the TinySec that come into play, during the execution.

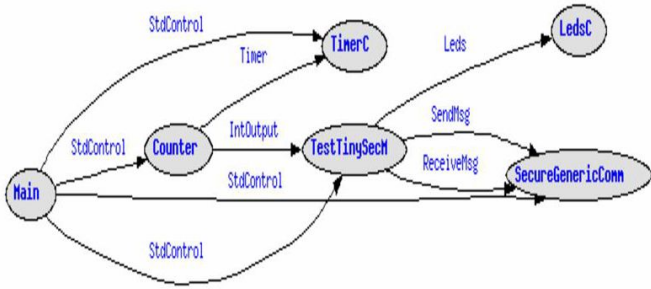


Fig. 1. The Call-graph of the Test Application

TinySec has been designed to be modular with respect to the selection of the block cipher and the modes of operation. But, as shown in Fig. 2, the component SkipJackM that implements the Skipjack cipher and the component CBCModeM that wires Skipjack cipher in CBC mode; are the default cipher and mode of operation.

The authentication support is implemented in the component CBCMAC. Thus, SkipJackM, CBCMAC and CBCModeM components are not implemented by us.

We implement our nesC components of the ciphers RC6, XXTEA (Fig. 3) and AES. We implement AES in two versions viz. speed-optimized (AESSPO) and the size-optimized (AESSIO) ones. We also implement the block ciphers modes nesC components viz. CCMM and OCBM.

For implementation, we have used the size-optimized C-versions of AES in [48] and the XXTEA version in [16] and converted these versions into the nesC. We have also used the speed-optimized version of AES from [41], to implement two different versions to suit different environmental constraints.

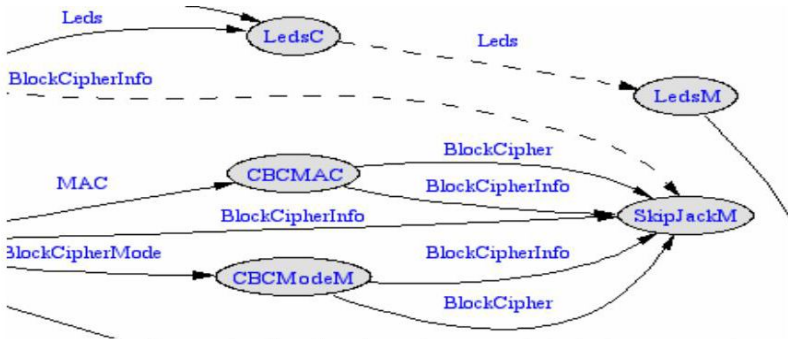


Fig. 2. Partial Call-graph of the application with Skipjack cipher in CBC Mode

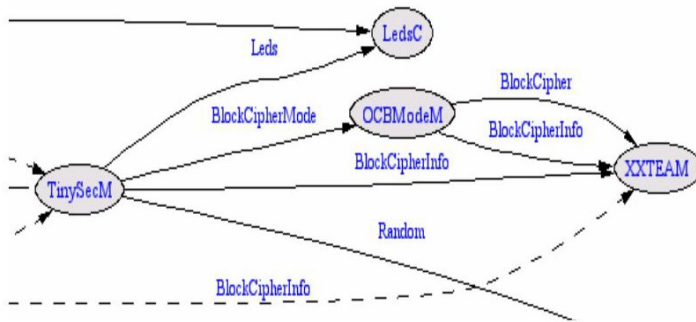


Fig. 3. Partial Callgraph of the application with XXTEA in OCB Mode

We then modified the configuration files of TinySec to use the ciphers Rijndael, RC6 and XXTEA and the OCB and the CCM as the modes of operation in various combinations viz. Skipjack-CBC, Skipjack-OCB, XXTEA-CBC, XXTEA-OCB and AES-CBC. In Fig. 3, we show the sample partial snapshots of the TestTinySec callgraph with XXTEA cipher in OCB mode.

We compared the openssl version of AES with the version described in [48]. The openssl version uses one 8-bit 256-entries S-box and four 32-bit 256-entries forward and reverse tables each, thus consuming a total static storage of 8.448 KB. Due to the lack of the need for generating the F-table and the S-table dynamically during the execution, this version is expected to offer higher speed at the cost of requirement of higher storage. We call this version of AES, as the AESSpeedOptimized (AESSPO).

As compared to openssl version, the size-optimized nesC version of AES that is based on the one in [48], uses dynamic computation of tables using only one 8-bit 256 entries S-box and one 32-bit 256 entries forward and reverse tables each. Thus it consumes a total static storage of 2.304 KB. The reduction in storage is 72%, over the openssl version. We call this version of AES as the AESSize/StorageOptimized (AESSIO). Also, since the AES and RC6 are 128-bit ciphers as compared to the 64-bit Skipjack and XXTEA, we made appropriate logical changes in the TinySec files, for obtaining this support. For XXTEA, RC6 and AES, we also changed the default tinys-keyfile to enable the support for 128-bit cipher keys.

5 Performance Results and Analysis

We now present the results of evaluations of the storage requirements, throughput (bits/sec) and the energy requirements for various modes. In Fig. 4 we show the percentage increase in memory when using the AES cipher.

From Fig 4, we can see that for the MICA2 motes with only 4KB of RAM, an overhead of only 13.46% and 12.98% results, when using the OCB or CCM mode with the AESSPO and AESSIO implementation. The significant advantage is the

increased security strength due to the standard 128-bit cipher, wired in OCB/CCM modes of operations.

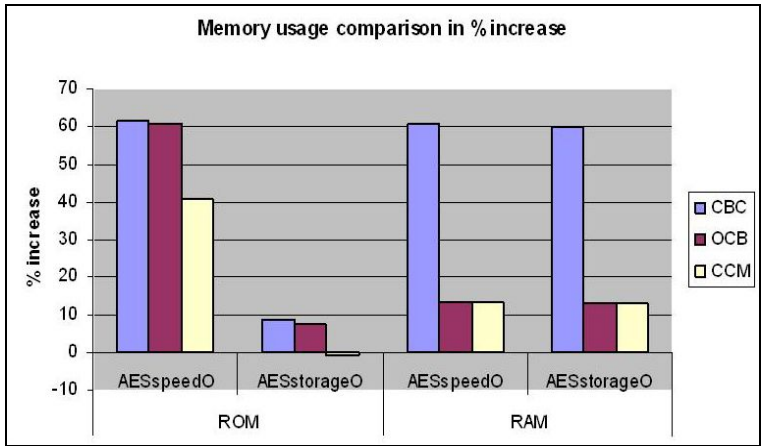


Fig. 4. Increase (%) in RAM/ROM over the Skipjack in the CBC mode

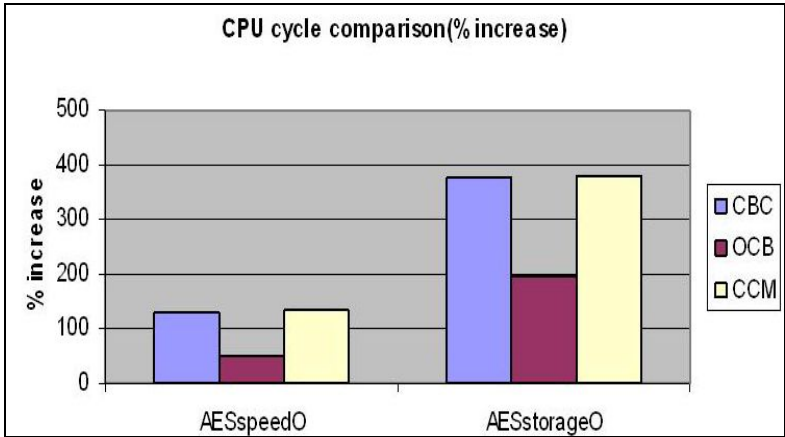


Fig. 5. Increase (%) in CPU cycles over the Skipjack in CBC mode

As per our results shown in Fig 5, we can see that the CPU cycles in the CBC, OCB and CCM modes (with AESSPO) are 129.08%, 48.09% and 131.31% higher respectively, over the same with Skipjack cipher in CBC mode.

The corresponding figures for the AESIO version are similar. Thus, when employing OCB mode, the penalty in terms of increased CPU resources, is much lesser than the same when employing CBC/CCM modes.

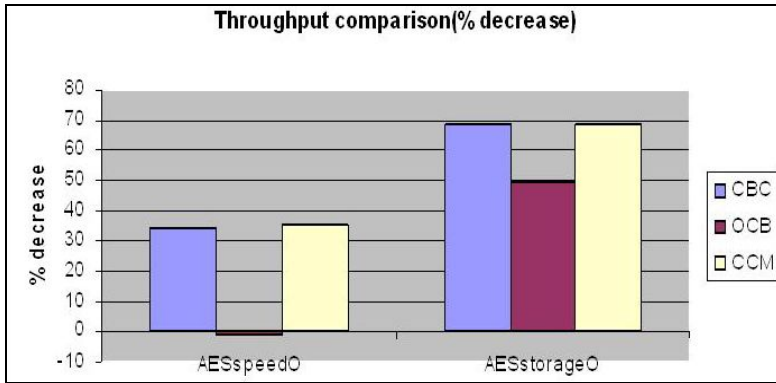


Fig. 6. (%) Penalty in throughput with OCB/CCM (AES) over CBC (Skipjack)

In Fig. 6 we show the penalty in the form of lesser throughput when employing the AES cipher. As can be observed, when employing our version of the AESSpeedOptimized in OCB, the percentage reduction in throughput is minimal, as compared to the CBC mode.

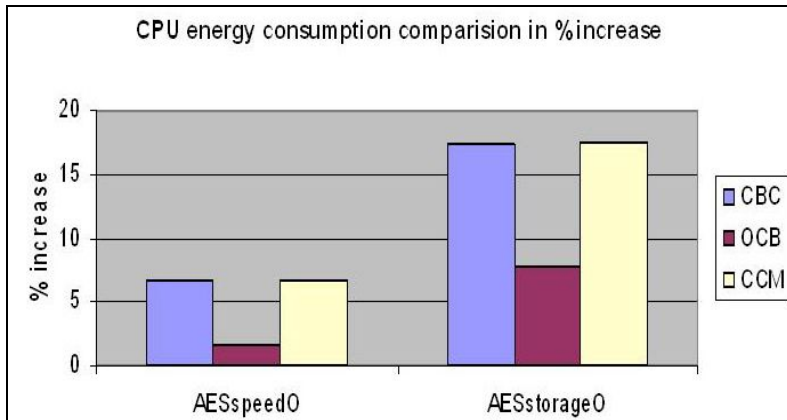


Fig. 7. (%) Penalty in Energy consumption with OCB/CCM (AES) over CBC (Skipjack)

This is a significant observation pointing to the fact that security strength due to 128-bit key-size with a stronger mode like OCB, is obtained at much lesser overhead. Hence, the OCB mode must be the preferred mode of operation.

As can be seen from Fig. 7, the penalty in energy consumption in Mica2 motes for the OCB/CCM modes employed in 128-bit key-size cipher is much less as compared to the Skipjack cipher in CBC mode. Even in the next generation motes, the energy availability has almost remained the same. So our results indicating lesser penalty in energy consumption for OCB/CCM modes are even more significant and vital.

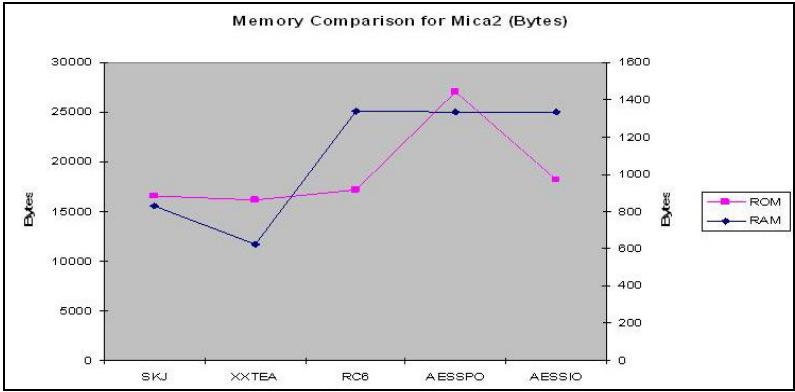


Fig. 8. RAM & ROM requirements for ciphers

In Fig. 8, we show the storage requirements for the ciphers under consideration. We observe the RAM requirement of the cipher XXTEA to be minimal - again which is a significant advantage because the size of RAM has not grown as significantly even in the next generation motes as ROM. Next, in Fig. 9, we show the throughput for encryption and decryption operations for the ciphers, under consideration.

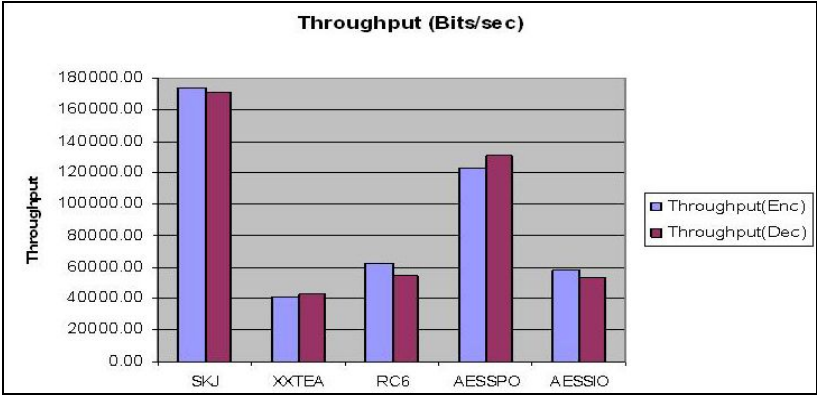


Fig. 9. Throughput in Encryption/Decryption for ciphers

The throughput of Skipjack as such is the highest. But when considering the key-setup operation also, the throughput of Skipjack is expected to be lower than the XXTEA cipher – since XXTEA cipher almost has minimal key-setup.

From the observed energy requirements of the ciphers, in Fig. 10, it is clear that (a) as compared to Skipjack (80-bit keysize), the energy requirement of the 128-bit key-sized lightweight cipher XXTEA is only slightly higher (b) the energy demands of the speed-optimized version of AES is quite low, but it requires higher memory (c) since energy constraints in motes are always severer, XXTEA cipher gives the best of

both the worlds – it requires lower memory, provides the security strength due to 128-bit keysize and demands lower energy resources as compared to other 128-bit key-sized ciphers like RC6 and AES.

But as mentioned before, the throughput of XXTEA is the least. This is due to the large number of rounds (32) involving combinations of rotations and exclusive OR operation. Note that XXTEA cipher does not have any substitution operation – it achieves the required non-linearity through only the combinations of additions modulo 2^{32} , rotations and Ex-OR operations. While the designers of XXTEA proclaim having considered substitution also as one of the options, it is worthwhile (with increasing storage availability in motes) to investigate a combination of substitution and rotation based operations in XXTEA with lesser number of rounds, than present. The energy consumption of the XXTEA cipher is higher than the speed-optimized version of AES, but obviously this gain is at the cost of the increased storage consumption. Hence, depending upon the availability of the resources either of the XXTEA or the AES versions can be employed to attain the higher security strength.

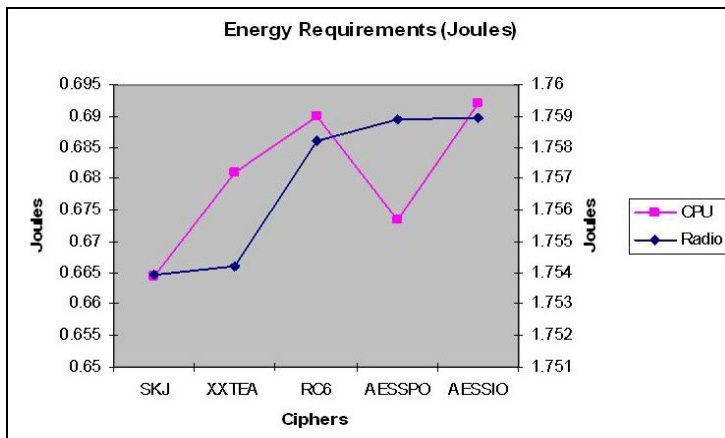


Fig. 10. Energy in Encryption/Decryption for ciphers

6 Conclusion and Future Work

Our experimental observations lead to vital significance and conclusions. (a) First, the experimentation clearly proclaims the possibility of employing 128-bit key-sized ciphers even in the software implementations of link layer security framework (b) the simplicity of XXTEA is worth exploited in the resource constrained WSN environments (c) other 128-bit ciphers like RC6 are not suitable for the resource starved sensor nodes (d) since the OCB mode proves to be highly efficient in reducing the overall overhead, it must be the natural block cipher mode of operation for WSNs. However, being an AE mode, OCB is suitable only for those applications demanding both the confidentiality of data as well as message integrity.

Therefore, we believe that the link layer security should be implemented in software with a configurable design: so that the application programmers can tune the

underlying security architecture to suit the demands of the applications with minimum overhead. That is, when the applications under consideration demand only the message integrity and not confidentiality, the security architecture should be configured to employ the CBC block cipher mode, while OCB otherwise. Such configurable architecture can then be suitably used employed in further research involving investigations into the link layer security in the WSNs.

Our current work is focused in the implementation of such configurable link layer security framework.

Acknowledgment

The authors would like to thank the anonymous reviewers for giving suggestions to refine the contents of this paper.

References

1. Wireless Sensor Networks: Getting Started Guide, <http://www.crossbow.com>
2. The IEEE 802.15.4 Standard, <http://www.ieee802.org/15/pub/TG4.html>
3. The Chipcon Radio Transceiver, CC2420/CC2430, <http://www.chipcon.com/>
4. Undercoffer, J., Avancha, S., Joshi, A., Pinkston, J.: Security in Wireless Sensor Networks. In: Proc. of CADIP Research Symposium (2002)
5. Karlof, C., Sastry, N., Wagner, D.: TinySec: Link Layer Encryption for Tiny Devices. In: Proceedings of the 2nd international conference on Embedded networked sensor systems, pp. 162–175. ACM, New York (2004)
6. Viega, J., Chandra, P., Messier, M.: Network Security with Openssl. O'Reilly & Associates, Inc., Sebastopol (2002)
7. IPSec: Requests For Comments. RFC 2401, RFC 2402, RFC 2406, RFC 2408, <http://www.ietf.org/rfc/rfc240n.txt>
8. Tieyan, L., Hongjun, W., Xinkai, W., Feng, B.: SenSec: Sensor Security Framework for TinyOS. In: Second International Workshop on Networked Sensing Systems, San Diego, California, USA, pp. 145–150 (2005)
9. Luk, M., Mezzour, G., Perrig, A., Gligor, V.: MiniSec: A Secure Sensor Network Communication Architecture. In: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, pp. 479–488. ACM, New York (2007)
10. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: ACM Transactions on Information and System Security, pp. 365–403. ACM, New York (2003)
11. NIST Special Publication 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf
12. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. In: Proceedings of 38th Annual Symposium on Foundations of Computer Science, pp. 394–403 (1997)
13. Bellare, M., Kilian, J., Rogaway, P.: The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences* 61(3), 362–399 (2000)

14. NIST-CSRC, SKIPJACK and KEA Algorithm Specifications, version 2 (1998), <http://csrc.nist.gov/CryptoToolkit/>
15. Daemen, J., Rijmen, V.: The Design of Rijndael AES - The Advanced Encryption Standard. Springer Series: Information Security and Cryptography (2002)
16. Wheeler, D., Needham, R.: XXTEA: Correction to XTEA. Technical report, Computer Laboratory, University of Cambridge (1998)
17. Rivest, R.: The RC5 Encryption Algorithm. In: Proceedings of the 1994 Leuven Workshop on Fast Software Encryption, pp. 86–96. Springer, Heidelberg (1995)
18. Telos Motes, <http://www.moteiv.com>
19. IEEE802.15.4/ZigBee – The Next Generation Low Rate Wireless Network IC Development – Press release (May 2004), <http://www.oki.com/en/press/2004/z04017e.html>
20. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: Twofish: A 128-Bit Block Cipher (1998), <http://www.schneier.com/paper-twofish-paper.pdf>
21. Specification of the 3GPP Confidentiality and Integrity Algorithms Document 2: KASUMI Specification (1999), <http://downloads.securityfocus.com/library/3GTS35.202.pdf>
22. Matsui, M., Tokita, T.: MISTY, KASUMI and Camellia Cipher Algorithm (2000), http://global.mitsubishielectric.com/pdf/advance/vol1100/vol1100_complete.pdf
23. Wheeler, D., Needham, R.: TEA, A Tiny Encryption Algorithm. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 363–366. Springer, Heidelberg (1995)
24. Law, Y., Doumen, J., Hartel, P.: Survey and Benchmark of Block Ciphers for Wireless Sensor Networks. ACM Transactions on Sensor Networks 2(1), 65–93 (2006)
25. Deng, J., Han, R., Mishra, S.: A Performance Evaluation of Intrusion Tolerant Routing in Wireless Sensor Networks. In: Zhao, F., Guibas, L.J. (eds.) IPSN 2003. LNCS, vol. 2634, p. 552. Springer, Heidelberg (2003)
26. The openSSL, <http://www.openSSL.org>
27. Großshädl, J., Tillich, S., Rechberger, C., Hofman, M., Medwed, M.: Energy Evaluation of Software Implementations of Block Ciphers under Memory Constraints. In: Proceedings of the 10th Conference to Design, Automation and Test in Europe, EDA Consortium, San Jose, USA, pp. 1110–1115 (2007)
28. Rivest, R., Robshaw, M., Sidney, R., Yin, Y.: The RC6 Block Cipher. Specification version 1.1 (August 1998), <http://www.rsasecurity.com/rsalabs/rc6/>
29. Anderson, R., Biham, E., Knudsen, L.: Serpent: A Flexible Block Cipher with Maximum Assurance. In: Proceedings of the The First Advanced Encryption Standard Candidate Conference (1998)
30. Needham, R., Wheeler, D.: Tea extensions; Technical Report, Computer Laboratory, University of Cambridge (1997)
31. Hill, J., Horton, M., Kling, R., Krishnamurthy, L.: The Platforms Enabling Wireless Sensor Networks. Communications of ACM 47(6), 41–46 (2004)
32. Guimarães, G., Souto, E., Sadok, D., Kelner, J.: Evaluation of Security Mechanisms in Wireless Sensor Networks. In: Proceedings of the 2005 Systems Communications, pp. 428–433. IEEE Computer Society, Washington (2005)
33. Luo, X., Zheng, K., Pan, Y., Wu, Z.: Encryption Algorithms Comparison for Wireless Networked Sensors. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp. 1142–1146. IEEE, Los Alamitos (2004)

34. Rogaway, P., Coppersmith, D.: SEAL Software-Optimized Encryption Algorithm. *Journal of Cryptology* 11(4), 273–287 (1998)
35. The RC4 Cipher Page,
<http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html>
36. Ganesan, P., Venugopalan, R., Peddabachagari, P., Dean, A., Mueller, F., Sichitiu, M.: Analyzing and Modeling Encryption Overhead for Sensor Network Nodes. In: *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, pp. 151–159. ACM, San Diego (2003)
37. Xuejia, L., Massey, J.: A Proposal for a New Block Encryption Standard. In: *Proceedings of the Advances in Cryptology*, pp. 389–404. Springer, Heidelberg (1991)
38. Rivest, R.: The MD5 Message-Digest Algorithm. RFC 1321,
<http://www.ietf.org/rfc/rfc1321.txt>
39. SHA-1, NIST. Secure hash standard. In: *Federal Information Processing Standard*. National Institute of Standards and Technology (1995)
40. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. *Journal of Cryptology* 18, 291–311 (2005)
41. RSA Laboratories, <http://www.rsa.com/rsalabs>
42. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System Architecture Directions for Networked Sensors. *ACM SIGPLAN Notices* 35, 93–104 (2000)
43. Gay, D., Levis, P., Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesC Language: A Holistic Approach to Networked Embedded Systems. In: *Proceedings of the ACM SIGPLAN: The Conference on Programming Language Design & Implementation*, pp. 1–11. ACM, New York (2003)
44. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In: *Proceedings of the 1st international conference on Embedded Networked Sensor Systems; SenSys 2003*, pp. 126–131. ACM, New York (2003)
45. Titzer, B., Lee, D., Palsberg, J.: Avrora: Scalable Sensor Network Simulation with Precise Timing. In: *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks*, pp. 477–482 (2005)
46. Intel’s Imote, <http://www.xbow.com/Products>
47. The Crossbow IRIS motes,
<http://www.xbow.com/Products/wproductsoverview.aspx>
48. The AES smaller version,
http://hostap.epitest.fi/wpa_suppliment/devel/aes_8c.html

Privacy Management for Facebook

Enkh-Amgalan Baatarjav, Ram Dantu, and Santi Phithakkitnukoon

Department of Computer Science and Engineering
University of North Texas, Denton, Texas, 76203, USA
{eb0050, rdantu, santi}@unt.edu

Abstract. As more people adopt the Internet as a medium of communication, the Internet has developed into a virtual world and this has resulted in many online social networks (SN). MySpace and Facebook, two leading online SN sites, have a combined user base of 170 million as of 2008. SN sites started to offer developers open platforms that provide users' profile information to the developers. Unfortunately, the applications can also be used to invade privacy and to harvest the users' profile information without their acknowledgement. To address this vulnerability, we propose a privacy-management system that protects the accessibility of users' profile. The system uses probabilistic approach based on information revelation of users. Our experimental result shows that the system can achieve high accuracy rate of 75%.

Keywords: Privacy, Privacy management, Social network, Facebook.

1 Introduction

Communication among friends, colleagues, and family members has been changing from strictly face-to-face interaction to increasingly include cyberspace or online social networking (SN) where individuals can receive/send messages, share photos, join groups, read gossips, and meet with strangers. The number of people with Internet access has fostered an environment where real-life social activities transform into online social activities [1]. As with real-life social networks, we can cluster this online social networking into sub-networks with common values. Among such values may be family bonds, common interests, regions, political preferences, activities, ideas, and religions. Anyone can join a particular sub-network, which makes online SNs quite diverse.

Since they do not require face-to-face interaction, online SN sites make it easy to find and meet people. To ensure online interaction takes place, users tend to post personal information such as their actual names, birthdays, political preferences, religions, relationship status, interests, activities, favorite music, listings of movies, and other information they believe will attract others. This posting provides credibility (through identifying credentials) and suggests areas of compatibility between parties. In addition, the online SN allows users distribute their information through the network efficiently and quickly. For example, users having parties can send event invitations to their networks. Users may learn more

about friends on the social network site than they would in face-to-face meetings. Thus, sharing common interests and ideas with friends makes online SNS attractive cyber-places to hang out [2].

In this study, we will study architecture of Facebook platform and its privacy hole. Using the privacy hole of Facebook, we will show a way to harvest users' profile information. Finally, we will propose a possible solution to protect privacy of Facebook users.

2 Background

Privacy is an inevitably issue for online SN. It has become much easier these days to find almost any person's personal information through highly sophisticated online technologies with search engines such as Google, Yahoo, and Ask. Social Networking Sites (SNS) provide the next big step toward invading users' privacy because users willingly post personal information either without considering the consequences or believing that their information is somehow protected. However, practically this is not always the case and privacy on social networking sites has received more attention from individuals in many fields of study, particularly, computer science, information science, and sociology. In 2007, Acquisti and Gross of Carnegie Mellon University [3] reported some privacy issue on Facebook. Their survey of 40 questions relating to Facebook privacy was taken by 506 respondents. Acquisti and Gross analyzed survey-takers behavior on Facebook based on before and after learning information revealed on Facebook. They also pointed out misconceptions of members' profile visibility in their network based on the survey.

Social networking has become increasingly popular among teenagers who can easily become victims of privacy invasion because of lack of awareness of privacy issues[4] [1] [5]. Consequently, members of this group reveal more information on their profile sites than older users. This lack of awareness can lead to unexpected consequences. To address this issue, Susan B. Barnes [6] proposes three approaches to solve this problem: social, technical, and legal. Social networking is one of the technologies that changes our everyday life-style. Danah Boyd's study [4] showed four properties: persistence, search-ability, exact copy-ability, and invisible audiences that SN sites had, but conventional face-to-face interaction did not. These properties had been changing the way people interact, especially for young people.

3 Facebook

Facebook is the Internet's fastest growing SNS. Facebook, founded by Mark Zuckerberg and launched on February 4, 2004, was initially restricted to Harvard University. However, because of its rapid success, Zuckerberg expanded it to users at Ivy League schools. Facebook was quickly spreading throughout institutions around the world. Any user with a valid university email address could join in. From September 2006 to September 2007, Facebook's Internet traffic ranking

jumped from 60th to 7th [7]. From September 2006, Facebook became accessible to any user 13 or older [8]. As of November 2007, Facebook had more than 55 million active users and 60 million users by the end of 2007. Facebook's average new user registration per day since January 2007 was 250,000 [8]. Open platform, one of Facebook's main features, attracts a large audience to Facebook from a variety of fields including multi-million dollar corporations, entrepreneurs, and student developers [8]. Facebook made its platform available to application developers in April 2007. As of May 2008, the number of applications available to users had grown to 24,800 [7].

3.1 Facebook Platform

Introducing open platform on a social networking site makes a breakthrough that gives both developers and companies creative freedom. As of June 2007. On the Facebook social network, there were 40,000 Facebook application developers, and it attracted 1,000 developers daily [9]. Three reasons may account for this. Firstly, using the open platform, advertising firms can exploit social graph of users to have a clearer understanding demographic of the users, so it is shown to be effectively way to distribute information to potencial customers using the social graphs [8]. Secondly, developers can develop applications quickly on the Facebook platform, making it attractive from a profit measure. Thirdly, the platform is available in many programming environments, such as PHP, ASP, ColdFusion, Java, C, C++, and Python, so the developers can select their comfortable environments [8].

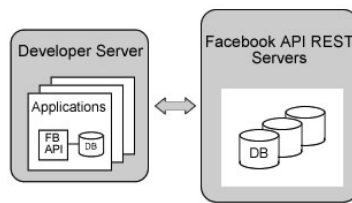


Fig. 1. Data transaction mechanism between Facebook platform and Facebook API REST servers

The Facebook platform is based on a representational state transfer (REST)-like interface. Figure 1 depicts a high-level architecture of the Facebook platform. Information resource is located at Facebook API REST server, and this information is retrieved by method calls in the Facebook API located on the developer's server. Data transaction between the REST server and the application uses either HTTP GET or POST request methods [8]. Using the Facebook API, applications can access a database of information as given in Table 1, can be considered as sensitive information for some users. A complete list can be found at [8] ¹.

¹ <http://developers.facebook.com>

Table 1. Using Facebook platform, application developers can access users’ personal and social information

Tables	Description
user	User profile information first name, last name, birthday, sex, hometown location, current location, political preference, religion, work history, education history, interests, activities, etc.
friend	All friends of a user. Facebook API method returns list of user IDs (uid).
group	Groups a user belongs to along with group IDs (gid), names, group types, and descriptions.
group_member	Member list of a specific group
event	Upcoming event organized by group or friend along with that event’s unique ID (eid).
event_member	Invited members’ status of an event.

3.2 Facebook Privacy Policy

Facebook believes in openness of information on the its network. Its privacy setting is based on an opt-in policy, in which Facebook users’ information is accessible to all of Facebook’s SNS users and to platform applications by default. This means that Facebook considers its members to be legitimate users will not violate Facebook rules and policy. In reality, not everyone is legitimate user and obeys its policy. Therefore, Facebook does not enforce its privacy policy rigorously enough to protect its legitimate users from invasions of privacy and even criminal activity. As shown in Table 1, the amount of information available to strangers is high on Facebook. In default setting, miscreants can easily identify a user’s first name, last name, birthday, email address, physical address, phone number, relationship status, political reference, religion, hometown, favorite music, TV shows, books, groups that user belongs to, and a variety of other personal information.

Facebook clearly publishes the following statements on its new users’ term. First, Facebook does not review and approve any application before it is published on the network:

“...Platform Applications have not been approved, endorsed, or reviewed in any manner by Facebook, and we are not responsible for your use of or inability to use any Platform Applications, including the content, accuracy, or reliability of such Application and the privacy practices or other policies of Developers. YOU USE SUCH PLATFORM APPLICATIONS AT YOUR OWN RISK ...” [8].

Second, any application that is installed on user's friend's site can access all user's information that is allowed by the user: "...If you, your friends or members of your network use any Platform Applications, such Platform Applications may access and share certain information about you with others in accordance with your privacy ..." [8].

Facebook's privacy statement suggests that users who are agree to its terms, know that they are agreeing to make any of their information placed on the site accessible by any users on a same network and platform application. In our study, we find that most Facebook users are unaware of the default privacy setting they are agreeing to. In section 4, we discuss how this information can be exploited and how this exploitation can be implemented by the Facebook API.

4 Facebook Privacy Issue

Demographic factors, such as age, education level, and wealth, influence level of privacy concerns [5]. In this section we explore about how much information is revealed by different demographics. Having an open platform that makes it easy to access social and personal information means maintaining users' privacy must be in a careful consideration. In the case of Facebook, open platform creates a privacy hole. Many users post personal information without knowing that malicious users (hackers) can harvest and exploit their information. In addition, the application privacy setting by default is configured to allow all platform applications to access users' profile information. This, again, makes that information available to potentially dishonest hackers and other criminals. To find out how much information Facebook users reveal on their profile, we analyze sample of 4,919 Facebook users on the University of North Texas network. (The online SN has 34,790 registered members.) Gender ratio is 35% female and 65% male. Our research shows that 75% of the users reveal their education history after high school (Fig. 2); 70% disclosed their high school's name; more than 60% posted their favorite movies, music preferences, interests, relationship status,

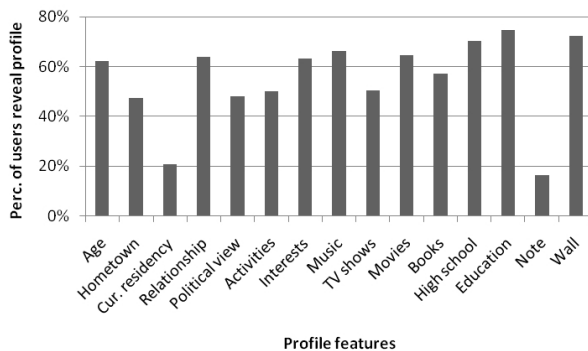


Fig. 2. Example of how much personal information is revealed on UNT social network site, and result is based on 4,919 users

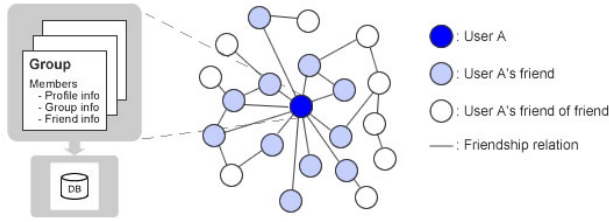


Fig. 3. Once a user installs an application, his social network is accessible by the application, which creates privacy hole in the social graph

and age; 57% listed books they like; between 45-51% revealed their hometown, their favorite TV shows, activities, and political preference which can be one of libertarian, apathetic, very conservative, conservative, moderate, liberal, or very liberal; and, 21% disclosed the city where they currently resided.

Facebook's policy of neither approving nor reviewing platform applications developed by third parties and high number of users uninformed about their privacy settings make Facebook users' personal and social information easily harvestable using Facebook API. An unscrupulous developer can exploit this vulnerability. Figure 3 shows the harvesting process. Each user's site carries information of profile, group, and friend. Information can be harvested based on this information.

Figure 3 shows a user's social graph. Exploiting Facebook's privacy hole, a malicious application harvests the user's personal and social information. In addition, the user's social information can be used for further information harvesting. The following is an example of a pseudocode to harvest a user's social graph: after *UserA* installs an application, the application harvests profile information of *UserA*'s social network. *G* and *F* denotes groups and friends, respectively. A function *Get()* returns profile information, group information, friend information, or group member information depending on one of the arguments: *profile*, *group*, *friend*, or *member*, respectively.

A malicious program installed by User A

```

1: A = Get (profile (User A)) // Input: User A's profile information
2: G = Get (group (A)) // Extract groups where A belongs
3: F = Get (friend (A)) // Extract A's friends
4: StoreData // Variable or database to store profile information
5: for  $f \in F$  do
6:   G += Get (group (f))
7: end for
8: Delete duplicating groups in G
9: for  $g \in G$  do
10:  V += Get (member (g))
11: end for
12: Delete duplicating data in V

```



```
13: for  $v \in V$  do  
14:   StoreData += Get (profile (v))  
15: end for
```

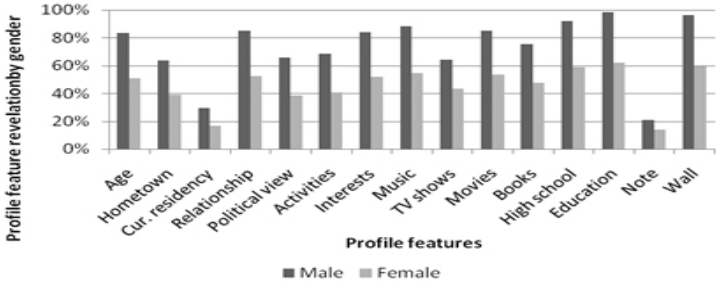
Once the user installs an application, the open platform’s API gives the application access to the user’s profile, friend, and group information. On Facebook, each user and group has unique ID (uid and gid). Once their harvesting program recovers these IDs, its developers can easily collect and use these information. As this example demonstrates, by writing a simple PHP code on top of the Facebook platform, developers can find harvesting information that users believe is private and personal to be a trivial task. In this section we demonstrated how much information can potentially be revealed by Facebook users using an SNS.

5 Analysis on Information Revelation

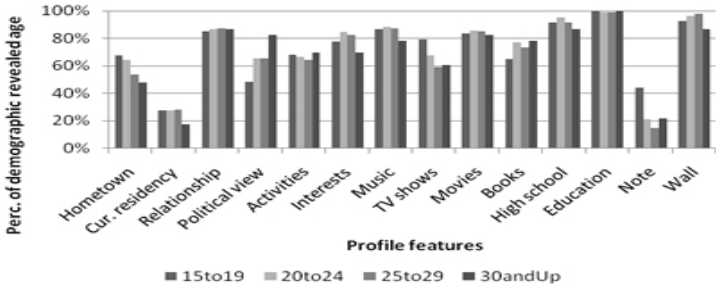
In Sect. 4 we demonstrated that Facebook users’ information can be harvested because of the default privacy setting by platform applications. In this section, we report our analysis of data from four categories: gender, age group, relationship status, and political preference. In fact, we further investigated to build privacy protection system based on these results in Sect. 6. Each category exposes interesting results. The analysis is based on 4,919 Facebook users in the UNT social network. Although Facebook did not require that users reveal the information,

Table 2. Demographic of data revelation by gender, age, relationship, and politic preference

Category	Group	Revelation (%)
Sex	Male	35
	Female	65
Age	15-19	4
	20-24	82
	25-29	14
	30-up	0.5
Relationship	Singe	47
Status	In Relationship	35
	Engaged	6
	Married	12
	Complicated	0
	Open Relationship	0.4
Political Preference	Very Liberal	6
	Liberal	28
	Moderate	34
	Conservative	24
	Very Conservative	2
	Apathetic	4
	Libertarian	3



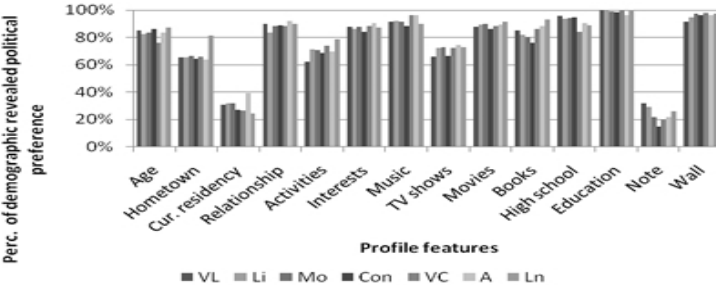
(a) Information revealed by different genders



(b) Information revealed by different age groups



(c) Information revealed by users in different relationship status



(d) Information revealed by users in different political references

Fig. 4. Informtion revelation based on gender, age, relationship status, political preference

62% (3,045) revealed their age; 64% (3,143) exposed their relationship status; and 48% (2,358) showed their political preference.

In Table 2, we further analyze the data by gender, age, relationship status, and political preference. From 4,919 users profiles examined, 65% revealed the user as female and 35% male. We find that 82% of 3,045 are between 20 and 24 years old and 18.5% of 3,045 are in age group of 15-19, 25-29, or 30-Up. Table 2 (Relationship Status) shows that majority of the users who reveal their relationship are singles. However, no one was in complicated relationship. Table 2 (Political Preference) shows that 84% of 2,358 reveal their political preference are conservative, moderate, or liberal.

From Fig. 4(a), we find that female users are less likely to reveal their personal and social information than male users. Figure 4(b) shows that on average age group between 20 and 29 reveals more personal and social information than any other age groups. In addition, Fig. 4(c) illustrates on average that users are engaged or married reveal more information than any other status. Apathetic and libertarian users disclose more information than any other political references as shown in Fig. 4(d).

6 Privacy Protection Mechanism

In this research, we attempt to mathematically formalize users' data revelation behavior on social network sites. The preliminary model is developed to facilitate privacy protection mechanism. Facebook users can configure what information to be available to platform applications. However, Facebook privacy setting is configured after opt-out. In other words, all of a user's profile information is accessible unless the user knows that specific information must configure as inaccessible. As we can see from Sec. 4 and 5, an opt-out model does not provide sufficient protection against data harvesting. In this section, we address this issue and develop a privacy-protection system (PPS) that automatically configures a user's privacy settings based on the user's profile information. Figure 5 shows the model of our privacy-management system. The system uses three components: profile information (PI), privacy manager (PM), and profile zoning (PZ).

Profile Information (PI) contains two types of information: personal and social information. The personal information contains age, relationship status, and political views. On the other hand, social information consists of hometown, current residency, activities, interests, music, TV shows, movies, books, high school information, education history, profile note, and wall postings. We select personal information as our main features that can characterize users' personality and can be easily obtained from profile information.

In profile zone (PZ), PPS divides the user's profile information into two zones. One zone carries information which is accessible by platform application. The other zone carries information, which can not be accessed by platform application.

Privacy configuration takes place in the privacy manager (PM). PM performs its task in two phases. Phase 1 is sampling the network to discover information—revelation behavior. Sampling should not take place every time a new user

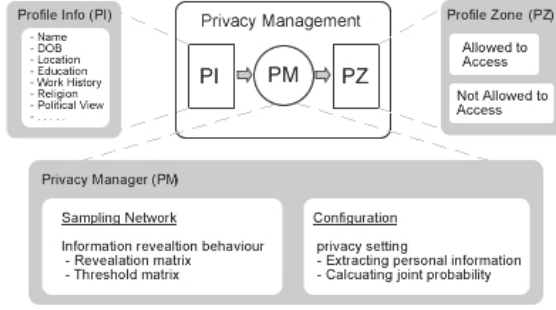


Fig. 5. Privacy management system for configuring users' privacy setting for applications

joins the network, but it only after a significant change in network population or substantial number of users changing their personal information. In this study, we randomly select 4,919 users in the University of North Texas (UNT) network. In phase 2, PM configures the user's privacy setting using *revelation matrix* and *threshold matrix*. With this system, users would have to opt-out of privacy rather than opt-in to making all their information public.

6.1 Building Revelation Matrix

Revelation matrix is built using statistical analysis on personal information of 4,919 UNT users. Personal information is categorized as follows: gender with two subgroups of male and female; age with four subgroups of ages range 15 to 19, 20 to 24, 25 to 29, and 30 and up; relationship status with five subgroups of single (S), in relationship (IR), engaged (E), married (M), and open relationship (OR); and political preference with seven subgroups with very liberal (VL), liberal (Li), moderate (M), conservative (Con), very conservative (VC), apathetic (A), and libertarian (Ln). On each subgroup, the same statistical analysis is applied. Let's take an example of male subgroup. First step is finding all male users from the sample of 4,919 users, and then calculate percentage of the users who reveal age, hometown, current residence, etc. Equation 1 is applied for each subgroup. $R_{i,j}$ is percentage of users who are in j subgroup reveal their feature i , where $i = \{Age, Hometown, Cur.resident..., Wall\}$, $j = \{Male, Female, 15 - 19, ..., Ln\}$, n_i and N_j are total number of users corresponding to i and j , respectively.

$$R_{i,j} = \frac{n_i}{N_j}. \quad (1)$$

An element of the revelation matrix can also be interpreted as a probability of users revealing profile features based on their personal information.

6.2 Building Threshold Matrix

The second step in sampling network phase is building threshold matrix. Threshold matrix shows what is average feature revelation for each subgroup. Using

Table 3. Demographic of features revelation by gender, age, relationship, and politic preference

	Gender		Age					Relationship status					Political preference							
	Male	Female	15-19	20-24	25-29	30-Up	S	IR	E	M	OR	VL	Li	Mo	Con	VC	A	Ln		
Age	0.83	0.51	x	x	x	x	0.84	0.84	0.83	0.86	0.92	0.85	0.82	0.84	0.86	0.76	0.84	0.87		
Hometown	0.64	0.39	0.68	0.64	0.54	0.48	0.63	0.64	0.72	0.67	0.67	0.65	0.65	0.66	0.64	0.66	0.64	0.81		
Cur. resident	0.29	0.16	0.28	0.28	0.28	0.17	0.27	0.28	0.31	0.36	0.25	0.31	0.32	0.31	0.27	0.26	0.39	0.24		
Gender	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
Relationship	0.85	0.53	0.85	0.86	0.87	0.87	x	x	x	x	x	0.90	0.83	0.88	0.89	0.88	0.92	0.90		
Political view	0.66	0.38	0.48	0.65	0.66	0.83	0.65	0.64	0.70	0.70	0.50	x	x	x	x	x	x	x		
Activities	0.68	0.40	0.68	0.66	0.65	0.70	0.68	0.67	0.67	0.67	0.50	0.62	0.71	0.70	0.68	0.74	0.69	0.79		
Interests	0.84	0.52	0.78	0.85	0.83	0.70	0.85	0.84	0.83	0.84	0.75	0.88	0.86	0.88	0.84	0.88	0.91	0.87		
Music	0.88	0.55	0.87	0.88	0.87	0.78	0.90	0.89	0.86	0.85	0.75	0.91	0.92	0.91	0.88	0.96	0.96	0.90		
TV shows	0.64	0.43	0.79	0.67	0.59	0.61	0.68	0.66	0.72	0.69	0.75	0.66	0.72	0.73	0.66	0.72	0.74	0.73		
Movies	0.85	0.54	0.83	0.86	0.85	0.83	0.87	0.86	0.89	0.83	0.92	0.88	0.89	0.90	0.86	0.88	0.89	0.91		
Books	0.75	0.47	0.65	0.77	0.73	0.78	0.78	0.75	0.78	0.77	0.58	0.85	0.82	0.80	0.76	0.86	0.88	0.93		
High school	0.92	0.59	0.92	0.95	0.92	0.87	0.93	0.93	0.92	0.95	0.92	0.96	0.94	0.94	0.95	0.84	0.91	0.89		
Education	0.98	0.62	1.00	0.99	0.99	1.00	0.99	0.99	0.99	0.99	1.00	1.00	1.00	0.99	0.98	1.00	0.96	1.00		
Note	0.21	0.14	0.44	0.21	0.15	0.22	0.25	0.21	0.15	0.17	0.42	0.31	0.29	0.22	0.14	0.20	0.21	0.26		
Wall	0.96	0.60	0.93	0.96	0.98	0.87	0.96	0.96	0.94	0.98	1.00	0.91	0.94	0.97	0.96	0.98	0.96	0.97		

revelation matrix, threshold is calculated as Eq. (2), where T_j is average profile —feature revelation for subgroup j and $|i|$ is size of the feature set.

$$T_j = \frac{1}{|i|} \sum_i R_{i,j}. \quad (2)$$

Threshold matrix shows that what is average probability for each subgroup.

6.3 Configuring Privacy Setting

To find a suitable privacy setting for profile features, we use joint probability technique on the four —main features: age, gender, relationship status, and political preference. Because the statistical analysis is done on main features with replacement (*independent variables*), the probability that users reveal their profile features is the product of the main features.

After completing phase 1: building revelation and threshold matrices, configuration of privacy settings can take place. Unlike phase 1, phase 2 takes place every time a new user joins a network. Probability of revealing a feature is calculated by joint probability of four main features of personal information, and it is compared against joint probability of threshold value of given subgroups. If the value is greater then threshold's value, the feature is set to be accessible by platform applications (3), otherwise the feature is not accessible (4), where U_p is a set of person p 's personal information e.g., $U_1 = \{Male, 24, S, Mo\}$.

$$if \left(\prod_{k \in U_p} R_{i,k} \right) > \left(\prod_{k \in U_p} T_k \right), accessible. \quad (3)$$

Table 4. Threshold matrix is built for categories of gender, age, relationship status, and political preference

	Gender		Age					Relationship status					Political preference							
	Male	Female	15-19	20-24	25-29	30-Up	S	IR	E	M	OR	VL	Li	Mo	Con	VC	A	Ln		
Threshold	0.73	0.46	0.73	0.73	0.71	0.69	0.73	0.73	0.74	0.74	0.71	0.76	0.77	0.77	0.74	0.76	0.78	0.79		

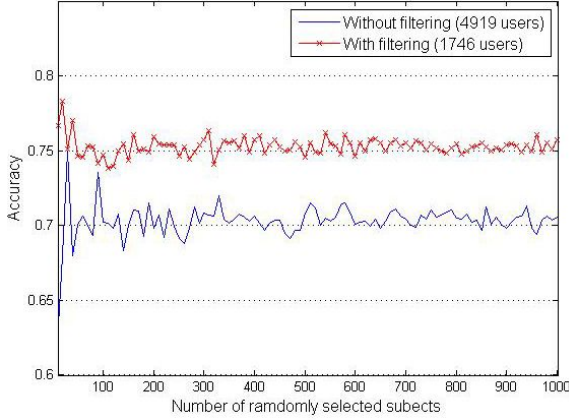


Fig. 6. Comparing accuracy of privacy management system using filtered dataset of 1,746 users with unfiltered dataset of 4,919 users. The system shows notable high error tolerance.

$$\text{if } \left(\prod_{k \in U_p} R_{i,k} \right) \leq \left(\prod_{k \in U_p} T_k \right), \text{ not accessible.} \quad (4)$$

Let's take an example to show how automatic configuration of privacy setting works. A new user who is male, 24 years old, single, and moderate-political preference joins the UNT network. By default, all features are inaccessible. Based on Tab. 3, we observe that probability of the user's revealing age is comparing joint probability (revelation matrix) of 83% (Male), 84% (S), and 84% (Mo) against joint probability (threshold matrix) of 73% (Male), 73% (20-24), 73% (S), and 77% (Mo). Result is 59% (revelation matrix) > 30% (threshold matrix), so the user's age is accessible by the applications.

7 Performance

To evaluate the performance of PPS, we test PPS with two sets of data. Based on our dataset, we find that 3,173 (65%) of 4,919 users do not provide one or more of the main features (age, relationship status, or political views). One dataset has all 4,919 members without any alteration, and the other one is filtered such that there are only users who have all four main features provided. The performance of PPS in terms of accuracy rate is shown in Fig. 6 where the accuracy rate is measured by correctly configured privacy settings of randomly selected users from the dataset. After the system configures the users' privacy settings for each feature based on the their personal information, we compare the configuration with the user's actual setting. Without filtering the data, the accuracy converges to 70%. On the other hand, with filtered the data, the accuracy of privacy configuration converges to 75%. Even though 65% of 4,919 users lack one or

more of the personal information, PPS still able to perform 70% accuracy. In other words, only five percent less than 75% indicates that system has *high error tolerance*.

8 Conclusion

Social networking boundaries appear to have been pushed in every way possible to allure new users and to keep current users. Open platform gives great flexibility to application developers to be creative and innovative as possible. Even though it gives benefit to both Social Network Sites and to application developers, customers' private information can be unawaresly accessed. Not all developers are legitimate. Without careful consideration of privacy management, open platforms result in information harvesting for illegal or unethical purposes. We purpose a privacy management system as a solution to the privacy problems on SN sites. The system uses probabilistic approach based on information revelation of users to recommend a more appropriate privacy setting for the user. The system restricts access to users' personal information unless they wish to make the information available. Our experiment shows that our approach can achieve 75% accuracy. In addition its high error tolerance makes it a suitable technique for user content management environment.

It is arguable to apply opt-out privacy model, where all profile information are inaccessible as default. However, from business point of view, the users' profile information are asset. This could be a reason that Facebook uses opt-in privacy model. Disadvantages of using opt-in model are that new users trust the system and platform applications, that all profile information are accessible as default, and that inexperienced users are unaware of their personal information are being harvested. Thus, we believe that the PPS can facilitate new user's profile settings by not having to reveal all personal information which are not intended to share with unknown users.

Acknowledgments

This work is supported by the National Science Foundation under grants CNS-0627754, CNS-0619871 and CNS-0551694. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We would like to thank our anonymous reviewers for their insightful and helpful comments and suggestions.

References

1. Hargittai, E.: Whose space? differences among users and non-users of social network sites. *Journal of Computer-Mediated Communication* 13(1) (2007)
2. Nie, N., Hillygu, S.: Where does internet time come from?: A reconnaissance. *IT & Society* 1(2), 1–20 (2002)

3. Acquisti, A., Gross, R.: Imagined communities: Awareness, information sharing, and privacy on the facebook. In: Danezis, G., Golle, P. (eds.) PET 2006. LNCS, vol. 4258, pp. 36–58. Springer, Heidelberg (2006)
4. Boyd, D.: Why Youth (Heart) Social Network Sites: The Role of Networked Publics in Teenage Social Life. Massachusetts Institute of Technology, Cambridge, MA (2008)
5. Zukowski, T., Brown, I.: Examining the influence of demographic factors on internet users information privacy concerns. In: SAICSIT Conf., pp. 197–204 (2007)
6. Barnes, S.B.: A privacy paradox: Social networking in the united states. First Monday 11(9) (August 2006)
7. Alexa: Facebook.com - facebook. Technical report, alexa.com (2007)
8. Facebook: Facebook developers. Technical report, facebook.com (2007)
9. McCarthy, C.: Facebook platform attracts 1,000 developers a day. Technical report, CNET News.com (2007)

HyDRo – Hybrid Development of Roles

Ludwig Fuchs and Günther Pernul

Department of Information Systems
University of Regensburg, 93053 Regensburg, Germany
{Ludwig.Fuchs, Guenther.Pernul}@wiwi.uni-regensburg.de

Abstract. Defining valid enterprise-wide roles needs to be carried out on the basis of a predefined Role Development Methodology. Hybrid role development combining elements from Role Engineering and Role Mining is the most promising way to define enterprise-wide roles, however no such model has been published yet. We close this gap by analysing existing approaches and proposing HyDRo, a tool-supported methodology that facilitates existing identity information and access rights without neglecting the importance of information like managers' knowledge about their employees.

Keywords: Role Development Methodology, Role Engineering, Role Mining, Identity Management, Information security.

1 Introduction and Motivation

As a result of ineffectual account management within organisations, users accumulate a number of excessive rights over time, violating the principle of the least privilege [1]. Major security problems arise because of employees gaining unauthorised access to resources as a result of manually handling user accounts ([2], [3]). This situation results in the so called identity chaos. In-house Identity Management (IdM) has become a means to solve the aforementioned identity chaos. It deals with the storage, administration, and usage of digital identities during their lifecycle. Roles acting as intermediary between employees and their access rights are an essential element of IdM. They allow companies to ease and secure provisioning processes, i.e. the allocation of digital and non-digital assets to employees, and access to resources in their IdM Infrastructure (IdMI) [4]. However, the most expensive challenge before achieving the benefits of role usage is the preliminary definition of valid roles [5]. Some companies deal with this issue by installing resource-intensive procedures based on organisational and operational structures. These approaches are known as Role Engineering Methodologies. In contrast, Role Mining Methodologies create roles using data mining tools that analyse and cluster existing user permissions providing a high degree of automation. This paper underlines the need for hybrid role development combining Role Engineering and Role Mining as the most promising approach for defining enterprise-wide roles. Up to now, to the best of our knowledge, no such model has been published. Hence, the main goal of this work is to close this gap by proposing HyDRo, a hybrid Role Development Methodology (RDM) that integrates

Role Engineering and Role Mining elements into a comprehensive framework for role creation. Central modelling requirements of HyDRo are the shortcomings of existing models, literature analysis, practical experiences, and requirements from industry partners.

This paper is structured as follows. In section 2 we present and compare existing Role Development Methodologies in order to show their shortcomings. Subsequently, section 3 introduces HyDRo, a methodology for hybrid development of roles. In section 4 we provide an overview over contrOLE, a role development tool that supports HyDRo. Conclusions and future work is given in section 5.

2 Existing Role Development Methodologies

Role Development Methodologies can in general be categorised according to the input information they are based on (see figure 1): *Role Engineering* is considered as the theoretical way of developing roles where roles are derived Top-Down based on information from the OOS (Organisational and Operational Structures) layer within an enterprise. This includes knowledge about hierarchical structures, process- or workflow definitions, or employees’ task bundles. Role Engineering following an aggregation or decomposition approach offers the chance to define a role catalogue that is closely aligned to the business perspective within a company. Decomposition approaches define roles and break them down into permissions needed while aggregation works the opposite way [6]. *Role Mining* on the contrary is the tool-based Bottom-Up approach discovering roles using existing identity information and access rights from the Directory layer. It in general investigates users and their existing access rights and is usually based on clustering algorithms which can be divided into statistical clustering or usage of neuronal networks.

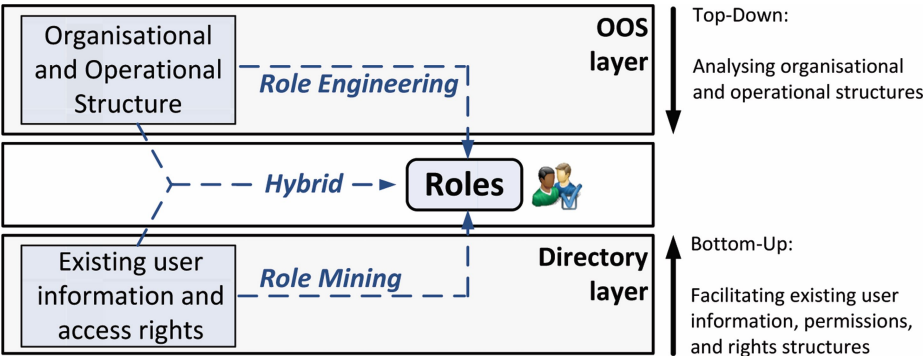


Fig. 1. Role Development Methodologies

We define role development as the umbrella term for Role Engineering and Role Mining. Role development can be carried out hybrid or non-hybrid. Several publications explicitly mention that a hybrid combination of Role Engineering and Role Mining is necessary to define a good collection of roles ([7], [8], [9], [10], [11], and

[12]). However, although considered to be the most promising way for developing roles on a company-wide level, no such hybrid RDM has been published up to now.

2.1 Role Engineering (Top-Down)

The importance of Role Engineering was first mentioned by Edward Coyne [13] after the upcoming of the original RBAC (Role-Based Access Control) model [14] in 1996. Role Engineering following the **decomposition approach** involves an in-depth analysis of business processes and functional structures in order to break down these elements to system-specific features needed to fulfil certain tasks. Crook et al. [9] showed that using organisational structure to define roles has significant advantages by providing a clear focus for analysts and users eliciting requirements. Roeckle et al.'s approach [8] on the contrary integrates business processes into the Role Engineering duties. They aim at finding the complete IT supported set of job functions performed in an organisation. While decomposition is used mainly for defining system-independent roles, **aggregation approaches** are adopted in the process of developing application-specific roles. They are based on use case- or scenario descriptions (scenarios can be regarded a specific representation of a use case [15]), goals, or other input information. In general, aggregation approaches define the way of interaction with an application and the bundles of permissions needed to fulfil certain tasks within this application. In order to streamline the mainly manual aggregation process, Strembeck presented a tool-based technique for defining scenarios ([16], [17]) and extract RBAC-models from BPEL4WS processes [18].

Shortcomings

Role Engineering significantly depends on human factors and the amount and quality of input information available. Above all in settings where the quality of organisational charts and job descriptions is high, Role Engineering is a promising approach to find role candidates. However, on the other hand it is primarily a manual task involving extensive communication between stakeholders [19]. A comparison of existing Role Engineering methods showed that only decomposition approaches are feasible for developing system-independent roles. Aggregating single elements like tasks comprehensively into roles is not applicable in an enterprise-wide project as most approaches are lacking any tool support. With dozens of business processes, thousands of users, and millions of authorisations in big organisations, this is seemingly a difficult task. Besides the high complexity and costs the collection and preparation of input information are the main drawbacks ([8], [19]). Practical experience has moreover shown that Role Engineering neglects existing access rights and thus the actual situation within a company. Hence, relying solely on Role Engineering for defining company-wide roles is not feasible.

2.2 Role Mining (Bottom-Up)

As a result of the presented shortcomings of Role Engineering, Role Mining has over the last years evolved as the pragmatic approach to rapidly define adequate roles. It specifically focuses on the usage of data mining technology for definition of system-independent roles that can, amongst others, be used in IdMIs for user management.

Role Mining automates the development process by using tools to identify potential roles. In contrast to Role Engineering, Role Mining is based on the assumption that the actual roles already exist within the IT infrastructure. Existing permission assignments are aggregated to define role candidates using statistical clustering algorithms or neuronal networks. Statistical clustering can be carried out hierarchically or partitioning whereas neuronal networks use unsupervised learning methods for role development. In [7] Vaidya et al. surveyed existing Role Mining approaches that mostly present heuristic ways to find a set of role candidates. Kuhlmann et al. ([20], [21]), ORCA [19], and Vaidya et al. [22] are identified as the most important publications in that area. Kuhlmann et al. propose a clustering technique closely related to the k-means algorithm. In [19], Schlegelmilch et al. facilitate an agglomerative hierarchical clustering based algorithm, which discovers roles by merging permissions appropriately. Additionally, Vaidya et al. [22] propose RoleMiner, an approach based on subset enumeration. Recently [10], [23], [24], and [25] have presented specific improvements, integrating cost and performance decisions as well as semantics into Role Mining.

Shortcomings

Even though providing a high degree of automation, Role Mining has several serious unaddressed drawbacks: If the input quality is erroneous the role candidates discovered are also incorrect. Existing approaches assume that cleansing already took place before the role definition. We argue that this issue needs to be addressed by introducing a mandatory customisable data cleansing and -preparation phase as shown in [11] in order to ensure an appropriate quality level of the input information. Investigating existing literature has moreover shown that most publications only present algorithms for finding the optimal role set without taking into consideration that business needs have to be involved in a role development project. As it is not their main focus, none of them adheres to existing methodological requirements.

3 HyDRo – A Methodology for Hybrid Development of Roles

As aforementioned, neither pure Role Engineering nor pure Role Mining leads to an optimal role catalogue. Our analysis and practical experiences with existing RDMs underline these findings stating that a hybrid approach is the most promising basis for role creation. On the one hand the automation capabilities of Role Mining are needed while consideration of business functions and organisational structure is a mandatory element of a RDM on the other hand. None of the existing approaches shows how Role Engineering and Role Mining can be combined and how the information flows can be structured. Based on shortcomings of existing models, literature analysis, and practical experiences we are now going to introduce HyDRo, a new hybrid methodology for developing roles. The goal of HyDRo is the definition of system-independent roles usable within IdMIs. HyDRo considers existing user information and access right structures without neglecting the importance of organisational structures and information like managers' knowledge about their employees. It can be easily integrated into the proROLE framework [11] representing a role system lifecycle. The underlying philosophy is perform a joint Role Mining/Engineering approach and

integrate OOS layer representatives (managers, executives, CIO) as frequently as necessary but as infrequently as possible.

Methodological Background

In complex environments role development projects need to be carried out on basis of a predefined methodology in order to derive a consistent role catalogue and reducing failure risks. However, existing Role Engineering and Role Mining approaches lack a clear definition of mandatory method elements. HyDRo overcomes these significant shortcomings by being modelled based on method elements following well-defined Method Engineering principles ([26], [27]) (see figure 2, adapting the notation used by Brinkkemper [28]). We propose the **Procedure Model** as the central element of our methodology. Grey colouring marks elements directly integrated in the HyDRo procedure model: Necessary **Activities** and **Techniques** structured in six **Phases**, involved **Roles** (i.e. stakeholders), the documentation of **Results**, and the **Tool** used throughout the entire process. HyDRo is fully supported by the *contROLE* role development software which will be presented in section 4. HyDRo furthermore uses busi-ROLE [29] as **Meta-Model**.

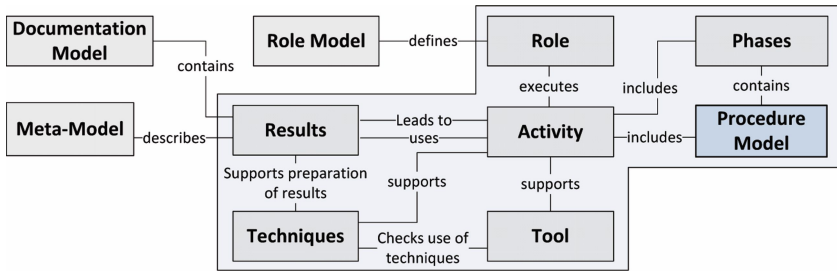


Fig. 2. Method Elements of HyDRo

3.1 Overview and Characteristics

In this section we analyse the different phases of HyDRo on basis of the aforementioned method elements. The methodology consists of six consecutive main phases and the respective interfaces in form of Quality Measurement (QM) and Execution Decision (ED) activities. Organisations applying HyDRo need to complete one phase to a predefined extent to be able to move on to the next phase. However, HyDRo is designed to provide maximum flexibility during role development; hence, users of the methodology can move back to previous phases or within phases in an incremental and iterative fashion. Companies applying HyDRo can re-run a single phase if the resulted quality is insufficient or new input is provided changing existing results. Figure 3 provides a high level overview of the main phases: The HyDRo process starts with the import of necessary input data (**Data Gathering**) and consecutive **Data Cleansing**. In order to define suitable roles, the input data is then classified and selected in a separate phase (**Data Preparation and Selection**). The role development process itself is split into three phases, namely the definition of **Basic**-,

Organisational-, and Functional Roles. We will discuss each of the phases in more detail in the following sub-section 3.2.

Quality Measurement and Execution Decision

One central business requirement for a hybrid RDM is the definition and measurement of partial results during the methodology execution. Business- as well as IT representatives desire milestones during the role development project that form the basis for further execution decisions. Figure 3 indicates the partial result measurement and -decisions at each transition between two phases. Every HyDRo phase ends with a QM and ED process step. Depending on the phase, different indicators provide information about the result quality. Data cleansing output quality can e.g. be measured by analysing the percentage of corrected input data. In contrast to the QM task, where the result quality of one phase is measured, ED activities take general project drivers into account. Even if the result quality of one phase is sufficient, companies applying HyDRo still might want to abort the role development as result of e.g. lacking funding, other prioritised projects, or time schedule issues.

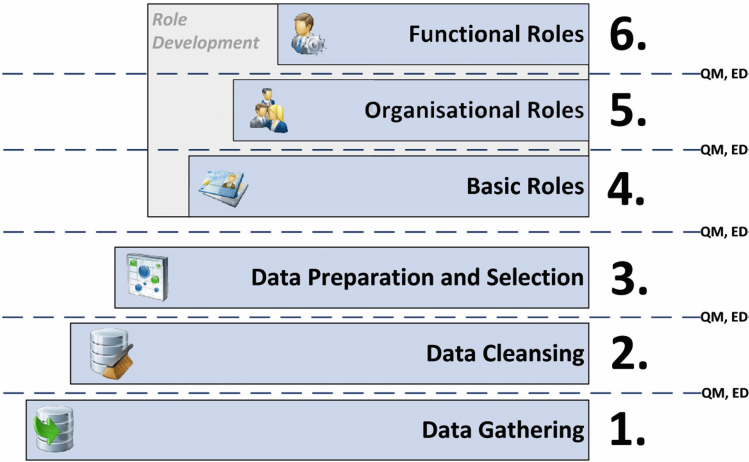


Fig. 3. Phases of HyDRo

3.2 HyDRo Phases

In the following we are going to introduce HyDRo on basis of its **Procedural Model** split into the six main **Phases** shown in figure 3. In order to highlight the interdependencies and information flows between the OOS- and Directory layer we use a visualisation schema integrating the **Phases** and **Activities**, the included stakeholders (**Roles**), the used **Tool**, as well as the derived **Results** (figure 4, 5, and 6). This allows for a clear distinction of Role Mining and Role Engineering elements.

3.2.1 Data Gathering

Within the data gathering phase input information from Role Engineering and Role Mining sources needed for hybrid role development is imported. The goal of this phase is the compilation of a consistent raw input information repository representing the basis for further data cleansing-, data preparation-, and role development activities. Thus, after the kick-off of the role development project using HyDRo the various available input sources are identified. Information needs to be imported and checked for consistency. HyDRo considers existing user rights in form of a LDAP-repository or a .csv-file as mandatory raw data from the Directory layer. If this information is not available, HyDRo is not applicable or, if exclusively input information from the OOS layer is available many Role Mining activities are not executable. Besides the mandatory identity information, input from the OOS layer is optional but highly desired. It might be available in forms of defined job positions, task bundles, processes, or already existing local role definitions from certain departments. Existing Top-Down knowledge is imported in order to compose the raw input information repository consisting of all available OOS- and Directory layer input information. After a basic quality measurement which checks if enough input information is available for consecutive HyDRo phases, business representatives like the manager of an organisational unit, need to review the information available for his department. This way he can alter basic information about his employees and identify employees with active user accounts but who are no longer working for the company.

3.2.2 Data Cleansing

If a certain minimum of input information is available, HyDRo continues with the data cleansing phase (see figure 4). The overall goal of this phase is to improve the data quality of the input information and thereby overcome the deficits of most existing approaches which do not include any data cleansing mechanisms. Data Cleansing in HyDRo is split into syntactic- and semantic cleansing. While syntactic checks might be fully automatable, semantic checks cannot be processed without human intervention. Consider an employee in a multinational organisation. One can imagine that misspelled user attributes like his location attribute within the global identity repository can easily be identified. However, if a user has a wrong assignment of a valid location, it is not possible to resolve this inconsistency without any information from the OOS layer. During this phase Role Engineering- and Role Mining activities need to be combined to achieve the best results. HyDRo provides tool support by facilitating various syntactic checks, including duplicate checks or attribute checks against valid values. Even more important, it also provides various functions for semantic analysis of the underlying input information. For instance Self-organising maps (SOM) [30] are used to identify users which have untypical attribute values assigned. Again, consider our previously mentioned example of an employee working for a multinational organisation. After he changed his location and has been assigned to new privileges he might still be assigned to a number of his former access rights. SOMs can be used to identify and highlight such a user because some of his assigned privileges are typical for a different location than the one he is assigned to.

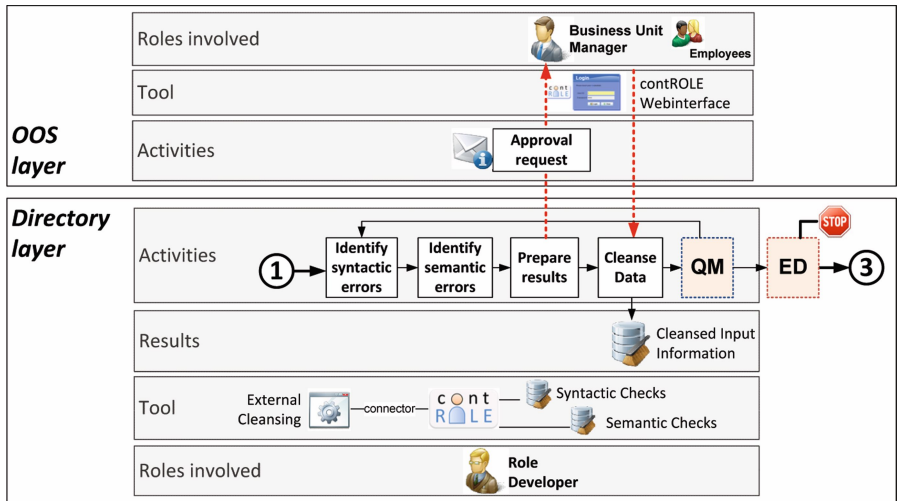


Fig. 4. The Data Cleansing Phase of HyDRo

3.2.3 Data Preparation and Selection

After input data has been cleansed the role development process moves on to the Data Preparation and Selection phase which exclusively comprises activities at the Directory layer (figure 5). Its goal is to generate additional knowledge about the underlying input information. We argue it is mandatory to allow companies to choose the appropriate part of the raw input data to be included in the role development. HyDRo starts by analysing the underlying input information on a global level. One aspect is the exclusion of further manually administered rights. Our experience has shown that a high number of rights are only held by a small number of users making them not feasible for role-based allocation. After the global classification and selection process, single hierarchical elements are classified locally. A hierarchical element is a unit in the organisational structure of an enterprise, for example a business unit, a department, or a unit within a department. Techniques like statistical analysis, clustering algorithms, or results from previous RDM phases can be used. The generated information might be of high relevance for business representatives as well as role developers: In contROLE, for example, a green traffic light in front of an organisational unit represents simple user- and access rights structures or a high amount of cleansed input information. In this case the respective organisational unit is a candidate for rapid role development. On the contrary, a red traffic light classifies organisational units as improper for easy role development as a result from e.g. a lack of cleansed input information. However, automatically classifying input information needs to be carefully parameterised. Fundamentals are the predefined classes of users and access rights, the amount of cleansed input information, or, even more complex, the grade of interdependencies between hierarchical elements.

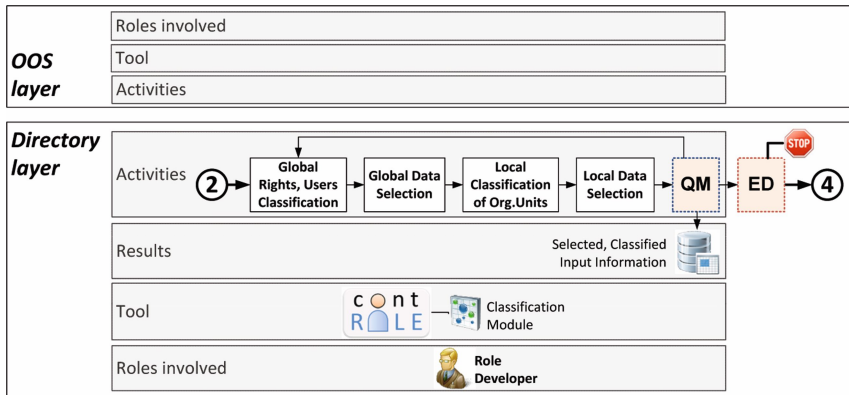


Fig. 5. The Data Preparation and Selection Phase of HyDRo

3.2.4 Role Development

After the input information has been cleansed and prepared, phases 4, 5, and 6 represent the actual role development phases of HyDRo. The outcome of each of those phases is a set of defined roles of a certain type: Basic Roles (phase 4) bundle common access rights. Organisational Roles (phase 5) represent job positions while Functional Roles (phase 6) correspond to the task bundles of employees. By allowing for the definition of business role types [29], HyDRo supports incremental role development. Phases 4, 5, and 6 from figure 3 are modelled similar, even though the underlying algorithms, the extent of hybrid communication, and the importance of OOS layer input information are varying. The amount of required Top-Down input is constantly increasing while the usage of Role Mining algorithms is decreasing as a result of growing complexity of role definition. However, companies can decide whether they want to define Functional Roles or whether they abort HyDRo after the definition of Basic- and Organisational Roles. In some cases a large part of the access rights might be already administered using those types of roles in which case the additional flexibility gained by the usage of Functional Roles is outweighed by the large amount of time and money spent for their definition.

Basic Roles

HyDRo starts with defining Basic Roles that are assigned on basis of organisational membership of users in different hierarchies. In general, they represent the bundle of access rights that are granted to every employee in a certain hierarchical element independent from his job position. They can be inherited, depending on the hierarchy type. Basic Roles could include rather common rights like "Internet Access" and are derived using Role Mining algorithms. The permissions assigned to a certain percentage of the employees within an element, e.g. more than 90%, are bundled and marked as a possible Basic Role. The manager of this hierarchical element is then informed via email that he needs to approve or alter the found role candidates.

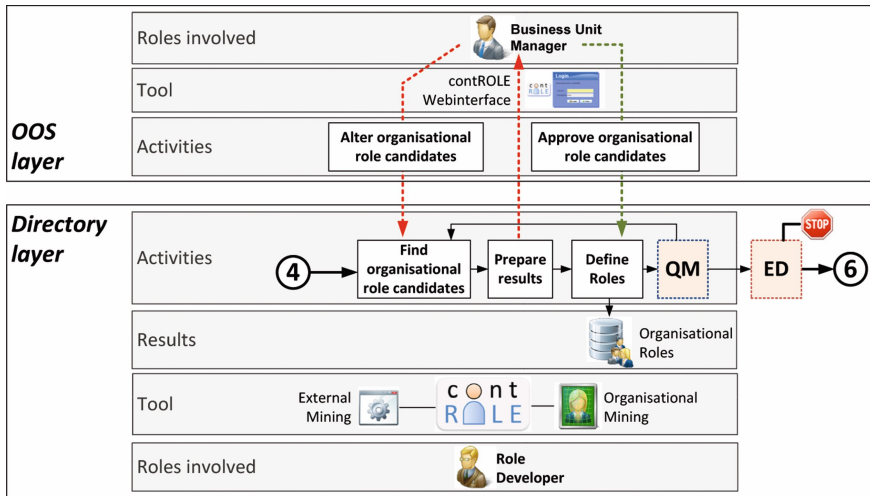


Fig. 6. Role Development in HyDRo (example: Phase 5 “Organisational Roles”)

Organisational Roles

In phase 5 Organisational Roles, i.e. job positions of employees within the organisational structure are derived (see **Fehler! Verweisquelle konnte nicht gefunden werden.6**). It is likely that in some hierarchical elements positions are already defined. In other situations this might not be the case and Role Mining technologies are needed to cluster employees with the same access rights. These clusters need to be visualised appropriately and presented to the OOS representatives. Quality criteria for this phase might be the number of employees which are assigned to a certain Organisational Role, the quality of feedback derived from the OOS representatives, or the percentage of permissions that is administered by the usage of the defined Organisational Roles.

Functional Roles

Functional Roles represent task bundles of employees and are amongst others used for delegation purposes. They might be specific for a hierarchical element or valid throughout the whole enterprise. On the one hand, the employees’ job positions have to be split up into various task bundles. On the other hand the task bundles might represent special duties of a number of employees, independent from any organisational hierarchy. The development of Functional Roles needs to be heavily supported by human interaction. Finding task bundles purely based on Role Mining algorithms is hardly possible. Automatically analysing employees with different job positions whose permissions overlap is one possible approach.

4 The contROLE Role Development Tool

HyDRo is fully supported by *contROLE*, a role development tool which currently is being implemented at our department. ContROLE is not designed as a Role Mining

tool according to section 2.2. It is rather an infrastructural tool that fosters the hybrid integration of Role Engineering by providing a maximum of process- and communication automation. Figure 7 shows the main interface of contROLE during Data Cleansing with the single phases of HyDRo seen in the upper window. Using client-server architecture, OOS representatives as well as Role Developers can use contROLE during the hybrid role development loops. Due to the limited space we only present selected features of contROLE and give a short quality analysis of implemented Role Mining algorithms using various predefined input data sets.

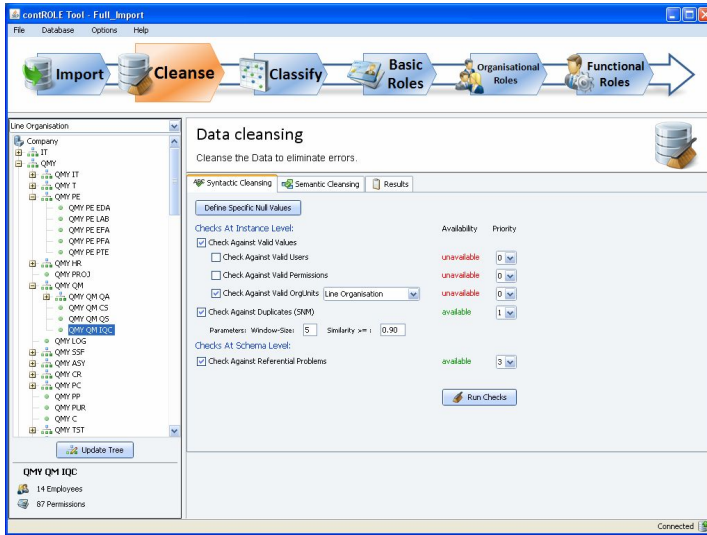


Fig. 7. ContROLE Role Developer Interface

As stated beforehand, contROLE offers a wide variety of data cleansing and data preparation functionalities that have to the best of our knowledge not been integrated in any other RDM. Syntactic- and semantic data checks can be used for cleansing the input information. This includes checks against valid permissions, users, and organisational hierarchy elements. Moreover contROLE is able to detect outliers and suspicious user-permission assignments. The role developer can design and parameterise a data cleansing process according to the available input information. For detecting and visualising outliers contROLE amongst others implements a connection to the *SOM Toolbox*, an already existing implementation of SOMs developed in the GHSOM Project [31]. We facilitate the capabilities of SOMs to find suspicious users in terms of wrong attributes or erroneous rights allocation. Figure 8 shows the visualisation of Directory layer information of one of our industry partners. One can see various suspicious data elements (arrows). These elements are automatically marked by contROLE and sent to the respective managers for approval together with a proposed attribute value.

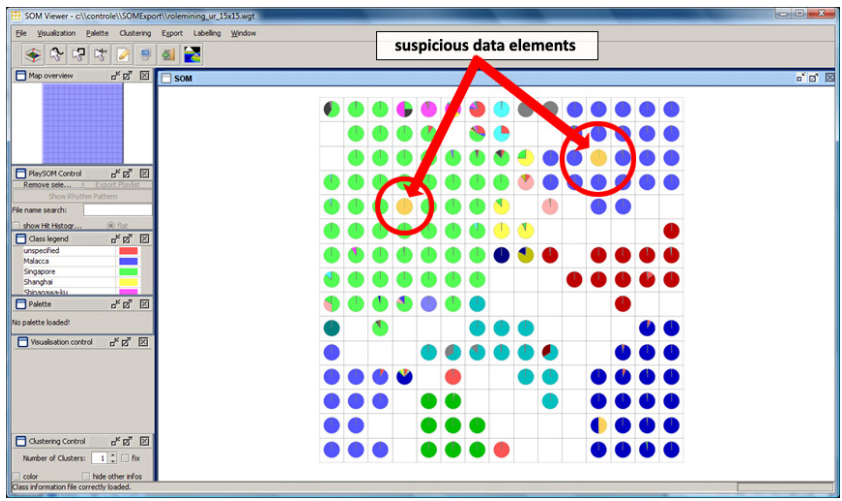


Fig. 8. SOM Representation of Access Rights

As mentioned beforehand many access rights might still be administered manually because they represent special permissions held by only a very small number of employees. Figure 9 represents the examination of access rights structures of one representative line organisation department of a large industrial company consisting of 103 users and 223 different access rights. The visualisation used orders the different access rights on the horizontal axis according to the number of users being granted this right (ascending). It can be seen that a very large number of rights are only held by one user (108 rights). This underlines the need for a careful data selection in order to ensure the definition of usable roles. ContROLE can automatically pick rights that should be further manually administered while the role developer can additionally disable rights for the consecutive role development phases. A further analysis of figure 9 points out that a relatively small number of rights are held by all users of this department, representing possible Basic Roles.

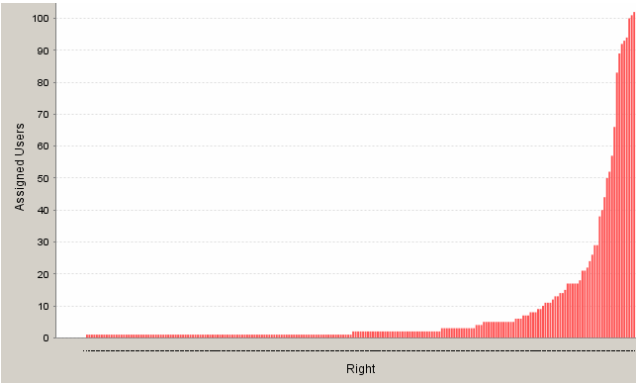


Fig. 9. Access Right Structures within a Hierarchical Element

ContROLE Role Mining Performance Analysis

Regarding the definition of role candidates, contROLE implements the Role Mining algorithms ORCA [19], FastMiner, and CompleteMiner (both presented in [22]). The quality of the clustering results presented in this section led to the development of our own Role Mining algorithm which is currently being implemented and tested. In order to validate the result quality of the already existing algorithms we carried out a performance and clustering analysis on basis of evaluation principles shown by Pries-Heje et al [32]. We decided to conduct an ex-post evaluation using artificial and naturalistic input datasets (see table 1).

Table 1. Input datasets for the Role Mining quality analysis

Dataset	Users	Permissions
Artificial Set 1 (AS1)	6	4
Artificial Set 2 (AS2)	13	4
ContROLE Artificial Set (AS3)	168	29
Small Naturalistic Set (NS1)	362	366
Big Naturalistic Set (NS2)	1211	805

AS1 and AS2 were included in the corresponding Role Mining publications [19] and [22] representing simple illustrative examples of user and permission structures. They include only a small number of user permission assignments. AS3 has been designed for contROLE functionality tests and represents a middle sized company with 168 employees. In contrast to the artificial sets AS1 and AS2 it includes organisational structures and hierarchies. NS1 and NS2 are both naturalistic datasets provided by our user partners, gathered from their global Identity Management System in place. They each represent user permission assignments of one department within the line organisation including a large number of sub-departments and hence complex hierarchical structures. A performance analysis revealed that most Role Mining algorithms were able to finish the computation process in a reasonable time (up to 360 seconds depending on the dataset size). Due to its complexity the CompleteMiner was the only exception not able to finish the role candidate discovery on basis of the big naturalistic set NS2. The hardware used for the computation was a machine with an Intel Core2Duo CPU and 3GB RAM. Our in-depth analysis moreover revealed that the result quality of the implemented algorithms strongly varies. Table 2 gives an overview over the number of derived role candidates (RC) using the given datasets from table 1 as input information for the Role Mining Algorithms implemented in contROLE. Note that the number of clusters using ORCA varies as a result of a random component within the algorithm. Hence we display an average value computed during various test loops. The reduced role candidate number computed using a minimum role membership (RM) limit is also given if that feature was provided by the corresponding algorithm.

Table 2. Found role candidates

Dataset	ORCA	FastMiner	CompleteMiner
AS1	4 Cluster	3 RC	2 RC
AS2	2 Cluster	4 RC	4 RC
AS3	18 Cluster	27 RC	27 RC
		<i>14 RC (min=10 RM)</i>	<i>14 RC (min=10 RM)</i>
NS1	~295 Cluster	1648 RC	2121 RC
		<i>108 RC (min=20 RM)</i>	<i>301 RC (min= 20 RM)</i>
NS2	~621 Cluster	14386 RC	
		<i>647 RC (min=20 RM)</i>	

Especially the analysis of the naturalistic datasets revealed that Role Mining algorithms tend to discover a large number of candidate roles which is not feasible in practical scenarios. Using NS2 as input information the FastMiner, e.g., discovered 14386 role candidates for 1211 users. Even though FastMiner and CompleteMiner both support the parameterisation with a minimum number of role members RM, the found role candidates still only can be seen as preliminary results. The biggest drawback of existing Role Mining algorithms is the missing integration of additional input information in the role discovery process. They exclusively facilitate user-permission assignments as input information, neglecting existing hierarchical and operational structures within a company. During the further development of contROLE we hence focus on the integration of OOS layer information into the role definition process.

5 Conclusions and Future Work

In this paper we have argued the need for a hybrid Role Development Methodology integrating Role Engineering and Role Mining functionalities. A literature analysis has investigated the existing models and shown that none of them sufficiently meets the requirements of a RDM. In order to close this gap we proposed HyDRo, to the best of our knowledge the first hybrid Role Development Methodology. HyDRo considers existing user information and access right structures without neglecting the importance of information like managers' knowledge about their employees. It overcomes the shortcomings of existing RDMs by being based on a well-defined method engineering basis and providing a comprehensive set of role development phases. Above all the Data Cleansing and Data Preparation and Selection phase have not been included in existing approaches. HyDRo moreover considers various business requirements in order to ensure applicability within real-life scenarios. One big advantage is the seamless tool-support. The contROLE software ensures the hybrid integration, cleansing, and selection of input information from various sources throughout an iterative and incremental role development process.

For future work we are focussing on the extension of the functionality of contROLE, especially the extension of classification- and Role Mining algorithms as well as the improvement of the user-interface. We furthermore are going to deal with performance issues arising when working with large datasets. This task affects above all the usage semantic Data Cleansing algorithms and the training of neuronal networks which can be a longsome process.

References

1. Ferraiolo, D.F., Kuhn, R.D., Chandramouli, R.: Role-Based Access Control. Artech House, Boston (2007)
2. Larsson, E.A.: A case study: Implementing Novell Identity Management at Drew University. In: Proc. of the 33rd annual ACM SIGUCCS conference on User services (SIGUCCS 2005), pp. 165–170. ACM, New York (2005)
3. Dhillon, G.: Violation of Safeguards by Trusted Personnel and Understanding Related Information Security Concerns. *Computers & Security* 20(2), 165–172 (2001)
4. Fuchs, L., Pernul, G.: Supporting Compliant and Secure User Handling – a Structured Approach for In-house Identity Management. In: Proc. of the 2nd Int. Conference on Availability, Reliability and Security (ARES 2007), pp. 374–384. IEEE Computer Society, Los Alamitos (2007)
5. Gallaher, M.P., O'Connor, A.C., Kropp, B.: The economic impact of role-based access control. Planning report 02-1, National Institute of Standards and Technology, Gaithersburg, MD (2002), <http://www.nist.gov/director/prog-ofc/report02-1.pdf>
6. Epstein, P., Sandhu, R.: Engineering of Role/Permission Assignments. In: Proc. of the 17th Annual Computer Security Applications Conference (ACSAC 2001). IEEE Computer Society, Washington (2001)
7. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: finding a minimal descriptive set of roles. In: Proc. of the 12th ACM Symp. on Access Control Models and Technologies (SACMAT 2007), pp. 175–184. ACM, New York (2007)
8. Roeckle, H., Schimpf, G., Weidinger, R.: Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In: Proc. of the 5th ACM workshop on Role-based access control, pp. 103–110. ACM, New York (2000)
9. Crook, R., Ince, D., Nuseibeh, B.: Towards an Analytical Role Modelling Framework for Security Requirements (2002), <http://mcs.open.ac.uk/ban25/papers/refsq02.pdf>
10. Colantonio, A., Di Pietro, R., Ocello, A.: Leveraging Lattices to Improve Role Mining. In: Proc. of the 23rd Int. Information Security Conference (SEC 2008) (2008)
11. Fuchs, L., Pernul, G.: proROLE: A Process-oriented Lifecycle Model for Role Systems. In: Proc. of the 16th European Conference on Information Systems (ECIS), Galway, Ireland (2008)
12. Shin, D., Ahn, G., Cho, S., Jin, S.: On modeling system-centric information for role engineering. In: Proc. of the 8th ACM Symp. on Access Control Models and Technologies (SACMAT 2003), pp. 169–178. ACM, New York (2003)
13. Coyne, E.J.: Role Engineering. In: Proc. of the 1st ACM Workshop on Role-based access control. ACM, New York (1996)
14. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-Based Access Control Models. *IEEE Computer* 29(2), 39–47 (1996)
15. Sadahiro, I.: A Critique of UML's Definition of the Use-Case Class. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 280–294. Springer, Heidelberg (2003)
16. Neumann, G., Strembeck, M.: A scenario-driven role engineering process for functional RBAC roles. In: Proc. of the 7th ACM Symp. on Access Control Models and Technologies, pp. 33–42. ACM, New York (2002)

17. Strembeck, M.: A Role Engineering Tool for Role-Based Access Control. In: Proc. of the Symp. on Requirements Engineering for Information Security (SREIS), Paris, France (2005)
18. Mendling, J., Strembeck, M., Stermsek, G., Neumann, G.: An Approach to Extract RBAC Models from BPEL4WS Processes. In: Proc. of the 13th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), pp. 81–86. IEEE Computer Society, Washington (2004)
19. Schlegelmilch, J., Steffens, U.: Role mining with ORCA. In: Proc. of the 10th ACM Symp. on Access Control Models and Technologies (SACMAT 2005), pp. 168–176. ACM, New York (2005)
20. Kuhlmann, M., Shohat, D., Schimpf, G.: Role mining - revealing business roles for security administration using data mining technology. In: Proc. of the 8th ACM Symp. on Access Control Models and Technologies (SACMAT 2003), pp. 179–186. ACM, New York (2003)
21. Kern, A., Kuhlmann, M., Schaad, A., Moffett, J.: Observations on the role life-cycle in the context of enterprise security management. In: Proc. of the 7th ACM Symp. on Access Control Models and Technologies (SACMAT 2002), pp. 43–51. ACM, New York (2002)
22. Vaidya, J., Atluri, V., Warner, J.: RoleMiner: mining roles using subset enumeration. In: Proc. of the 13th ACM Conf. on Computer and Communications Security (CCS 2006), pp. 144–153. ACM, New York (2006)
23. Colantonio, A., Di Pietro, R., Ocello, A.: A cost-driven approach to role engineering. In: Proc. of the 2008 ACM Symp. on Applied Computing, pp. 2129–2136. ACM, New York (2008)
24. Molloy, I., Chen, H., Li, T., Wang, Q., Li, N., Bertino, E., Calo, S., Lobo, J.: Mining roles with semantic meanings. In: Proc. of the 13th ACM Symp. on Access Control Models and Technologies (SACMAT 2008). ACM, New York (2008)
25. Vaidya, J., Atluri, V., Guo, Q., Adam, N.: Migrating to optimal RBAC with minimal perturbation. In: Proc. of the 13th ACM Symp. on Access Control Models and Technologies (SACMAT 2008). ACM, New York (2008)
26. Braun, C., Wortmann, F., Hafner, M., Winter, R.: Method Construction – A Core Approach to Organizational Engineering. In: Proc. of the 2005 ACM Symposium on Applied Computing, pp. 1295–1299. ACM, New York (2005)
27. Gutzwiller, T.: Das CC RIM-Referenzmodell für den Entwurf von betrieblichen, transaktionsorientierten Informationssystemen. Physica-Verlag, Heidelberg (1994)
28. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. *Information and Software Technology* 38, 275–280 (1996)
29. Fuchs, L., Preis, A.: BusiROLE: A Model for Integrating Business Roles into Identity Management. In: Proc of the 5th Int. Conference on Trust, Privacy, and Security in Digital Business (TrustBus), Torino, Italy (2008)
30. Kohonen, T.: *Self-Organizing Maps*. Springer, Berlin (2001)
31. The SOMLib Digital Library Project, Information & Software Engineering Group, Vienna University of Technology,
<http://www.ifs.tuwien.ac.at/~andi/somlib/index.html>
32. Pries-Heje, J., Baskerville, R., Venable, J.: Strategies for Design Science Research Evaluation. In: Proc. of the 16th European Conference on Information Systems (ECIS), Galway, Ireland (2008)

The Enlightened Era of Enterprise Security

(Invited Talk)

Basant Rajan

Chief Technology Officer, India, Symantec Corporation
India

1 Introduction

The more money and resources we seem to throw at security, the more the bad guys seem to catch up. With every new technology innovation, comes a new – and sometimes worse – threat to your business. It simply comes in a different form.

Loud, large scale virus attacks launched for glory have been replaced with stealth, financially-motivated attacks seeking confidential information. Spam is no longer just touting free Viagra. It's a delivery mechanism for phishing attacks, identity theft and malicious code. And malicious code propagates not just in email, but through web plug-ins, IM, smartphones and USB drives (just to name a few).

The biggest threat to a company's brand and bottom line these days isn't from hackers, it's from internal threats. It's the absent-minded contractor who leaves their laptop on a plane with unencrypted sensitive information. It's the disgruntled employee who steals thousands of customer credit card numbers from backend databases to sell on the black market.

While the range of risks can be mind numbing, they aren't unmanageable. Security just needs to be managed differently than it is today. To address internal and external threats in a way that doesn't inhibit collaboration or break the bank requires a fundamentally different approach to how we look at security.

First, security shouldn't be about eliminating risk altogether. Without risk, there is no opportunity. Instead businesses, and the IT and security professionals they rely on, should focus more on understanding the risks that will have the biggest impact and then determine the best way to eliminate those risks.

There is no longer an impenetrable wall around organizations. People are the new perimeter. Employees are everywhere, partners are faceless, and your brand isn't only in your hands. Companies are being forced to trust their data to third-parties and secure data that is not always in their control. In this environment, security can't only be about locking things down. Security should help guide organizations, enabling them to thrive and have confidence that their infrastructure, information and interactions are protected.

This is Security 2.0. Just as Web 2.0 offers new ways to boost productivity, increase revenue and costs, so will this next generation of security.

Security 2.0 isn't nirvana. It's an evolution. Security 1.0, which focused on making systems safe and keeping the bad guys out, is necessary but it's no longer good enough. Security 2.0 builds on 1.0 but expands protection to the information and interactions themselves. This requires a more dynamic view of security with

technologies and processes that adapt to the reputation or behavior of devices, people and applications. Security 2.0 is driven first by policy, then by technology, and it will be operationalized to speed progress and lower costs.

2 Protecting Information, Not Just Devices

The devices and systems we use are simply a suitcase for the real asset we're trying to protect. The information. Since the perimeter can't be shut down, security needs to focus on protecting the information itself. This requires knowing where your information is, what is sensitive or confidential, who has access to it, who needs access to it and how you make sure it's protected and available when you need it. Answering these questions requires security, operations and the business to work together.

According to a recent IT Policy Compliance Group report, 68 percent of organizations are experiencing six losses of sensitive data annually. With data breaches costing companies millions in lost revenue, share price declines, fines or customer loyalty, it's no surprise that companies are investing in new solutions to contain data leakage.

A variety of new solutions are coming to light to prevent data loss – solutions that enable companies to discover where information is in their organization, to set policies around entitlement or access, to filter confidential information from emails and IM, or to monitor security incidents and database patterns that could indicate malicious activity. But there's no silver bullet. Data loss prevention can't be addressed with a single piece of technology, and there is no substitute for understanding potential process weaknesses and training your people.

3 No More One-Size-Fits-All Security

Security should scale to the situation. The peanut butter approach to security doesn't work in today's threat landscape and changing business environment.

Take information controls for example. You wouldn't spend thousands of dollars protecting pictures from the company picnic, but you would in order to protect design documents, source code or credit card numbers. Your competitive position and brand reputation depend on it.

In Security 2.0, security adapts to the level of risk, what needs to be protected, and the reputation of those entities trying to access your systems and information. Security parameters should automatically change depending on whether a user is connecting to a network from inside the firewall or from an airport kiosk. Decisions should be made based on the behavior of users, historical information and the policies that are in place.

We see this happening already with anti-spam solutions, which analyze the behavior and reputation of IP addresses to determine what messages get blocked. Technologies like whitelisting and proactive threat protection in products like Symantec Endpoint Protection are another example of reputation-based security. These technologies consider both good and bad behavior to determine which applications and executables are permitted.

4 Operationalizing Security

Probably the biggest shift in Security 2.0 is how it's driven within an organization. Today most organizations are addressing security and risk in silos, with groups implementing distinct and often disconnected processes and technologies to mitigate the risks. These risks are often interconnected, but unfortunately the processes and technologies are not.

In order to lower operational costs and make security more effective, proactive and measurable, security needs to be embedded throughout business processes from the very start. Policies have to be consistently defined and socialized before controls can be put in place. The most successful companies look at policy first, and then implement the technology to automate it. Not the other way around.

By operationalizing security – standardizing, automating and driving down the cost of day-to-day security activities – companies and IT can be much more proactive when it comes to protection.

Security is an essential element to organizational health. We need to start looking at security the way we look at our own health – focusing on preventative care not simply seeing the doctor once you have a heart attack. Consider how much lower the medical bills are when you take your vitamins, eat right and exercise. The same could be said for security.

Author Index

- Aickelin, Uwe 173
 Amberker, B.B. 156
 Avancha, Sasikanth 124

 Baatarjav, Enkh-Amgalan 273
 Bhat K., Vivekananda 235
 Bonnecaze, Alexis 116
 Brumley, David 1

 Caballero, Juan 1
 Cavallaro, Lorenzo 203
 Cuppens, Frédéric 71
 Cuppens-Boulahia, Nora 71

 D'Souza, Deepak 26
 Dantu, Ram 273
 Das, Abhijit 235
 Das, Manik Lal 132
 Dasgupta, Kankar 258
 Düssel, Patrick 188

 Estan, C. 158

 Falcone, Yliès 41
 Fernandez, Jean-Claude 41
 Foschini, Luca 203
 Fuchs, Ludwig 287

 Gabillon, Alban 116
 Gehl, Christian 188
 Gupta, Phalguni 221

 Hicks, Boniface 56
 Hicks, Michael 56
 Holla, Raveendra 26

 Jaeger, Trent 56
 Jager, Ivan 1
 Jayaraman, Umarani 221
 Jha, S. 158
 Jinwala, Devesh 258

 Kang, Min Gyung 1
 King, Dave 56
 Kleiman, Dave 243
 Kruegel, Christopher 203
 Kulkarni, Janardhan 26

 Laskov, Pavel 188
 Le, Duc-Phong 116
 Liang, Zhenkai 1
 Liang, Zhiyao 86

 Mishra, Prasanna R. 154
 Mondal, Samrat 140
 Mounier, Laurent 41
 Mukkamala, Ravi 132

 Newsome, James 1
 Nitschke, Lukasz 102

 Pandey, Gireesh 154
 Patel, Dhiren 258
 Pernul, Günther 287
 Phithakkitnukoon, Santi 273
 Poosankam, Pongsin 1
 Prakash, Surya 221

 Rajan, Basant 303
 Ramesh, Raghavendra K. 26
 Rieck, Konrad 188

 Sahai, Amit 148
 Saxena, Prateek 1
 Sengupta, Indranil 235
 Siahaan, I. 158
 Smith, R. 158
 Song, Dawn 1
 Sprick, Barbara 26
 Sundhar R.S., Shyaam 243
 Sunitha, N.R. 156
 Sural, Shamik 140

 Tedesco, Gianni 173
 Thapliyal, Ashish V. 203
 Thomas, Julien A. 71

 Verma, Neelam 154
 Verma, Rakesh M. 86
 Vigna, Giovanni 203

 Wright, Craig 243

 Yin, Heng 1