TOWARDS MORE SECURE PROGRAM EXECUTION ENVIRONMENTS

SUFATRIO

(B.Sc., University of Indonesia,
M.Sc., National University of Singapore)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2010

Acknowledgments

First and foremost, all the praise and gratitude to Beloved True Source, our Source and Only Destiny, who always Loves and Blesses all beings so completely. May we all always embrace and accept Your Love and Will, which are the perfect and most beautiful ones ever. And may Your Name be exalted and glorified forever always.

I am very grateful to my supervisor, Associate Professor Roland Yap, for his continuous guidance, help and support throughout my Ph.D. years. I benefited very much from his vast range of knowledge on many areas of computer science, including operating systems, networks, and their related security aspects. The results reported in this thesis would not have been possible without his invaluable and constant support, and his set-by-example commitment to conducting research.

I am also much indebted to Professor Lim Hock for his generous support through Temasek Laboratories, NUS. With Temasek Laboratories, NUS, Professor Lim has always been very supportive in fostering an excellent environment on the University's campus for fruitful research in defence and security related areas.

I also would like to thank my team mates and friends on RISCI and VISCA projects: Wu Yongzheng, Rajiv Ramnath and Felix Halim. Working with them were inspiring, leveraging, and always enlightening. Thanks for the fruitful collaboration over these years. My thanks also go to my SoC and NUS friends: Dr. Andrew Edward Santosa, Dr. Li Qiming, Dr. David Lo and Dr. Vivy Suhendra. Special thanks to Dr. Zeyar Aung for his careful proofreading on the draft of this thesis. I also would like to sincerely thank the Administration and HR teams of Temasek Laboratories, NUS. They were always there to assist whenever I needed help.

My thanks beyond words go to my family for their continuous great love and support throughout my life. Special thanks to my wife, Elizabeth, and my baby son, Mike, for their love and support. Thanks and I love you all.

The support of DSTA and Temasek Laboratories NUS through RISCI and VISCA research grants are gratefully acknowledged. The excellent research facilities of School of Computing, National University of Singapore are also greatly appreciated.

With gratitude to Beloved True Source, Who always Loves all beings so completely.

Our love for You...

Contents

Summary

The increasing prevalence of cyber attacks is a worrying trend in the Internet age. By exploiting vulnerabilities in operating systems or applications, intruders are often able to circumvent the existing security mechanisms. This thesis proposes measures and infrastructure to provide more secure program execution environments so as to enhance host security. Our approach is based on securing the "Program Protection Life Cycle (PPLC)", which protects application programs throughout their life cycles against attacks, including zero-day attacks. A number of security mechanisms are proposed along the PPLC to substantially reduce attack vectors on a host.

Firstly, to mitigate the threat of zero-day attacks to a running program, we investigate a system-call monitoring Intrusion Detection System (IDS) which aims to detect any potential anomalous behavior of the execution. Using an automated attack generation approach, we show how a non-parameterized Self-based IDS model is vulnerable to mimicry attacks. We then propose an improved IDS model to mitigate mimicry attacks by employing a privilege and argument abstraction technique. We also move on to propose a general framework based on a notion of "attack-space search" to demonstrate how the attack construction approach can apply to various IDS models. The framework is then used to measure the resistance level of IDSs against attacks targeted on them.

Secondly, to secure program invocations on a host, we propose a lightweight executable authentication scheme which provides secure program distribution and integrity assurance on the invoked program. This is complemented by an automated vulnerability management scheme, which is aimed at performing automated vulnerability checks on operating system components and application programs to ensure vulnerability-free executions.

Thirdly, we address a supporting infrastructure which is needed to provide an efficient and secure program distribution and associated usage. Since existing Public Key Infrastructure (PKI) certificate revocation mechanisms are not sufficiently lightweight and timely, we propose two lightweight and practical near real-time revocation schemes. Our schemes are based on the use of the recently available Extended-Validation Certificate infrastructure, and can offer timeliness guarantees on the order of minute(s) with low performance cost. We also propose a formalism to reason with PKI-based systems and protocols by enhancing BAN Logic to deal with modern PKI-based protocols.

In summary, the contribution of this thesis is to give additional layers of protection, which give greater assurances of secure program executions amidst the increasing malware threats in today's Internet-connected systems.

List of Tables

List of Figures

List of Algorithms

List of Notations

List of Abbreviations

CA	Certificate Authority
CMAE	Certificate Management Ancillary Entity
COTS	Commercial Off The Shelf
CPE	Common Platform Enumeration
CRL	Certificate Revocation List
CRS	Certificate Revocation Status
CRT	Certificate Revocation Tree
CSI	Certificate Status Information
CVE	Common Vulnerabilities and Exposures
DoS	Denial of Service
EV	Extended Validation
EVC	Extended Validation Certificate
FSA	Finite State Automaton
IETF	Internet Engineering Task Force
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
LFC	Locality Frame Count
MAC	Message Authentication Code
MHT	Merkle Hash Tree
NVD	National Vulnerability Database
OCSP	Online Certificate Status Protocol

OS	Operating System
OSVDB	Open Source Vulnerability Database
OVAL	Open Vulnerability and Assessment Language
PAC	Privilege and Argument Categorization
PC	Program Counter
PDA	Push Down Automaton
PDF	Probability Density Function
PKI	Public Key Infrastructure
PPLC	Program Protection Life Cycle
SCAP	Security Content Automation Protocol
SDL	Security Development Lifecycle
SSL	Secure Sockets Layer
TCG	Trusted Computing Group
TLS	Transport Layer Security
TPM	Trusted Platform Module
TSA	Time Stamping Authority
UAC	User Account Control

Chapter 1

Introduction

Malicious software (malware), which is designed to infiltrate a system and cause damage to it, is a critical threat today. A report by F-Secure [45] indicates that there was as much malware produced in 2007 as in the previous 20 years altogether. A report by OECD [131] additionally highlights a worrying trend that malware has now evolved from occasional exploits to a global multi-million dollar criminal industry, and is threatening the Internet economy. On the other hand, computer systems, particularly software, are much more susceptible to attacks now than in the past. Hoglund and McGraw [68] attribute three factors to the present-day software security problem, namely: increasing connectivity, complexity and extensibility.

Despite numerous built-in security measures deployed by modern operating systems (OSes), the growing number of successful attacks shows that quite often intruders are able to circumvent the measures by exploiting flaws in the OSes or applications. To mitigate this situation, additional measures that can be deployed on top of the existing OS security mechanisms are needed in order to harden the OS and provide an increased protection. In addition, security infrastructure that critically supports host security mechanisms, such as Public Key Infrastructure (PKI), must be reliably available.

1.1 Securing Program Execution Environments

The central theme of this thesis is how to enhance host security by providing more secure environments for program execution. This thesis focuses on a concept of protecting software, i.e. programs, to ensure that they run as intended without violating host security. In particular, we focus on the challenges posed by commercial-off-the-shelf (COTS) software where no source code is available. Our aim is to safeguard software, from their distribution to their execution, from a variety of attack by the attackers.

We secure a program's execution environments based on our model of "*Program Protection Life Cycle (PPLC)*". The final objective of PPLC is to establish a belief on host H concerning "*good intended execution*" property of a program P. That is, a host H believes that program P performs the operations as intended by its (trusted) developer D, and that the operations will not violate H's security policies.[1] Figure 1.1 shows the overall components and steps of the PPLC model.

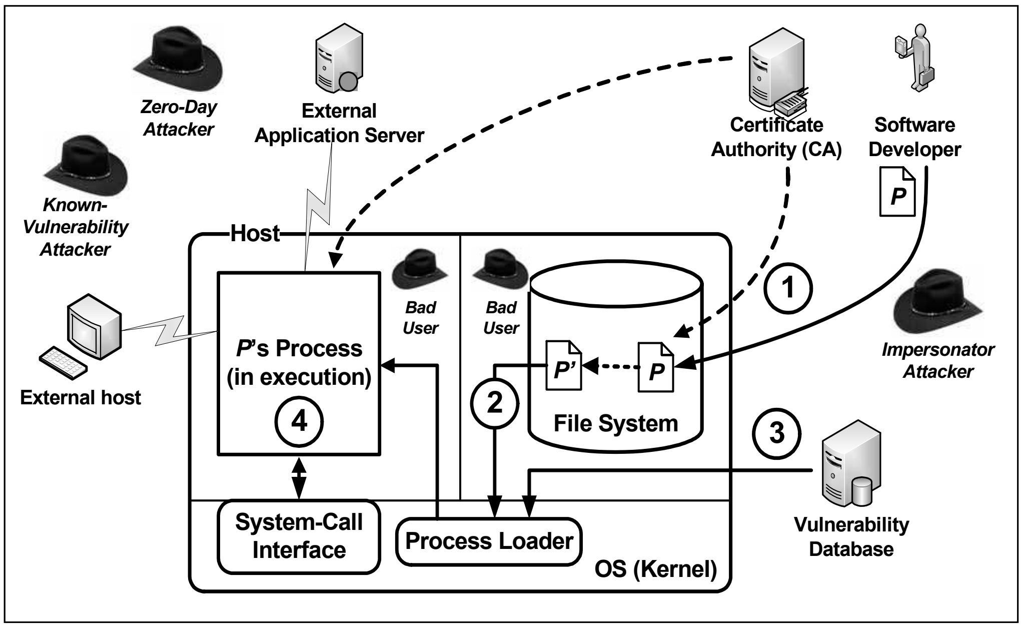

Figure 1.1: *Program Protection Life Cycle*: securing a program and its execution.

The four main steps of PPLC are:

1. **Secure program distribution**. This step is to give assurance to host H that program P originates from software developer D, and it is received unmodified. This goal can be achieved using public-key based signing on the program. Given a strong cryptosystem used, the security thus relies on ensuring that the public and private key pair of D is still valid (unrevoked) when H verifies P. By installing P into its system at time $t_{installed}$, host H implicitly agrees that P is a *good program*[2] which if it is run, it will not violate H's security policies. Hence, besides its content, P actually also carries with itself a belief statement that it is a good program.

2. **Preserved program integrity for execution**. To protect system security, it is imperative to ensure that both the *content* and *pathname* of P at time $t_{invoked}$ (i.e. just prior to its execution by the OS) is the same as those at time $t_{installed}$. In other words, P's location and content must stay intact on the file system of H.

[1]Chapter 8 discusses logic systems, such as Burrows-Abadi-Needham (BAN) Logic [26, 27], which deal with beliefs pertaining to security properties on network protocols and computer systems.

[2]More specifically, by "*good program*", we mean that the program is non-malicious in nature, and has no intention to violate any security policies of the target hosts (beyond the program's known functionalities) or any acceptable use policies.

3. **Vulnerability-free program execution**. Despite the deployment of numerous security measures, a host becomes highly susceptible to attacks when executing P with publicly known vulnerabilities during its "*unpatched time interval*". This interval spans from the time when a vulnerability alert affecting P is published until patches are applied or other preventive/corrective measures are taken by users. Given the increasingly high volume of reported vulnerabilities in recent years, the advantage is clearly with the attackers. To ensure that P runs as intended without any compromise due to attacks exploiting known vulnerabilities, automated vulnerability checking on P prior to program execution is important.

4. **Intrusion-free (unaltered) process execution**.

   Although P is believed to be good for execution, it still needs to be protected from possible attacks exploiting previously unknown vulnerabilities, which are also known as zero-day attacks. One useful mechanism to deal with this is anomaly detection Intrusion Detection System (IDS), which has an ability to detect novel attacks by flagging any abnormalities in program behavior. Online IDS systems which are well-integrated into the OS kernel can also block any suspected operations from being executed, thus preventing any potential security breach on H.

Safeguarding the four PPLC steps brings two main benefits to host security. Firstly, it reduces or even possibly eliminates the need for some defense mechanisms, such as anti-virus scanning on the executables, or network-packet scanning for known attack patterns. Secondly, it naturally deals with multi-modal attack entries. Program integrity protection, for instance, prevents file modification or addition either due to social engineering techniques, illegal user's operations, or various malware activities. Vulnerability checking also prevents vulnerable programs from being executed, therefore inhibiting various modes of attacks attempting to exploit the vulnerabilities.

In view of PPLC's critical role in ensuring secure operations of systems and applications, we position it to complement secure software development methodologies such as Microsoft's Security Development Lifecycle (SDL) [112]. While secure software development methodologies (and the associated techniques) aim to help software developers to reduce the presence of vulnerabilities in software products, the PPLC aims to ensure that the installed software products on deploying hosts run securely as intended.

Based on the PPLC model, in this thesis we thus argue the following statement:

> *"Security measures to protect the program life cycle are important to establish, and that they will reduce many attack vectors on a host and ensure more secure program executions on that host. With the right approach and techniques, these measures can be deployed with acceptable performance cost while substantially increasing host security."*

1.2 Challenges in Securing Program Protection Life Cycle

There are a number of challenges faced in securing the PPLC as well as providing the required infrastructures to support it. We mention several main challenges, which have provided motivations to our work reported in this thesis.

1.2.1 Difficulty in Evaluating the Security of IDS

In IDS research, it is common for a proposed IDS to be shown, usually using several known attacks and available dataset(s), for being able to detect the attacks with a high detection rate while exhibiting a low false positive rate. The question of *how secure* the IDS against attacks on itself is usually overlooked. A "smart attacker", who knows all the IDS internals, can exploit IDS limitations to craft attacks targeted on the IDS. One example highlighting this issue is the possibility of mimicry attacks [180, 164] on a system-call monitoring IDS called Stide [67, 152, 153]. Hence, one step towards IDS security analysis is finding a systematic way to measure the *resistance level* of an IDS against attacks targeted on itself. The derived *attack generation time* is particularly useful to estimate how fast an attacker can come up with a stealthy exploit given a newly discovered vulnerability on the protected program.

1.2.2 Making Anomaly Detector IDS More Secure

Given possible attacks on anomaly detection IDSs such as Stide IDS [67, 152, 153], the question to be asked is how we can improve the IDS, as well as other IDSs sharing a similar basic model (e.g. [146, 48]), to prevent the attacks. One approach is to make use of all available information from the observable events, such as system call arguments and user credential information. While increasing the IDS robustness against attacks, the inclusion of these new details should not prohibit fast (preferably online) detection operations, or increase the false positive rate.

1.2.3 Practicality of OS-based Executable Authentication System

The concept of executable integrity, i.e. checking the integrity of executable codes stored on a host, is not new. Tripwire [82] is one of widely known systems that provide (offline) file integrity protection. In securing the PPLC, we are more concerned with ensuring *data origin authentication* on executable codes, i.e. giving assurance to a party executing a code of the identity of the party which originated the code. Data origin authentication on an executable code implicitly provides data integrity since, if the code was modified during transmission or while stored on the host's file system, the sender of the code would no longer be the originator [109]. There are a number of OS-integrated executable authentication implementations, although mostly on Unix [9, 183, 175]. Despite the potential great benefits brought, current commercial OSes, such as Windows and Solaris, do not generally come with built-in support for executable authentication yet. One challenge here is to investigate how to devise a mandatory in-kernel executable authentication scheme, conducted just prior to program execution, in a lightweight manner on a real-world complex OS like Windows.

1.2.4 Automating Vulnerability Alert Processing

In order to keep track of security alerts on relevant OS and applications, system administrators usually rely on various sources of narrative vulnerability alert repositories. Given the speed at which an exploit becomes available once a vulnerability is known, and the frequency of occurrence of such alerts, manual intervention is too slow, time-consuming and possibly ineffective. It is therefore a real challenge on how to develop a coordinated vulnerability database which is designed to be machine readable and processable. This will allow automated processing of alerts and corresponding vulnerability scanning to be done on a host. One particular issue is that the scanning process should provide assurance that it does not perform any operation beyond what is necessary. Thus, the scanner operations must be minimal and open to inspection.

1.2.5 Providing a Lightweight and Near Real-Time Certificate Revocation Service

Many host security mechanisms, including validating a digitally-signed program, rely on ensuring that the signer's public key is not revoked. In X.509-based PKI, certificate revocation is mainly supported by a mechanism called Certificate Revocation List (CRL) [37]. However, CRL is widely perceived to be costly [96, 64]. Several alternatives have been proposed, such as Online Certificate Status Protocol (OCSP) [124] which requires the Certificate Authority (CA) to respond to any incoming query in real time with a signed message. To secure software distribution and PKI-based program's interaction with external hosts, a certificate revocation scheme must provide a *near real-time* timeliness guarantee and enable fast computation on the verifiers. This is particularly important given the proliferation of lightweight computers and mobile devices. In addition, the scheme must not put excessively high overheads on any of the entities involved, such as the CA or the revocation repository, which will create undesirable service bottlenecks.

1.2.6 Concise yet Practical Formal Reasoning on PKI-based Protocols

Nowadays, it is common for many programs running on a host to interact with external entities using PKI-based operations. PKI-based protocols, including those used in secure program distribution and certificate revocation management, must be shown to be secure. Designing a correct protocol specification is however well recognized as a difficult task. A formal protocol analysis is therefore necessary to establish the security of a protocol. Although protocol analysis on PKI-based protocols does not constitute a step in the PPLC, it is however an important prerequisite for securing the PPLC due to the assurance needed on both PKI-based application programs and infrastructure supporting the PPLC. Moreover, such analysis is usually not a focus of most secure software development methodologies such as [112]. Despite the availability of numerous formal techniques, one challenge is to devise techniques which can be handily utilized to verify real-world protocols by many protocol designers. One approach is by leveraging the popularity of widely-known techniques such as BAN Logic [26, 27]. BAN Logic however does not properly deal with the detailed aspects of PKI-based authentication, such as certificate processing, as commonly practiced nowadays. Given the ubiquity of PKI-based protocols and their importance to host security, there is a need to update such logic to be more concise yet remain practical to use.

1.3 Contributions

Based on the PPLC model and various security issues faced in securing it, we have proposed a number of schemes to help ensure more secure program environments. Most of the results here have been reported in the publications [158, 66, 186, 160, 159]. Appendix F gives a list of the author's published as well as submitted works throughout his Ph.D. candidature. The contributions of this thesis can be summarized as follows.

- Using our notion of *pseudo subtrace* construction, we demonstrate the security weakness of the Self-based IDS (Stide) [67, 152, 153], which compares the system call trace of a process with a set of k-grams (see Chapter 3). We then outline an efficient algorithm that transforms a previously detectable attack into the one that will pass the IDS detection. Using a number of real-world vulnerable programs, we show that mimicry attacks can be easily constructed with execution times of at most a few seconds, even when a relatively large window size is used. A variant model using a more precise *graph* profile representation can also be easily attacked. Based on this automated attack generation approach, we then propose a general framework which shows how the attack construction can be applied to various IDS models, including a Finite State Automaton (FSA) based IDS model [146]. This allows us to find out how computationally expensive it is to perform a search to craft attacks on the IDS. This result shows the feasibility of obtaining quantitative measurement on IDS resistance against targeted attacks.

- We also propose an extension to the Self-based IDS to make it more resistant against mimicry attacks by making use of the system call arguments and process credentials (see Chapter 4). The proposed data-flow technique can be easily combined with other system-call based IDS techniques. To avoid increasing the false positives, a user-supplied specification is used to abstract the arguments and credentials. This specification takes into account security semantics of the operations and their potential effects to system security, and highlights the occurrence of "dangerous" operations. With this simple extension, we show that the robustness of the IDS is increased. We demonstrate that the enhanced IDS provides resistance to attacks previously successful on the Self-based IDS as well as a variety of common attack strategies. We also have some evidence that the increase in detection accuracy does not lead to more false positives. Unlike other proposed data-flow IDS techniques [91, 123, 167, 23], our IDS enhancement using system call arguments highlights the virtue of incorporating the host's *security model* into the IDS model to result in a more concise and robust IDS.

- In the area of software authentication protection, we demonstrate the practicality of a mandatory, in-kernel, pre-execution executable (binary) authentication mechanism in Windows (see Chapter 5). We first analyze numerous design factors for a binary authentication system and compare a variety of existing systems. We then present a prototype, BinAuth, which exemplifies a lightweight binary authentication scheme and its integration within an OS. The main contribution of this work is that we believe that it provides the first comprehensive infrastructure for trusted binaries in Windows. It provides mandatory authentication for all binaries in Windows, and goes beyond authenticated code in XP and Vista. This is significant given that much of the problems of security on Windows stems from a host's inability to distinguish between trusted and untrusted software. With BinAuth, binaries are allowed for execution only if they come from developers that the host trusts, and that the binary's contents as well their file properties were not illegally modified. Our benchmarking shows that the overhead of the scheme can be quite low, around 2%, with a caching strategy. In addition, we also leverage the authentication with a Software_ID scheme which is useful for easier software management and automated vulnerability assessment.

- We propose a framework for machine-oriented vulnerability database geared towards automated vulnerability checking on a host (see Chapter 6). We developed a prototype sample scanner for Unix/Linux systems tailored for Red Hat Linux and FreeBSD, although the basic principles are applicable to other OSes. Our work was developed when vulnerability representation and automated processing were still not widely addressed by the security community. Our results are in line with present on-going standardizations efforts on vulnerability processing by the U.S. Government-sponsored organizations, such as CVE [116], OVAL [117], CPE [115] and SCAP [126]. Given these latest standardization efforts, we also address how these standardization results still need to be extended in order to realize the fully automated vulnerability processing as envisioned in our proposed framework.

- We also propose two lightweight, practical and inherently-distributed certificate revocation schemes, which are based on the recently available Extended-Validation (EV) certificate infrastructure (see Chapter 7). Our schemes enhance CRS/NOVOMODO [110, 111] and OCSP [124] to support efficient revocation with near real-time timeliness guarantees (1–10 minutes). Using a more realistic cost-evaluation model derived from empirical data, we demonstrate that it is feasible to keep the overheads manageable on all the involved entities, while maintaining lightweight requirements on the verifier even under a timeliness guarantee of one minute. The scheme has an additional advantage in that status queries are protected from external parties including the CA, which may be important for privacy reasons. Another contribution is our development of a more realistic cost-analysis framework for evaluating revocation schemes. Our proposed framework follows the approach taken in [99, 72] which makes use of empirical revocation data. In addition, our framework incorporates a number of novel aspects, including: a more accurate calculation of CRL data size over [99, 72] (which also works for fine-grained certificate generation and CRL issuance), a more realistic query model on revoked certificates, and the derivation of query probability on revoked and valid certificates.

- Finally, we propose various public-key related enhancements to BAN Logic [26, 27] to allow for more concise reasoning on PKI-based protocols (see Chapter 8). Our extension is along the lines of the work by Gaarder and Snekkenes [51] but better captures the current aspects of PKI. In particular, our extension redresses the reasoning on the goodness of private keys, and considers certificate revocation. We also address common pitfalls in public-key based protocol design due to insufficient attention placed on the "intended recipient" and "stated sender" of a message. Our extension makes the recipient and sender explicit, and these requirements are incorporated into the logic. As a result, our logic reduces the likelihood of allowing such flaws in the protocol and the corresponding formalism. In addition, our logic also deals with cases of (public-key) signed encrypted message and encrypted signed message. Examples and notes on the usage of our logic are also given. We additionally apply our logic to analyze a session establishment protocol, which is employed in our proposed certificate revocation scheme.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 gives the background information on various related problems addressed in this thesis. Chapter 3 presents our automated attack algorithm on the Self-based IDS and the framework for analyzing IDS resistance level. Chapter 4 proposes an enhancement to the Self-based IDS using system call argument and privilege abstraction technique. Chapter 5 outlines our lightweight, in-kernel software authentication protection scheme. In Chapter 6, we expound the automated vulnerability alert processing scheme. We present our lightweight and practical certificate revocation schemes in Chapter 7. Chapter 8 elaborates our extension to BAN Logic for reasoning with PKI-based protocols. Finally, Chapter 9 gives a summary on what has been achieved in this thesis and points out possible future work.

Chapter 2

Background

In this chapter, we give some background on related aspects of system and network-based infrastructure that support secure program environments. This chapter can be skipped given the familiarity of the subjects, or its parts can just be referred to as necessary.

2.1 Intrusion Detection Systems

2.1.1 Overview and Motivation

Intrusion detection is defined as the process of monitoring events occurring in a computer system or network and analyzing them for signs of possible *incidents*, namely violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices [140]. An Intrusion Detection System (IDS) is a device or software that automates the intrusion detection process. Intrusion Prevention System (IPS) has all the capabilities of an IDS, and can also attempt to stop possible incidents. Prevention features in IPSs can usually be disabled by system administrators, causing them to function as IDSs. Since this thesis focuses on intrusion detection mechanisms, for ease of discussion, the term IDS is used throughout this thesis to refer to both IDS and IPS.

IDSs represent a system's second line of defense, which are aimed to provide additional layer of security protection. The adoption of IDSs is motivated by the following observations and trends [89], which have led many experts to belief that a computer system even with all its attack prevention measures will never be absolutely secure [20]:

1. Statistics show that the number of reported vulnerabilities has increased greatly in the past decade [30].
2. Meanwhile, the number of produced malware is continuously increasing [45].
3. Administrators are generally very slow in applying fixes to vulnerable systems [137].

4. There is an increasing concern on zero-day attacks to computer systems. Unreleased zero-day exploits even now have formed a “zero-day market” [114].

For an introduction to IDS, we recommend [80, 40].

2.1.2 IDS Classification

In this section, we give a very short overview of IDS with respect to the classification of IDSs. A comprehensive survey and taxonomy of IDSs can be found in [14, 148, 98].

IDSs are commonly classified by their *audit (event information) source location* and *detection method*. Based on the audit source location, the IDSs are classified into:

- **Host-based IDSs:** which deal with audit data generated on a single host.
- **Network-based IDSs:** which monitor network traffic.

Based on the detection method, they can be broadly classified into:

- **Misuse detection IDSs:** which model known attacks using attack patterns (also called *signatures*), and detect them by means of pattern matching. The benefit of these IDSs is a high degree of detection accuracy, while their main drawback is the inability to identify new attacks.
- **Anomaly detection IDSs:** which define what is the “normal behavior” on the system (i.e. *normal profile*), and flags any deviations from normal as potential attacks. They thus operate based on the assumption that all anomalous activities are malicious [89]. Anomaly detection IDSs, also sometimes called *anomaly detectors*, are capable of detecting novel attacks. However, there are some challenges such as defining accurate “normal” behaviors and keeping a low false positive rate. In fact, these challenges are not unique only to the anomaly detection IDSs, but represent known issues in the *anomaly detection problem* (see [32] for a comprehensive survey of the problem). Patcha and Park [132] gave a recent survey of anomaly detection techniques used specifically for intrusion detection purpose.

2.1.3 IDS Effectiveness Metrics

The effectiveness of an IDS is usually measured in terms of detection (true positive) and false positive rate. Functioning as a classifier on a binary decision problem, an IDS determines examples as either positive (intrusive) or negative (non-intrusive). The decision can be represented in a structure known as a confusion matrix shown in Table 2.1.

The confusion matrix has the following four cases, and makes the four corresponding sets:

- *True Positives (TP)*: refer to intrusive behaviors correctly determined as intrusion.

		Actual Condition	
		Intrusive Behavior	**Non-Intrusive Behavior**
Detection	**Intrusion Alarm**	*True Positive (TP)*	*False Positive (FP)*
	No Alarm	*False Negative (FN)*	*True Negative (TN)*

Table 2.1: Confusion matrix comparing actual intrusive condition and detection result.

- *False Positives (FP)*: are non-intrusive behaviors incorrectly labeled as intrusion.
- *False Negatives (FN)*: represent a failure of an IDS to detect actual intrusions.
- *True Negatives (TN)*: when no intrusive behaviors have taken place and no alarm is raised by the IDS.

In a benchmark measurement scenario, based on the sizes of four defined sets, we can determine the following metrics [13] (let I and $\neg I$ denote intrusive and non-intrusive behaviors respectively, and let A and $\neg A$ denote the presence or absence of an alarm respectively):

- *True positive (detection) rate*: is defined by the conditional probability $P(A|I) = \frac{|TP|}{|TP| + |FN|}$, where $|\ |$ denotes the size of a set.
- *False positive rate*: is defined by the probability $P(A|\neg I) = \frac{|FP|}{|FP| + |TN|}$.
- *False negative rate*: is $P(\neg A|I) = 1 - P(A|I) = \frac{|FN|}{|TP| + |FN|}$.
- *True negative rate*: is $P(\neg A|\neg I) = 1 - P(A|\neg I) = \frac{|TN|}{|FP| + |TN|}$.

2.1.4 IDS and Alert Correlation

Alert correlation has recently attracted the attention of the IDS community as an extension to intrusion detection. It is defined as a process that takes as input the alerts produced by one or more IDS sensors, and provides as output a more succinct and high-level view of occurring or attempted intrusions. See [89] for a survey on alert correlation.

2.2 System-Call Monitoring IDSs: Self-based IDS, Attacks, and Related Models

In this part, we provide some background on system-call monitoring IDSs, particularly the Self-based IDS which is the focus of Chapters 3 and 4. A recent paper by Forrest et al. [50] gives a comprehensive survey on the development of system-call monitoring IDSs as inspired by the Self-based IDS.

2.2.1 Self-based IDS

Inspired by how the natural immune system distinguishes "self" from "other", a seminal work [67] proposed a host-based anomaly detection IDS based on an observation that *short sequences* generated from a program's system call traces can serve as a stable indicator of the program's behavior. Monitoring system calls has been recognized as an effective mechanism to observe what a program really does. System calls are the gateway to privileged kernel operations. Thus, by monitoring and restricting them, a running program can be observed and (if necessary) be prevented from violating the host's security policy.[1] The works [152, 153] expanded the IDS with an automated response to a detected intrusion. For ease of reference, in this thesis we will call this type of IDS a *Self-based IDS*.

The Self-based IDS takes an unparameterized system call trace of a program as its input. A system call trace can be viewed simply as a string where the alphabet consists of all the possible system calls in the underlying OS (∼200 for Unix/Linux systems). The IDS defines normal behavior of a program in terms of short k-grams of system calls. A k-gram is a sequence of system calls with length k. Conceptually, we define a small fixed size (k) window and then "slide" it over each normal trace. The parameter k is thus usually known as the *(sliding) window size*. Every unique system calls within the sliding window, i.e. k-gram, is recorded into the normal profile database. Sliding-window sizes of 6–8 are commonly used in practice [67, 152, 182].

Previous works on Self-based IDS make use of a few different techniques for representing a normal database entry, namely *full sequence*, *(forward) lookahead pair* and *backward lookahead pair* technique. The Self-based IDS using full sequence representation is also known as *sequence time-delay embedding* (Stide). It was first proposed in [67], and is later analyzed in a widely cited performance comparison work [182]. The Stide model has been widely used for performance analysis mainly due to its better discrimination power than the lookahead pair methods. However, a recent work comparing Stide and the backward lookahead pair method [74] shows that the latter exhibits lower false positives. Notwithstanding this, in our attack construction in this thesis, we mainly consider Stide since it is theoretically a more resistant model against mimicry attacks [74, 182].

Experiments were conducted in [67] to evaluate the effectiveness of the Self-based IDS. In [182], Warrender et al. reported the comparison results of the Self-based IDS and several system-call analyzing IDSs including an IDS based on data mining and Hidden Markov Models. Although no definite conclusion can be made, the results show that the Self-based IDS performs reasonable well despite its relative simplicity.

[1]This system-call based monitoring approach cannot detect intrusions which *do not* invoke system calls. Hence, this approach performs simpler measures in ensuring the intrusion-free (unaltered execution) property (see Section 1.1) of a running program. In practice, however, security-relevant interactions typically take the form of system calls [180].

2.2.2 Mimicry Attacks on Self-based IDS

Despite the good experimental results in [67, 182], the security of the Self-based IDS did not receive much attention until later when Wagner and Soto [180] and Tan et al. [164] independently published their security analysis results on the IDS.

Wagner and Soto [180] used Finite State Automaton (FSA) as a framework for studying and evaluating mimicry attacks. They showed that a mimicry attack is possible because additional system calls which behave like no-ops can be inserted into the original attack trace so that the resulting trace is accepted by the automaton of the IDS model. Independently, Tan et al. [164] showed attack construction as a process of moving an attack sequence into the IDS detection's *"blind region"* through successive attack modifications. The focus in these two works is to demonstrate the feasibility of mimicry attacks. However, they do not take a detailed look at designing and constructing automated attacks.

In a later work, Gao et al. [53] performed a study of the "black-box" Self-based IDS and also several "gray-box" IDSs such as [146, 48].[2] They investigated mimicry attacks with window sizes of up to 6 and showed the existence of mimicry attacks across the methods and the window sizes studied. However, they did not go into the details of attack generation. Later works, including the more recent mimicry attack construction techniques [88, 58], are surveyed in Section 3.1 when we discuss our own automated mimicry attack construction.

2.2.3 Improved System-Call based IDSs

Due to possible attacks on the basic Self-based IDS, a variety of improved IDS models have been proposed. Within the context of black-box and gray-box anomaly detectors, there are two main approaches, namely:

- **Incorporation of additional execution context**. Two main examples of this approach are [146, 48]. In [146], Sekar et al. proposed the use of FSA-based IDS which is generated by observing both system calls and program points during the normal program executions. A *program point* is the Program Counter (PC) at the point from where a system call is made. A stack traversal mechanism is used to recover the PC within the program segment where a system call is actually invoked. Each distinct PC is made as a state, whereas system calls are used as the labels

[2]We follow the terminology in [53] which classifies various system-call monitoring anomaly detectors into "black-box", "gray-box" and "white-box" detectors. Black-box detectors (e.g. [67, 152]) do not acquire any additional information other than the system call number and arguments. In contrast, white-box detectors (e.g. [179, 47]) examine all available pieces of information including those acquired by statically analyzing (and potentially modifying) the source or the binary codes. Gray-box approaches (e.g. [146, 48]) lie in between: the anomaly detector does not utilize static analysis of the program, but does extract additional runtime information.

for transitions. The FSA can then be used to monitor the program execution in an online fashion. If an error occurs while performing the stack traversal mechanism, or if a state or transition does not exist, there may be an anomaly. Since the FSA model is deterministic, the efficiency is high. It, however, allows the possibility of false positives as some legal transitions or states may never occur during training. In addition, it also suffers from the impossible path problem [48]. Feng et al. [48] then proposed the use of call stack information obtained through runtime monitoring in addition to the program's system call trace. Later works [53, 88], however, showed that attacks which will escape the detection of this IDS model can still be constructed.

- **Data-flow approach**. Examples of IDS models incorporating data-flow analysis which can be used to improve the Self-based IDS against mimicry attacks are [91, 123, 167, 23]. We discuss more about these models when we present our work on improving the Self-based IDS in Chapter 4.

A survey paper [50] also mentioned other works making use of static analysis and other observables, which are beyond the scope of this thesis.

2.3 Software Authentication Protection

Chapter 5 addresses the problem of assuring data origin authentication on a piece of software (i.e. executable code) prior to its execution on a host. For ease of reference in this thesis, we refer to an executable code stored on the file system as a *binary*. Data (i.e. binary) origin authentication implicitly also provides data integrity since, if the binary was modified during transmission or while stored on the host's file system, the sender of the binary would no longer be the originator [109]. Note that on a host, a binary is associated with an entry on the host's file system. The goal of a binary authentication system is thus to ensure that an executed binary only comes from a developer that the host trusts, and that the binary's content as well its properties on the file system were not illegally modified. Below, we provide some background on executable authentication problem, and survey several existing systems aimed to ensure executable authentication.

2.3.1 Executable Authentication Problem

The first step in securing the PPLC is to establish a secure program distribution. Hence, all installed binaries must be shown to be originated from a developer that a host trusts, and that they are received unmodified. This is particularly relevant nowadays since a large proportion of software packages are downloaded over the Internet, which is a public untrusted network. Thus, only trusted binaries should be kept on a host's file system.

This is however still insufficient to ensure that only trusted binaries are allowed for execution on the host. A modern OS has numerous built-in security measures designed to prevent illegal operations on a system, including any illegitimate addition or modification of executable codes on the file system. However, due to various vulnerabilities, an OS component or an application program can be attacked. Once an attacker succeeds in compromising a system, the next step is usually to install/modify executable code(s) on the victim's file system. This could be done for several reasons: to install a backdoor, to plant a spyware, or to use as a step for subsequent privilege-escalation attacks. A mechanism to deal with illegal addition or modification of binaries on a host's file system is thus required. This would also help prevent social engineering attacks which attempt to trick a user to install an application software package that illegitimately replaces important system libraries. Besides protecting application programs, binary authentication is also beneficial to protect the OS against illegal loading of malicious drivers into its kernel.

2.3.2 Overview of Existing Authentication Systems

Although the concept of binary authentication protection is not new, it is not commonly incorporated as a standard in-kernel mechanism in popular commercial off-the-shelf OSes, such as Windows and Solaris. Below, we will briefly mention some existing systems as well as other security approaches related to providing software authentication protection.

Tripwire [82] is one of the first to provide file integrity protection.[3] However, Tripwire is a user-mode application program. It checks file integrity in an off-line manner, and does not provide any mandatory form of integrity checking.

There exist a number of kernel-level binary authentication implementations. These are mainly for Unix, such as DigSig [9], Trojanproof [183] and SignedExec [175]. They modify the Unix kernel to verify a binary's digital signature before executing the binary. For efficiency, DigSig employs a caching mechanism to avoid checking binaries which have already been verified.

Partly motivated by the emergence of mobile code (e.g. Java applets), some OSes extend the concept of signed code to standard binary types. They check the validity of a binary's signature particularly if it is downloaded from an untrusted zone like the Internet. Windows has a technology called Authenticode [61]. In Windows XP and earlier versions, Authenticode alerts the users of the results of signature verification under a few situations. However, it is not mandatory, and can be bypassed. Windows Vista User Account Control (UAC) adds binary signature checks in some special cases, but it only deals with EXE binaries. It is also limited to privilege escalation situations. One common drawback of existing Windows mechanisms is that they do not authenticate the binary's pathname.

[3]A *file integrity protection* system ensures that a file on a host's file system is not illegally modified. Such a system may provide only data (i.e. file) integrity assurance, and no data (i.e. file) origin authentication assurance.

Moreover, they always perform digital signature verification when verifying a protected binary. Besides the incurred performance overhead, a certificate revocation service is thus always required. Hence, they may be too heavy for an online (pre-execution) program integrity checking mechanism, particularly on lightweight computers.

There is also the Trusted Computing initiative by the Trusted Computing Group (TCG) [173]. The Trusted Platform Module (TPM), promoted by the TCG, is a certified hardware in which all the basic trusted operations and the cryptographic functions are securely handled. It thus constitutes the "root of trust" aimed to provide an uninterrupted "chain of trust" which starts at the system's startup up to the application's executions. TPM can therefore facilitate a secure startup service on a host, thus ensuring the integrity of the OS kernel. This is important since most binary authentication systems including BinAuth require the OS kernel to be trusted. In addition, an authentication system like BinAuth can take advantage of TPM by having the latter securely store keys and authentication data.

The concept of layered integrity protection, which aims to ensure the integrity of the overall system starting from the lowest level (hardware) to the highest level (application), was analyzed among others by Arbaugh [10]. In the layered model described in [10, page 54], an in-kernel authentication scheme deals with integrity assurance at level 5 (user applications) and part of level 4 (some components of the OS, e.g. kernel drivers).

2.3.3 Authentication Issues in Microsoft Windows

Most works on binary authentication are on Unix/Linux [9, 183, 175]. The problem of malware is however more acute in Microsoft Windows. Windows is perhaps the most commonly exploited commercial OS. This is partly due to its sheer complexity, which leads to a high number of security flaws. Below, we briefly mention the complexities and special problems of Windows which make it more difficult to implement binary authentication than in other OSes such as Unix. This is to highlight issues which need to be taken into consideration when establishing a binary authentication system on Windows.

Windows NT (including later Windows versions based on it such as Server 2000, XP, Server 2003, and Vista) is a microkernel-like OS. Programs are usually written for the Win32 API, but these are decomposed into microkernel operations. Windows is closed source — only the Win32 API is documented but not the microkernel API. As a result, it is not possible for an enhancement work related to Windows kernel infrastructure to make any guarantees on the completeness of the security mechanisms.

Some specific issues in Windows related to implementing binary authentication are as follows [66, 186].

- **Proliferation of binary types:** It is not sufficient to ensure the integrity of `EXE` files alone. In Windows, binaries can have any file name extension, or even no extension. Some of the most common extensions include `EXE` (regular executables), `DLL` (dynamic linked libraries), `OCX` (ActiveX controls), `SYS` (drivers), `DRV` (drivers) and `CPL` (control panel applets). Unlike in Unix, binaries in Windows cannot be distinguished by their execution flags.

- **Complex process execution and DLL loading:** In Windows, a process is created using `CreateProcess()` which is a Win32 library function. To load a DLL, a process uses `LoadLibrary()`. However, these two operations are broken up into several operations at the native API level. Hence, it is more complex to incorporate a mandatory authentication in Windows. We need to properly intercept the right API(s) with correctly intended operation semantics with respect to Windows' behavior.

Compared to other open platforms, Windows potentially also makes the issue of locating vulnerable software components more complicated. A great deal of binaries created by Microsoft contain an internal file version, which is stored as the file's meta data. The Windows update process, however, does not inform the user which files have been modified. Furthermore, the meta data of the modified file might remain unchanged. Thus, by merely inspecting the meta data of a binary, one cannot ensure whether the binary is still affected by a particular vulnerability. A "software naming" scheme, which associates binaries with unique names, can simplify the software vulnerability management. This issue is addressed later in Section 5.6.

2.4 Managing Host Vulnerabilities

2.4.1 The Problem of Host Vulnerabilities

A *vulnerability* is defined as a weakness in a system that allows the use of a product beyond its design intent with an adverse effect on the software, system, or data [49]. Security loopholes found in programs and OS thus lead to vulnerabilities in the system.

In recent years, the number of security vulnerabilities discovered in computer systems has increased explosively [30]. To keep track of known vulnerabilities, system administrators usually rely on vulnerability alert repositories/databases[4], such as CERT Coordination Centre Vulnerability Notes Database (now hosted by the US-CERT [174]), Securityfocus BugTraq [145] and Open Source Vulnerability Database [171]. Unfortunately, with the observed rate of alerts (e.g. 5,500 vulnerabilities reported in 2002), it has been estimated that it would take more than 200 days for an administrator to digest all the alerts published in that year alone [31, page 41]. Hence, a mechanism that requires

[4]Following a common practice in vulnerability management field, we refer to a vulnerability alert repository as vulnerability database although the alert entries are narrative and not structured in a well-defined database model, e.g. relational model.

human intervention to understand and deal with such alerts is too time consuming, and more importantly is too slow to respond to vulnerabilities in this day and age.

This situation, we argue, has contributed quite significantly to the noticed trend in the *Vulnerability Exploit Cycle* [94] shown in Figure 2.1. The Y-axis in the graph represents the number of incidents for a given vulnerability. It quite clearly shows the existence of *significant time lag* between the release of a vulnerability report and the decrease of incidents following corrective measures by users/administrators.

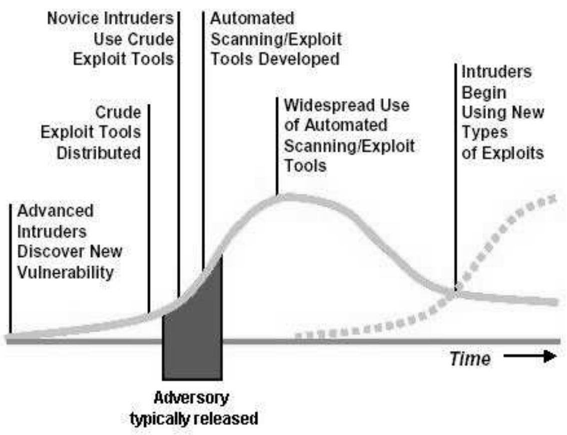

Figure 2.1: Vulnerability Exploit Cycle (from CERT Coordination Center [94]).

2.4.2 Vulnerability Assessment and Self-based IDS

A *vulnerability assessment tool* (also known as *vulnerability scanner*) tests to determine whether a network or host, which is generally called a target, is vulnerable to a set of known attacks [49]. For each vulnerability, it performs a *vulnerability check*, which is a set of items to check on a target that may reveal the presence of the vulnerability [49]. As acknowledged by many including [16], vulnerability analysis is a very powerful security management technique. However, it also has its own limitations, and is suitable only as a *complement* to using an IDS, *not as* a replacement. It thus works together with an IDS in protecting the PPLC as depicted in Section 1.1. In [16], Bace views that a vulnerability assessment can be treated as a special case of intrusion detection process. To better highlight the differences between a vulnerability assessment tool and the Self-based IDS (as an anomaly detector), we compile the comparison of the two systems in Table 2.2.

Vulnerability scanners can be broadly classified into two categories:

1. Network-based scanners: which probe a target machine remotely to find vulnerabilities. Widely known examples are SATAN [144] and Nessus [128][5]. These scanners

[5]Nessus is traditionally and still mainly a network-based scanner. Recent development however also equips it with an option to perform host-based assessments. Nonetheless, Nessus achieves this by making use of network-based access using system credentials as opposed to a software agent approach [41]. It logs on to Unix systems via the SSH protocol and to Windows systems via an NTLM network API.

Aspect	Vulnerability Assessment Tool	Self-based IDS
Goal	To detect vulnerable programs in a system	To detect whether an intrusion is taking place on a program
Scope of detection	Whole system (OS and application programs)	A program of interest
Information examined	System state, i.e. installed programs and environmental settings	System call trace invoked by the observed program
Reference data	Vulnerability alert entries from vulnerability database center(s)	Normal program profile (database of *k*-grams)
Type of analysis	Misuse detection	Anomaly detection
Timing	Interval-based (each execution makes a snapshot of the system's state)	Whole program execution
Output	To report whether there exist vulnerable programs in the system	To raise an alarm when anomalous behavior is observed
Monitoring agent	System information collector as part of the vulnerability scanner	System call interposition (reference monitor)
Main strength	Exhaustively test a system for large numbers of known vulnerabilities	Able to detect novel attacks or anomalous behaviors
Main drawback	Offer no protection for attacks in between scanning times and no ability to detect novel attacks	Must continuously observe the program and inspect its activity

Table 2.2: The comparison between vulnerability assessment tool and the Self-based IDS.

can help administrators automate the process of testing target systems for known vulnerabilities that can be exploited via the network. In its typical operation, Nessus begins by performing a port scan to determine which ports are open on the target, and then tries various exploits on the open ports.

2. Host-based scanners: which are installed and executed on the target host itself, such as COPS [46] or Ferret [147]. The host-based scanners have an advantage over network-based scanners as they can directly access the host's configuration information and various running services. They can also work together with a "scanning manager", which is deployed within the same administrative network domain. In this way, the scanners can be made as lightweight as possible. As such, we can mitigate the main drawback of host-based scanners, namely the need for installations or regular updates of the scanners on every target machine.

Our work reported later in Chapter 6 focuses on developing a host-based scanner to automatically process host vulnerability alerts.

2.5 PKI and Certificate Revocation

Public Key Infrastructure (PKI) is defined in [75] as "the infrastructure able to support the management of public keys able to support authentication, encryption, integrity or non-repudiation services". The dominant PKI standards used are the ITU-T X.500 standards, in particular X.509 [75]. X.509 has also been adopted by the IETF (Internet

Engineering Task Force) for use in the Internet [37]. PKIs have been developed and deployed to facilitate secure communication and transactions over insecure public networks such as the Internet. Electronic commerce and secure server-access applications that use Transport Layer Security (TLS) or Secure Sockets Layer (SSL) [42] do rely on PKI. Verification of network-based (signed) software distribution and mobile code execution also critically depend on a timely and efficient PKI service.

In order for a principal to trust a public key belonging to another principal, the public key must be issued by a trusted source commonly known as a Certificate Authority (CA). A CA certifies a public key by issuing a digitally signed certificate, which binds the public key to the entity holding the corresponding private key. An issued certificate is expected to be in use for its entire validity period. However, it can become invalid prior to its expiration due to several reasons, such as change of name, change of association between the subject and the CA (e.g. the subject is no longer with an organization), and compromise or suspected compromise of the corresponding private key. Under such circumstances, the CA needs to revoke the certificate.

From a technical viewpoint, certificate revocation is probably the most challenging task of certificate management [96]. It thus deserves a special attention, and needs to be satisfactorily addressed so that the PKI service can be reliably available to host systems.

2.5.1 Issues in Certificate Revocation

In X.509-based PKI, certificate revocation is mainly supported by a mechanism called Certificate Revocation List (CRL) [37]. However, CRL is widely perceived to be costly and is attributed as one of the main impediments to successful global PKI deployment [96, 64]. Among others, CRL suffers from a high bandwidth requirement and (generally) a low timeliness guarantee.

Several alternatives, such as Online Certificate Status Protocol (OCSP) [124], have been proposed. OCSP offers a potentially real-time recency, but the CA needs to respond to any incoming query in real time with a signed message. As such, it imposes significantly high computation and network requirements on the CA. CA/Browser Forum, for instance, mandates OCSP support for its premium certificates only starting from 2012 [28].

To secure software distribution and programs' interaction with external hosts (e.g. using TLS/SSL), certificate revocation service must provide a *near real-time* timeliness guarantee and enable a fast computation on the verifier. This is particularly important given the proliferation of lightweight computers and mobile devices. In addition, the revocation scheme must not put excessively high overheads on any of the entities involved. Otherwise, undesirable service bottlenecks will be formed. Chapter 7 proposes two lightweight and practical certificate revocation schemes which can provide a near real-time timeliness guarantee.

2.5.2 Survey of Existing Certificate Revocation Systems

This section surveys various existing revocation schemes, and examines the potential issues with them. Throughout this thesis, we will refer to the subject of a certificate as a *principal*, and the party accepting/verifying certificates as a *verifier*. We also adopt the term *Certificate Status Information (CSI)* [73] to denote information pertinent to the validity of a certificate. Hence, CSI encompasses CRL data, OCSP messages and hash tokens in CRS/NOVOMODO. We refer to *Certificate Management Ancillary Entity (CMAE)* as a generic term for a *repository* or *directory*, from which a verifier may obtain the CSI.[6] Also, we define *revocation latency* (based on [4]) as the time interval between when a CA makes a revocation record and when it makes that information available to the verifiers. A low revocation latency means a high *revocation timeliness guarantee*.

Standardized in X.509 [75] and also profiled in the IETF [37], CRL is the most widely supported revocation scheme. Rivest [138] criticizes CRL by elaborating its several important drawbacks. McDaniel and Rubin [106] however responded to the criticism by pointing out that CRL still can provide a useful and efficient service in some environments. For many typical online transaction environments, however, CRL is widely known to have serious shortcomings. Firstly, the CRL may eventually grow to a cumbersome size in very large PKIs, e.g. ∼750 KB in Verisign's [141] and ∼40 MB in the U.S. DoD's [129]. Secondly, the downloaded CRLs may be mostly redundant given that more than 90% of the information is irrelevant to the verifiers [141]. Lastly, CRL does not offer adequate timely revocation guarantees. It is common for a CA to update its CRL only *daily*, as suggested in [177], although the CA may have a service for a shorter window period presumably with a higher charge.

There are several methods for improving the basic CRL mechanism, such as CRL Distribution Points, Delta CRLs, and Indirect CRLs (see [4] for a survey). However, all these improved schemes still put the same requirement on the verifier to obtain a complete revocation list, which includes the unrelated entries.

OCSP [124] was proposed to provide a more timely certificate status checking. An OCSP Responder is required to return the status information about a specific certificate in a digitally signed response. Since OCSP is an online service, it necessitate a prompt and reliable OCSP Responder system with a high level of security. It basically shifts the demands from network bandwidth in CRL to the Responder's processing power [141]. The high requirement put on the Responder to promptly generate signed replies may however be an issue [96]. A report on day-to-day operation of the EuroPKI infrastructure [93] highlighted the signature operations as the Responder's main bottleneck. Furthermore, the Responder's throughput is always bounded by the maximum number of signatures per second that it is able to perform.

[6] A CMAE may be trusted or untrusted depending on the revocation scheme.

Several modifications have been proposed on OCSP. H-OCSP [122] is as an improvement over OCSP that allows a verifier to verify the validity of a pre-produced response beyond its documented life time (as indicated by `thisUpdate` and `nextUpdate` fields in OCSP Response). The scheme uses a hash-chaining technique so that a H-OCSP Response can later be updated with reduced information and processing needs. However, H-OCSP requires the CA to maintain a hash-chain for each verifier. Given a potentially large number of verifiers, the CA's bandwidth and storage requirements are thus high.

Merkle Based Server (MBS)-OCSP [21] is a further improvement over H-OCSP, which employs Merkle Hash Tree (MHT) for faster verification by the verifier. In MBS-OCSP, a Responder associates a MHT with an OCSP Response, and stores in its local database the signed OCSP Response together with the associated MHT. We note that the use of MHT in MBS-OCSP is essentially the same as that suggested by Naor and Nissim [125]. The enhancement, however, comes with at the expense of larger messages to be downloaded by the verifier and extra storage on the Responder for all the MHTs.

Another approach, sometimes called "trusted directories", includes Certificate Revocation Status (CRS) [110] and NOVOMODO [111]. CRS/NOVOMODO, which we improve upon in Chapter 7, are summarized in Section 7.2.2.

Aiello et al. [6] proposed an improvement to CRS called "Hierarchical Scheme". It is aimed at reducing the CA-to-CMAE communication while still maintaining the low query communication. The improvement however comes at the price of a significant increase in the certificate transmission costs due to the increase in certificate size.

Kocher [83] proposed Certificate Revocation Tree (CRT) which employs MHT. In CRT, the CA breaks its revoked certificate list into a series of ranges that describes the certificate statuses. The lower end of each range contains the unique serial number of a revoked certificate. The ranges are then packaged as leaf nodes. A tree is built on these leaves by hashing two sibling values to obtain their parent node. The scheme however suffers from a high computational cost needed to update the tree.

Naor and Nissim [125] extended CRT by using a more suitable data structure, a 2-3 tree, rather than a binary hash tree. In a 2-3 tree, insertion and deletion of an element only affects a few interior points. Thus, there is no need to recalculate the entire tree.

2.5.3 Extended-Validation (EV) Certificates

In Chapter 7, our proposed scheme makes use of the Extended-Validation Certificate infrastructure as the basis for a lightweight and practical revocation scheme. We give some background on EV Certificate (EVC) here.

Recent attacks, such as certification-chaining [102], homograph phishing [52, 69, 102], and man-in-the-middle based SSLstrip attack [102], have shown the weaknesses of the standard X.509 certificate's usage. As a response, the EVC initiative was launched by the

CA/Browser Forum [28] to provide stronger assurance on certificates. This is achieved through a stricter issuance process based on the EV Guidelines [28]. The Guidelines standardize the identification process for EVCs, so that those issued by different CAs are still of the same quality with respect to identification. In addition, the Guidelines specify more rigorous checkings by a CA before issuing an EVC, which include:

- verifying the legal, physical and operational existence of the entity;
- verifying that the entity has an exclusive right over the specified domain name;
- verifying the identity and authority of the individuals acting for the website owner, and documents pertaining to legal obligations are signed by an authorized officer.

There have been a number of studies conducted to ascertain the usability and effectiveness of EVC and its purported benefits, particularly in securing SSL transactions [77, 168, 149]. These studies however focused mostly on the user's perception of EVC based on the perceived browser's interface. Jackson and Barth [76] evaluated the security of EVC with respect to the browser's security policy, and reported a possible vulnerability in its usage by the browser. We however view the issue more as a loophole in the browser's SSL usage of EVC rather than the flaw of EVC infrastructure. It is worth noting that despite the suggested conclusions, all of the studies acknowledge the value of EVC in providing stronger protections for secure transactions.

2.6 Formal Protocol Verification and BAN Logic

Designing a correct protocol specification which satisfies certain security properties is well recognized as a non-trivial task. Many logics and formal techniques have been proposed for verifying security protocols. Among various authentication logics, Burrows-Abadi-Needham (BAN) Logic [26, 27] is one of the best known and most widely used [107, 143, 108]. One reason for its popularity is that BAN Logic is comparatively easy to use. As pointed out by Meadows [107, 108], BAN Logic's intentional avoidance of many advanced features makes it a simple and straightforward logic that is easy to apply yet of substantial use for detecting flaws. This may well explain the constant appearance of publications applying BAN Logic even till now [5, 24, 155, 188, 35, 33], with application domains as diverse as wireless network [188], mobile communication [33], and voting system [155].

2.6.1 Overview of BAN Logic

BAN Logic [26, 27] is a modal-sorted logic constructed on several sorts of objects: principals, keys, messages and well-formed formulae. In BAN Logic notation, the symbols P, Q and R are usually used to denote principals; whereas X and Y are statements; and K is an encryption key. *Predicate constructs* are used to organize objects into well-formed formulae. BAN Logic defines the following constructs:

$P \mid\equiv X$:	P **believes** X. P may act as though X is true.
$P \triangleleft X$:	P **sees** X. Someone has sent a message containing X to principal P, who can read and repeat X (possibly after doing some decryption).
$P \mid\!\sim X$:	P **once said** X. P at some time sent a message including X. It is not known whether the message was sent long ago or during the current run of the protocol, but it is known that P believed X when he sent the message.
$P \mid\!\Rightarrow X$:	P **has jurisdiction over** X. P is an authority on X and should be trusted on this matter.
$\sharp(X)$:	X **is fresh**. X has not been sent in a message at any time before the current run of the protocol.
$\{X\}_{K_{PQ}}$:	X **encrypted with a secret key** K_{PQ}.
$P \overset{K_{PQ}}{\longleftrightarrow} Q$:	P and Q **may use a secret key** K_{PQ}. The key K_{PQ} is good, i.e. it will never be discovered by any principal except P or Q, or a principal trusted by either P or Q.

The only propositional connective used in BAN Logic is conjunction, which is denoted by a comma. A set of *inference rules*, also called logical postulates, are used to reason on well-formed formulae. Appendix C lists all the rules of BAN Logic which are relevant to our discussion in this thesis. A sample application of BAN Logic's rules to analyze a protocol can be found later in Appendix E.

BAN Logic has been criticized by many largely for its idealization process. It lacks a systematic way of transforming a protocol into the form that will be used by the logic. Additional criticisms were also pointed out in [127, 59, 101]. Fortunately though, along with these criticisms, suggestions for improvement as well as precautions on the improper usage of BAN Logic were made [51, 176, 81]. [176] attributes inaccurate idealization rather than the logic itself to be the root of many problems associated with BAN Logic.

For more discussion on BAN Logic, see [62, 107]. Syverson and Cervesato [163] give a tutorial on BAN Logic, and put it within a broader context of authentication logics.

2.6.2 Issues on BAN Logic Application to PKI-based Protocols

Despite being useful and widely applied, BAN Logic gives a rather simplified treatment of public-key authentication processing. It does not deal with deeper aspects of public key authentication such as certificate processing. This is presumably because PKI was not well established when the logic was designed. The situation is now very different since PKI becomes common. There exist some works such as [5, 51, 156] which attempted to extend BAN Logic to reason better with public key authentication. We find that the extension proposed in [51] does improve the expressiveness of BAN Logic while keeping the logic's secret-key aspects intact for easy application. However, it still falls short in capturing many important concepts and practices of the modern PKI usage.

Chapter 3

Self-Based IDS: Security Analysis and Automated Attack Construction

Section 2.2.1 gives an overview of the Self-based IDS [67, 152, 153], which compares an unparameterized system call trace of a process against the normal profile using k-grams of system calls. While the IDS has been shown to be effective in detecting intrusions [67, 182], it can be susceptible to evasion or mimicry attacks [180, 164]. The attacks disguise an attack trace so that it appears "normal" to the IDS. Our aim in this chapter is to systematically investigate the susceptibility of the Self-based IDS to mimicry attacks. In particular, we focus on deriving *automated* and *practical* attack constructions, and whether changing parameter(s) and normal profile representation of the Self-based IDS can help prevent such attacks. Here we assume a Self-based IDS which makes use of the full k-grams, an IDS model sometimes referred to as Stide [182][1], instead of the lookahead pair representations.

In the discussion to follow, we present a branch-and-bound algorithm for automatically constructing the *shortest mimicry attack trace* on the Self-based IDS (Stide) and its variant where the normal profile is represented as a *graph of k-grams*. This variant (defined later in Section 3.2.3) is a more precise model since it also records a temporal relationship between two consecutive k-grams generated from the normal trace. We evaluate these two models under two attack scenarios of *trojan attack* and *code-injection attack*. Our experimental results show that using sliding-window sizes larger than what is commonly used (6–8) and even using a more precise graph profile representation cannot prevent mimicry attacks on both attack scenarios. Furthermore, the execution times of

[1]Stide (sequence time-delay embedding) makes use of a more precise representation compared to the lookahead pair counterparts. As a result, it gives better discrimination in detecting potential anomalous behaviors, and is theoretically more resistant against mimicry attacks [74, 182].

the attack construction on several sample vulnerable programs are at most a few seconds, even for the relatively large sliding-window sizes (k=9–11). These running times thus show the practicality and efficiency of our attack construction algorithm.

In addition to producing the mimicry attack trace, our construction technique also provides us with useful information on how computationally expensive it is to perform a search to craft attacks on an IDS. Hence, it provides a systematic method to effectively measure the *resistance level* of the IDS against targeted attacks. Based on this observation, we generalize our attack construction method into a framework for measuring IDS resistance against attacks using a notion of *"attack space"*. We show the generality of the framework by applying it to other system-call based IDS models. Parts of the results on our mimicry attack construction on the Self-based IDS have been reported in [158].

The remainder of this chapter is organized as follows. Section 3.1 discusses related works on mimicry attacks. We give an algorithm for constructing mimicry attacks in Section 3.2. Section 3.3 presents the results of our experiments on automatically attacking the Self-based IDS and its variant. Section 3.4 discusses and analyzes these results. The generalization of our attack construction into a framework for measuring the robustness of IDS is expounded in Section 3.5. Finally, Section 3.6 gives a chapter summary.

3.1 Motivation and Limitations of Existing Works

Wagner and Soto [180] showed how the Self-based IDS which takes a non-parameterized system call trace can be vulnerable against mimicry attacks. They achieved this by inserting system calls which semantically behave like "no-ops", i.e. dummy operations that do not change any system effects, into the attack trace. In their work, they model the detection mechanism of the Self-based IDS as a Finite State Automaton (FSA). The attack is defined as a regular language. By adding no-ops, an attack is thus transformed into one that falls into the language modeled by the FSA of the IDS. Although the work contributed to the establishment of theoretical framework for the Self-based IDS from FSA viewpoint, the question of whether such attacks are practical is not fully answered. In our work, we address the practical construction of attacks and its efficiency issue.

Independently, Tan et al. [164] showed that it is possible to modify a foreign subtrace to be accepted by the Self-based IDS by taking advantage of the limited sliding-window size. They gave a hint on how this process could work involving an example with a sliding-window of length two. Their work viewed the construction problem as how to move an attack sequence into the IDS detection's *"blind region"* through *successive attack modifications*. However the work is primarily descriptive, and does not give a systematic procedure for attacking the IDS. A later paper by Gao et al. [53] showed some experimental results of attack construction on the Self-based IDS (Stide), which is a black-box detector, and several gray-box detectors [146, 48]. They investigated mimicry attacks on the IDSs with window sizes of up to 6, and showed the existence of mimicry attacks across the methods and the window sizes studied. But there was also no mention in the work on how the attack construction procedure was actually performed, which is the main focus of this chapter. Furthermore, we also consider mimicry attack construction algorithm in both trojan and code-injection attack scenarios.

Based on these earlier works, we can conclude that although mimicry attack on the Self-based IDS is known, and examples of turning an attack trace into a stealthy one do exist, it is however not completely clear how to generate a *practical* attack, which is fully automated and efficient in terms of the construction time. Additionally, we are also interested to find out whether varying parameter(s) and the normal profile representation of the the Self-based IDS can help prevent mimicry attacks.

Other related works on mimicry attack construction are [88, 58]. Mimicry attack on gray-box detectors [146, 48] requires forging the call stack and also temporarily relinquishing control to the victim application before regaining it for the next system call execution. Gao et al. [53] suggested a technique to forge the call stack and transfer the execution control to the injected code. To regain control, they suggested the modification of any code pointer following the system call so that it points back to the attack code. The work [53] shows the feasibility of this technique on a small example program, but manual development of mimicry attacks for realistic programs based on the technique poses some challenges. Kruegel et al. [88] addressed this challenge with a novel technique based on symbolic code execution on a program's binary to automate the steps needed for regaining control. Using example programs and applications, they showed that about 90% of the times, control could be successfully returned to the attack code. This result highlights the susceptibility of gray-box detectors against mimicry attacks. The technique in theory reduces the difficulty level of constructing mimicry attacks on gray-box detectors into that on black-box ones (e.g. Self-based IDS). The problem now becomes that of creating a "stealthy" attack trace interspersed with no-ops, the process which our attack construction algorithm addresses.

Giffin et al. [58] generated mimicry attack sequences using techniques based on model checking. The input to the model checker includes a specification of the OS model, the program model in the form of Push Down Automaton (PDA), and a specification that characterizes "unsafe" OS states. By using the model checking, their technique automatically finds system call sequences with the necessary arguments, which are allowed as a valid execution by the PDA that however produce unsafe OS state(s). However, the work has its own limitations too. Firstly, although the system does not require an initial exploit (or *basic attack trace* in our terminology as discussed in Section 3.2.5), it however requires a manually-specified model of the OS and its unsafe configurations.

Secondly, the "proofs" of detection hold only *with respect to* the developed OS abstraction and may not hold in the actual OS implementation. Thirdly, depending on the model-checker used, a reported attack sequence may not be the shortest attack trace possible. Lastly, the sample generated attack sequences given in [58] do not attempt to reach the `exit()` system call in order to gracefully terminate the attack. Nevertheless, the work still supports the conclusion of our results that the IDS model omitting data flow is vulnerable to mimicry attacks.

3.2 Automated Mimicry Attack Construction

3.2.1 Definitions

Before explaining our mimicry attack generation algorithm, let us first establish some definitions. A *trace* is a sequence of system calls invoked by a program during its execution. In our context, a trace can be viewed simply as a string over some defined alphabet. In the Self-based IDS model [67, 152, 153], the alphabet for traces are the system call numbers. A *normal trace* is the trace generated by a program during its training stage, which represents the normal behavior of the program. This normal trace is used to construct the *normal profile* of the program.

The objective of a Self-based IDS is to examine traces and determine whether they are normal or anomalous based on the program's normal profile. We call the attack trace that is detected by the IDS a *basic attack trace.* A mimicry attack disguises a basic attack trace into a *stealthy attack trace*, which the IDS classifies as being normal.

In our analysis, we will also look at *subtraces*, which are simply *substrings* of a trace. In addition, we also consider *subsequences*, which are a subset of letters from a trace that is arranged in the original relative order. Thus, subsequences are different from subtraces, i.e. the former need not be contiguous in the trace.

3.2.2 Pseudo Subtraces

A weakness of the Self-based IDS which makes use of a normal profile represented as a set of k-grams is that it can accept subtraces which actually do not occur in the normal trace(s). For example, consider the following two subtraces of a normal trace N: $\langle \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E} \rangle$ and $\langle \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F} \rangle$.[2] Suppose the window size is 5, and assume that the subtrace $\langle \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F} \rangle$ never occurs in the normal trace. This subtrace, however, will be accepted as normal by a Self-based IDS since its two 5-grams are both present in the normal profile. We call such a subtrace *a pseudo subtrace*, which is defined as follows.

[2]For easier illustration, we opt to use English letters instead of system call names or numbers for representing system calls in the example trace(s).

Definiton 3.1 (Pseudo subtrace). A subtrace is a *pseudo subtrace* for sliding-window size k if it is not a substring of the actual normal trace, yet passes the IDS detection since all of its k-grams are present in the normal profile.

A pseudo subtrace can be constructed by finding a common substring of length $k-1+\ell$ with $\ell \geq 0$ in two separate subtraces of length m ($\geq k+\ell$) and n ($\geq k+\ell$) respectively, and then joining them to form a new subtrace of length $m+n-k-\ell+1$. For our attack construction, we set $\ell = 0$ so as to put the weakest constraint on the mimicry attack construction with sliding-window size k.

Example 3.1. Given two unrelated subtraces $\langle \texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E} \rangle$ and $\langle \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}, \texttt{J}, \texttt{K} \rangle$ with $k = 5$, we can construct a pseudo subtrace $\langle \texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}, \texttt{J}, \texttt{K} \rangle$ of length $5+6-5-0+1 = 7$ as illustrated in Figure 3.1.

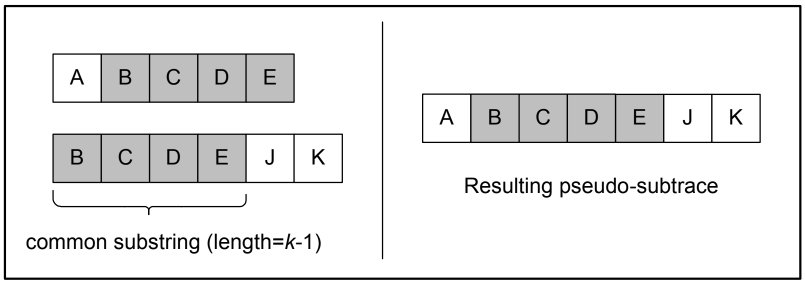

Figure 3.1: An example of pseudo subtrace construction with $k = 5$.

We can concatenate a pseudo subtrace with a normal subtrace, or another pseudo subtrace, to create a longer pseudo subtrace. *A stealthy attack trace* version of a basic attack trace is simply a pseudo subtrace in which the basic attack trace is its subsequence. Figure 3.2 illustrates such a complete process which combines subtraces into a stealthy attack trace containing basic attack sequence interspersed with no-ops.

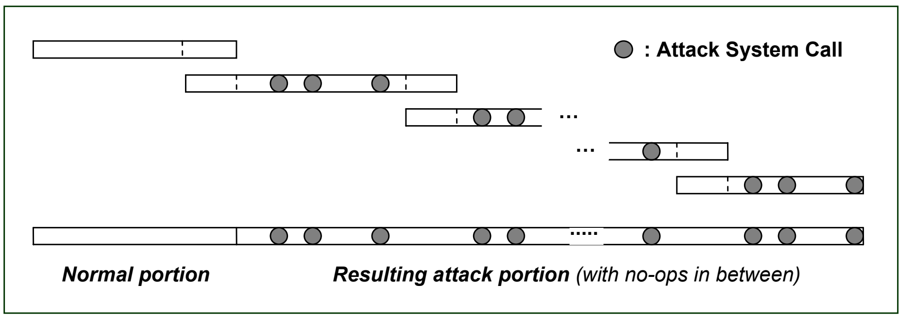

Figure 3.2: Mimicry attack construction by composing pseudo subtraces.

In our work, we use the term *pseudo subtrace* to specifically refer to the resulting subtrace which is obtained by joining two separate subtraces according to Definition 3.1. This resulting subtrace contains a *foreign sequence* (of *foreign order* type in the terminology of [166, 165]) of length k+1+ℓ as a substring. When $\ell = 0$ in the joining operation, the foreign sequence is a *minimal foreign sequence.* In the previous example, $\langle \texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}, \texttt{F} \rangle$ is a minimal foreign sequence of length 6 for $k = 5$. The process in Figure 3.2 constructs a pseudo subtrace for a mimicry attack where *multiple* minimal foreign sequences of length k+1 may exist along that subtrace. Each minimal foreign sequence combines two unconnected subtraces of the normal trace(s) together.[3]

3.2.3 Overlapping Graph Representation

Given a normal trace, we represent the corresponding normal profile using a construct called an *overlapping graph.* This is similar to the *De-Bruijn* graph, which has been applied to various problems such as the "sequencing by hybridization" problem in computational biology [134].

Consider the normal trace of a program P, which is denoted as $N : \langle n_1, n_2, n_3, ..., n_t \rangle$, where n_i (with $1 \leq i \leq t$) is the i-th system call of the trace N. We first augment N by adding a suffix consisting of the $k - 1$ occurrences of sentinel symbol ('$\texttt{\$}$'), which signifies the end of the trace. This adds $k - 1$ extra k-grams, and is done to simplify the attack construction algorithm. We now define K as the set of all k-grams derived from N according to the sliding-window based profile generation rule of the Self-based IDS.

In the presence of multiple normal traces of P, each collected from an execution session of P, we thus have N_i for $1 \leq i \leq x$, where x= the number of normal trace sessions. For each trace $N_{1 \leq i \leq x}$, we augment it with a suffix of $k - 1$ sentinel symbols, and then derive the corresponding set of k-grams K_i. The (consolidated) normal profile set is $K = \bigcup_{i=1}^{x} K_i$.[4]

Given two strings p and q, the function *overlap*(p, q) gives the maximal length of a suffix of p that matches a prefix of q. We define the *overlapping graph* constructed from K as follows.

Definiton 3.2 (Overlapping Graph). The *overlapping graph* G for a normal profile K, which is generated from the normal trace(s) and sliding-windows size k, is defined as: a directed graph (V, E), where the vertices V are the k-grams in K, and $E \subseteq V \times V$ with each edge $e \in E$ connects two vertices p and q whenever *overlap*$(p, q) = k - 1$.

Example 3.2. Given a normal trace N : $\langle \texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}, \texttt{F}, \texttt{G}, \texttt{A}, \texttt{B}, \texttt{E}, \texttt{F}, \texttt{H} \rangle$ with k=3, the corresponding overlapping graph G is as shown in Figure 3.3. For simplicity, we do not show the nodes corresponding to 3-grams of $(\texttt{F}, \texttt{H}, \texttt{\$})$ and $(\texttt{H}, \texttt{\$}, \texttt{\$})$ which are in G.

[3]We remark that not all foreign sequences according to the definition in [166, 165] are related to the notion of pseudo subtraces. Here, we are not concerned with foreign sequences containing system calls that are not in the k-grams. This is because they cannot be used to generate mimicry attacks.

[4]A single-session normal trace N thus can be considered as N_1 where $x = 1$.

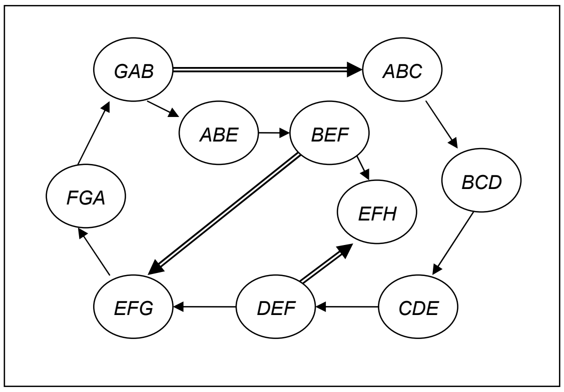

Figure 3.3: The overlapping graph G for $N : \langle \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{A}, \mathtt{B}, \mathtt{E}, \mathtt{F}, \mathtt{H} \rangle$ with $k = 3$. For simplicity, nodes corresponding to 3-gram $(\mathtt{F}, \mathtt{H}, \$)$ and $(\mathtt{H}, \$, \$)$ are not shown.

Notice that there are two types of edges in G: *direct edges* and *pseudo edges*. A direct edge is an edge that connects vertices p and q corresponding to two *consecutive k-grams* derived from a normal trace $N_{1 \le i \le x}$. More formally, we can define the set of direct edges $D = \{d \in E \mid d$ connects two vertices $p = (n_1, n_2, \ldots, n_{k-1}, n_k)$ and $q = (n_2, n_3, \ldots, n_k, n_{k+1})$ if there exists a subtrace $\langle n_1, n_2, n_3, \ldots, n_{k-1}, n_k, n_{k+1} \rangle$ in a normal trace $N_{1 \le i \le x}\}$. Hence, both p and q contain a common substring of length $k-1$, namely $\langle n_2, n_3, \ldots, n_{k-1}, n_k \rangle$, which stems from *the same* subtrace in a normal trace. Pseudo edges are edges created not due to the same substring of length $k-1$ in all N_i with $1 \le i \le x$. The set of pseudo edges is thus $U = E - D$. A pseudo edge is of special importance as it can be used to generate a pseudo subtrace. In Figure 3.3, the direct edges are drawn with single arrows, while the pseudo edges are drawn with double arrows.

The graph G can also be viewed as an FSA model for recognizing normal traces. The work [180] develops an FSA from the k-gram database profile. Their approach however does not distinguish between what in the overlapping graph corresponds to direct and pseudo edges. Since our concern is to address the limitations of the Self-based IDS, the overlapping graph gives us a natural capability where we can evaluate the difference between allowing pseudo edges and removing them.

So far, by our definition of G, we always assume that all system calls can be turned into no-ops. Wagner and Soto [180] analyze system calls in Unix/Linux environment, and conclude that apart from just a few system calls, almost all can be easily nullifiable. The exception applies mainly to `exit()`, `pause()` and `vhangup()`. We thus can define S^- as the set of unnullifiable system calls. To avoid any subsequent effect of using S^- in the stealthy trace, we can remove all outgoing edges from every node whose k-gram starts with any letter in set S^-. Thus, a stealthy trace can only contain a system call $s \in S^-$ as its last system call in the trace.[5]

[5]If deemed necessary, we can additionally remove all incoming edges into every node whose k-gram starts with any letter in set $S^- - \{$ `exit()`$\}$ to avoid executing unnullifiable system calls altogether.

3.2.4 Mimicry Attack Construction

Now, we would like to construct mimicry attacks by using the overlapping graph. Given an overlapping graph G and a basic attack trace $A : \langle a_1, a_2, a_3, \ldots, a_m \rangle$ which is detectable by the IDS, we want to automatically construct the shortest stealthy attack trace $L_{min} : \langle l_1, l_2, l_3, \ldots, l_n \rangle$, where $n \geq m$, which contains $a_1, a_2, a_3, \ldots, a_m$ as a subsequence and where the other system calls in $\{L_{min} - A\}$ behave as no-ops with respect to A.

We consider automated attack constructions in two following attack scenarios:

1. *Trojan attack scenario*: where the attacker replaces a victim program with a trojan that executes a stealthy attack trace.
2. *Code-injection attack scenario*: where the resulting stealthy attack trace is to be injected into a victim program's process memory, and subsequently be executed. This can be achieved, for example, by using a buffer overflow attack technique, where the stealthy trace is crafted as the shellcode of the buffer-overflow exploit.

The code-injection attack scenario poses more challenges since the transition between the pre-injection subtrace and the stealthy attack trace must still pass the IDS detection. In addition, from the attacker's viewpoint, the shortest stealthy attack trace is desirable in this scenario for the following important reasons. Firstly, an attack trace with a minimum number of system calls will achieve attack efficiency. Secondly, and more importantly, a rather long shellcode injected as input data into the program may be noticed by other security measure(s) deployed on the host. A security measure monitoring the length of an input fed into the program, for instance, may suspect a long shellcode as a malicious input or an anomalous request [90]. Lastly, the most straightforward buffer overflow technique is one that puts the whole shellcode inside the overflowed buffer [130]. Hence, for this technique to work, the shellcode is restricted by size constraints.

3.2.5 Attack Construction Algorithm under Trojan Attack Scenario

In a trojan attack scenario, transforming a basic attack trace A into *a stealthy attack trace* L is equivalent to:

> Finding a path P on the overlapping graph G which monotonically visits nodes whose k-gram label begins with the symbol a_i for all $1 \leq i \leq m$.

The shortest stealthy trace L_{min} is a stealthy attack trace with the minimum number of system calls among all the stealthy attack traces.

To allow for an attack construction, we augment the previously constructed overlapping graph G into the *extended overlapping graph* G' defined as follows.

Definiton 3.3 (Extended Overlapping Graph). The *extended overlapping graph* $G' = (V{+}W, E{+}Occ)$ is the resulting graph constructed by augmenting G with an additional subgraph called the *occurrence subgraph* $O = (W, Occ)$. The nodes in the occurrence subgraph, W, are the unique individual letters from the k-grams in G. The edges Occ, which connect W and V, are constructed as follows: for each node $w \in W$, we add an outgoing edge to every node in G where the first letter in its k-gram label is the same as the letter for w.

Example 3.3. Figure 3.4 shows the extended overlapping graph for the overlapping graph in Example 3.2 (which is shown in Figure 3.3).

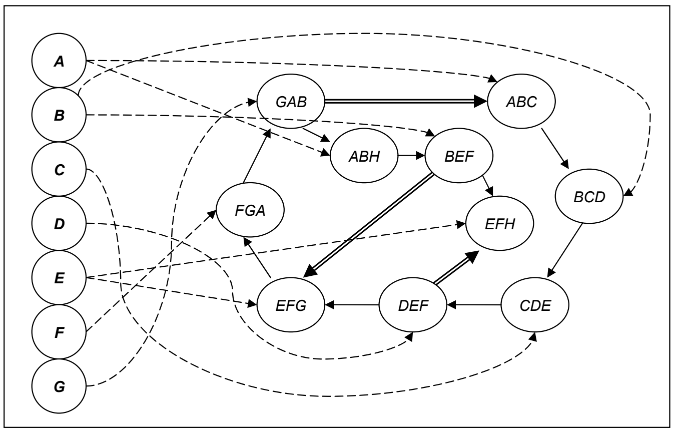

Figure 3.4: The extended overlapping graph G' from graph G in Figure 3.3.

Before outlining the construction algorithm, we illustrate the mimicry attack construction with the following example.

Example 3.4. Suppose that we want to construct a stealthy attack trace from a basic attack trace A:$\langle \mathtt{G}, \mathtt{C}, \mathtt{D} \rangle$ using the extended overlapping graph G' in Figure 3.4. Note that trace A is detected by the IDS since its corresponding 3-gram $(\mathtt{G}, \mathtt{C}, \mathtt{D})$ is not in the normal profile. Inspecting graph G', we however find the stealthy path: GAB - ABC - BCD - CDE - DEF. Thus, the stealthy attack trace is the sequence of $\langle \underline{\mathtt{G}}, \mathtt{A}, \mathtt{B}, \underline{\mathtt{C}}, \underline{\mathtt{D}} \rangle$, with A and B added as no-ops. This example uses the pseudo edge (GAB, ABC).

Our attack construction builds a search tree, and performs a search on the tree to find the shortest stealthy attack trace. Apart from the specially-added root node v_0, each node with depth i (for $1 \le i \le m$) in the search tree is a node $v_i \in V$ whose label starts with a_i in A. The node v_i also represents a path $P_{v_i}: v_0, v_1, \pi_1, v_2, \pi_2, \ldots, v_{i-1}, \pi_{i-1}, v_i$ on G, where π_i (for $1 \le i < i{-}1$) denotes a subpath with zero or more nodes. By extracting the first system call in each node along the path, the path P_{v_i} thus corresponds to a trace containing $a_1, a_2, \ldots, a_i$ as its subsequence. From v_i (for $1 \le i < m$), the search explores all the next nodes in $\{v_{i+1} \in V \mid v_i \rightsquigarrow v_{i+1} \;\wedge\; (a_{i+1}, v_{i+1}) \in Occ)\}$, where the connection relation ('$p \rightsquigarrow q$') denotes that nodes p and q on G are connected. The branches from v_i

(for $1 \leq i < m$) thus represent the choices of extending the path P_{v_i} into the path $P_{v_{i+1}}$, which corresponds to the trace containing $a_1, \ldots, a_i, a_{i+1}$ as its subsequence. A stealthy attack trace is found when we can reach v_m which corresponds a trace containing A as its subsequence.

In order to make the search more efficient, we employ a *branch-and-bound* strategy to prune the constructed attacks which exceed the best solution found so far. Our implementation constructs an all-pair shortest-path distance table, which is used both to test connectivity between two nodes on G and also to assist in pruning for the branch-and-bound search. A sketch of the algorithm is given in Algorithm 3.1.

Algorithm 3.1 Attack construction on Self-based IDS under trojan attack scenario

```
Input:
- Extended overlapping graph $G'$ constructed from the program's normal profile $K$
- Basic attack trace $A : \langle a_1, a_2, a_3, \ldots, a_m \rangle$
Output:
- The shortest stealthy attack trace $L_{min} : \langle l_1, l_2, l_3, \ldots, l_n \rangle$
- Or failure, if no solution trace can be found.

1. Construct the all-pair shortest-path distance table, with the following initialization:
     Between any two adjacent nodes $p$ and $q$, set $distance(p, q) := 1$.
     If two nodes $p$ and $q$ are not connected Then $distance(p, q) := \infty$.
2. $Min\_distance := \infty$ and $Min\_path := \langle\rangle$.
   Create a special node $v_0$ as the root of the search tree, and $\forall v \in V$, $distance(v_0, v) := 1$.
3. (Perform branch-and-bound search on the search tree as follows:)
   $current\_cost := 0$. (It keeps track of the distance from $v_0$ to $v_i$ in the path being explored.)
   For $i$:=1 to $m$ choose $v_i$ from $\{v_i \in V \mid (a_i, v_i) \in Occ\}$:
     - If $distance(v_{i-1}, v_i) = \infty$ Then backtrack.
     - Add $distance(v_{i-1}, v_i)$ to $current\_cost$.
     - If $current\_cost \geq Min\_distance$ Then backtrack.
     - If complete solution is found ($i = m$) Then
         If $current\_cost < Min\_distance$ Then
           $Min\_distance := current\_cost$;
           $Min\_path := current\_path$.
           (Note that a path includes zero or more nodes in between $v_{i-1}$ and $v_i$ for $2 \leq i \leq m$.)
4. Once the search tree is fully explored:
   If $Min\_distance = \infty$ Then return failure;
   Else return $L_{min}:\langle l_1, l_2, l_3, \ldots, l_n \rangle$ with $n = Min\_distance$, and $l_i$ (for $1 \leq i \leq n$) being the first
       system call in the $k$-gram label of ($i$+1)-th node in $Min\_path$.
       (Note that the first node in $Min\_path$, i.e. the root node $v_0$, is eventually discarded.)
```

3.2.6 Attack Construction Algorithm under Code-Injection Attack Scenario

Under the code-injection attack scenario, a stealthy attack trace must be introduced only after the "*attack-introduction point*", that is after the code injection succeeds and the victim process' control flow is gained and redirected. Here, we need to make note of the k system calls prior to the attack point, which we call the *border sequence* $B : \langle b_{-k}, b_{-k+1}, \ldots, b_{-2}, b_{-1}\rangle$. This sequence corresponds to a *border k-gram* in the normal profile; which then translates into a particular node v_s in G, which we call the *border-start node*.

To ensure that the concatenation of the pre-injection trace and the stealthy attack trace still passes the IDS, we extend Algorithm 3.1 as follows:

- **(E_1)**: The border sequence $B : \langle b_{-k}, b_{-k+1}, \ldots, b_{-2}, b_{-1}\rangle$ must be included as an input to the search algorithm.
- **(E_2)**: This border-start node v_s is made as the root of the search tree.
- **(E_3)**: A first-level node in the search tree, i.e. node $v_1 \in \{v_1 \in V \mid (a_1, v_1) \in Occ\}$, is explored during the search only if $k \leq distance(v_s, v_1) < \infty$.
- **(E_4)**: Once the search tree is fully explored with a solution, then return $L_{min} : \langle l_1, l_2, l_3, \ldots, l_n\rangle$ with l_i (for $1 \leq i \leq n = Min\_distance - k$) being the first system call in the k-gram label of $(i+k)$-th node in *Min_path*. (Thus, in the end, we discard the first k nodes of *Min_path*).

We remark that the construction actually requires only the last $k-1$, instead of k, system calls prior to the attack point. In other words, the sequence $B' : \langle b_{-k+1}, b_{-k+2}, \ldots, b_{-2}, b_{-1}\rangle$ would be sufficient. Our actual objective is to first form a set of *border-end nodes*, denoted as Z, where each node $v_z \in Z$ is determined as follows: Find a path $P_v : v_{-k+1}, v_{-k+2}, \ldots, v_{-2}, v_{-1}$ (of length $k-2$) on G, where v_i (for $-k+1 \leq i \leq -1$) is a node whose label starts with b_i in B'. The last node in P_v, i.e. node v_{-1}, is a border-end node v_z. Given this set Z, the "*attack trace connectivity constraint*" to satisfy under the code-injection attack scenario is that the first-level nodes in the search tree, i.e. set $\{v_1 \in V \mid (a_1, v_1) \in Occ\}$, are explored during the search *only if* they are connected to $v_z \in Z$. In other words, a node v_1 must satisfy $1 \leq distance(v_z, v_1) < \infty$ to be explorable.

The set of border-end nodes Z is in fact equivalent to the set of all reachable nodes from the border-start node v_s by distance $k-1$, which we denote as set V_{k-1}. We can show this equivalence as follows. Suppose a node $v \in Z$. By the definition of a border-end node, there exists a path $P_v : v_{-k+1}, v_{-k+2}, \ldots, v_{-2}, v$ on G where: v_i (for $-k+1 \leq i \leq -2$) is a node whose label starts with b_i in B', and v has a label starting with b_{-1} in B'. Given the fact that an edge in G connects two nodes p and q when $overlap(p, q) = k-1$ (see Definition 3.2), the node v_{-k+1} in the path P_v has $(b_{-k+1}, b_{-k+2}, \ldots, b_{-2}, b_{-1}, b_0)$ as its label.

As a result, there exists an edge connecting v_s, whose label is $(b_{-k}, b_{-k+1}, \ldots, b_{-2}, b_{-1})$, to v_{-k+1}. The node v is thus reachable from v_s by distance $k{-}1$, i.e. $v \in V_{k-1}$. Therefore, $Z \subseteq V_{k-1}$. Now, suppose that a node $v \in V_{k-1}$. Thus, there exists a path of length $k{-}1$ on G connecting v_s, whose label is $(b_{-k}, b_{-k+1}, \ldots, b_{-2}, b_{-1})$, to v. Let us call this path $P_z : v_{-k}, v_{-k+1}, \ldots, v_{-2}, v$. From the definition of the overlapping graph, node v_i (for $-k \leq i \leq -2$) is a node whose label starts with b_i in B, and the node v has a label starting with b_{-1} in B. By the definition of the border-end node, v is thus a border-end node, i.e. $v \in Z$. Therefore, $V_{k-1} \subseteq Z$. We thus have shown that $Z = V_{k-1}$.

For the attack construction under the code-injection attack scenario, having the border sequence B (instead of B') allows us to easily check if a first-level node in the search tree satisfies the attack trace connectivity constraint. As mentioned, the border sequence B allows us to determine the border-start node v_s. Also recall that we only explore a first-level node v_1 in the search tree only if $\exists\, v_z \in Z$ such that $1 \leq \mathit{distance}(v_z, v_1) < \infty$. Given the equivalence between Z and V_{k-1} as shown above, we thus can easily satisfy the constraint by exploring a first-level node v_1 only if $k \leq \mathit{distance}(v_s, v_1) < \infty$ (as required by step E_3). Note that we later discard the first k nodes of *Min_path*, which corresponds to a sub-path connecting v_s to a border-end node $v_z \in Z$ (see step E_4).

3.2.7 Proof of Optimality of the Attack Construction

The construction algorithm produces the *shortest* stealthy attack trace for a given set of k-grams as the normal profile database, with k being the sliding-window size. Note that there may be more than one shortest stealthy attack trace, all of the same length. Our attack algorithm returns one of them.

We now prove the optimality of the attack construction algorithm for both the trojan and the code-injection attack scenarios.

Theorem 3.1 (Stealthy Attack Trace Optimality). Let k be the chosen sliding-window size and K be a set of k-grams as the normal profile database, then the stealthy attack trace $L_{min} : \langle l_1, l_2, l_3, ..., l_n \rangle$ returned by the attack construction algorithm is the *shortest* stealthy attack trace of the basic attack trace $A : \langle a_1, a_2, a_3, \ldots, a_m \rangle$.

Proof. We will prove the optimality by contradiction. Suppose there exists a stealthy attack trace shorter than L_{min}, which is denoted by $L' : \langle l'_1, l'_2, l'_3, ..., l'_\ell \rangle$, with $\ell < n$. Given that all k-grams derived from trace L' must be present in the set K, there exists a node corresponding to each k-gram of L' in the overlapping graph G which is constructed based on K.[6] Moreover, since any two consecutively generated k-grams share a common substring of length $k{-}1$, then the two nodes corresponding to these two consecutive

[6]Recall that graph G also contains $k{-}1$ nodes whose label contains sentinel symbol ('\$') due to our addition of $k{-}1$ occurrences of sentinel at the end of the trace (see Section 3.2.3).

k-grams are connected by an edge in G. As a result, the assumed stealthy trace L' corresponds to a path $P' : \pi'_0, v'_1, \pi'_1, v'_2, \pi'_2, v'_3, \pi'_3, \ldots, v'_{m-1}, \pi'_{m-1}, v'_m$ on G, where the following conditions hold (here v'_i, with $1 \leq i \leq m$, is a node in G; and π'_i, with $0 \leq i < m$, denotes a path on G with *zero or more* nodes):

1. For each node v'_i on path P', with $1 \leq i \leq m$, the first letter of its label is the attack system call a_i.
2. For each node v'_i, with $1 \leq i \leq m$, there exist a node $w_i \in W$ whose label is a_i and an edge $(w_i, v'_i) \in Occ$ (i.e. an edge connecting w_i to v'_i).
3. Node v'_i, with $1 \leq i < m$, is connected to node v'_{i+1} through a path π'_i on G which consists of zero or more nodes.
4. Under the trojan attack scenario, path π'_0 is empty. Under the code-injection attack scenario (discussed in Section 3.2.6), path π'_0 connects a border-end node $v_z \in Z$ to node v'_1.

Following the same reasoning on the correspondence between a stealthy attack trace and a path on G, we also have a path P_{min} on G which corresponds to the reported stealthy trace $L_{min} : \langle l_1, l_2, l_3, \ldots, l_n \rangle$. Due to the 1-to-1 correspondence between a system call in a stealthy attack trace and a node in the corresponding path on G, the number of system calls of a stealthy trace is the same as the number of nodes on the corresponding path. Here, we use $|P|$ to denote the number of nodes in a path P. We thus have $|P'| = \ell$, $|P_{min}| = n$, and $\ell < n$. Note that $|P| = 1 +$ the length of path P.

By the definition of our search tree in the attack construction algorithm and the fact the our search explores all possible solution paths, there exists an explored solution path P_{equiv} which also contains $v'_1, v'_2, v'_3, \ldots, v'_{m-1}, v'_m$ as its subsequence. That is, P_{equiv} is in the form of: $\pi_0, v'_1, \pi_1, v'_2, \pi_2, v'_3, \pi_3, \ldots, v'_{m-1}, \pi_{m-1}, v'_m$ where π_i, for $0 \leq i < m$, denotes a path on G with zero or more nodes. Similar to π'_0 in P', the path π_0 is empty under the trojan attack scenario. Under the code-injection attack scenario, π_0 connects a border-end node to v'_1. Note that this border-end node may or may not be the same as the one used by the path π'_0 in P'.

Now let us consider paths P' and P_{equiv}. We have two possible cases:

1. If $|P_{equiv}| \leq |P'|$:
   Since the branch-and-bound algorithm returns a trace with the shortest length among all solution paths, we therefore have $|P_{min}| \leq |P_{equiv}|$. By transitivity, we have $|P_{min}| \leq |P'|$. Thus, $n \leq \ell$. However, we assume earlier that L' is shorter than L_{min}, that is: $\ell < n$. This leads to a contradiction.

2. If $|P_{equiv}| > |P'|$:
   Since both paths share a common subsequence $v'_1, v'_2, v'_3, \ldots, v'_{m-1}, v'_m$, the path P' can be shorter than P_{equiv} only if:

$$\sum_{i=0}^{m-1} |\pi'_i| < \sum_{i=0}^{m-1} |\pi_i| \tag{3.1}$$

However, the attack construction algorithm always makes use of the shortest path between any two nodes v'_i and v'_{i+1} for $1 \leq i \leq m{-}1$ to produce P_{equiv}. Hence, $|\pi_i| \leq |\pi'_i|$ for all $1 \leq i \leq m{-}1$. Now, recall that $|\pi_0| = |\pi'_0| = 0$ under the trojan attack scenario. Under the code-injection attack scenario, both π_0 and π'_0 contain their respective (prefix) subpath of length $k{-}1$, which connects v_s to their respective choice of a border-end node. In constructing P_{equiv}, the attack construction algorithm also makes use of the shortest path between v_s and v'_1. As a result, P_{equiv} also has the shortest path possible between its border-end node and v'_1. Hence, $|\pi_0| \leq |\pi'_0|$ under the code-injection attack scenario. We therefore have $|\pi_i| \leq |\pi'_i|$ for all $0 \leq i \leq m{-}1$. Consequently:

$$\sum_{i=0}^{m-1} |\pi_i| \leq \sum_{i=0}^{m-1} |\pi'_i| \tag{3.2}$$

Therefore, $|P_{equiv}| \leq |P'|$. This contradicts our case assumption that $|P_{equiv}| > |P'|$.

Hence, the stealthy attack trace L' which is shorter than L_{min} does not exist. $\square$

3.3 IDS Attack Experiments

3.3.1 Experimental Set-Up

We perform an automated stealthy attack construction under the trojan and the code-injection attack scenario on the following two variations of the Self-based IDS models:

- **Set Represented Self-based IDS (SET-SELF)**: where the normal profile is represented as a *set* of k-grams. This is the Stide model used in [67, 182].
- **Graph Represented Self-based IDS (GRA-SELF)**: where the normal profile is a *graph* of k-grams. This graph is essentially an overlapping graph with the edge set $E = D$ (see Definition 3.2). That is, only direct edges are allowed in this graph profile. Since a path on this direct-edge only overlapping graph corresponds to an accepted subtrace in GRA-IDS, our attack construction algorithms which find a trace on the overlapping graph (outlined in Section 3.2.5 and 3.2.6) and their proof of optimality (given in Section 3.2.7) apply to GRA-SELF as well.

Our attack construction is implemented in C, and executed on a PC with an Intel Core i7 processor (3.20 GHz) and 12 GB of RAM running Linux Fedora 12. We have also used various older versions of the Red Hat Linux distribution in order to obtain the traces of sample vulnerable programs together with their exploits. The traces are captured in Linux by using the `strace` utility. For simplicity, we have omitted system calls related to signal events, such as SIGALRM and SIGCHLD, due to their asynchronous nature.

The following remarks apply to our experiments:

- To construct a mimicry attack, the attacker requires the knowledge of the vulnerable program's actual profile (K) in the victim host. Generally, K is not available to the attacker. Yet, given possible information and insights on the OS environments and the program's configurations, the attacker can attempt to simulate the targeted program and derive an approximated profile (K') to be as close as possible to K. This is particularly the case if the attacker is an insider, or receives some help from one. Furthermore, the normal profile K is meant to be generated by exploring all possible execution paths of the program which are deemed normal. In our experiments, we use K as an input to the algorithm so as to investigate the actual susceptibility of the programs against mimicry attacks. This assumption is relevant particularly when the smart attacker assumed could derive a good approximation of K, or when it somehow manages to obtain K.
- The three exploits below make use of `execve()` system call to spawn a root shell. However, `execve()` is not present in the normal trace. Therefore, we use an alternative strategy to write an entry into the file "`/etc/shadow`". This actually corresponds to *Attack-strategy* A_2 from our list of strategies given in Table 4.5 (see Section 4.3.2). This particular attack strategy is chosen for comparison here as it has been used for mimicry attacks on the Self-based IDS before (e.g. [164]). We remark that it is perfectly reasonable to modify the original attack since we assume an intelligent adversary.

3.3.2 Sample Vulnerable Programs and Attack Construction

As our goal is to investigate the practicality of our automated attack construction, we experiment with real programs using real available exploits.

Traceroot2 (Traceroute Exploit)

This traceroute exploit was previously used in [164]. It is available at `http://www.packetstormsecurity.org/0011-exploits/traceroot2.c`. The exploit attacks LBNL traceroute v1.4a5 which is included in the Linux Red Hat 6.2 distribution.

The original attack sequence is: `setuid(0),setgid(0), execve("/bin/sh")`. This is changed into: `open(),write(), close(),_exit()` as in [164]. The results of the attack construction on normal traces generated from three traceroute's sessions (with a total of 2,789 system calls) for window sizes from k=5 to 11 are given in Table 3.1.

As can be seen in Table 3.1, the attack construction algorithm on traceroute is able to find stealthy attack traces on the two IDS variants and under the two attack scenarios. The shortest stealthy attack traces under the trojan attack mode are of below 60 system

Traceroute Search	*k*=**5**	*k*=**6**	*k*=**7**	*k*=**8**	*k*=**9**	*k*=**10**	*k*=**11**
Number of System Calls in Resulting Stealthy Attack Trace:							
SET-SELF (Trojan-Attack)	39	43	43	48	51	54	54
GRA-SELF (Trojan-Attack)	43	43	48	51	54	54	56
SET-SELF (Code-Injection)	44	48	48	64	64	112	114
GRA-SELF (Code-Injection)	48	48	64	64	112	114	125
Average Search Time (User+Sys)	0.021s	0.021s	0.025s	0.028s	0.032s	0.034s	0.030s

Table 3.1: Attack construction results for traceroute with k=5 to 11 (with 2,789 system calls in the normal trace). SET-SELF and GRA-SELF represent the Self-based IDSs with the normal profile stored as a *set* of k-grams and a *graph* of k-grams respectively.

calls even with k=11. The stealthy attack traces under code-injection attack scenario require longer traces than those under trojan attack scenario given the same k. Storing the normal profile as a graph of k-grams (GRA-SELF) does not make the Self-based IDS substantially more robust against mimicry attacks. It makes the resulting stealthy traces only slightly longer than those required in the SET-SELF.

JOE Text Editor Exploit

The victim program that we chose is a popular Linux terminal text editor JOE, available at `http://sourceforge.net/projects/joe-editor/`. The exploit for Red Hat Linux was available at `http://www.uhagr.org/src/kwazy/UHAGr-Joe.pl`[7], and was run on Red Hat 7.3. JOE is not normally run as a setuid program. As a proof of concept, we assume that JOE has been run from root (or setuid to root). The original attack sequence is: `setuid(0), execve ("/bin/sh")`. Again, we changed it to: `open(), write(), close(), _exit()`.

The results of the attack construction algorithm on JOE's normal traces generated from three JOE sessions (with a total of 9,802 system calls) for sliding-window sizes from k=5 to 11 are given in Table 3.2.[8]

Since JOE is a text editor, it falls into the class of general purpose programs as opposed to the more privileged processes targeted for monitoring by the Self-based IDS [67, 152]. We include it here to highlight some points on the results of attack construction. From the results shown in Table 3.2, we have made some interesting observations. Here, we find that for the trojan attack scenario, a stealthy attack trace with 7 system calls is sufficient for k=5 to 11. This is because a subtrace in the normal trace of JOE happens to

[7]The site seems to be no longer in operation (as of October 2010).

[8]From the normal traces collected for JOE, we note that there are actually some differences between the normal traces and the exploit trace before the attack-introduction point due to some `brk()` system calls. This is probably due to an increased memory allocation for the buffer overflow attack. However, as reasoned by [180], small differences may be tolerated by the Self-based IDS depending on the parameters used in its anomaly-signal measurement function (e.g. Locality Frame Count).

JOE Search	*k*=**5**	*k*=**6**	*k*=**7**	*k*=**8**	*k*=**9**	*k*=**10**	*k*=**11**
Number of System Calls in Resulting Stealthy Attack Trace:							
SET-SELF (Trojan-Attack)	7	7	7	7	7	7	7
GRA-SELF (Trojan-Attack)	7	7	7	7	7	7	7
SET-SELF (Code-Injection)	20	30	49	76	79	80	81
GRA-SELF (Code-Injection)	30	49	76	79	80	81	82
Average Search Time (User+Sys)	0.039s	0.043s	0.048s	0.054s	0.059s	0.070s	0.081s

Table 3.2: Attack construction results for JOE with k=5 to 11 (with 9,802 system calls in the normal trace).

contain the basic attack trace as its subsequence. Hence, *no* pseudo subtrace construction was needed. However, this stealthy trace does not work under the code-injection attack scenario. Instead, longer stealthy attack traces are required.

Autowux WU-FTPD Exploit

This is the same exploit used in [180]. The `autowux.c` exploits "site exec" vulnerability on the WU-FTPD FTP server. It is available at `http://www.securityfocus.com/bid/1387/exploit/`. We ran the `wu-2.4.2-academ [BETA-15]` version of WU-FTPD that comes with Red Hat 5.0 distribution on the 2.2.19 kernel.

We use the same attack trace as [180] which is: `setreuid(),chroot(),chdir(),chroot(),open(),write(),close(),_exit()`. The results of the attack construction on the WU-FTPD normal traces generated from 17 sessions (19,582 system calls) for sliding-window sizes from k=5 to 11 are given in Table 3.3.

WU-FTPD Search	*k*=**5**	*k*=**6**	*k*=**7**	*k*=**8**	*k*=**9**	*k*=**10**	*k*=**11**
Number of System Calls in Resulting Stealthy Attack Trace:							
SET-SELF (Trojan-Attack)	64	155	165	174	216	241	258
GRA-SELF (Trojan-Attack)	155	165	174	216	241	258	276
SET-SELF (Code-Injection)	79	170	180	203	238	256	294
GRA-SELF (Code-Injection)	170	180	203	238	256	294	308
Average Search Time (User+Sys)	0.547s	0.738s	0.947s	1.183s	1.456s	1.730s	1.927s

Table 3.3: Attack construction results for WU-FTPD with k=5 to 11 (with 19,582 system calls in the normal trace).

We can see from Table 3.3 that the resulting stealthy attack traces happen to be relatively longer than those in the first two sample programs. The work [180] give a stealthy trace for k=6 with 135 system calls based on their normal profile. Their result, however, is not comparable to ours as the normal traces used are different. In their case, they had collected normal traces for an existing WU-FTPD with large numbers of downloads over two days. On the other hand, we have used a small normal profile.

We additionally discuss some patterns which commonly apply to the three sample programs below.

3.4 IDS Evaluation Discussion

We have shown that the two variants of the Self-based IDS are vulnerable against mimicry attacks even with sliding-window size longer than that is usually employed (k= 6 to 8). The running times also show that our automated attack construction algorithm is practical and efficient. Execution times for all cases are at most a few seconds even on relatively large window sizes.

Based on the results, we also observe the following common trends:

- There can be a considerable difference in length between the stealthy code-injection attack traces and the trojan attack ones. In some cases, like in JOE, the stealthy attack trace under the trojan attack scenario is very short. Here an attack with 7 system calls works for window sizes from k=5 to 11.[9]

- The length of the shortest stealthy attack trace varies from program to program. It confirms the earlier reports [164, 53] that a larger window size (k) tends to require also a longer stealthy attack trace. In practice, however, there is a limit on the choice of k due to the increase in normal profile size and processing overheads of the IDS [182, 74]. Nevertheless, it clearly shows that relying on the Self-based IDS with a certain size of sliding-window of, such as 6 as suggested in [67], is insufficient. Rather, other improvements are necessary.

- We can see that removing pseudo edges on graph G (i.e. imposing the GRA-SELF IDS model) does not make the Self-based IDS significantly stronger against mimicry attacks. In other words, pseudo subtraces can still exist even if we store the normal profile as a *graph* of k-grams. We find that, besides pseudo edges, there is another source of imprecision in the GRA-SELF model which would allow for a pseudo subtrace construction. To understand this, let us now consider a normal trace $\langle \texttt{A},\texttt{B},\texttt{C},\texttt{D},\texttt{E},\texttt{A},\texttt{B},\texttt{C},\texttt{M}\rangle$ with $k = 3$. Figure 3.5 shows the graph without pseudo edges as the normal profile for this trace.

  A stealthy attack trace $\langle \underline{\texttt{E}},\texttt{A},\underline{\texttt{B}},\texttt{C},\underline{\texttt{D}}\rangle$ can be constructed for a basic attack trace $\langle \texttt{E},\texttt{B},\texttt{D}\rangle$. This is achieved by a pseudo subtrace construction of $\langle \texttt{E},\texttt{A},\texttt{B},\texttt{C},\texttt{D}\rangle$ from two unrelated subtraces $\langle \texttt{E},\texttt{A},\texttt{B}\rangle$ and $\langle \texttt{A},\texttt{B},\texttt{C},\texttt{D}\rangle$. In Figure 3.5, the constructed stealthy trace corresponds to the path: `EAB-ABC-BCD-CDE-DEA`. The reason why

[9]The actual trojans will usually have longer sequences since there are additional system calls invoked at the beginning of a program related to libraries loading or memory allocation. However, the reported number of system calls does establish the lower-bound of mimicry attacks in the trojan attack setting.

the path exist in the graph is that a "*common node*" `ABC` facilitates such a path construction. To illustrate this construction, Figure 3.5 is drawn with two different outgoing edges from node `ABC` based on the actual overlapping relationships with its adjacent nodes (k-grams) as they appear in the normal trace. A stealthy path can make use of a common node like `ABC` to combine two unrelated paths (shown as solid and dotted lines in the figure). Such common nodes exist due to the existence of a common substring of length k, e.g. $\langle \mathtt{A}, \mathtt{B}, \mathtt{C} \rangle$ in the example, between any two unrelated subtraces in the normal trace.

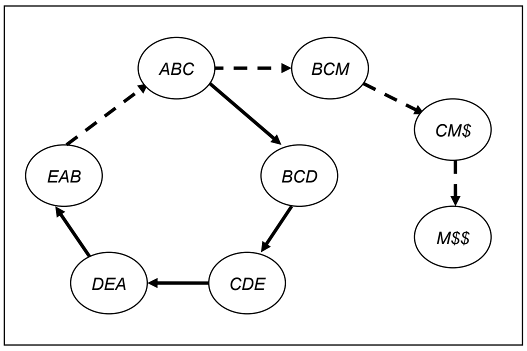

Figure 3.5: A graph of 3-grams (without pseudo edges) used in GRA-SELF model for the sample trace $\langle \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{M} \rangle$. Note that a "common node" `ABC` allows for pseudo subtrace construction.

3.5 Using Attack Construction to Measure IDS Security

Although our main focus is the automated attack construction on the Self-based IDS, an important contribution of our work is the demonstration of the attack construction as a systematic method to effectively measure the *resistance level* of an IDS against targeted attacks. This section describes our generalized framework for attacking IDSs, which can be used to measure the resistance levels of various IDS models.

3.5.1 Approach and General Framework

Our generalized framework is constructed upon the notion of "*attack-space search*". An attack space is a state representation derived from the properties and detection mechanism of the IDS *given that* one of its security assumptions is broken. We will show how actual attack spaces look like by means of examples in the subsections to follow. Once the attack search objective has been defined, the problem is then recast into how we can perform an efficient search on the attack space.

Our framework for evaluating the IDS security involves the following steps:

Step 1. Study and formalize the IDS' *definition and its security assumptions*;

Step 2. Find a *security assumption that can be made invalid*;

Step 3. Define the *attack space*;

Step 4. Establish the *search objective*;

Step 5. Construct a corresponding *search algorithm*;

Step 6. Apply the search on the *gathered experimental data*;

Step 7. (Optionally) modify the IDS model so that the search process becomes *computationally harder* or its search objective becomes *unfeasible* in the new attack space. This step thus increases the strength of the IDS against attacks which attempt to exploit the invalidated assumption.

Steps 1–5 above realize the objective of obtaining the attack crafting time as well as the constraints for a successful attack, which will be our main focuses here. Step 6 is the step which analyzes the attack occurrence given a normal profile dataset. Defining an attack space (Step 3) is an important step, which unfortunately requires some insights to be carried out. In defining an attack space, we can always come up with different representations for it. The efficiency of an attack construction, of course, depends on the right representation and its corresponding search algorithm.

Rather than giving detailed definition of the above steps, we instead show below how we can apply the framework. We revisit the attack construction on the Self-based IDS by describing the construction process within the developed framework. Details of the attack process are now fitted into the framework's defined steps. This makes the higher-level strategy of constructing mimicry attacks on the Self-based IDS become much clearer. Additionally, we then show how the framework can be applied to another IDS model.

3.5.2 Applying the Framework to Self-based IDS

Security Assumption

Self-based IDS (Stide) assumes that k-grams of a running program are good predictors of the program's behavior. More specifically, it assumes that:

> "It is *difficult* for an attacker to inject system call(s) into the program's system call trace without introducing *foreign k-grams* [10] which will be detected by the IDS."

Invalidating the Assumption

Stide's assumption above, however, can be made invalid by the following observation.

> "It is *quite possible* to combine two unrelated subtraces to make a longer subtrace that derives no foreign k-grams (i.e. all of the k-grams of the combined subtrace are present in the program's normal database)".

[10]Foreign k-grams are k-grams which are not present in the normal profile database.

We have shown this constructively using our pseudo subtrace construction described in Section 3.2.2.

Attack Space, Search Objective, and Algorithm

Here, we define the attack space for Stide that fulfills the requirements that:

1. Two k-grams derived from $(k + 1)$-length of the normal subtrace are related since they can be joined to produce (part of) a stealthy attack trace;
2. A pseudo subtrace can be constructed from two unrelated k-grams provided that they share a common substring of length $k - 1$.

We have used overlapping graph for the attack space (see Section 3.2.3), in which the requirement 1 is satisfied by the direct edges, whereas requirement 2 is realized by the pseudo edges in the overlapping graph. Accordingly, we also have shown the search objective on attack space and the construction algorithm in Section 3.2.

3.5.3 Applying the Framework to the FSA-based IDS

We now examine a different IDS model, which is based on FSA as proposed by Sekar et al. [146]. We have briefly summarized this IDS model in Section 2.2.3. In our attack construction, we make use of the technique to forge a Program Counter (PC) through a buffer overflow attack as suggested in [53]. Hence, we extend the results here by formalizing the search space for the IDS, and constructing the search algorithm using our framework.

To make the discussion of the attack construction below clearer, we include a sample FSA as shown in Figure 3.6, which was also used as an example in [146]. Recall that in the FSA-based IDS, each distinct Program Counter (PC) within the program segment, which is recovered by a stack traversal mechanism, is made as a state. System calls invoked on a PC are used as the labels for transitions from the corresponding state.

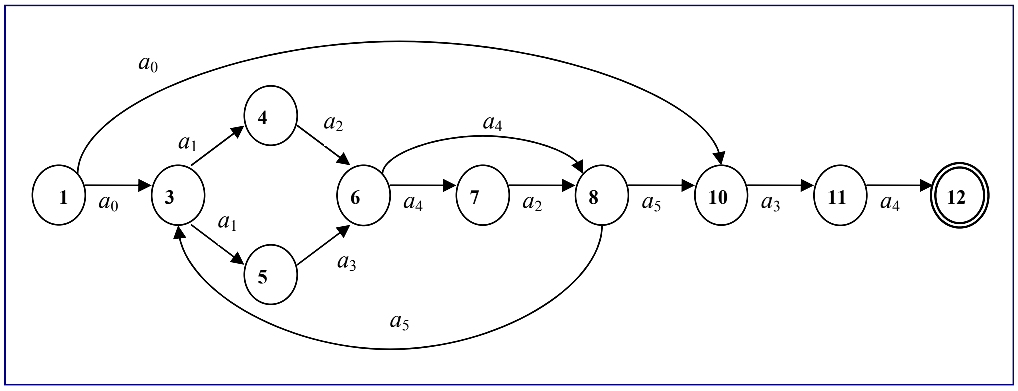

Figure 3.6: A sample FSA as a program's normal profile used in [146].

Security Assumption

The FSA-based IDS incorporates the PC information into the IDS model. It assumes that the PC information can be reliably obtained, and its inclusion can enhance the detection and security of the IDS. More specifically, we can state this security assumption as follows:

> "PC represents a program point where a system call is made within the program. It can be accurately obtained by using a stack traversal mechanism. The attacker *cannot* forge the stack (or other runtime environment information) in order for the stack traversal to recover the correct PC information."

Invalidating the Assumption

The work [53] points out how one can forge PC and return address through a buffer overflow attack.[11] The attack technique makes use of `ret` instead of `call` instruction to launch an exploit system call without inserting a stack frame into the stack. The hijacked control flow is then made to jump and re-execute the vulnerable operation[12] again and again until the complete exploit sequence has been executed. To achieve one system call execution, three forged stack frames are inserted into the stack in the following "push order": (i) a stack frame with a PC address of the vulnerable function; (ii) another stack frame with a PC that falls within the program's code region; and (iii) one with a PC of the exploit system call somewhere within the library. Note that since we assume the occurrence of buffer overflow in the user space, such a vulnerable operation does not involve any invocation of system call.

From our framework's viewpoint, the attack technique actually provides a mechanism to invalidate IDS' security assumption on the stack integrity and the correctness of the recovered PC addresses. Since the IDS does not check system call arguments, an attack will therefore successfully fool the IDS as long as its transitions (PCs and executed system calls) are accepted by the normal FSA.

Attack Space and Search Objective

Different from the Self-based IDS (Stide) where we need to define the overlapping graph, the FSA-based IDS generates an FSA as the normal representation. This FSA directly provides an attack space for the IDS. We define an FSA representing the IDS normal profile as a tuple $(\mathcal{Q}, S, \delta, q_0, \mathcal{Q}-\{q_\perp\})$, where: $\mathcal{Q}$ as the set of states (Program Counters); S as the set of all system calls; q_0 as the initial state; $q_\perp$ as a special state indicating that an anomaly has occurred; and $\delta : \mathcal{Q} \times S \to \mathcal{Q}$.[13]

[11] A technique proposed by Kruegel et al. [88] can additionally be used on the program's binary to automate the process of modifying code pointers following a system call so as to transfer the execution control back to the attacker for the invocation of subsequent attack system calls.

[12] We assume that the overflow occurs in the user space due to the program's use of an unsafe function.

[13] It is assumed that every state $q \neq q_\perp$ is an accepting state.

The state and the system call before the buffer overflow occurs are of particular importance. Let us call that state q_{prev} and the system call s_{prev}. Given an FSA $(\mathcal{Q}, S, \delta, q_0, \mathcal{Q}-\{q_\perp\})$, q_{prev}, s_{prev}, and an intended (basic) attack trace $A : \langle a_1, a_2, \ldots, a_m\rangle$, our search objective is thus to form the shortest stealthy attack path P_{min} on the FSA of the following form:

$$P_{min} = q_0 \xrightarrow{s_1} q_1 \xrightarrow{s_2} q_2 \; \ldots q_{n-2} \xrightarrow{s_{n-1}} q_{n-1} \xrightarrow{s_n} q_n \tag{3.3}$$

where: $n \geq m$; state q_0 is reached by a transition $q_{prev} \xrightarrow{s_{prev}} q_0$; the stealthy attack trace $L : \langle s_1, s_2, \ldots, s_{n-1}, s_n\rangle$ contains A as a subsequence; and all system calls in $\{L - A\}$ behave as no-ops with respect to A. The terminating state q_n is a special state after the invocation of s_n as the last system call in the exploit (usually `_exit()`), thus allowing the exploit to make a grateful exit.

The key to the attack construction is the fact that we can forge the PC for a state executing an exploit system call so that it appears to fall within the valid program's code region, and that the forged PC is connected to the previous valid state in the FSA. The PC forging is achieved by the crafting of the second stack frame in the corrupted stack after a buffer overflow attack occurs as discussed in [53].

Search Algorithm

To facilitate the search, we add an auxiliary subgraph similar to the one added for the Self-based IDS (Stide) attack construction. *Occurrence subgraph* $O = (W, Occ)$ is defined with W $(\subseteq S)$ as a set of unique system calls invoked in the FSA. For each node $w \in W$, we add an outgoing edge to every state reached by the invocation of the system call w, i.e. every state in the set $\{q \in \mathcal{Q} \,|\, q_{adj} \xrightarrow{w} q\}$. These edges form a set of edges Occ.

Our attack construction algorithm performs the search starting from all the states that are (directly) adjacent to q_{prev} by s_{prev}. We denote this set as $R = \{r \in \mathcal{Q} \mid q_{prev} \xrightarrow{s_{prev}} r\}$. Hence, there are $|R|$ search trees, each with state $r \in R$ as the root node q_0. Each node with depth i (for $1 \leq i \leq m$) in the search tree is a state $q_i \in \mathcal{Q}$ with $q_{adj\_i} \xrightarrow{a_i} q_i$, i.e. the state $q_i \in \mathcal{Q}$ where $(a_i, q_i) \in Occ$. This state q_i also represents a path $P_{q_i} : q_0, \pi_0, q_1, \pi_1, \ldots, q_{i-1}, \pi_{i-1}, q_i$ on the FSA, where π_i (for $0 \leq i < i-1$) denotes a subpath with zero or more states. The path P_{q_i} therefore corresponds to an execution of a system call trace containing $a_1, a_2, \ldots, a_i$ as its subsequence. From q_i (for $1 \leq i < m$), the search explores all the next states in $\{q_{i+1} \in \mathcal{Q} \,|\, q_i \rightsquigarrow q_{i+1} \;\wedge\; (a_{i+1}, q_{i+1}) \in Occ)\}$, where the connection relation ('$p \rightsquigarrow q$') denotes that states p and q on the FSA are connected. The branches from q_i (for $1 \leq i < m$) thus represent the choices of extending the path P_{q_i} into the path $P_{q_{i+1}}$, which corresponds to the extended execution trace containing $a_1, \ldots, a_i, a_{i+1}$ as its subsequence. A stealthy attack trace is found when we can reach q_m which corresponds an execution trace containing A as its subsequence.

Before outlining the construction algorithm, we illustrate the mimicry attack construction on the FSA-based IDS with the following example.

Example 3.5. Given the FSA in Figure 3.6 as the normal profile, suppose that the basic attack trace is $A : \langle a_5,\ a_1,\ a_4\rangle$, with state 3 being the last state before the buffer overflow occurs, and a_1 is the corresponding system call invoked. (Note that this sample FSA (from [146]) employs numbers to indicate states and a_i to indicate system calls.) Although the occurrence subgraph is not shown in Figure 3.6, we can easily form the following attack path by using the search construction algorithm outlined above:

$$4 \xrightarrow{a_2} 6 \xrightarrow{a_4} 8 \xrightarrow{\boxed{a_5}} 3 \xrightarrow{\boxed{a_1}} 4 \xrightarrow{a_2} 6 \xrightarrow{\boxed{a_4}} 7.$$

The exploit system calls performing real attacks are shown inside boxes, whereas the rest are no-ops.

Algorithm 3.2 Attack construction on the FSA-based IDS under code-injection scenario

```
Input:
- Normal-profile FSA (as a directed graph);
- Basic attack trace $A : \langle a_1, a_2, a_3, ..., a_m\rangle$;
- The last state before the buffer overflow occurrence ($q_{prev}$);
- The last system call before the first buffer overflow occurs ($s_{prev}$).
Output:
- The shortest stealthy attack path $P_{min} : q_0 \xrightarrow{s_1} q_1 \xrightarrow{s_2} q_2 \ldots q_{n-2} \xrightarrow{s_{n-1}} q_{n-1} \xrightarrow{s_n} q_n$;
- Or failure, if no solution trace can be found.

1. Construct the *all-pair shortest-path distance table*, with the following initialization:
     Between two adjacent states $p$ and $q$, set *distance*$(p, q)$:=1.
     If two states $p$ and $q$ are not connected Then *distance*$(p, q)$:= $\infty$.
2. Set *Min_distance*:= $\infty$ and *Min_path*:=$\langle\rangle$.
3. For all $r \in \{r \in \mathcal{Q} \mid q_{prev} \xrightarrow{s_{prev}} r\}$:
     (Perform branch-and-bound search on the search tree of $r$ as follows:)
     Set $q_o$:=$r$.
     *current_cost*:= 0 (it keeps track of the distance from $q_0$ to $q_i$ in the path being explored)
     For all $i$:=1..$m$ choose $q_i$ from $\{q_i \in \mathcal{Q} \mid (a_i, q_i) \in Occ\}$:
       - If *distance*$(q_{i-1},\ q_i) = \infty$ Then backtrack.
       - Add *distance*$(q_{i-1},\ q_i)$ to *current_cost*.
       - If *current_cost* $\geq$ *Min_distance* Then backtrack.
       - If complete solution is found ($i = m$) Then
           If *current_cost* $<$ *Min_distance* Then
             *Min_distance* := *current_cost*;
             *Min_path* := *current_path*.
       (Recall that a path includes zero or more states in between $q_{i-1}$ and $q_i$ for $2 \leq i \leq m$.)
4. Once the search tree is fully explored:
   If *Min_distance* = $\infty$ Then return failure;
   Else return *Min_path* (as the shortest stealthy attack path $P_{min}$).
```

As before, we employ a branch-and-bound strategy to prune a constructed path which exceeds the best solution found so far. The shortest distance between any two states is stored in a "distance table". It is employed both to test connectivity between two states and also to assist in pruning for the branch-and-bound search. The algorithm for constructing attack on the FSA-based IDS is a variation of the one on the Self-based IDS (see Algorithm 3.1). They both make use of the same branch-and-bound search procedure. However, we now use a different search tree definition to be explored. Algorithm 3.2 shows a sketch of the algorithm to automatically construct the shortest stealthy attack execution trace on the FSA-based IDS under code-injection scenario.

3.6 Chapter Summary

We have presented an efficient algorithm for automated mimicry attack construction on the Self-based IDS (Stide) and its variant model (GRA-SELF). Using several real programs and exploits, we also have shown the practicality of the attack construction, with execution times of at most a few seconds even for the relatively large window sizes. We also have shown how the construction method can be generalized into an approach to evaluate the robustness of an IDS against targeted attacks. The attack construction time is particularly useful to evaluate the resistance of an IDS against attacks in a zero-day attack setting. If we know that crafting an attack is possible but computationally hard, then we can have greater confidence that the IDS can function sufficiently well as the second line of defense during the program's unpatched time interval. This result is important since it shows how a quantitative measurement on IDS security strength against targeted attacks can be established.

Chapter 4

Improving Self-based IDS using Privilege and Argument Abstraction

The previous chapter has shown that the Self-based IDS is susceptible to mimicry attacks. In this chapter, we consider a simple enhancement to the Self-based IDS, which can either prevent mimicry attacks or make them more difficult. The enhancement makes use of system call arguments and process privilege information. By using system call arguments, a data-flow aspect is thus incorporated into the IDS model. Taking the arguments into account is also important for mitigating non control-flow attacks, whose threat has been shown in [36] to be sufficiently realistic in practice.

Besides being applicable to the Self-based IDS, our improved model can also strengthen various system-call monitoring IDS, such as the FSA-based [146] and the VtPath model [48]. To simplify our discussion, we will elaborate the enhancement to the Stide model of Self-based IDS, which keeps track of system call numbers in terms of a set of (full sequence) k-grams [67].

Our enhanced model has been published in [158], and also surveyed in [50]. The remainder of this chapter is organized as follows. Section 4.1 discusses the related works on improving the Self-based IDS using system call arguments. Section 4.2 elaborates our scheme which abstracts system call arguments and process privilege information. Our experiments are reported in Section 4.3, and the discussions on the experimental results are given in Section 4.4. Section 4.5 finally gives a summary of this chapter.

4.1 Related Works on Data-Flow based IDS

The general idea of analyzing arguments of operations for detecting behavior deviance appeared in a number of works. For example, [105] showed how the use of enriched command-line data can enhance the detection of masqueraders. Our work reported in this chapter is based on the established Self-based IDS model (described in Section 2.2), and focuses on system call arguments. Below, we discuss several IDS models which also make use of system call arguments.

Kruegel et al. [91] made use of statistical analysis of system call arguments which can be used to evaluate features of the arguments such as string length, string character distribution, structural inference and token finder. It is however unclear whether the approach is sufficiently robust against targeted attacks such as mimicry attacks [18]. In addition, the work solely examined the arguments and disregarded the code flow altogether. The work was later extended in [123] by additionally using a Bayesian network to combine individual model scores into a single aggregate score. Another work which made use of machine learning technique on system call trace is [167]. It proposed a scheme to learn the attributes of system call arguments using a rule learning algorithm.

Taking a static analysis-based approach, Giffin et al. [57] made use of inter-procedural data-flow analysis to model statically-known arguments passed to system calls. Due to the nature of static analysis on program behavior, the scheme was only able to detect attacks that cause a program's runtime behavior to deviate from its statically extracted data-flow model. In our work, we focus on the gray-box and the black-box techniques which do not require the analysis of source codes or the binaries of the executables. Yet, the white-box technique and gray/black-box detector can complement each other as discussed in [95].

In a work proposed after ours, Bhatkar et al. [23] modeled the temporal aspects of data flow by performing a learning on system call arguments. The learning establishes unary relations which correlate arguments with constant values, and binary relations of which each correlates two arguments of system calls on two different PC locations. Although our proposed technique does not relate arguments from two system calls, it has a particular strength in that the supplied simple policy represents an effective security model to prevent potentially dangerous operations according to their *direct effects* on the host's security. The work [23] did share the same conclusion as ours here that it is necessary to include data-flow model to be layered into the code-based IDS model.

Finally, there exist sandboxing techniques which also make use of system-call argument checking. The systrace scheme [135] uses system call policies to specify that certain system calls with specific arguments can be allowed or denied. This can be viewed as a the Self-based IDS with a window size of one, which is then enriched with system-call argument checking. We additionally remark that, compared to a basic policy definition used in [135], our policy definition (see Section 4.2.1) allows for a generalized mapping of arguments. Our scheme allows operations on files/directories to be grouped together into a set of specified categories according to their impact on the host's security.

4.2 Privilege and Argument Categorization (PAC) based IDS

In Unix, every process environment contains credentials which are evaluated by the OS access control mechanism when the process makes a system call. The credentials that determine the current privileges of a process are its effective user-id (euid) and effective group-id (egid). The euid/egid is either the actual real uid/gid of the user, or the one changed by invoking a setuid/setgid executable. Hence, euid and egid are simply a subset of all the user and group-id values defined in a system.

We propose to enhance k-grams to include not only the system number but also the (abstracted) information about the euid, egid and system call arguments. We call our enhanced IDS model the *Privilege and Argument Categorization (PAC)-based IDS*. It is common for attacks to attempt exploiting programs while they are running in a privileged mode, or to elevate the privilege of the hijacked programs prior to performing damaging operations. The idea behind our privilege profiling is that such attacks can be detected if the corresponding executed system calls are unprivileged in the normal trace(s). This is particularly useful if a program conforms to a good setuid programming practice, which generally drops privileges as soon as they are no longer required. Rather than using the actual values, we abstract the euid, egid and system call arguments into categories based on a configuration specification. This is mainly to reduce the false positives which can be higher since the number of possible values is much greater. The abstraction technique also provides flexibility to group the arguments and the privileges together in terms of their importance/sensitivity levels.

4.2.1 Privilege and Argument Categorization

Formally, we can represent the privilege and argument categorization in the OS model with the following *mapping* functions:

- Function $EuidCat : U \rightarrow U'$, where: U = the set of euid values, and $U' \subset \mathbb{N}_0$ (the set of natural numbers starting from zero).
- Function $EgidCat : G \rightarrow G'$, where: G = the set of egid values, and $G' \subset \mathbb{N}_0$.
- Let $S \subset \mathbb{N}_0$ be the set of system call numbers in the OS model. For each $s \in S$, function $ArgCat_s : A_{s,1} \times A_{s,2} \times \ldots \times A_{s,no\_of\_args(s)} \rightarrow C_s$, where $A_{s,i}$ ($1 \leq i \leq no\_of\_args(s)$) = the set of possible entries for i-th argument of the system call s, and $C_s \subset \mathbb{N}_0$ as the categorized value for the specified arguments of system call s.

We enhance the Self-based IDS using the privilege and argument categorization technique by extending the *k-gram definition* from a sequence of system calls of length k, into a sequence of tuples ⟨categorized euid, categorized egid, system call, system call argument category⟩ of length k. Since there are two variants of the Self-based IDS defined earlier, namely SET-SELF and GRA-SELF (see Section 3.3.1), we also can apply the categorization technique to result into two following PAC-based IDS models:

- SET-PAC: which stores the normal profile as a *set* of enhanced k-grams. This model thus enhances SET-SELF.
- GRA-PAC: which stores the normal profile as a *graph* of enhanced k-grams. Note that this model, which is enhancement of GRA-SELF, is a stricter model than SET-PAC.

As mentioned earlier, we can formalize system-call monitoring IDS as an FSA [180, 53, 146]. Let us define the two PAC-based IDS models more precisely. We first define $s_z \in \mathbb{N}_0$ and $s_z \notin S$ to denote the "*sentinel*" system call. We then define the extended system call set $S^+ = S \cup \{s_z\}$. The automaton based on our PAC-based IDS can be defined as $(\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q} - \{q_\perp\})$, with:

- The set of states $\mathcal{Q} = \{q_0, q_\perp\} \cup \Sigma^k$, with k as the size of sliding window.
- The set of input $\Sigma = U' \times G' \times S^+ \times C$, with $C = \bigcup_{s \in S} C_s$.
- The initial state q_0.
- A special state $q_\perp$ which indicates that an anomaly has occurred. The other states $\mathcal{Q} - \{q_\perp\}$ are considered as accepting states.[1]
- A transition $(\sigma_1, \sigma_2, \ldots, \sigma_{k-1}, \sigma_k) \xrightarrow{\sigma_{k+1}} (\sigma_2, \sigma_3, \ldots, \sigma_k, \sigma_{k+1})$, where $\sigma_i \in \Sigma$ (for $1 \leq i \leq k{+}1$), is added into the FSA of the GRA-PAC model if there exists a subtrace $(\sigma_1, \sigma_2, \ldots, \sigma_{k-1}, \sigma_k, \sigma_{k+1})$ in the normal trace. As for the FSA of the SET-PAC model, we add a transition $(\sigma_1, \sigma_2, \ldots, \sigma_{k-1}, \sigma_k) \xrightarrow{\sigma_{k+1}} (\sigma_2, \sigma_3, \ldots, \sigma_k, \sigma_{k+1})$ for any two states $q = (\sigma_1, \sigma_2, \ldots, \sigma_{k-1}, \sigma_k)$ and $q' = (\sigma_2, \sigma_3, \ldots, \sigma_k, \sigma_{k+1})$ in the FSA. As can be seen, a transition in the FSA of GRA-PAC also exists in that of SET-PAC. The converse is however not necessarily true.

Here, we assume that a state $q \in \mathcal{Q} - \{q_0, q_\perp\}$ is always a k-tuple, i.e. $(\sigma_1, \ldots, \sigma_k)$. A complication however arises in the beginning or end of the normal trace, where only $i < k$ of σ_i entries are available. A workaround to this is by defining a "sentinel" entry $\sigma_z = (u', g', s_z, c)$, with s_z as the sentinel system call; and u', g', and c being arbitrarily selected entries in U', G' and C, respectively. We can add this sentinel as padding entries either in the first $k-1$ k-grams of the trace (as suggested in [74]), or the last $k-1$ k-grams as discussed in Chapter 3.

[1]As elaborated in [180], every state $q \neq q_\perp$ is an accepting state. The special state $q_\perp$ is non-accepting and contains a self-loop $q_\perp \xrightarrow{\sigma} q_\perp$ for every $\sigma \in \Sigma$. When a state q contains no outgoing transitions on $\sigma \in \Sigma$, we add an implicit transition $q \xrightarrow{\sigma} q_\perp$. Note that this FSA definition takes a single *transition mismatch*, i.e. a non-existent transition from a state $q \neq q_\perp$, as an intrusion. In practice, mainly to keep false positive rate reasonably low, the IDS raises an alarm only if the output of its anomaly-signal measurement function (e.g. Locality Frame Count) has reached a specified threshold value.

As can be easily observed, the new automaton is richer in comparison to that of the Self-based IDS. In the basic Self-based IDS, the alphabet (Σ) is the the set of possible system call numbers S. In PAC-based IDS, the alphabet is now defined as a tuple $U' \times G' \times S^{+} \times C$. Note that while we have focused on Unix, the categorization approach can be extended to other OSes.

4.2.2 A Simple Category Specification Scheme

We now give a simple scheme for defining the abstraction and the categories. The category specification is constructed by taking into account the importance or the sensitivity level of files/directories in the underlying OS from the security standpoint. The main goal of the specification is to separate operations which have potential security risks from the benign ones.

A fragment of an example specification is given in Figure 4.1. It consists of four sections: *euid abstraction*, *egid abstraction*, *argument abstraction* and *illegal transition*, which are explained more below. For each section, the category specifications are processed in a sequential manner, from the start to the end of the definition. In this fashion, the more specific mappings are specified first and the most general ones last. This is similar to the ordering in firewall configuration files. Note that this example is only meant to be illustrative on how the category specification works. A more complete example specification can be seen in Appendix A.

```
# EUID Abstraction Section
# Format:  <categorized-euid>:<euid1>,<euid2>,...
0:0
1:2000,2001,2003
100:*
# EGID Abstraction Section
# Format:  <categorized-egid>:<egid1>,<egid2>,...
0:0
1:1,2,3,5
100:*
# Argument Abstraction Section
# Format:  <cat-value> <syscall> <arg1> <arg2> <arg3> ...
1 open p=/etc/passwd o=O_WRONLY|o=O_RDRW *
2 open p=/etc/shadow o=O_WRONLY|o=O_RDRW *
18 open * * *
1 chmod p=/etc/{passwd,shadow,group,hosts.equiv}|p=/proc/kmem *
# Illegal Transition Section
# Format:  <syscall> <cat-values> [<cat-euid>,<cat-egid>]*
open [1..6,8-11,13-15] 0,* *,0
{chmod,fchmod,chown,fchown,lchown,mknod,unlink,init_module,execve} 1 0,* *,0
```

Figure 4.1: An example of category specification for the PAC-based IDS.

Privilege Abstraction Section

The euid and egid sections are meant to provide the actual value mapping for $EuidCat : U \rightarrow U'$ and $EgidCat : G \rightarrow G'$ respectively. The example specification uses the following syntax for euid and egid:

$$\begin{aligned} \langle u'_i \rangle &: \langle u_{i1} \rangle, \langle u_{i2} \rangle, \ldots, \langle u_{in} \rangle \\ \langle g'_i \rangle &: \langle g_{i1} \rangle, \langle g_{i2} \rangle, \ldots, \langle g_{in} \rangle \end{aligned} \tag{4.1}$$

where: $u'_i \in U'$, $u_{ij} \in U$, $g'_i \in G'$ and $g_{ij} \in G$.

To ensure that $EuidCat/EgidCat$ is a total mapping, a special entry "*" is employed to indicate other (remaining) euids/egids so that the mapping satisfies the requirement for a function. As euid=0 and egid=0 signify important privileges in Unix, each of them should have a distinguished mapping.

Argument Abstraction Section

Each entry in this section maps a system call together with its corresponding arguments into a number representing its category. We now briefly discuss some considerations in creating a definition:

- The approach taken in our work is to focus on a subset of system calls $S' \subset S$ which should be checked in order to prevent attacks from gaining full control of the system. Our choice for the system call subset S' is based on the work of Bernaschi et al. [22] which classified Unix system calls according to their *threat level* with respect to the system penetration. Here, we consider S' to be the system calls in the *Threat-level 1 Category* in [22], namely: `open()`, `chmod()`, `fchmod()`, `chown()`, `fchown()`, `lchown()`, `rename()`, `link()`, `unlink()`, `symlink()`, `mount()`, `mknod()`, `init_module()` and `execve()` [22].

  Other system calls in $S - S'$, which are not defined in the specification are simply mapped to a unique default value. We do not address the issues raised by the system calls in the Threat-level 2 (which can be used for a denial of service) and the Threat-level 3 (which can be used for subverting the invoking process). Otherwise, we might need a richer IDS model which also deals with issues such as memory/storage consumption metering and file usage pattern, which are beyond the scope of this work. One advantage of the system call subset, which is approximately 10% of the total number of system calls, is that it reduces the monitoring overheads which is important when the IDS is to run in an on-line fashion.

  Bernaschi et al. also grouped the setuid/setgid system call family into the Threat-level 1 list. However, we take a different approach here by capturing the effect of the setuid/setgid system call family as the *changes in process credential* *values* —in the form of (euid, egid) pairs— to form part of the state information in our enhanced IDS model.

- Given a system call $s' \in S'$, a simple approach for the choice of abstraction is to ensure that any critical operations on security-sensitive objects are mapped to a value different from non or less critical ones.
- To indicate a pathname in our specification, we use a special notation: `p=`<*pathname*>. Since pathnames in Unix are not unique, they have to be made canonical by turning them into a normalized absolute pathname (see also [135]).
- Additionally, we also make use of a special tag `o=`<*option_token*> to indicate that the *option_token* is to appear as an entry in a list of entries in the corresponding argument space. An example is `oflag` as the second argument of `open()`, where `O_RDWR` can co-exist with other modes such as `O_CREAT` or `O_APPEND`. A categorization specification entry is triggered if the specified *option_token* is present in the list.

4.2.3 Disallowing Transitions

It is also useful to specify the transitions that can immediately lead to "bad states". The idea is to identify those singleton system calls with the corresponding privileges which can be sufficient to compromise the system's security. An example would be the operation of `chown()` on `/etc/passwd` with root privileges. Thus, the usual way of measuring anomaly signal by means of Locality Frame Count (LFC) function as used in [152] is inadequate. This can also be used as an enhancement to the access control to actually deny such a system call invocation in a running program.

In the ileggal transition section (see Figure 4.1), the syntax for an entry line is:

$$s' \quad c \quad [u', g']^* \tag{4.2}$$

where: $s' \in S'$; c is the abstracted value for the arguments of s'; u' and g' are the abstracted user and group privileges respectively. We denote the set of bad transitions as D_0. Disallowing D_0, however, may be too strict, and needs to be adjusted with respect to the normal traces. When extracting the normal profile from the normal trace, we can construct the set D_N from all $\sigma \in D_0$ which are also present in the normal traces. The final adjusted negative transitions are: $D = D_0 - D_N$.

In the detection stage, the PAC-based IDS flags any system call execution that matches an illegal transition $\sigma_d \in D$ as intrusive.[2] Since the PAC-based IDS can function as an IPS, it may also prevent such operation from being executed.

[2]The PAC-based IDS can additionally log any execution of $\sigma \in D_N$.

4.3 Experiments on PAC-based IDS

Our aim here is to evaluate whether the inclusion of abstracted process privilege and system call arguments in the PAC-based IDS can make it more resistant against mimicry attacks. In addition, we also would like to evaluate the behavior of the PAC-based IDS, particularly whether there is any increase in its number of false positives.

4.3.1 Attack Construction on PAC-based IDS

We extend the attack construction algorithm on the Self-based IDS (described in Section 3.2.5) to also work with the PAC-based IDS. This can be easily done since we just need to extend the k system call labels in each node's state in the overlapping graph G by adding the euid, the egid and the argument category value.

To compare the PAC-based IDS with the two variants of the Self-based IDS (i.e. SET-SELF and GRA-SELF), we evaluate the attack construction on the PAC-based IDS using the three vulnerable programs used in evaluating SET-SELF and GRA-SELF (see Section 3.3). Recall again that we can successfully construct stealthy attacks on the three vulnerable programs protected by both SET-SELF and GRA-SELF. These vulnerable programs are:

- Traceroute: with the basic attack trace: `open()`, `write()`, `close()`, `_exit()`.
- JOE: with the basic attack trace: `open()`, `write()`, `close()`, `_exit()`.
- WU-FTPD: with the same following basic attack trace as in [180]: `setreuid()` `chroot()`, `chdir()`, `chroot()`, `open()`, `write()`, `close()`, `_exit()`.

In the trace generation, we purposely set the euid/egid value of all system call entries in the normal trace to 0 (root). In other words, we assume a poorly written setuid program. This is to provide the worst-case condition for attacks to occur, since system calls in the attack trace are typically to be executed with root privilege.

In these experiments, the argument category specification makes use of the "dangerous" system call subset discussed in Section 4.2. For the choice of arguments, from [22, Table 4], we can see that the dangerous arguments for system calls in S' are mainly files/directories. The work by Garfinkel and Spafford [54, Appendix B] gave a comprehensive list of security sensitive and important files/directories that one might want to consider monitoring in Unix. In our experiments, we pay special attention to several security critical files in the Unix/Linux environment, which are listed in Table 4.1. For simplicity, we omit most of the system configuration files in `/etc` (such as: `/etc/inetd.conf, /etc/hosts, /etc/cron/*`) and devices files in `/dev`. We however include entries for various directories commonly found in the Unix/Linux file system hierarchy, which conforms to the *Filesystem Hierarchy Standard* (`http://www.pathname.com/fhs/pub/fhs-2.3.`

File ID	**File name**
F_1	`/etc/passwd`
F_2	`/etc/shadow`
F_3	`/etc/group`
F_4	`/proc/kmem`
F_5	`hosts.equiv`

Table 4.1: Several important files in Unix/Linux to be protected from security viewpoint.

`html`). While one can use a more detailed specification, ours is already sufficient to show an increase in the robustness of the IDS.

Based on our experiments for window sizes from k=5 to 11 using the three sample programs above, we found that no stealthy attack could be constructed on both SET-PAC and GRA-PAC under both the code-injection and the trojan attack scenarios. Table 4.2–4.4 show the running times for the attack constructions on SET-PAC and GRA-PAC using the three sample programs. Note that if no attack can be constructed on SET-PAC, then no attack can be constructed on GRA-PAC either. This is since GRA-PAC is a stricter model than SET-PAC (see their definitions in Section 4.2.1). As can be seen, the attack constructions on SET-PAC and GRA-PAC are almost as efficient as those on SET-IDS and GRA-IDS. The construction times on all considered cases are at most a few seconds even for the relatively large sliding-window sizes (k=9–11).

Traceroute Search on PAC-based IDS	**Attack Construction Time (User+Sys)**						
	k**=5**	k**=6**	k**=7**	k**=8**	k**=9**	k**=10**	k**=11**
SET-PAC (Trojan-Attack)	0.020s	0.025s	0.026s	0.031s	0.032s	0.033s	0.031s
GRA-PAC (Trojan-Attack)	0.019s	0.022s	0.027s	0.028s	0.031s	0.029s	0.029s
SET-PAC (Code-Injection)	0.022s	0.025s	0.026s	0.033s	0.034s	0.037s	0.032s
GRA-PAC (Code-Injection)	0.021s	0.026s	0.028s	0.027s	0.032s	0.027s	0.029
Average Time	0.021s	0.025s	0.027s	0.030s	0.032s	0.032s	0.030s

Table 4.2: Execution times for the attack constructions on the PAC-based IDSs using traceroute program (used earlier in Section 3.3) with k=5 to 11. No stealthy attack trace can be found on SET-PAC and GRA-PAC on all the examined cases.

JOE Search on PAC-based IDS	**Attack Construction Time (User+Sys)**						
	k**=5**	k**=6**	k**=7**	k**=8**	k**=9**	k**=10**	k**=11**
SET-PAC (Trojan-Attack)	0.048s	0.051s	0.057s	0.064s	0.077s	0.089s	0.102s
GRA-PAC (Trojan-Attack)	0.047s	0.050s	0.056s	0.061s	0.071s	0.085s	0.099s
SET-PAC (Code-Injection)	0.049s	0.052s	0.059s	0.067s	0.077s	0.088s	0.102s
GRA-PAC (Code-Injection)	0.048s	0.049s	0.058s	0.064s	0.074s	0.085s	0.095s
Average Time	0.048s	0.051s	0.058s	0.064s	0.075s	0.087s	0.100s

Table 4.3: Execution times for the attack constructions on the PAC-based IDSs using JOE with k=5 to 11. No stealthy attack trace can be found on all the examined cases.

WU-FTPD Search on PAC-based IDS	**Attack Construction Time (User+Sys)**						
	k**=5**	k**=6**	k**=7**	k**=8**	k**=9**	k**=10**	k**=11**
SET-PAC (Trojan-Attack)	0.574s	0.837s	0.997s	1.232s	1.494s	1.916s	2.139s
GRA-PAC (Trojan-Attack)	0.581s	0.846s	1.000s	1.233s	1.523s	1.723s	1.984s
SET-PAC (Code-Injection)	0.577s	0.835s	0.994s	1.238s	1.490s	1.774s	2.118s
GRA-PAC (Code-Injection)	0.590s	0.797s	0.998s	1.354s	1.521s	1.718s	1.971s
Average Time	0.581s	0.829s	0.997s	1.264s	1.507s	1.783s	2.053s

Table 4.4: Execution times for the attack constructions on the PAC-based IDSs using WU-FTPD with k=5 to 11. No stealthy attack trace can be found on all the cases.

4.3.2 Behavior of PAC-based IDS

We now experiment with the PAC-based IDS against various mimicry attack strategies, and also investigate its false positives.

Resistance Against Various Attacks

Having shown that the PAC-based IDS can better withstand mimicry attacks on the three sample programs when the attacker attempts to write an entry into the file `/etc/shadow`, we now evaluate the IDS against a number of different *attack strategies*.

Attack ID	**Operation (respectively)**
$A_1 - A_5$	Open and write an entry into F_1, F_2, F_3, F_4, F_5
$A_6 - A_{10}$	Chmod on F_1, F_2, F_3, F_4, F_5
$A_{11} - A_{15}$	Fchmod on F_1, F_2, F_3, F_4, F_5
$A_{16} - A_{20}$	Chown on F_1, F_2, F_3, F_4, F_5
$A_{21} - A_{25}$	Fchown on F_1, F_2, F_3, F_4, F_5
$A_{26} - A_{30}$	Lchown on F_1, F_2, F_3, F_4, F_5
$A_{31} - A_{35}$	Rename F_1, F_2, F_3, F_4, F_5 into some other file
$A_{36} - A_{40}$	Rename some other file into F_1, F_2, F_3, F_4, F_5
$A_{41} - A_{45}$	Link F_1, F_2, F_3, F_4, F_5 into some other file
$A_{46} - A_{50}$	Link some other file into F_1, F_2, F_3, F_4, F_5
$A_{51} - A_{55}$	Unlink F_1, F_2, F_3, F_4, F_5
$A_{56} - A_{60}$	Mknod F_1, F_2, F_3, F_4, F_5
A_{61}	Execve a shell or command

Table 4.5: Attack strategies (on important files listed in Table 4.1) to be prevented.

In Table 4.5, we list a number of common attack strategies in the Unix/Linux environment on the files listed in Table 4.1 when the system calls are executed with a root euid/egid privilege. While the list is not comprehensive, it is sufficient to demonstrate the improvements in the IDS' resistance level. We choose the traceroute program for this experiment. The experiment was done on normal traces described earlier (2,789 system calls) with the sliding-window size (k) set to 5, which is a relatively short one. We found that all the attack strategies listed in Table 4.5 fail on the tested normal traces even in the trojan attack scenario. For most of the strategies (A_6–A_{61}), the attacks fail because the needed attack system calls do not appear in the normal traces.[3] In attacks A_1–A_5, given the category specification, the searches fail because the normal trace does not contain the required system call argument categories.

False Positives

Here, we give some preliminary results of comparing the PAC-based IDS and the baseline Self-based IDS in terms of the number of false positives. We choose two programs, ls and traceroute in Red Hat Linux 7.3. For each program, we record 10 traces from 10 different program runs. We then randomly choose one to be tested against the other 9. We compare SET-PAC with SET-SELF which also stores its normal profile as a set of k-grams. The comparison results for k=5 to 11 are shown in Table 4.6. Here, we simply measure the number of foreign k-grams. As can be seen, the enhancement does not increase the number of false positives.

	traceroute		**ls**	
k	**SET-SELF**	**SET-PAC**	**SET-SELF**	**SET-PAC**
5	0	0	2	2
6	0	0	2	2
7	0	0	2	2
8	0	0	2	2
9	1	1	2	2
10	2	2	2	2
11	3	3	2	2

Table 4.6: Number of foreign k-grams in traceroute and ls program with window sizes k=5 to 11. SET-SELF refers to the Self-based IDS (Stide), whereas SET-PAC indicates our new PAC-based IDS that stores its normal profile as a set of enhanced k-grams.

4.4 Discussions

We have shown that the PAC-based IDS model is more resistant to mimicry attacks since the basic attacks in our experiments could not be turned into the mimicry attacks.

Furthermore, we have the following observations:

- As we have shown that the Self-based IDS with certain sizes of the sliding window, such as k=6 to 8 as suggested in [67, 152, 74], is insufficient, other improvements to the IDS model are thus necessary. Our privilege and argument abstraction technique appears to be able to answer the need to make the Self-based IDS more robust. In addition, one can always specify his/her own specification rules for the PAC-based IDS in order to suit a particular program and its operational setting in preventing possible attacks.

[3]These attack strategies thus would not work to attack SET-SELF and GRA-SELF either.

- Our experimental results show that with the given basic attacks, it was not possible to turn them into the mimicry attacks on the PAC-based IDS although it was possible to do so in the two Self-based IDS models (SET-SELF and GRA-SELF). Most results that we are aware of for analyzing an IDS, in particular using the mimicry attacks, are usually of the negative variety in that they show the existence of attacks or ways of attacking the IDS. It is significant that our result here is a positive one, since it shows that certain systematic attacks fail to work on the PAC-based IDS. However, we do not guarantee that no attacks are possible since the evaluation is relative with respect to a given basic attack and the normal traces. The question of a security guarantee is in fact an open problem in the most IDS models, and we argue that our work here paves a way towards the more robust evaluation methods.
- The false positive results of the experiments are encouraging as we find that improving the IDS with a more fine-grained detection mechanism does not increase the number of false positives over the baseline IDS.
- The privilege and argument abstraction technique can also be applied to the other gray-box IDS models, such as the FSA-based model [146]. In this new model, we have the set of states $\mathcal{Q} = \{q_0, q_\perp\} \cup \{U' \times G' \times P\}$ with $P =$ set of possible Program Counter values, and $\Sigma = S \times C$. The transition is thus enhanced using a 2-tuple of the system call number and the argument category value.

4.5 Chapter Summary

We have proposed an extension to the Self-based IDS using privilege and argument abstraction. The new IDS, called the PAC-based IDS, is a more-fine grained model which takes into account the security aspects of the operations. We argue that this extension is both simple to use and also makes the IDS more robust. Our experimental results show that mimicry attacks, which were able to work in the baseline Self-based IDS, fail in the PAC-based IDS using both the SET-PAC and GRA-PAC variants. We also have some evidence that the increase in detection accuracy does not lead to more mis-predictions.

An important advantage of our IDS extension is its *simplicity*. Directly using the arguments or process credentials will not work well due to the possibility of increasing the number of false positives. However, a simple classification technique, which abstracts away the irrelevant information and takes into account a security model of the OS, appears to work well. Furthermore, the simplicity means that the scheme can be easily integrated into various IDS models to make them significantly more robust.

Chapter 5

Lightweight Executable Authentication Protection

Ensuring that a host only executes codes from trusted sources, which were also unmodified while in transit and stored on the host's file system, is an important step in the Program Protection Life Cycle (PPLC). It helps establish a stronger trust on good program execution. Moreover, it is also beneficial to the OS since it can help protect the OS against malicious kernel drivers, i.e. attacks attempting to load malware into the kernel. Section 2.3 has provided some background on the executable authentication problem, as well as briefly mentioned existing authentication systems and issues in providing executable authentication on Windows. As mentioned before, we refer to an executable code stored in the file system as a *binary*.

Despite its important role, binary authentication protection mechanism is yet to be commonly incorporated as a standard in-kernel mechanism in popular commercial-off-the-shelf OSes, such as Windows and Solaris. Demonstrating the practicality of a binary authentication system on a complex commodity OS is thus significant. This is since many of the security problems on commodity OSes, particularly Windows, stem from a host's inability to distinguish between trusted and untrusted software. In this chapter, through a proof-of-concept implementation called BinAuth, we demonstrate the practicality of establishing a binary authentication system on Windows. BinAuth is a practical, lightweight and in-kernel mandatory binary authentication system. It is lightweight in terms of both performance overheads and its reduced reliance on PKI for the certificate revocation service. Our benchmarking shows that the overhead of BinAuth can be quite low, around 2%, with a caching strategy.

In addition to proposing BinAuth, we also develop a framework for comparing binary authentication systems based on a number of key design factors. We use this framework to compare a variety of existing authentication systems, and identify the desirable properties of an robust and efficient binary authentication system. In fact, we specifically design BinAuth to meet these desirable properties. Besides this framework, we additionally propose a simple scheme called Software_ID which leverages on the authentication infrastructure. This scheme is similar to the Common Platform Enumeration (CPE) initiative [115], which can help simplify the task of vulnerability alert management as discussed more in Chapter 6.

Most of the results reported in this chapter have been published in our collaborative works [66, 186]. The author of this thesis focuses mainly on the design of BinAuth including the notion of Software_ID, the security analysis of MAC-based authentication (as given in Section 5.4), and on a comparative analysis of the authentication schemes. The implementation of BinAuth on Windows was done by other team members. The remainder of this chapter is organized as follows. We elaborate the security goals to be achieved in Section 5.1. Section 5.2 outlines our framework for comparing authentication systems. Section 5.3 describes the design, and briefly mentions some implementation aspects of BinAuth. We analyze the security of BinAuth in Section 5.4, and provide the benchmarking results on BinAuth in Section 5.5. Section 5.6 discusses the Software_ID scheme which can take advantage of BinAuth. Section 5.7 summarizes this chapter.

5.1 Security Goals

Let us first establish the security goals of a binary authentication system. Throughout this chapter, we define "*administrator*" to be the trusted privileged user, such as system administrator in Windows or superuser in Unix/Linux. We define the security goals of a binary authentication system as follows. A binary authentication system aims to ensure that a binary file B stored on a host is allowed for execution at time $t_{invoked}$ only if the following *security goals* are satisfied:

- (G_1) ***Trusted Origin***: the binary originates from a source (developer) that the host trusts. The host trusts the developer that the binary produced by the latter is non-malicious in nature and has no intention to violate any security policies of the host (beyond the program's known functionalities) or any acceptable use policies.
- (G_2) ***Binary-Content Integrity***: the *content* of the binary was unmodified while in transit (during the software distribution) and while stored on the host's file system. This goal ensures that the executed binary has not been illegally tampered with. For example, `cmd.exe` in a Windows system is not replaced with a trojan.
- (G_3) ***Binary-Pathname Integrity***: the *pathname* (location) of the binary on the host's file system must not be illegally modified based on the pathname mapping when it was securely added into the host at $t_{installed}$. Note that on a host, a binary is kept in a file and is identified by its pathname. As such, the pathname associated

with a particular binary must not be illegally modified.[1] This goal is needed so that we execute a binary which we actually want. For instance, suppose that the binary of a file-system format and that of a shell both satisfy G_1 and G_2. However, if an attacker swaps their pathnames, then running the shell would cause the file system to be formatted.

Goals G_1 and G_2 can be achieved by ensuring *data origin authentication*[2] on the invoked binary; whereas goal G_3 is attained by ensuring *data integrity* on the binary's pathname. In practice, data origin authentication on a binary is achieved by using digital signature[3], whereas binary's pathname integrity is easily attainable using hash function.

5.2 Framework for Analyzing Binary Authentication Schemes

This section describes our framework for characterizing authentication systems based on a number of key design factors. We elaborate the available choices for the design factors, and discuss their impact on the effectiveness of the resultant systems. A set of desirable design factors are then identified. We also compare a variety of existing authentication systems using the framework.

5.2.1 Security Assumptions

First, we make clear some assumptions which are commonly made by software-based binary authentication systems including BinAuth:

- **Trusted host's kernel**: Since the authentication systems run on top of the kernel, and thus depend on the kernel to achieve their security objectives, it is assumed that the host's kernel is uncompromised. Following a host (but not kernel) breach, an authentication system is still expected to remain trusted and to function properly in preventing the execution of illegal binaries. As such, an authentication system should allow for the possibility of malware exploitation, e.g. a buffer overflow attack hijacking a privileged process, but not the ability to alter the kernel code [187].
- **Protected authentication system's information**: The systems also assume the security (integrity, confidentiality and availability) of all the keys used for the authentication as well as any database and configuration files stored on the host.

[1]Extending this security goal to additionally ensure the integrities of a binary file's other properties, such as permission modes (in Unix/Linux), is trivial. For simplicity, here we focus on protecting a binary file's pathname, which is a file's main property.

[2]Recall again that ensuring data origin authentication on a binary implicitly also assures data (i.e. binary-content) integrity [109] (see also some background in Section 2.3).

[3]It is actually possible to use symmetric key mechanism to ensure data origin authentication on a binary. A Message Authentication Code (MAC), which is a keyed hash function, can be used to provide data origin authentication [109]. However, it requires a host and *each* software developer to share a secret key, which is rather impractical in practice. Note that, unlike the signature-based authentication, MAC-based authentication can not additionally provide non-repudiation service.

- **Trusted administrator account**: Since the administrator in most OSes virtually has unblocked access to all the files, registries, and possibly kernel settings, we also need to assume the security of the administrator account.
- **The goodness of developer's private key**: Each software developer protects the secrecy of its private key (which is used for signing its binaries), and notifies the CA *immediately* in the event that the key has been compromised. This assumption on a principal's responsibility to safeguard its private key is commonly taken in practice (e.g. [56, 169]). Without such a strong guarantee on the goodness of a principal's secret key, it is practically impossible to provide both data origin authentication and non-repudiation service on documents/codes digitally signed by a principal. However, there exists a legitimate concern with possible compromise of an honest user's private key [104], particularly under the "*undetected key compromise*" scenario [79].[4] To better protect a private key against potential compromise, some mechanisms have been proposed [192, 191, 79], which go beyond timestamping and countersigning by a trusted Time Stamping Authority (TSA) [65, 3].

5.2.2 Authentication System Design Options

We now list the main design options in constructing a binary authentication system.

(D_1) **Location of the authentication verification module**:

(a) *In-kernel module*: The authentication verification module is implemented as part of the kernel, and thus runs in the kernel space.

(b) *User-mode application*: The module runs as an application in the user space.

(D_2) **Time of authentication**:

(a) *Pre-execution*: The binary file is verified *just right before* it is loaded for execution. This does not necessarily imply that the authentication is done by in-kernel module since wrappers can be placed on various entry points of the program's execution, e.g. command shell and GUI-based shell.

(b) *User-specified time*: The authentication timing is set independent from the binary-invocation time. It can be directly triggered by the user, or based on a user-specified time in the case of a scheduled job. A system like this usually verifies all protected binaries within one verification session. It thus takes a "snapshot" of all the binaries at one point in time.

(c) *Installation-time*: The verification is performed only at the installation time.

[4] An *undetected private key compromise* takes place where a principal suspects a compromise of its private key (and subsequently reports the compromise to the CA) only *after* it has signed a number of documents/binaries [79]. In Section 5.4, we discuss potential security implications to BinAuth if we allow such a possible compromise despite the developer's best protection efforts in safeguarding its private key.

(D_3) **Coverage of mandatory authentication**:

- **(a)** *All binaries*: All binaries must be verified before being executed.
- **(b)** *Selected binaries*: Only certain classes of binaries are subject to verification. For example, only binaries run with the administrator privilege and those with embedded digital signatures are checked. The rest are allowed to run without authentication.
- **(c)** *Configurable*: Here, a system's level of enforcement is configurable. The administrator is responsible to select one from a set of preset options to apply at a time. He/she, for instance, can apply one of the following options: all binaries must be authenticated, all binaries with the administrator privilege are authenticated, or no authentication is performed.

We note that for the mandatory authentication enforcement on all binaries, an in-kernel pre-execution mechanism is the most robust option.

(D_4) **Implementation platform**:

- **(a)** *Microsoft Windows*
- **(b)** *Unix/Linux*

We note that it is theoretically possible to port an authentication system implemented in one OS to another OS. In practice, however, such porting is rather difficult due to the differences in the security model and the kernel operations of the underlying OSes.

(D_5) **Security goals achieved**:

- **(a)** *Trusted Origin*: goal G_1 as defined in Section 5.1.
- **(b)** *Binary-Content Integrity*: goal G_2 as defined in Section 5.1.
- **(b)** *Binary-Pathname Integrity*: goal G_3 as defined in Section 5.1.

(D_6) **Placement of authentication information (digital signature/hash value)**:

- **(a)** *Embedded into the executable*: In this method, the digital signature or hash value is embedded into the executable file. Putting the information in the file itself seems to result in a cleaner model of the authentication system. However, as we will see later, there is a potential weakness in such a system.
- **(b)** *(Secure) centralized database*: The information from all the protected binaries is stored in a (secure) database. If all binaries are to be mandatorily authenticated prior to execution, then the database serves as a "whitelist" of accepted applications. This contrasts with the "blacklist" approach taken, for example, by anti-virus software. The advantage of using a centralized database is that the authentication system can also monitor non-binary files.

(D_7) **Producer of the binaries' digital signatures/hash values**:

(a) *Authentication system*: Here, all the signatures or hash values come solely from the authentication system. This option is useful when all installed binaries do not come signed by their respective developers. The administrator then assumes that a binary file is good at some point in time (usually at $t_{installed}$), and then invokes the authentication system to produce a signature/hash value as the reference for the binary's future executions.

(b) *Developer*: All binaries are assumed to be signed by their respective software developers.

(c) *Both*: It is possible that the administrator produces the signatures/hash values on *some* of the binaries. It is also possible that the administrator reproduces the signatures/hash values based on the existing developer's signature for security or performance reasons as in BinAuth system.

(D_8) **Authentication mechanism used**:

(a) *Symmetric key*: The authenticity/integrity of a binary is checked by using a symmetric-key based hash function.

(b) *Public key*: The checking is done by using digital signature operation and related PKI mechanisms.

(D_9) **Authentication Caching**:

(a) *With/without caching*: Whether a caching technique is employed by an in-kernel authentication system to keep track of the previously authenticated binaries. This technique is employed to reduce the overheads of always checking the binary files every time they are executed. It is particularly useful for some frequently executed binaries, especially if their file sizes are large. Files which are stored externally, e.g. files on NFS, however cannot be cached [9]. One main drawback of the caching technique is that the authentication system now needs to monitor if a binary in the cache list has been modified.

(b) *Not-applicable*: The caching technique is not applicable to a user-mode authentication system, or one such as [187] which checks the integrity during binary installation.

(D_{10}) **Reduced reliance on PKI**:

(a) *Reduced/full reliance*: Whether there is any mechanism employed to reduce the reliance on PKI in the case where verification using public key mechanism is used.

(b) *Not-applicable*: This is not applicable if only symmetric key mechanism is used.

(D_{11}) **Support for secure update of a protected binary**:

(a) *With/without support*: Whether there is any mechanism to support secure update of a protected binary file.

5.2.3 Comparison of Existing Authentication Systems and BinAuth

We now survey and compare various existing authentication systems using the developed framework. We select a representative system for each category. We also highlight how well each authentication scheme meets the authentication goals G_1–G_3, and contrast them with BinAuth. Table 5.1 summarizes the surveyed systems including BinAuth.

Tripwire [82] is one of the first schemes to perform file integrity protection. Tripwire ensures Binary-Content Integrity (G_2) as well as Binary-Pathname Integrity (G_3), but not Trusted Origin (G_1).[5] Tripwire is limited as it is a user-mode application program, and checks the file integrity in an off-line manner. It also does not provide any mandatory form of integrity checking. In addition, there exist many known attacks such as: file modification in between authentication times, and attacks on system daemons (e.g. `cron` and `sendmail`) and system files that it depends on [12].

There are a number of in-kernel binary authentication implementations such as DigSig [9], Trojanproof [183], SignedExec [175] and one proposed in [29]. These are mainly for Unix or Linux. They modify the Unix kernel to verify an executable's digital signature prior to its execution. **DigSig**, **SignedExec** and the scheme proposed in [29] embed signatures within the binary itself by making use of the ELF format. The works assume that the installed binaries are unsigned by their developers. Hence, the administrator signs the binaries using his/her own private key. It appears that these systems provide Trusted Origin (G_1) and Binary-Content Integrity (G_2), but not Binary-Pathname Integrity (G_3). The problem is that, in an embedded-signature authentication system, one signed binary replacing another signed binary would go undetected. One solution to this problem is that the authentication system must include the pathname in addition to the binary content when producing a digital signature. Even with this inclusion, the system would still suffer from a "old-attack" problem, when the attacker replaces a signed binary with an older, perhaps vulnerable, signed binary of the same pathname. DigSig employs a technique called signature revocation list in order to blacklist a signed binary. Catuogno and Visconti similarly proposed a file revocation list and an alternative tree-based scheme [29]. However, such techniques require a separate centralized configuration file which can grow over time. Maintaining such a list also poses its own challenges. In light of this, we opt for an authentication system with a centralized database.

For efficiency, DigSig and the scheme proposed in [29] employ a caching mechanism to avoid checking binaries which have been previously verified. In developing BinAuth, we also address the detailed implementation issues of caching technique on Windows, which are much more complex than in Unix.

Cryptomark [19] is another in-kernel authentication system on Linux, and is similar

[5]Tripwire (as outlined in [82]) does not attempt to ensure Trusted Origin (G_1) as it aims mainly to function as a file system's integrity checking tool.

Authentication System	**Verifier Module Location**	**Time of Authentication**	**Enforced Mandatory Authentication?**	**Platform**	**Security Goals Achieved**	**Placement of Integrity Information**	**Signature /Hash Producer**	**Mechanisms Used**	**Caching Used?**	**PKI Reliance Reduction?**	**Binary Update Support?**
Tripwire	Application-level	User-specified	No	Unix	G_2,G_3	Centralized	Auth. System	Symmetric Key	No	-N/A-	No
DigSig	In-kernel	Pre-execution	No	Linux	G_1,G_2	Embedded	Auth. System	Public Key	Yes	No	No
Trojan-proof	In-kernel	Pre-execution	Yes	Linux	G_2,G_3	Centralized	Auth. System	Symmetric Key	No	-N/A-	No
SignedExec	In-kernel	Pre-execution	Yes	Linux	G_1,G_2	Embedded	Auth. System	Public Key	Yes	No	No
CryptoMark	In-kernel	Pre-execution	Configurable	Linux	G_1,G_2	Embedded	Auth. System	Public Key	No	No	No
Emu System	In-kernel	Pre-execution	Yes	Windows	G_2,G_3	Centralized	Auth. System	Symmetric Key	No	-N/A-	No
SignTool	Application-level	User-specified	No	Windows	G_1,G_2	Embedded	Auth. System	Public Key	No	No	No
Sigcheck	Application-level	User-specified	No	Windows	G_1,G_2	Embedded	Auth. System	Public Key	No	No	No
Vista UAC (On Signed Executables)	In-kernel	Pre-execution (on privilege-elevation)	No	Windows	G_1,G_2	Embedded	Developer	Public Key	No	No	No
Wurster-Oorschot	In-kernel	Installation time	No	Unix/Linux	G_1,G_2	Embedded	Developer	Public Key	-N/A-	Yes	Yes (weak)
BinAuth	In-kernel	Pre-execution	Yes	Windows	G_1,G_2,G_3	Centralized	Auth. System, Developer	Symmetric, Public Key	Yes	Yes	Yes

Table 5.1: Comparison of binary authentication systems using the design-option based framework.

to DigSig in a number of ways. The special feature of Cryptomark is that it can be configured to require valid digital signatures for *all* or *some* binary files. The most secure configuration requires a valid signature for every binary. A more permissive configuration mandates signatures only for binaries that run with the root privilege.

There are some mechanisms in Windows related to binary authentication. **Authenticode** [61] is a Microsoft infrastructure for producing and verifying signed binaries. In Windows versions prior to Vista, such XP with SP2, Authenticode is used as follows:

- During an ActiveX installation: Internet Explorer uses Authenticode to examine the ActiveX plugin, and shows a prompt containing the publisher's information and the result of the signature verification.
- Following a user's download of a file using Internet Explorer: If this file is later invoked using the Windows Explorer shell, a prompt will be displayed giving the information of the publisher's information and the signature verification result. Internet Explorer uses an NTFS feature called Alternate Data Streams to embed the untrusted Internet zone information into the downloaded file. The Windows Explorer shell detects this zone information and then displays the prompt. This mechanism is however *not* mandatory, and relies on the use of zone-aware programs.

Since Authenticode runs in the user space, its signature verification can be bypassed in a number of ways, e.g. using the non zone-aware command shell. It is also limited only to files downloaded using Internet Explorer. Authenticode examines EXE binaries, but not the DLLs. Thus, one possible attack is to put malware into a DLL and then execute it, e.g. using `rundll32.exe`. Furthermore, Authenticode relies heavily on digital certificates. Ensuring that the developer's public key is still valid may add extra delay including timeouts due to the need to contact the CA and download the latest certificate status information, such as CRL. In some cases, this causes significant slowdown.

Windows Vista improves on signed-code checking since its **User Account Control (UAC)** can be configured for mandatory checking of all signed executables. However, this is quite limited since the UAC mechanism only kicks in when a process requests privileged elevation, or for certain operations on protected resources. Vista does not seem to prevent the loading of unsigned DLLs and other non-EXE binaries. The 32-bit versions of Windows (including Vista 32-bit) do not check whether drivers are signed. Similar to DigSig, both the UAC and Authenticode do not authenticate the binary pathname. Moreover, a system that always requires PKI infrastructure, we believe, poses various challenges for a general purpose online (pre-execution) authentication mechanism.

The closest existing work to our BinAuth is the **Emu** system [142], which also runs in Windows. It intercepts a process creation by intercepting the `NtCreateProcess()` system call. It is unclear, however, whether they are able authenticate all binary codes since trapping at `NtCreateProcess()` is insufficient to deal with DLLs. No performance benchmarks were given, so it is difficult to assess its efficiency.

There is also a binary protection system that protects unauthorized modification of existing binaries during the installation of a new piece of software [187]. This system does not restrict the software which can be installed on a system, but it denies one which modifies existing binaries. Hence, the system does nothing to prevent malware from being installed on the system, but it restricts the files that the malware can modify. One particular feature of the system in [187] is that it aims to allow software updates in a controlled manner. A binary can be replaced by another binary only if the latter is of the same name and contains a digital signature derived from the same public key previously used on the former. However, there is a potential attack of downgrading the binary. BinAuth, in contrast, makes use of a naming scheme like Software_ID (described in Section 5.6) to ensure a controlled software upgrading.

Lastly, we remark that there was also a related work which compares existing authentication systems. In [121], Motara and Irwin looked at six authentication systems and evaluated them in terms of the following criteria: ease of use, executable checking, pre-execution validation, active development and transparent checking. They also discussed several design options, namely: in-kernel vs user-mode (D_1), pre-execution vs offline validation (D_2), embedded-signature vs centralized (D_6), and caching mechanism (D_9). Our work [186] conducts a significantly more comprehensive analysis on the design options when compared to [121]. More importantly, our analysis is performed as a part of our effort to devise a robust yet lightweight binary authentication system. As such, the analysis is geared towards identifying desirable characteristics for such an authentication system, which are then realized in the BinAuth system.

5.3 System Architecture for Lightweight Authentication

Based on the analysis using the design-option based framework discussed above, our goal is to devise BinAuth as a robust and practical authentication scheme with the following characteristics: an in-kernel (D_{1a}), pre-execution (D_{2a}), mandatory (D_{3a}) authentication scheme for Windows (D_{4a}), which ensures security goals G_1–G_3 by storing the authentication information in a centralized database (D_{6b}). BinAuth accepts both digital signatures from the developer as well as producing MAC values on protected binaries. It thus exercises (D_{7c}), (D_{8a}) and (D_{8b}), and has a reduced reliance on PKI ($D_{10a-Reduced\ reliance}$). It also makes use of caching ($D_{9a-With\ caching}$), and supports controlled update of the protected binaries ($D_{11a-With\ support}$). Additionally, it should incur low overhead and allow for a better vulnerability management on the protected binaries.

5.3.1 BinAuth Architecture

In the following discussion, we assume that binaries already come tagged with a necessary naming information such as Software_ID (see Section 5.6) or CPE [115].[6] BinAuth achieves goals G_1–G_3 by performing the following two-step guarantees (under the assumptions stated in Section 5.2.1):

- **(Step$_1$)** ***Trusted-binary installation***: Immediately after a binary is installed into the host's file system at time $t_{installed}$, BinAuth should be invoked to ensure both Trusted-Origin (goal G_1) and Binary-Content Integrity (goal G_2) on the binary. If the binary is signed by its developer, BinAuth verifies the binary's signature. Should the binary come unsigned, BinAuth consults the administrator to manually decide whether the binary can be deemed trusted. Once an installed binary is considered trusted, BinAuth generates the Message Authentication Code (MAC) value for the binary. In addition, BinAuth also records its pathname.
- **(Step$_2$)** ***Unmodified-binary invocation***: When an installed binary is later invoked at time $t_{invoked}$, BinAuth ensures that both the MAC value and the pathname of the binary are the same as those recorded at Step$_1$. In this way, BinAuth finally ensures that the binary does satisfy goals G_1–G_3 prior to its execution.

From the way BinAuth works as elaborated above, *only one* public-key operation is done per signed binary. We choose to use the HMAC algorithm [86] for generating MAC value. Note that HMAC is a keyed hash construction, therefore there is a secret key for the administrator. This secret key could be stored on a secure external storage such as a TPM module [173]. Having this secret key provides an additional host protection. Suppose that an attacker manages to illegally add a binary and also modify the MAC database. Given the use of the secret key, an attempted execution of the added binary will still be detected by BinAuth. To ensure the authenticity of a protected binary on its future invocation, only the generated HMAC value needs to be checked. In what follows, we mostly write MAC which already covers the choice of HMAC algorithm.

One way of storing the generated MAC is to embed it into the binary. However, doing so may interfere with file format of the binaries, and may also have other complications, such as the inability to provide Binary-Pathname Integrity (goal G_3). As such, we instead use a repository (database) file which stores all the MAC values of protected binaries together with their pathnames. During the boot-up process, the kernel creates its own in-memory data structures for binary authentication from this repository. We can also customize binary authentication on a per-user basis rather than on a system-wide basis, thus producing a white list of binaries approved for execution for each user.

[6]Unlike our Software_ID, CPE does not include the binary file name. Hence, the file name must be added should the CPE be used.

There are two main components of BinAuth: the **SignatureToMac** and the **Verifier**. The SignatureToMac maintains the authentication repository, called `Digest_file`. The Verifier is a kernel driver, which makes use of `Digest_file` and decides whether the execution of a binary is to be allowed.

5.3.2 SignatureToMac Module

Once a piece of software is installed on the system, Figure 5.1 shows how SignatureToMac processes the installed binaries. The steps involved can be elaborated as follows.

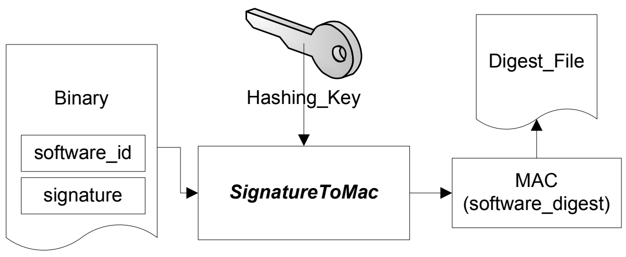

Figure 5.1: SignatureToMac: deriving the MAC for a signed binary.

1. If a binary is signed, check the validity of the binary's digital signature and the corresponding certificate. If the signature or certificate is invalid, report failure. In the case where the binary is unsigned, inform the administrator.
2. Consult the administrator whether the software is to be trusted or not. This is similar to the Vista UAC dialog, but only happens once. Additional checking policies (possibly mandatory) can also be implemented.
3. Generate the MAC value (called `software_digest`) from the binary's content (including its `Software_ID` string) using a system-wide secret key called `Hashing_key`. This `Hashing_key` is accessible only by the authentication system, e.g. obtained at boot time from a secure (external) storage. Our prototype implements the HMAC-MD5 [86], HMAC-SHA-1 and HMAC-SHA-256 [44] hash algorithms.[7]
4. Add an entry for the binary as a tuple ⟨`path_name`, `software_digest`⟩ into the `Digest_file` repository, and inform the Verifier to perform the necessary in-memory updates.

5.3.3 Verifier Module

The Verifier performs a mandatory binary authentication — it denies the execution of any kind of Windows binary that fails to match the MAC or the pathname. There are two general approaches for the checking. One is *cached MAC*, which avoids generating

[7]The stronger hash functions SHA-1 and SHA-256 are used due to recent concerns on the weaknesses of MD5 and the associated attacks on it [181].

and checking the MAC for a previously authenticated binary. The other is *uncached MAC*, which always generates and checks the MAC. As we will see, these two approaches have various tradeoffs. The cached MAC implementation needs to ensure that binaries are unmodified. Hence, the Verifier monitors all file operations on binaries being cached, and removes them from the cache if they can potentially be modified.

The core data structure of the Verifier can be viewed as a table of tuples in the form ⟨`Kernel_path`, `FileID`, `MAC`, `Authenticated_bit`⟩ representing the allowed binaries. The four fields in the tuple are as follows.

- The `Kernel_path` is Windows kernel (internal) pathname representation of a file. In Window's user space, a file can have multiple absolute pathnames, due to: (i) 8.3 file naming format (e.g. "`C:\Program Files\`" and "`C:\progra~1\`" are the same); (ii) symbolic links; (iii) hard links; (iv) volume mount points; or (v) the `SUBST` and `APPEND` DOS commands. The `Kernel_path` is a unique representation for all the possible pathnames. When the system loads a tuple ⟨`path_name`, `software_digest`⟩ from `Digest_file` either during the startup or when an entry was just added into the `Digest_file`, the `path_name` is converted into `Kernel_path` since all subsequent checks by the Verifier use the latter.
- The `FileID` is a pair of ⟨`device_name`, `NTFS_object_ID`⟩. The `device_name` is a Windows internal name to identify a disk or partition volume. For instance, the device name `HarddiskVolume1` usually refers to `C:\`. The `NTFS_object_ID` is a 128-bit number uniquely identifying a file in an NTFS file system.[8] This `FileID` is used by the Verifier to identify the same binary file given more than one hard links. This prevents an attacker from creating a hard link in order to modify a binary (which has been authenticated before) without invalidating the binary cache.
- The `MAC` is same as the `software_digest` entry in `Digest_file`.
- The `Authenticated_bit` remembers whether a binary has been previously authenticated. It is initially set to false, and then set to true after a successful binary authentication. If the binary is modified at a later time, the bit is set back to false.

Figure 5.2 shows the authentication steps performed by the Verifier (with caching technique used) when a binary is invoked for execution. The steps can be elaborated as follows.

1. Check if the binary's `Kernel_pathname` exists in the Verifier's table. If not, deny the execution (and optionally log the event). A notification is accordingly sent to the user and the administrator. The user then could ask the administrator to perform the step of generating the binary's MAC using the SignatureToMac.

[8]NTFS is the standard file system of Windows NT, including its later versions Windows 2000, Windows XP, Windows Server 2003, Windows Server 2008, Windows Vista, and Windows 7. Hence, we assume that NTFS is the file system type used by the Windows systems to be protected by BinAuth. If the older FAT file system is used instead of NTFS, we simply make use of the pathname to identify a binary.

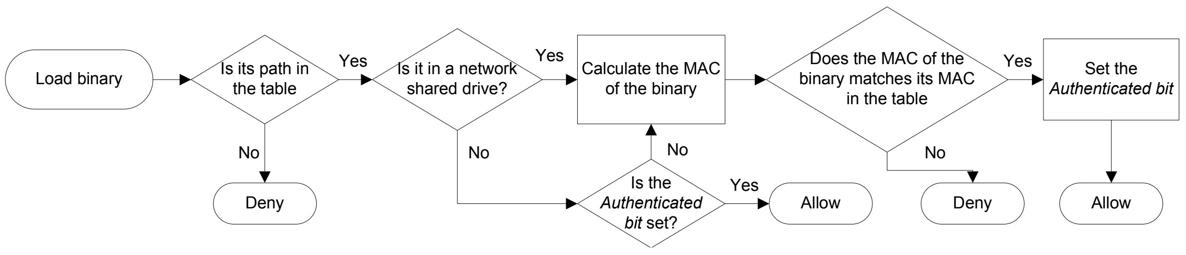
Load binary
Is its path in the table
Yes
No
Deny
Is it in a network shared drive?
Yes
No
Calculate the MAC of the binary
Is the *Authenticated bit* set?
No
Yes
Allow
Does the MAC of the binary matches its MAC in the table
Yes
No
Deny
Set the *Authenticated bit*
Allow

Figure 5.2: Verifier: the in-kernel authentication process.

2. If the file is on a network shared drive or removable media, then goto step 4.
3. If the `Authenticated_bit` is set, then go to step 7.
4. Perform MAC generation operation on the binary.
5. If the resulting MAC value does not match with the `MAC` stored in the Verifier's table, deny the execution.
6. Set the `Authenticated_bit` of the binary to true.
7. Pass the control to the kernel to continue the binary's execution.

For additional details on the implementation aspects of BinAuth, such as the use of system-call interception mechanism and file-monitoring mechanism for secure caching, the readers can refer to [66, 186].

5.4 Security Analysis

In this section, we give the security analysis of BinAuth.

Since BinAuth makes use of MAC operations, its security thus also relies on the strength of the chosen hash functions (i.e. MD5, SHA-1, SHA-256) as well as the HMAC construction. Here, we assume that any change in a binary can be detected through a changed MAC value.

Following BinAuth's successful verification of a digitally signed binary, the MAC value for the binary is generated and then securely recorded. Subsequent invocations on this binary is verified by BinAuth based on the binary's MAC value as opposed to its original digital signature. Thus, BinAuth considers that a successful MAC authentication at time $t_{invoked}$ "preserves" the data origin authentication assurance on a binary, that was previously established from a valid digital signature shortly after time $t_{installed}$. A subtlety however arises when the developer's certificate expires or is revoked at some point in time after the MAC generation. Given the assumption on the goodness of a principal's private key (up until shortly prior to the revocation as discussed in Section 5.2.1), signatures created before the compromise are deemed still valid. Such a reasoning is also taken in practice in the case of a securely timestamped digital signature used as non-repudiation evidence for a signed document (e.g. [43, 55, 3]).[9] In this signed document application, the use of a trusted timestamp can prevent a malicious party, who obtains a principal's private key after the certification revocation/expiration, from fabricating a signed message and subsequently claiming that the principal had signed the document when the certificate was still good (i.e. at one point in time prior to the revocation/expiration). In BinAuth, the host *directly* verifies both the developer's certificate and the corresponding signature before *immediately* generating the MAC value at Step$_1$ (see Section 5.3.1). As such, BinAuth can work on digital signatures with no added secure timestamps.

Alternatively, we can view that the question of whether one should keep trusting the binary (after the corresponding certificate's revocation/expiration) depends on one's interpretation of certificate expiration/revocation. If the certificate expiration/revocation means that the corresponding public key *must no longer be used*, but the fact that the *previously established goodness properties of a binary* still hold (under the assumption of the goodness of the private key prior to the revocation/expiration), then we can keep trusting the binary as long as we still believe in its issuer.

We can reason this issue more concisely using a BAN Logic-based reasoning [26, 27]. An overview of BAN Logic is given in Section 2.6, and its extension is the subject of Chapter 8. Since we need to deal with a belief established in the past, we thus need to incorporate a temporal aspect into BAN Logic.[10]

For our purpose, we can use a temporal construct defined in [161], namely $\Diamond$, which means "*at some point in the run prior to the current one*". Given a formula φ, a run r, and a time t, we define that:

$$(r,t) \models \Diamond\, \varphi \ \text{ iff } \ (r,t') \models \varphi \ \text{for some } \ t' < t. \tag{5.1}$$

Given this $\Diamond$ construct, we can restate the validity of the employed MAC authentication in BinAuth as follows:

> The question of whether we can still believe in a binary that has been successfully verified using the MAC authentication, but whose public/private key pair used to generate the digital signature has expired or has been revoked after the MAC generation, is equivalent to that of determining whether the following *Carried-forward-belief* Rule is applicable:
>
> $$\frac{P \mid\equiv Q \Rightarrow X,\ \Diamond(P \mid\equiv X)}{P \mid\equiv X}. \tag{5.2}$$

That is, if we know that $(r,t') \models (P \mid\equiv X)$ for some $t' < t$, with t being the present time, and

[9] In this setting, each newly generated digital signature is timestamped and countersigned by a trusted TSA, which establishes evidence that the signed message had existed at the time of the timestamping. Given the goodness of the public/private key pair up to its revocation/expiration, the TSA's assurance allows a revoked or expired certificate to be used for verifying timestamped signatures created before the time of revocation or expiration (see also [3]).

[10] This is because the original BAN Logic only allows the promotion of "once said" (in the past) to "believe" (in the present) using the Nonce-verification Rule (i.e. R_2 in Appendix C).

that $(r,t) \models (P \mid\equiv Q \Rightarrow X)$, we are to determine whether we can derive $(r,t) \models (P \mid\equiv X)$. For our purpose of authenticating a binary under the assumptions stated in Section 5.2.1, we chose to accept this rule. As such, BinAuth can perform binary authentication with low performance overhead.

Note however that, given an expired or revoked certificate, we cannot derive the *present* belief of a binary (i.e. $P \mid\equiv X$) using our MPKI-BAN Logic which is defined in Chapter 8. This is because the New certificate-validation Rule (R_{15} in Appendix D) cannot be used to derive the belief on the goodness of the developer's public key (i.e. $P \mid\equiv \wp\kappa\,(Q, K_Q)$) and private key (i.e. $P \mid\equiv \Pi(K_Q^{-1})$). As a result, we cannot use the New message-meaning for signed message Rule (R_{13}) to derive $P \mid\equiv Q \mid\sim X$, which is a requirement for the desired belief $P \mid\equiv X$.

One additional issue with regard to the use of MAC instead of the developer's digital signature is the potential delay in obtaining a timely certificate revocation information, which may happen in practice. As explained, prior to generating a MAC value for a newly added binary, BinAuth first verifies the validity of the developer's certificate. However, it may be the case that the certificate has been already revoked by that time, but the revocation notification fails to be received in time. To overcome this, we stipulate BinAuth to monitor the status of a recently used certificate until the possibility of such a delay diminishes. That is, BinAuth will still ensure that the certificate must still be valid at time $t_{ensured} > t_{installed} + \delta_{installed\_to\_verified} + \delta_{rev\_delay}$, where: $\delta_{installed\_to\_verified}$ is the time interval between $t_{installed}$ and the time where the signature verification is done, and δ_{rev_delay} is the upper-bound margin for the delay in obtaining revocation information. The value for δ_{rev_delay} can be set in accordance to the revocation scheme's inherent timeliness delay (see Section 2.5). BinAuth can thus maintain a list of certificates to be revalidated, which are periodically checked in the background.

Let us now analyze the security of BinAuth, if we relax the assumption on the goodness of a developer's private key as stated in Section 5.2.1. That is, we now allow a possibility of an "undetected key compromise" scenario [79] where a principal (i.e. developer) suspects a compromise of its private key *only after* it has signed a number of binaries. In this scenario, BinAuth needs to keep revalidating all the certificates previously used to generate the corresponding MAC values. We can thus stipulate BinAuth to periodically check these previously validated certificates in the background.[11] This way, BinAuth still provides an advantage of shifting the required pre-execution certificate validations (if we do not generate and use MAC values at all) to periodic certificate validations in the background.

Upon finding out that a particular certificate is revoked, the administrator needs to

[11]Note that the number of certificates is much less than the number of binaries. This is because all binaries from the same software package are signed by one developer using a single certificate.

determine whether future executions of *all* binaries signed using the revoked certificate should no longer be allowed. Disallowing all these binaries represents a prudent action. However, it can be too restrictive in some environments. One possible compromise is to examine the reason code of the revocation. In [38], Cooper analyzes possible revocation reason codes, and categorize them into *repudiable* and *non-repudiable* sets depending on whether there is a possibility that someone other than the principal may have illegally used the private key associated with the revoked certificate. Thus, an authentication system like BinAuth may still allow the future executions of a binary if the corresponding certificate is revoked with a reason code belonging to the non-repudiable category. Yet, the work [38] also identifies some potential problems that may arise due to an attacker's attempts to manipulate the use of reason codes in certificate revocation systems.

In our analysis on BinAuth security, we also have additionally analyzed various possible system (OS-related) attacks to BinAuth. These attacks mainly attempt to target the caching system, i.e. the attacker attempts to modify an already authenticated binary without causing the `Authenticated_bit` to be set to false. The analysis on how BinAuth can deal with these attacks can be found in [66, 186].

Lastly, we remark that if an additional hardware-based infrastructure, such as TPM [173], is available to support secure booting, then BinAuth can enjoy an increased security. The `Hashing_key` can also be stored securely by TPM.

5.5 Experimental Results and Discussion

We now briefly discuss the impact of BinAuth on the system's performance. The two main factors that impact the overall system performance are: (i) the Verifier's checking upon binary invocations; and (ii) file modification monitoring (when caching is used). These factors affect the user's waiting time for a binary's execution and file operations. We use both micro and macro benchmarks to determine the worst case and the average performance overheads.

The benchmarks are run on an Intel Core 2 Duo PC with 2-GB of RAM running Windows XP with SP2. Each benchmark is run five times. As we want to investigate the effect of the cached Verifier, each benchmark is run with and without caching.

To see the difference of using different hashing algorithms, we implement and benchmark three algorithms: MD5, SHA-1 and SHA-256. For clarity, only the results of MD5 and SHA-256 are shown in Table 5.2. (The results of SHA-1 are always in between those of MD5 and SHA-256). When caching is enabled, the results of different hashing algorithms are not distinguished (shown as Cached-MAC in the table), because BinAuth does not perform any MAC checkings during the benchmark. This is since the first run is ignored[12], and no binary is modified during the benchmark.

To see the difference against digital signature based authentication systems, we also compare the performances of BinAuth with those of the Microsoft official Authenticode utility called `SignTool` [113], and another Sysinternals (now acquired by Microsoft) Authenticode utility called `Sigcheck` [139]. Note that these two tools are user-mode programs. They are considered here only to illustrate the difference between non-mandatory strategies used in Authenticode and our in-kernel mandatory authentication.

We have conducted a total of three benchmarks. The first two benchmarks investigate the system's performance, whereas the third investigates the tradeoffs between the cached and the uncached verifications. The first two benchmarks are conducted under two following scenarios:

1. **Micro benchmark:** The micro benchmark aims to measure the *worst case* performance overhead incurred by the authentication scheme. Note that this is primarily intended to measure the authentication costs, and not other system's overheads. Here, we have two micro benchmark scenarios:

   (a) **EXE-Intensive:** This scenario executes the `noop.exe` program, which is a dummy program that immediately exits, for 10,000 times. We use different binary sizes (40 KB, 400 KB, 4 MB, and 40 MB respectively). This scenario aims to show how executable size impacts the performance. Note that due to the way Windows works, an execution of `noop.exe` still involves some DLL invocations. When sufficiently large EXE files are used, this scenario thus investigates the overheads of EXE-intensive authentications.

   (b) **DLL-Intensive:** This executes the `load-dll.exe` program for 100 times. This scenario is used to find out how loading a significant number of DLLs impacts the system's performance. The program `load-dll.exe` loads 278 standard Microsoft DLLs with a total file size of ~75 MB. The size of the `load-dll.exe` itself is only 60 KB.

2. **Macro benchmark:** The macro benchmark measures the overhead under a user's *typical usage* scenario. The selected scenario here is a user's creations of Windows DDK sample projects using the `build` command. In each test run, 482 C/C++ source files in 43 projects are built. This benchmark is chosen as it is deterministic and non-interactive in nature; and it creates many processes and uses many files.

We benchmark `SignTool` and `Sigcheck` in the following fashion. For the micro benchmark, we first sign `noop.exe` and `load-dll.exe` using `SignTool`'s signing opera-

[12]When caching is enabled, we ignore the result of the first run because this authentication overhead is already shown in the uncached cases. Even if we take the first run into account, its impact will be very small due to the high number of runs that the benchmarks are subject to.

tion. We then measure the total times for both authenticating these two programs (using the `SignTool`'s and `Sigcheck`'s verification operations respectively) and executing them. The results of two following evaluation modes on `SignTool` and `Sigcheck` are reported under the micro benchmark: (i) evaluating `EXE`s only; and (ii) evaluating all binaries (`EXE`s + `DLL`s). For the macro benchmark, we replace each development tool in the DDK (i.e. `build.exe`, `nmake.exe`, `cl.exe` and `link.exe`) with a wrapper program. This wrapper first authenticates the actual development tool and then invokes it. Unlike in the micro benchmark, only the overheads of evaluating `EXE`s are reported in the macro benchmark. This is because many supporting binaries are invoked during the macro benchmark. Each invocation of these binaries also loads a different set of DLLs depending on the input file(s) being processed. Thus, it is rather hard to keep track of what DLLs are loaded. We therefore do not report the results for evaluation `EXE`s + `DLL`s.

The results are presented in Table 5.2. For clarity, the micro benchmark results for "`noop.exe` 400 KB" and "`noop.exe` 4 MB" are not shown because they are bounded by those of "`noop.exe` 40 KB" and "`noop.exe` 40 MB" in a linear fashion. The slowdown of executing a binary b is defined as: $slowdown_b = (time_b - time_{baseline})/time_{baseline}$.

Authentication System	**Micro Benchmark**						**Macro Benchmark**	
	EXE-Intensive				**DLL-Intensive**			
	`noop.exe` (40 KB)		`noop.exe` (40 MB)		`load-dll.exe`		`build`	
	time	slowdown	time	slowdown	time	slowdown	time	slowdown
Baseline	22.76	–	30.07	–	45.32	–	66.26	–
EXEs Only:								
`SignTool`	2822	11637%	4850	16033%	73.49	62.16%	97.00	46.39%
`Sigcheck`	1720	7457%	5629	18623%	62.82	38.62%	110.5	66.72%
Uncached-MD5	25.96	14.08%	2150	7052%	45.34	0.05%	70.85	6.93%
Uncached-SHA256	30.29	33.07%	9005	29851%	45.34	0.05%	71.79	8.35%
Cached-MAC	23.20	**1.93%**	30.63	**1.88%**	45.33	**0.02%**	67.62	**2.06%**
All Binaries:								
`SignTool`	11867	52043%	14030	46565%	16018	35244%	–	–
`Sigcheck`	4283	18772%	6186	20478%	12548	27587%	–	–
Uncached-MD5	26.10	14.67%	3881	12811%	128.8	184.1%	79.31	19.69%
Uncached-SHA256	30.42	33.67%	9302	30839%	201.3	344.0%	91.80	38.55%
Cached-MAC	23.25	**2.14%**	30.58	**1.72%**	45.35	**0.07%**	67.88	**2.45%**

Table 5.2: Performance benchmark results showing the authentication+execution times (in seconds) and the slowdown factors. The worst slowdown factor for each scenario is shown with underline, whereas the best is in bold.

We can see that the overheads of `SignTool` and `Sigcheck` makes them unusable under DLL-intensive scenario when `EXE`s and `DLL`s are evaluated (∼352 times slower for `SignTool` and ∼276 times slower for `Sigcheck`). If only `EXE`s are checked, the overhead is at least ∼39%. Again, our main purpose here is just to show the difference between what can be done in user-mode versus in-kernel scheme. We can see that the uncached-MD5 and uncached-SHA256 are considerably faster than `SignTool` and `Sigcheck` in almost all cases. The only exception is the micro benchmark case of invoking huge 40-MB `noop.exe`. Here, the uncached-SHA256 runs slower than `SignTool` if both `EXE`s and `DLL`s are evaluated, and runs slower than both `SignTool` and `Sigcheck` if only `EXE`s are evaluated. This is since we have BinAuth repeatedly authenticate the 40-MB length binary for 10,000 times in each run using the more computationally expensive SHA-256 algorithm. As mentioned earlier, the results are not fully comparable for this case since `SignTool` employs SHA-1 algorithm.

Now, let us compare the results of the cached and the uncached BinAuth. Under EXEs only measurements, as also mentioned above, the running time for the uncached BinAuth is very high in the micro benchmark scenario with SHA-256 on the 40-MB `noop.exe`. The micro benchmark results on the smaller file size (40 KB) however shows that the overhead of the uncached BinAuth is acceptable (∼14–33%). Furthermore, the macro benchmark shows that under a typical usage, the uncached BinAuth incurs overheads of only ∼7–8%, while the cached mode brings this down to very small value of ∼2%. In the `load-dll.exe` benchmark, the overhead of cached BinAuth is even almost negligible (0.02%).[13] Moving to the overheads of authenticating both `EXE`s + `DLL`s, we can see the effect of programs which generally use many DLLs in Windows. (Typically, there are more codes contained in invoked `DLL`s than `EXE`.) The overheads incurred by cached BinAuth are still small (∼2% on `noop.exe` and ∼0.07% on `load-dll.exe` in the micro benchmark; and ∼2% in the macro benchmark). The overheads of the uncached mode however can grow to between ∼20-40% in the macro benchmark depending on the hash algorithm used.

The third benchmark is a micro benchmark investigating the tradeoffs between the cached and the uncached verification. Using caching means that the MAC verification is amortized over executions, but it has added an overhead for monitoring file modifications. The uncached mode is just the opposite. Our micro benchmark opens a file for writing 100,000 times in order to measure the *worst case* overhead incurred by the file modification monitoring. We consider three following cases: (i) a clean system without BinAuth as the baseline case; (ii) cached BinAuth when the modified file is a BinAuth-protected binary file; and (iii) cached BinAuth when the modified file is a non BinAuth-protected file. The results are shown in Table 5.3.

Micro Benchmark Case	**Time (s)**	**Slowdown**
Baseline	4.051	–
Cached-MAC with a modified BinAuth-protected binary file	6.809	68.1%
Cached-MAC with a modified non BinAuth-protected file	6.266	54.7%

Table 5.3: Micro benchmark results showing the overheads of file modification monitoring.

[13]Note that as this overhead is quite small, the results are dominated by the non-determinism in timing measurements.

The results for the file modification benchmark show that for the cached BinAuth, it does not matter whether the file being written to is a binary or not. File monitoring in case (ii) and (iii) incur ∼60% overhead compared to a clean system. Note that there is no overhead associated with file monitoring in the uncached BinAuth. Hence, under some usage scenarios where files are frequently modified, the uncached strategy may be preferable over the cached one although the Verifier's overhead is higher in the former.

5.6 BinAuth and Software_ID Scheme

We also complement binary authentication with a scheme called Software_ID in order to simplify binary management issues. The idea is that we associate a *unique string* to a particular binary of a software product. This is similar to the recent CPE initiative by MITRE [115]. Software_ID is however different from CPE in that Software_ID is associated not with a family of software, but rather with *each* individual (binary) file. Additionally, we suggest that Software_ID is to be signed together with a binary file. The benefits of combining binary authentication and Software_ID are twofold. Firstly, we can obtain the ID of an executed binary in an accurate manner right to *the file level*. This allows us to perform any necessary check, such as pre-execution vulnerability check (described in Chapter 6), on a binary. The vulnerability alert advisory may already contain the affected Software_ID. Compared to CPE, it is now possible to identify and block only the affected file(s) in a vulnerable software product.[14] Secondly, due to the use of binary authentication, a Software_ID string is securely protected against illegal modifications.

Software_ID should ideally come from the software developer. Alternatively, it can be assigned by the system administrator. The key to ensuring the uniqueness of Software_ID lies on its standardized format. We can define Software_ID as follows:

$$\text{Software\_ID} ::= \langle \text{vendor\_ID} \,||\, \text{product\_ID} \,||\, \text{module\_ID} \,||\, \text{version\_ID} \rangle. \quad (5.3)$$

Here, '||' denotes string concatenation. Module_ID records the file name of the binary. Different from CPE, version_ID does not keep track of a software product version. Rather, it is meant to keep track of *file versioning* in the case where a binary is updated or patched.[15] For the software product version information, we make it as part of product_ID. Similar to CPE, we can leverage on the possible existing trust infrastructures [178], such as the domain name of the developer, for the vendor_ID in the absence of a centralized naming authority for software vendors.

[14]This policy of blocking only the affected file(s) can be acceptable, for instance, when the vulnerable components are the non-essential components of the software package.

[15]To deal with module updates, module_ID can also be used to record the module's version information. Having a separate version_ID, however, is useful to easily track different versions (or patched versions) of the same program.

Software_ID can also be used to help ensure controlled update on (protected) binaries. We can stipulate that a signed binary can only be replaced with another signed binary that satisfies the following requirements:

- The new binary is of the same file name.
- It is correctly signed by the same developer.
- The binary has the same Software_ID, but with higher version_ID information.

Upon a valid binary update, the database (`Digest_file`) is thus accordingly updated. Using this mechanism, we can therefore protect the binaries against old-attacks attempting to exploit the software update process. Note that this mechanism does not need an additional file/signature revocation list as in [9, 29].

5.7 Chapter Summary

We have analyzed the problem of ensuring trusted execution of binaries on a host. We also have developed a framework that characterizes a binary authentication system, and how its design options may affect its ability in ensuring trusted executions. Using BinAuth, we have shown how a mandatory in-kernel authentication system can increase the trust on good software execution but with acceptable performance overheads, even on a complex OS like Windows. BinAuth integrates well with PKI without having to heavily rely on it. Additionally, we have shown how BinAuth can be combined with a simple Software_ID scheme to simplify binary version management and vulnerability alert processing.

Chapter 6

Towards Automated Vulnerability Alert Processing

The number of security vulnerabilities discovered in computer systems has increased explosively [30]. In order to keep track of security alerts, administrators generally rely on vulnerability alert repositories/databases[1] such as [174, 145, 171]. Such databases are however designed primarily to be read and understood by humans. Given the speed at which an exploit becomes available once a vulnerability is known, and the frequency of occurrence of such vulnerabilities, manual human intervention becomes too slow, time-consuming and possibly ineffective [31] (see also background information in Section 2.4).

In this chapter, we address the challenge to provide a vulnerability-free execution of a program with respect to the (publicly) known vulnerabilities. This constitutes an important step in securing the PPLC. Given the limitations of the manual processing of alert information, we incorporate a framework for automating vulnerability alert processing on a host. The framework is based on our work [160], which was developed when vulnerability alert databases for automated processing were still not addressed as widely as today. More specifically, our framework aims to achieve the following objectives:

- To define a coordinated vulnerability database, whose entries that are meant to be machine-readable and processable. The database must be comprehensive and up-to-date in its contents, and is coordinated from multiple different existing sources in a well-structured manner. In our approach in [160], we assume a representation using the relational database model.
- To represent each alert information as a *specification data* by means of a vulnerability description expression scheme rather than as a code (i.e. binary or script). The specification should enable efficient operation of a vulnerability scanner. In

[1]As previously mentioned, we follow a common practice in vulnerability management field to refer to a vulnerability alert repository as a vulnerability database. This is despite the alert entries are narrative and not structured in a well-defined database model, e.g. the relational model.

addition, it should make it possible for the scanner to *relate* different vulnerability entries and determine the possibility of a "*chain of exploits*".

- To define and develop a proof-of-concept vulnerability scanner which performs an automated alert processing and vulnerability scanning on a host. Such a scanner must be *general purpose* in that its mechanisms are independent from any specific entry definitions and not tied down to limited vendor-specific entries. Furthermore, its operations must be minimal and should be *open* to possible scrutiny and verification.

Our work published in [160] proposes example mechanisms that achieves the abovementioned objectives. Since the creation of an integrated database should involve many relevant parties, the proposed mechanisms are however not meant to be a standalone definitive solution. Rather, the main contributions of our work are the definition of the framework and the identification of its key components and required properties for an effective automated vulnerability processing. Despite being published several years back, our results are in line with the present standardization efforts to automate vulnerability processing, such as OVAL [117], CPE [115] and SCAP [126]. With respect to these standardization results, we are pleased to see how the vulnerability management field has been progressing in recent years, and that our published work may have played a part in spurring the developments of machine-oriented vulnerability alert processing. Later in this chapter, we contrast our framework with these on-going standardized results. We also point out how the standardization results still need to be extended to realize the fully-automated vulnerability processing solution as envisioned in our framework.

The rest of this chapter is organized as follows. Section 6.1 surveys the existing works. Section 6.2 gives an overview of our proposed framework. Section 6.3 then describes our proof-of-concept vulnerability database, and Section 6.4 covers our vulnerability description expression scheme. A prototype scanner is described in Section 6.5. Section 6.6 discusses deployment issues as well as other aspects, and Section 6.7 gives a summary.

6.1 Existing Works and Challenges

We discuss below various works related to the three components of our proposed framework, namely: machine-oriented database, automated vulnerability scanner, and vulnerability description scheme. We additionally highlight the challenges found in some of the existing systems, which have provided motivations for our framework's design goals.

6.1.1 Machine-Oriented Vulnerability Database

There exist a number of vulnerability databases which reorganize and integrate various vulnerability alerts into one repository that is searchable based on specified attribute values. Examples are ICAT[2], Public Cooperative Vulnerability Database [136], and the Open Source Vulnerability Database (OSVDB) [171]. Although these databases are searchable, they are however not specifically designed for any automated applications. Krsul [87] proposed a comprehensive taxonomy of vulnerabilities for possible further processing or automated manipulation. A database definition was also proposed. However, no specific applications were co-designed or shown to work together with it.

National Vulnerability Database (NVD) [170] is an active vulnerability database managed by NIST, based on a recent initiative supported by the U.S. Department of Homeland Security. It hosts the U.S. government repository of vulnerability management data, and can be searched based on CVE [116] or OVAL [117] query. NVD was specially created with the intent of reorganizing and aggregating vulnerability information from multiple existing databases into a standardized database. MITRE also maintains OVAL Repository [119], which hosts *OVAL Definitions*, i.e. machine-readable tests written in the OVAL Language [117]. These two databases are thus in line with our proposal for an integrated and unified machine-oriented database.

6.1.2 Host-based Vulnerability Scanner

There have been a number of popular tools that scan for any presence of vulnerability or configuration weaknesses in a system. Some examples are COPS [46], Ferret [147], SATAN [144], and Nessus [128]. In code-based scanners such as [46, 144], the logic of vulnerability checking is embedded tightly in the scanner's code. This means that including a new vulnerability check requires one to update the scanner's code or its sub-component(s). Most recent scanners such as [147, 128] take a modular approach. A vulnerability check is usually represented as a "plug-in module", which can be fed to the scanner to perform the test. Nessus writes its vulnerability checks in a scripting language called NASL [11]. Vulnerability checks written in languages like NASL however can potentially be too powerful, which may allow for possible misuse. NASL resembles code rather than data. Thus, an attacker may target the plug-in modules so as to perform its own unauthorized operations on a target host. In contrast, our proposed framework uses a declarative language to specify a vulnerability check. The actions are performed by a generic scanner which is made available for third party inspection.

Windows Update [185] is a Microsoft online tool for automatically updating Windows OSes and and its installed components with recent patches. It illustrates some important issues with vendor-specific automated tools. Windows Update and its more automatic cousin Windows Software Update Services [184] are closed systems, and follow a "black-box" software update model. This leads to the following issues:

[2]The database used to be available at `http://icat.nist.gov/icat.cfm`. The site seems to be no longer in operational, and access to it is referred to the National Vulnerability Database (NVD) [170].

1. **Trust and privacy issue**: As there is no open specification or possible inspection on the scanner, no complete trust can be put on the scanner. It is difficult to determine if the scanner performs the correct actions while preserving the local system security policy. Since the update system is typically hosted on the vendor's server, there is also no guarantee that any local sensitive information will not leak to external entity(ies).
2. **Non-standard vulnerability checking issue**: Windows Update behaves more as a vendor's patch-updating mechanism rather than a standardized vulnerability entry checking. Thus, little feedback is given back to users in terms of a standardized vulnerability report information. This might be too limiting for an administrator who, for example, wants to ensure that his/her systems are up-to-date against recent vulnerability reports regardless of whether patches for the vulnerabilities are available or not.

Thus, in contrast to this black-box and vendor-specific update model approach, we propose an open system which can cater to heterogeneous environments.

6.1.3 Vulnerability Description

Besides describing a vulnerability entry, our proposed vulnerability description scheme also aims to allow the scanner to analyze a potential *chain of exploits* involving multiple vulnerabilities. More specifically, the specification must take into account of the effects of a vulnerability in a way that allows us to verify whether they can induce the necessary condition(s) for the exploitation of other vulnerabilities.

Modeling vulnerabilities and their interactions can be traced back to the Kuang system [17]. Baldwin proposed a rule-based system called Kuang, and developed a sample Unix scanner called U-kuang [17]. Based on a set of deduced access privileges on a host, the system tries to find any ways the host's access policy can be violated. COPS [46] incorporated the U-kuang system into its host-based scanner. NetKuang [189] extended the rule set in Kuang to allow it to also work in a networked environment. Based on the concepts from Kuang, the KuangPlus system [70, 172] was developed as a Perl-based scanner. Our prototype scanner is also written in Perl, and is inspired by KuangPlus due to Perl's versatile text processing facilities. Given this Perl-based scanner, we propose our vulnerability description scheme which we find easy to describe and amenable for vulnerability-chaining analysis. Unlike KuangPlus [70, 172], we take a vulnerability check as a declarative data rather than a code.

Open Vulnerability and Assessment Language (OVAL) [117] is a security assessment language from MITRE, which is supported by the U.S. Department of Homeland Security. The OVAL Language specifies how to check a host for the presence of software vulnerabilities using XML. The vulnerability checks are to be verified by a scanner called OVAL Interpreter. The present OVAL Definition Language however does not allow us to abstract the effects of the exploitation of a vulnerability. [100, 34] presented the DESEREC vulnerability definition language, which extends our Movtraq description language [160] by using the OVAL format in order to enable vulnerability-chaining analysis.

6.2 Movtraq Framework: System Overview

We call our proposed framework for automated host vulnerability alert processing as *Movtraq (Machine Oriented Vulnerability and Tracking)*. Figure 6.1 shows the system overview of Movtraq. The integrated Movtraq vulnerability database is designed to be compiled from multiple sources and is usable directly by an automatic scanner.

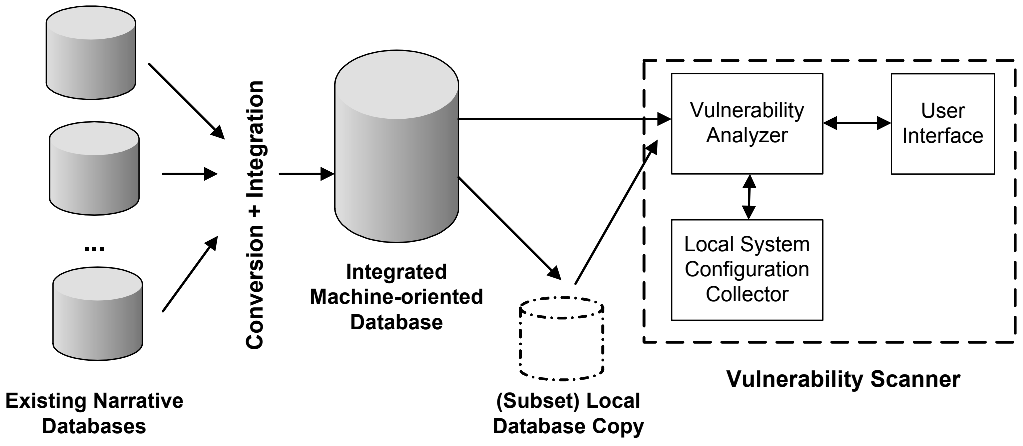
...
Existing Narrative Databases
Conversion + Integration
Integrated Machine-oriented Database
(Subset) Local Database Copy
Vulnerability Analyzer
User Interface
Local System Configuration Collector
Vulnerability Scanner

Figure 6.1: System overview of Movtraq automated framework, showing the vulnerability database and scanner. Note that (a subset of) the database may be replicated in the target's host or a proxy server within the same administrative domain.

We now explain the three main components of the framework, namely: integrated database scheme representation, vulnerability description expressions, and automated scanner.

6.3 Movtraq Vulnerability Database

6.3.1 Design Goals

As mentioned, our philosophy for Movtraq database is that vulnerabilities should be stored as well-structured data/description. The data can be stored in a database (e.g. relational database), or any data description language such as XML. Our proposed database is designed with the following criteria:

- Each vulnerability entry includes the general information of the vulnerability, the conditions (*pre-requisites*) in which it can affect a host, and its *consequences*.
- The pre-requisites and the consequences of a vulnerability are described using an abstraction which we call a *vulnerability description expression scheme*. It allows a precise formulation of the nature of the vulnerability, and is machine processable.
- The structure of the database should allow easy retrieval both by users and automated-tools via SQL.

6.3.2 Content of a Vulnerability Entry

The main challenge in designing the new database is to determine what the actual contents of each vulnerability entry should be. For our proof of concept, we focus on what the database *should contain* rather than on constructing an optimal database schema. The data fields corresponding to a vulnerability entry fall into the following three categories:

General Information

This portion mostly contains references to several public vulnerability databases such as CERT and Bugtraq. The purpose of these fields is to give the user a reference to the original source of information. This portion is mainly for human understanding.

Vulnerability Conditions

This second category provides the main content of the vulnerability information. A vulnerability normally exists within a context. Therefore, it can be described in terms of its *component factor* and the associated *environmental factors*. By “component factor”, we mean the system component (i.e. application or OS) where the vulnerability originates. “Environmental factors” refers to the settings or services in the local host which make it subject to the vulnerability.

We distinguish two following types of presences of a vulnerability on a host:

- vulnerability that *currently* exists on the system; and
- vulnerability that *potentially* exists on the system.

There are four possible combinations of the checking results on the component and environmental factors of a vulnerability, which are diagrammatically shown in Table 6.1:

Case 1: Component factor: *match* & Environmental factors: *match*. We will obtain this result when a vulnerable component exists on a host and the host’s settings match all the required environmental factors. In this case, we will conclude that the vulnerability *exists* on the host.

		Environmental Factors	
		Match	**No-Match**
Component Factor	**Match**	*Vulnerability Exists*	*(Potential) Vulnerability*
	No-Match	*Vulnerability Free*	*Vulnerability Free*

Table 6.1: Combination of checking results between component and environment factors.

Case 2: Component factor: *match* & Environmental factors: *no match.* This occurs when we can detect a vulnerable component on the host, however the host's settings do not match the environmental requirements. While the vulnerability is currently not applicable yet, it has the *potential* to affect the system should its settings change. For example, consider the case of "Apache Web Server Chunk Handling Vulnerability" (`http://www.cert.org/advisories/CA-2002-17.html`). Even if Apache is installed in our system, the host will not be affected by this vulnerability as long it does not provide http services.

Case 3: Component factor: *no match* & Environmental factors: *match.* In this case, the vulnerability would appear to be not applicable. However, there is a subtle issue. Consider the case of OpenSSL which previously had several exploitable stack overflow vulnerabilities (`http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0656`). OpenSSL may not be installed as an individual component. As such, even if there is a database entry for the OpenSSL vulnerability, this would return a negative result in terms of the vulnerability component factors. However, OpenSSL is commonly included in various applications such as Apache, Sendmail and Bind. Thus, it is necessary to check for the existence of such applications. To deal with this case, we therefore need a vulnerability entry to list all the affected applications based on the dependency of the vulnerable software component(s).

Case 4: Component factor: *no match* & Environmental factors: *no match.* The vulnerability does not exist on the local host. However, the remark on vulnerability component dependency in Case 3 also applies here.

Vulnerability Impact (Consequences)

The third category of data fields concerns with the impact of vulnerability, which describes the possible consequences of a vulnerability if it is successfully exploited. In our proposed database, it is represented using the vulnerability description expression. This enables checking of the relationship among different vulnerabilities and whether they can affect one another.

6.3.3 Database Structure

As we have argued, the exact structure of the database is not very important. Rather, it is the selected content and having it in a machine processable format. In our proof-of-concept design, the database has seven main entities (tables). The description of these entities is given in Appendix B.

6.4 Vulnerability Description Expressions

As shown above, both the environmental pre-requisites and consequences of a vulnerability entry require a machine friendly specification. After studying ∼1,000 vulnerability alerts from CERT advisory database, we found that most of the information for these two categories can be described effectively using the *vulnerability description expressions* as describe below. These expressions are designed to be easily processable using text-processing based vulnerability scanners, such as our Perl-based prototype scanner inspired by KuangPlus [70, 172].

An expression is written with the following syntax:

$$\langle Vulnerability\_Expression \rangle := \langle Target\_Object \rangle \mid \langle Action \rangle \, \langle Target\_Object \rangle. \qquad (6.1)$$

An action is written prefixed by '@'. Table 6.2 illustrates the actions and the types of the corresponding target objects. Here, we use a concise notation mainly for brevity. A real system may also support multiple syntaxes.

Syntax	Semantics
@G $\langle u \mid g \rangle$	Gain $\langle$ *user_object u* $\mid$ *group_object g* $\rangle$
@R | W $\langle f \mid m \rangle$	Read | Write $\langle$ *file_object f* $\mid$ *memory_object m* $\rangle$
@A $\langle f \mid m \rangle$	Access (read and write) $\langle$ *file_object f* $\mid$ *memory_object m* $\rangle$
@C $\langle f \rangle$	Create $\langle$ *file_object f* $\rangle$
@K $\langle f \mid m \rangle$	Corrupt $\langle$ *file_object f* $\mid$ *memory_object m* $\rangle$
@X $\langle f \mid c \rangle$	Execute $\langle$ *file_object f* $\mid$ *code_object c* $\rangle$
@S | I $\langle n \mid a \rangle$	Crash | Disrupt $\langle$ *node_object n* $\mid$ *application_object a* $\rangle$
@D $\langle n \mid a \mid s \rangle$	Deny $\langle$ *node_object n* $\mid$ *application_object a* $\mid$ *service_object s* $\rangle$
@U $\langle r \rangle$	Use $\langle$ *resource_object r* $\rangle$
@E $\langle r \rangle$	Exhaust $\langle$ *resource_object r* $\rangle$

Table 6.2: Actions in the vulnerability description expressions.

Rather than giving a formal definition of the target objects, we have listed the target object examples in Table 6.4. The following prefixes are used: '%' is used to denote an actual value; '#' is used to denote a symbolic value; and '&' is used to express users/groups associated with an application/service. As our proof-of-concept implementation is for Unix systems, the examples and objects are also Unix based. Vulnerabilities for other OSes may require extension to the types of target objects and actions.

Syntax	Semantics
⟨**u**⟩:	**User_Objects:**
u#R	Remote user
u#L	Local user
u#S	Super user
u#*	All users
u#P	Physical user
u#U	User whose privilege is beyond that of the current user
u%⟨*uid*⟩	User with UID=*uid* (e.g. 100)
u%⟨*user_name*⟩	User with the specified *user_name* (e.g. '`nobody`')
u&App	User running the corresponding application process
u&Svc	User running the corresponding service (i.e. daemon)
u&Kernel	User who can access or control OS kernel
⟨**g**⟩:	**Group_Objects:**
g#*	All groups
g&⟨*App*|*Svc*⟩	Group of the corresponding application process/service
g%⟨*gid*⟩	Group with GID=*gid* (e.g. 50)
g%⟨*group_name*⟩	Group with the specified *group_name* (e.g. '`sys`')
⟨**f**⟩:	**File_Objects:**
f#*	All files
f#passwd	Pathname corresponding to passwd file (i.e. '`/etc/passwd`')
f#shell	Pathname corresponding to shell files (e.g. '`/bin/bash`')
f#system	Pathname corresponding to system files in OS
f#*(4777)	All files with permission 4777
f#F	Files beyond the current user's access rights
f%⟨$file\_name$⟩	File with the specified $file\_name$ (e.g. '`/etc/httpd/httpd.conf`')
f&App	File associated with the running application process
⟨**m**⟩:	**Memory_Object:**
m#M	Memory area beyond the current user's access right
⟨**c**⟩:	**Code_Object:**
c#(⟨u⟩)	Piece of code with execution privilege of user_object u, e.g. privilege escalation
⟨**n**⟩:	**Node_Objects:**
n#S	Target node where an application is installed and/or related service is running
n#L	Nodes in a local area network
n#N	Network
n%⟨$IP_1[-IP_2]$⟩	Node with the specified IP address (may be a range)
⟨**a**|**s**⟩:	**Application_Objects and Service_Objects:**
a%⟨*AppName*⟩	Application with the specified *AppName* (i.e. as listed by '`ps`' command)
s%⟨*SvcName*⟩	Service with the specified *SvcName*
⟨**r**⟩:	**Resource_Objects:**
r#M	Memory
r#CPU	CPU
r#B	Network bandwidth
r#D	Disk space

Table 6.3: Objects in the vulnerability description expressions. The prefix '%' is used to denote an actual value, '#' for a symbolic value, and '&' for expressing users/groups of an application or service.

6.4.1 Examples using Vulnerability Expressions

The following examples use the expressions to describe various vulnerability consequences:

- @D n#N: Denial of service for the whole network (in: Cisco IOS Interface Blocked by IPv4 Packet, CERT_ID: VU#411332).
- @G u#S : Gain superuser right (in: Linux Kernel Privileged Process Hijacking Vulnerability, Bugtraq_ID: 7112).
- @G u#R : Gain remote user right (in: Apache htpasswd Password Entropy Weakness, Bugtraq_ID: 8707).
- @R f%/etc/passwd : Read file '`/etc/passwd`'.
- @X f#*(4777) : Execute a file with setuid permission.

The following are examples of portions of the machine oriented fields in the database for several vulnerabilities:[3]

- *MySQL Password Handler Buffer Overflow Vulnerability*:
  CVE_ID: CAN-2003-0780
  Bugtraq_ID: 8590
  Vul_Con: @G u%mysql; @G u#L; @X c#(u%mysql)
  Vul_OS: null
  Vul_App: (*Various Mysql versions*)
  Env_User: u#L
  Env_File: null
  Env_OS: null
  Env_App: mysql
  Env_Remote: No
  Exploit: No

- *Linux Kernel IOPERM System Call IO Port Access Vulnerability*:
  CVE_ID: CAN-2003-0246
  Bugtraq_ID: 7600
  Vul_Con: @A f#F
  Vul_OS: (*Various Linux distributions*)
  Vul_App: null
  Env_User: u#L
  Env_File: null
  Env_OS: Linux kernel 2.4.0 - 2.4.21, 2.5.0 - 2.5.69
  Env_App: null
  Env_Remote: No
  Exploit: No

[3]For simplicity, multiple expressions are separated by semicolon.

- *Linux 2.4 Kernel execve Race Condition Vulnerability*:
  CVE_ID: CAN-2003-0462
  Bugtraq_ID: 8042
  Vul_Con: @A f#F; @X c#(u#S); @G u#S
  Vul_OS: (*Various Linux distributions*)
  Vul_App: null
  Env_User: u#L
  Env_File: f#*(4111)
  Env_OS: Linux Kernel 2.4.0 - 2.4.21
  Env_App: null
  Env_Remote: No
  Exploit: Yes

- *Multiple Vulnerabilities In OpenSSL*:
  CVE_ID: CAN-2002-0656
  Bugtraq_ID: 5363
  Vul_Con: @X c#(u&App); @G u#L
  Vul_OS: null
  Vul_App: (*Various Apache versions and OpenSSL-based applications*)
  Env_User: u#R
  Env_File: null
  Env_OS: null
  Env_Svc: (*Corresponding service provided by the vulnerable application*)
  Env_Remote: Yes
  Exploit: Yes

6.4.2 Translation Issues

From our experiments in translating text-based vulnerabilities into vulnerability expressions, we encountered the following issues:

- The vulnerability description in the narrative database sources is sometimes rather vague. Some examples are: "could expose sensitive information to local attackers" (Bugtraq_ID: 8233), "gain access to sensitive information" (Bugtraq_ID: 9558), or "leads to unauthorized access to attacker-specified resources" (Bugtraq_ID: 9778). We however require a more precise consequence, which either means settling for a general (high-level) consequence, or that much more work is required to assess the vulnerability.
- Our vulnerability expression language is designed to capture general expressions at the OS level. It does not express various application specific descriptions, such as: "to access variables outside the Safe compartment" (Perl, Bugtraq_ID: 6111), or "could compromise the private keys of ElGamal signing key implementation"

(GnuPG, Bugtraq_ID: 9115). Such consequences are approximated by our translation into the closest vulnerability expressions capturable by our language. In the two examples above, we can rewrite them into: access of memory and files that are beyond the current user's right respectively.

- Some vulnerability consequence entries, particularly those of CAN (didate) type, are listed as "unknown consequence" (e.g. Bugtraq_ID: 10428). Hence, we either have to ignore such consequences until they are known and specified, or use a special notation to indicate unknown consequences.

6.5 Movtraq Vulnerability Scanner

6.5.1 Design Goals

Our objective is build an automatic scanner that can use the defined database to perform the following tasks:

- Check whether a given vulnerability exists on a local host;
- Notify the existence of potential vulnerabilities on a local host;
- Scan a local host for all possible vulnerabilities;
- Analyze the relationship among different vulnerabilities, e.g. whether one vulnerability can be exploited to lead to another.

6.5.2 Implementation

To demonstrate the use of the Movtraq database, we implement a prototype automatic vulnerability scanner called the *Movtraq vulnerability scanner*. The scanner runs on two different versions of Unix, namely Red Hat Linux and FreeBSD. This is to demonstrate some degree of platform independence.

Our prototype scanner runs as a user-mode application following its invocation by the administrator. An alternative operation mode is to integrate it into the kernel, which then checks a particular binary prior to the binary's execution. To avoid any significant delay in binary execution, this pre-execution vulnerability scanning would be more feasible if the host maintains a local (up-to-date) copy of the vulnerability database, or that the database is hosted by a server within its network. These possible deployment scenarios of the database are discussed more in Section 6.6.1. To reduce the database size, the host can choose to copy only the relevant entries based on its installed software packages. In our prototype, which is aimed at highlighting the feasibility of the machine-processable database and corresponding automated scanner, we opt to implement a user-mode scanner which can be run interactively by the administrator.

The overall structure of the scanner together with the database is depicted in Figure 6.1. The integrated Movtraq database is stored in MySQL. The scanner consists of a local system configuration collector which collects information about the OS (i.e.: which processes are running, which ports are open, hardware details), applications, and services on the system. Software versions are obtained by using the `rpm` utility on Red Hat and the `pkg_info` utility on FreeBSD. The scanner is written in Perl, and queries the MySQL Movtraq database using SQL.

An abbreviated sample log from running the scanner illustrates how OS, application, and environmental checkings are performed:

```
1. Apache Mod_Auth_Any Remote Command Execution Vulnerability
   Application version check: positive.
   Application environment (service port) check: negative.
   Conclusion: source application is detected, default port required
   is not open. Potential vulnerability exists but does not affect
   current system configuration.

2. Sun One/iPlanet Web Server Vulnerability to DOS
   Application version check: negative.
   Conclusion: source application not detected, safe from vulnerability.

3. Linux Kernel IOPERM System Call IO Port Access Vulnerability
   OS version check: positive.
   OS environment (running kernel version) check: positive.
   Conclusion: vulnerability detected!

4. MySQL Password Handler Buffer Overflow Vulnerability
   Application version check: positive.
   Application environment (process object) check: positive.
   Conclusion: vulnerability detected!
```

Only some of the pertinent checks from the log are shown above to illustrate the following cases:

Example 1: Apache vulnerability exists, but the environmental factor check fails since the required port is not open.

Example 2: no vulnerability since the vulnerable application is not installed.

Example 3: OS vulnerability exists, so only OS component and the (environmental) kernel status checking are used.

Example 4: vulnerability inherent to MySQL, OS environment checking is skipped as it is not required.

6.5.3 Vulnerability-Chain Analysis

An interesting use of the scanner is that it can be used to test if existing vulnerabilities can be combined (chained) together to create more serious vulnerabilities. This mimics what a hacker might do to take advantage of indirect weaknesses on the system.

Consider the following example which is typical of a privilege escalation attack. Suppose the system has the following two vulnerabilities:

Name:	Buffer Management Vulnerability in OpenSSH		
Vul_ID:	57		
CVE_ID:	CAN-2003-0693	Bugtraq_ID:	8628
Vul_Con:	@G u#L	Vul_OS:	null
Vul_App:	(*Openssh applications*)	Env_Usr:	u#R
Env_File:	null	Env_OS:	null
Env_App:	(*Services provided by the vulnerable applications*)		
Env_Remote:	Yes	Exploit:	No

Name:	Linux 2.4 Kernel execve Race Condition Vulnerability		
Vul_ID:	48		
CVE_ID:	CAN-2003-0462	Bugtraq_ID:	8042
Vul_Con:	@G u#S	Vul_OS:	(*various Linux distributions*)
Vul_App:	null	Env_Usr:	u#L
Env_File:	f#*(4111)	Env_OS:	Linux kernel 2.4.0 - 2.4.21
Env_App:	null		
Env_Remote:	No	Exploit:	Yes

In this example, the scanner discovers that both vulnerabilities are present. From Vul_ID: 57, a remote user (u#R) can gain the local user access (@G u#L), and this can chain onto Vul_ID: 48 which has two environmental factors (i.e. the local user access: u#L and a setuid executable file:f#*(4111)). Thus, the scanner discovers that a remote user may be able to exploit the two vulnerabilities to gain the local root access.

The vulnerability-chaining analysis illustrates the benefit of a machine oriented approach and the use of vulnerability expressions to analyze the relationships among the vulnerabilities.

6.6 Discussion

6.6.1 Deployment Strategies for Movtraq

The prototype Movtraq system is sufficiently useful to be deployed in a number of ways. Some of the potential scenarios are depicted in Figure 6.6.1:

Scenario 1: Local vulnerability database, local scanner.

Here, each local machine hosts its own database. The Movtraq database is meant to have been (securely) downloaded from an authoritative server. This scenario has the advantage that all operations can be done locally. The disadvantage is that an up-to-date database has to be maintained on every host.

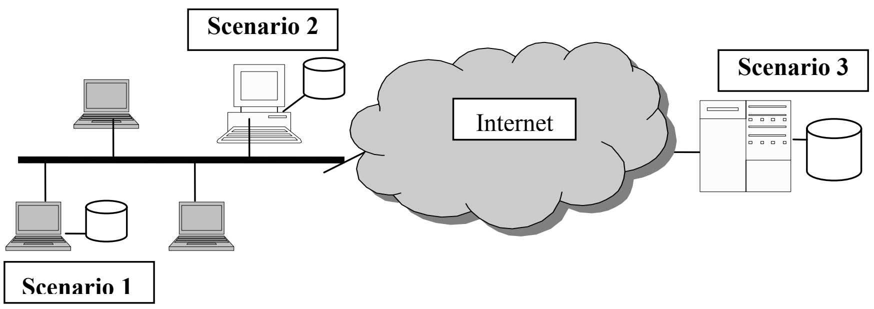

Figure 6.2: Deployment options for Movtraq vulnerability database.

Scenario 2: Organization-wide database, local scanner.

This simply extends Scenario 1 to an organizational context where there is an organization-wide database server. Where multiple machines have exactly the same configuration, one may choose to check only a subset of the machines.

Scenario 3: Internet-based database, local scanner.

Lastly, as in the automated update systems, a database server somewhere on the Internet serves as the database repository.

6.6.2 Movtraq and Recent Standardization Efforts

Our Movtraq framework was proposed several years back when machine-oriented vulnerability database and automated processing were still not widely addressed. At present, there exist several on-going collaborative standardization efforts, including those conducted by the U.S. Government-sponsored organizations, such as OVAL [117], National Vulnerability Database (NVD) [170], CPE [115] and SCAP [126].

Both OVAL Repository [119] and NVD [170] were specially created to be a unified and machine-oriented vulnerability database. These two databases are thus in line with our proposal of Movtraq vulnerability database. The OVAL initiative also has a language, called OVAL Language [120], which specifies how to check a host for the presence of vulnerabilities. A vulnerability entry is specified in XML in order for a scanner, called OVAL Interpreter, to perform the corresponding check(s) on a target machine. A sample OVAL Interpreter, although not a fully functional scanning tool yet, is freely available [118]. Thus, as an on-going collaborative efforts, the OVAL initiative does share the same philosophy and many common goals with our proposed framework.

The present OVAL Definition Language however does not allow us to abstract the effects of the exploitation of a vulnerability. As such, the language does not facilitate easy support for the vulnerability-chaining analysis, which we aim to provide with Movtraq. This deficiency led [100, 34] to present their own definition language, which extends OVAL definition with ideas taken from our Movtraq vulnerability description expressions.

They describe vulnerabilities in terms of the conditions (pre-conditions) that make them exploitable and the effects (post-conditions) on the victim system.

In our present approach with Movtraq, objects involved in the checking of environmental factors are stored as database fields. The scanner is assumed to implicitly know how to operate on them. For an actively-evolving standardization effort like OVAL, such a rather rigid model may be too limiting. A database model worth investigating is one that standardizes all possible scanner's basic checks on the target machine. Such standardized checks could also be parameterized to make them as generic as possible. These checks are then recorded *as database entries*. An entry thus specifies how the check should be performed (on various applicable OSes), fields containing the values of objects involved in the checking, and notational descriptions of the pre-conditions and the consequence as in Movtraq. In this model, the capabilities of a scanner are thus basically limited by the checks defined in the database. This would answer the growing nature of on-going standardization effort on vulnerability language. At the same time, it will establish a set of standardized basic operations that can be performed on a target host, an area which is presently still not well addressed in OVAL. Moreover, as discussed earlier, the model will also allow for vulnerability-chaining analysis.

To describe vulnerable components, OVAL makes use of CPE [115]. Unlike CPE, our proposed Software_ID scheme (see Section 5.6) is associated not with a family of software, but with *each* individual binary file. Using Software_ID thus gives us a flexibility in choosing to block the execution of either a particular vulnerable binary or the whole software package. The former is useful when the vulnerable binary is a non-critical component of a software package. In the latter, we can simply put wildcards ('*') on both module_ID and version_ID (see Eq. 5.3) in the Software_ID of the vulnerable component.

6.7 Chapter Summary

We believe that there is a pressing need for vulnerability databases that allow for automated host vulnerability processing. We have demonstrated a proof-of-concept database that allows effective integration of vulnerability data from multiple sources and can be used directly by a scanner. The proposed Movtraq scheme decouples the vulnerability information from its processing by storing a vulnerability description as declarative data. The scanner simply collects a system's local information and analyzes whether vulnerabilities apply on that host. Due to our use of the vulnerability description expression scheme in describing a vulnerability's pre-requisites and consequences, the analysis of potential chain of exploits involving multiple vulnerabilities becomes possible. Our prototype system was designed for Unix-based systems. However, we believe that the fundamental concepts in our design should still be substantially applicable to other OSes.

Chapter 7

Lightweight and Near Real-Time Certificate Revocation Schemes

As previously mentioned, timely and efficient certificate revocation service is a critical prerequisite for host security. This revocation service must be reliably available to ascertain that public keys belonging to external parties are not revoked. In our PPLC context, the service is thus essential for secure software distribution, including possible mobile code execution on a local host (Step 1 of the PPLC); and public-key based interaction of a running program with external hosts (Step 4). Disregarding such revocation verification would result in a catastrophic compromise to a host security.

In X.509-based PKI [75], certificate revocation is mainly supported by Certificate Revocation List (CRL) [37]. However, CRL is widely perceived to be costly, and is attributed as one of the main impediments to a successful global PKI deployment [96, 64]. Among others, CRL suffers from high bandwidth requirement and (generally) low revocation timeliness guarantee. Several alternatives, such as Online Certificate Status Protocol (OCSP) [124], have been proposed. OCSP offers a potentially real-time recency, but requires the Certificate Authority (CA) to respond to any incoming query in real time with a signed message. As such, it imposes significantly higher computational and network requirements on the CA. Additionally, there exists a privacy issue since the information of all verifiers validating a certificate becomes known to the CA.

In this chapter, we look at a satisfactory revocation solution to support two important emerging trends on the Internet. Firstly, the growth of transactions relying on certificates increases the need for *timely* certificate revocation service. Secondly, rapid growth in mobile devices, whose power and bandwidth are limited, mean that any revocation solution should be *low* in terms of computation, bandwidth and storage requirements on the verifier. The lack of lightweight and timely revocation today means that many applications, particularly on bandwidth- and processing-limited clients, are unable to offer proper revocation checking [141, 96]. Although this situation is slowly changing, it is still unsatisfactory given the above trends.

More specifically, our goal here is to have a *practical* and *scalable* revocation solution, which provides *near real-time* certificate revocation scheme (e.g. from 1 to 10 minutes) yet allows for *fast verification* and *low bandwidth* on the verifiers. In addition, we would like to avoid excessively high overheads on *any* of the entities so as to avoid any undesirable service bottlenecks. Lastly, we also aim to provide *privacy assurance* so that queries from the verifiers are unknown to external parties, including the CA. Although the CA is trusted for managing certificates, transaction information involving a certificate *should not necessarily* be made available to the CA.

We propose two new and simple revocation schemes, CREV-I and CREV-II, which take advantage of the availability of the "Extended-Validation SSL Certificate" (EVC) [28] to utilize a *principal's server* as the directory for *its own* revocation information. We argue that this new proposed revocation setting gives a simple and practical solution to the revocation problems, and allows CREV-I and CREV-II to achieve the above mentioned objectives. CREV-I scheme enhances CRS/NOVOMODO [110, 111] to achieve the stated objectives; whereas CREV-II scheme enhances OCSP and makes it more scalable. In addition, the new setting also provides suitable incentive models for all the involved parties to achieve both revocation timeliness and scalability.

Another contribution of this chapter is the use of a more realistic cost-analysis framework for evaluating certificate revocation schemes. We follow the approach taken in [99, 72] which uses empirical revocation data from VeriSign. Our framework is a substantial extension of [99, 72] by incorporating the following: the revision of the CRL size derivation using a more accurate calculation method; and the generalization of the CRL size estimate with both certificate generation and the CRL issuance in minute(s) instead of day(s). Furthermore, our analysis incorporates more realistic features in order to derive more realistic derivations of costs, which include: a more realistic query model on revoked certificates, the derivation of query probability on revoked and valid certificates, and more accurate accounting of message sizes based on the standardized CSI formats.

Our evaluation demonstrates that the two CREV schemes achieve good scalability on the part of the CA and the principals servers, and incur low computation and bandwidth costs on the part of the verifier.

The remainder of this chapter is organized as follows. Section 7.1 provides a framework for designing revocation schemes, and gives a brief survey of existing schemes and revocation-cost analysis works. Section 7.2 provides additional background on EV Certificate, CRS/NOVOMODO, and the cost analysis of [99, 72]. We explain our revocation setting and CREV schemes in Section 7.3. We then elaborate our realistic cost-calculation framework, and analyze CREV schemes as well as several existing ones in Section 7.4. Section 7.5 discusses the comparison results, and Section 7.6 concludes this chapter.

7.1 Certificate Revocation Framework and Related Works

Let us first establish some definitions for use in our discussion in this chapter. We refer to the subject of a certificate as *principal*, the principal of an Extended Validation Certificate (EVC) as *EVCP*, and the party verifying certificates as *verifier*. We also adopt the term *Certificate Status Information (CSI)* [73] to denote CA's released information pertinent to the validity of a certificate. Hence, CSI encompasses CRL data, OCSP messages, and hash tokens in CRS/NOVOMODO. *Certificate Management Ancillary Entity (CMAE)* is a generic term for a designated repository from which a verifier may obtain the CSI.[1] We define *revocation latency* (based on [4]) as the time interval between when a CA makes a revocation record and when it makes that information available to the verifiers. A low revocation latency means a high *revocation timeliness guarantee*. In a per-certificate revocation scheme, such as CRS/NOVOMODO, OCSP and ours here, we assume that the CA will not unnecessarily delay publishing the CSI indicating a revocation event in contrast to a periodic batch-processing mechanism.[2]

Below, we first outline a framework for designing certificate revocation schemes. We then briefly summarize related existing revocation schemes, and mention related works on the cost analysis of revocation schemes.

7.1.1 Framework for Certificate Revocation Schemes

We can characterize certificate revocation schemes based on the following four properties:

(P_1) **Placement of directory:** The directory (or sometimes referred to as cache), which holds a copy of CSI issued by the CA, can be placed in either: the *CMAE* as a designated repository, the *verifier*, or the *principal*. It is also possible that no directory is employed at all, such as in NOVOMODO and OCSP with the CA as the Responder.

(P_2) **Scope of released CSI:** A released CSI is verifiable to provide status information for either: *all* certificates, *a subset of* certificates, or *a single* certificate issued by the CA. CRL data, for instance, gives the statuses of either all (as in the standard CRL) or a subset of (as in the Partitioned CRL) certificates. In contrast, CSI in CRS/NOVOMODO or OCSP accounts for an individual certificate.

(P_3) **Positive or negative status representation:** The CA can state the statuses of certificates using either a *negative* (black-listing) approach, or *positive* approach where the status of each certificate is explicitly stated. Note that the negative

[1]A CMAE may be trusted or untrusted depending on the revocation scheme.

[2]Although OCSP is considered as a real-time service, its timeliness is only as up-to-date as the latency involved in obtaining the CSI from the Responder's definitive source. In our work here, we take an optimistic assumption that a CA providing OCSP service would provide *the lowest latency it can afford* as permitted by its applicable policy, in order to take advantage of the available real-time setting.

approach can only work when the scope of CSI is for all CA's certificates or a well-defined subset (partition) of CA's certificates.

(P_4) **Use of Linked CSIs for Subsequent Status Updates:** A CSI generally comes signed by the CA in order to ensure data origin authentication. It is also possible that the signed CSI additionally contains some value identifying a hash chain, which is usually the tip of the hash chain. Subsequent periodical releases of compact and unsigned hash token information update the status(es) of the certificate(s) in a lightweight manner. We thus can characterize a revocation scheme based on *whether or not* it employs a hash chain for lightweight subsequent status updates.

Table 7.1 characterizes several revocation schemes based on properties P_1 and P_2.

Directory	**CSI Coverage Scope**		
Placement	**All CA's Certificates**	**Multiple Certs**	**Individual Cert**
No directory (handled by CA)			NOVOMODO [111], OCSP [124], H-OCSP [122], MBS-OCSP [21]
CMAE	CRL [37], HS [6], CRT [83], 2-3 tree CRT [125]	Partitioned CRL, CSPR [60]	CRS [110]
Verifier	CPR [151], BCPR[150]		
Principal		*Optimized CREV-II*	*CREV-I, CREV-II*

Table 7.1: Possible combinations of directory placement and CSI scope for certificate revocation schemes. Note that we consider the OCSP scheme with the CA as the responder.

7.1.2 Related Works

We briefly summarize the related existing revocation schemes below. Our focus here is to highlight their main strengths and weaknesses, and also their differences with our schemes. Section 2.5 provides more detailed background information on some of the surveyed schemes.

Standardized in X.509 [75] and also profiled in the IETF [37], CRL is also the most widely supported revocation scheme. It is however widely known to have serious shortcomings. Firstly, the CRL may eventually grow to a cumbersome size in very large PKIs, e.g. ∼750 KB in Verisign's [141] and ∼40 MB in the U.S. DoD's [129]. Secondly, the downloaded CRLs may be mostly useless given that more than 90% of the information is irrelevant to the verifiers [141]. Lastly, CRL does not offer adequate timely revocation guarantees. It is common for a CA to update its CRL only *daily*, as suggested in [177], although the CA may have a service for a shorter window period presumably with a higher charge.

There are several methods for improving the basic CRL mechanism, such as CRL Distribution Points, Delta CRLs, and Indirect CRLs (see [4] for a survey). However, all these schemes still put the same requirement on the verifier to obtain a complete revocation list, which includes the unrelated entries. Furthermore, for our setting of EVCP servers as CSI distribution points, we require a scheme that carries a certificate's liveness status instead of the black-listing approach taken by CRL-based mechanisms.

OCSP [124] was proposed to provide a more timely certificate status checking. An OCSP Responder is required to return the status information about a specific certificate in a digitally signed response. Since OCSP is an online service, it necessitate a prompt and reliable OCSP Responder system with a high level of security. As such, it imposes significantly higher computational and network requirements on the Responder [93] (see also Section 2.5). Additionally, there exists a privacy issue with OCSP. Since the Responder is consulted whenever a verifier validates a certificate, it therefore knows all verifiers dealing with a principal. In our proposed schemes, the verifier obtain the CSI of an EVC principal directly from the principal's server. Hence, our schemes does not reveal certificate status queries to any third party, including the Responder (the CA).[3] Although the CA is trusted for managing certificates, the transaction information involving a principal and a verifier should not be made available to the CA.

Several modifications have been proposed on OCSP, such as H-OCSP [122] and MBS-OCSP [21]. Similar to our proposed scheme (CREV-I), H-OCSP and MBS-OCSP make use of a hash-chaining technique. However, they require the CA to cater for a *potentially large number of verifiers*. Hence, the CA's bandwidth and storage requirements remain high. Our CREV schemes operate differently since the CA maintains hash chains belonging to EVC principals, whose number is much smaller than that of the verifiers. Moreover, we also employ a session-based hash chain as to reduce costs further (see Section 7.3).

Aiello et al. [6] proposed an improvement to CRS called "Hierarchical Scheme" (HS) aimed at reducing the CA-to-CMAE communication while still maintaining a low query communication. The improvement however comes at the price of a significant increase in the certificate size. Kocher [83] proposed Certificate Revocation Tree (CRT) which employs Merkle Hash Tree (MHT). The scheme however suffers from a high computational cost needed to update the CRT. Naor and Nissim [125] subsequently extended the CRT by using a more suitable data structure, a 2-3 tree. All these three schemes are however rather different from CRS/NOVOMODO since the employed data structure maintains the statuses of *all* revoked certificates in contrast to an individual certificate status.

Certificate Push Revocation (CPR) [151] and Beacon CPR (BCPR) [150] suggest placing the cache at the verifiers. Although the schemes may work for highly connected and high-bandwidth verifiers, they however seem impractical for mobile verifiers con-

[3]Recall again that, in this thesis, we assume a scenario of OCSP deployment where the OCSP Responder is co-located with the CA.

nected intermittently or at low-speed. The works also suggest placing the cache on the ISPs. However, there exists a question of economic incentive for the ISPs to provide the service.

Certificate Space Partitioning with Renewals (CSPR) [60] was proposed to reduce the high CA-to-directory communication cost found for example in CRS [110]. In CSPR, the CA divides its certificates into partitions, and signs the CSI for each partition which contains the *status bits* of all certificates in the partition. If there was no status change for any of the certificates in a partition, the CA then renews the partition by releasing a hash-chain information whose tip is embedded in the partition's CSI. Unlike CREV which utilizes a new proposed setting with the *principals* as directories, CSPR employs *CMAEs* as directories.

One of our proposed schemes, CREV-II, can utilize an optimization technique where an OCSP Response carries the common *valid status* of multiple certificates. The technique is based on the certificate range definition in CRT [83]. A similar technique was also proposed by Koga et al. [84]. Our use of this technique in CREV-II thus demonstrates how it can be utilized in the assumed network setting. In addition, using the developed realistic analysis framework, we also show how well the optimization can reduce the overheads.

Our performance analysis framework offer a more realistic cost calculation of revocation schemes. Below, we briefly describe the differences of our developed framework from previous analysis works such as [190, 92, 99, 72].

The work by Zheng [190] is one of the most widely cited works on cost analysis of certificate revocation schemes. However, it simply assumes that a certificate is always revoked at the half of its issued lifetime. In addition, its analysis of NOVOMODO is based on an assumption that the hash token is released on a daily basis. In contrast, our framework is based on a realistic revocation model which enhances the one proposed in [99, 72]. We consider a revocation service with timeliness guarantee on the order of minute(s) instead of day(s). Moreover, in Section 7.4.2, we derive a realistic query model on revoked certificates. Given the timeliness guarantee in minute(s), this model has a significant effect on the cost calculation of hash-chaining based schemes. This is because there is a great difference in cost between verifying a valid and a revoked certificate.

Lim and Lakshminarayanan [92] also conducted a performance analysis of various revocation schemes. They proposed a more detailed framework than that of Zheng [190]. However, the work is still based on the same assumption that a certificate is revoked at the half of its lifetime. The work derived the probability that a queried certificate is valid/revoked. This allows for a more accurate cost estimate in hash-chaining based schemes. However, the probability derivation is simply based on *the number* of valid and revoked (but not yet expired) certificates in the certification system. In contrast, our work (see Section 7.4.2) develops a realistic exponential-based model of a query on revoked certificates. This is because, in practice, the number of queries on revoked certificates tend to decrease over time. Our performance analysis is thus more accurate with respect to the CRL size as well as the probability of the query on valid/revoked certificates. Furthermore, we also follow the standardized CSI message sizes in [133] to derive realistic message costs in various revocation schemes.

Our cost analysis framework enhances the one proposed in [99, 72]. We summarize the framework of [99, 72] in Section 7.2.3, and describe how we enhance it in Section 7.4.2.

7.2 Preliminaries

Below we provide a brief summary of EV Certificate, CRS/NOVOMODO schemes, and a realistic cost analysis model proposed in [99, 72].

7.2.1 Extended-Validation Certificates (EVC)

EV Certificates (EVCs) are a special type of the X.509 certificate to deal with the problems of phishing [8] and online fraud. It requires a more extensive investigation of the requesting entity by the CA before a certificate is issued. The investigation includes the existence of a domain name and its associated publicly-accessible server which are exclusively owned or controlled by the entity. EVC is important to provide a stronger assurance against some recent attacks on the X.509 certificate usage, such as certification-chaining attack [102], homograph-based phishing [52, 69, 102], and man-in-the-middle based SSLstrip attack [102]. (See Section 2.5.3 for additional background on EVC.)

We remark that, in theory, our proposed CREV schemes could equally work on the standard X.509 certificate with an included domain name. However, simply accepting a domain name and having the server disseminate the CSI falls short of providing strong assurances on both the certification and the revocation services. The above mentioned attacks in [102, 52, 69], among others, may apply. For our revocation schemes, we need an assurance that a server associated with an EVC principal is a valid server representing the principal as how general public would likely to assume (e.g. `mics0s0ft.com` is not `microsoft.com`). Additionally, a principal is bound to perform its best in following the revocation scheme. Our CREV schemes build upon the EVC infrastructure, since it is a widely-accepted standardized premium certificate mechanism that can provide the needed assurance. Any service provided by a PKI that operates under restricting policies at least as secure as those used for EVC can also work with our schemes.

7.2.2 CRS/NOVOMODO

Certificate Revocation Status (CRS) [110] makes revocation more efficient through its periodical release of compact hash chain information. CRS assumes the involvement of CMAE(s), which receive CSI from the CA and handle the verifier's queries.

The basic idea is as follows. Prior to issuing a certificate, the CA chooses a one-way hash function $H()$, and determines a time interval period d and a hash-chain length ℓ. The lifetime of the certificate is: $d\,(\ell{+}1)$. The CA then adds the following to the published certificate: $HashAlgID$ as the identification of $H()$, d, ℓ, issue time, expiration time[4], and two 100-bit values Y and N representing "valid" and "revoked" status, respectively. Y and N are calculated as follows: the CA first generates two secret 100-bit random numbers Y_0 and N_0, and then computes $Y = H^{\ell}(Y_0)$ and $N = H(N_0)$. The scheme works as follows: on the i-th time interval after the certificate's issuance, where $1 \leq i \leq \ell$, the CA submits the following information to CMAE(s): a signed timestamped string containing all serial numbers of issued and not-yet-expired certificates, and 100-bit V_i for each certificate, which indicates whether it has been revoked or not by the current time interval. V_i is set as follows: $V_i = H^{\ell-i}(Y_0)$ if a certificate is still valid, or $V_i = N_0$ if it is revoked. When a CMAE receives a verifier's query, it then sends V_i to the verifier. Using the current time and certificate's issue time, the verifier can check whether a certificate is valid or revoked by comparing $H(V_i)$ with N or comparing $H^i(V_i)$ with Y.[5]

NOVOMODO [111] was later proposed as an improvement on CRS. It suggests the use of SHA-1 as one-way hash function and the avoidance of the CMAE in the centralized NOVOMODO scheme. The hash-chaining mechanism is the same as in CRS. The work [111] also described how to build a distributed NOVOMODO, which makes use of a single vault and an unlimited number of "untrusted responders" to answer validation queries.

CRS and NOVOMODO have their own limitations. Firstly, CRS has a high CA-to-CMAE communication cost [125, 6]. Our CREV scheme mitigates this since an EVCP acts as a CMAE *only for its own certificate.* Secondly, CRS/NOVOMODO may require a high verifier's processing cost due to the a large number of repeated hash operations. A hash operation is much more efficient than a public-key operation. However, for a sufficiently high timeliness guarantee and long certificate lifetime, the total computation cost of hash operations can be significant particularly for lightweight verifiers. CREV scheme reduces the total length of the hash chain to make the verifier more efficient. Lastly, CRS/NOVOMODO increases the CA's storage requirements into $O(n \cdot \ell)$, where n is the number of certificates. This storage requirement can be reduced at the cost of more computation by the CA [78], thus trading-off storage with computational cost.

[4]The expiration time can actually be derived from: issue time + $d(\ell + 1)$.

[5]If a verifier has obtained a valid V_j at an earlier j-th time interval ($j < i$), the number of hashes required to check the validity is reduced to $i - j$.

7.2.3 Certificate Revocation Model using Empirical Data

The work [99, 72] aims to determine the CA's strategies in releasing CRL so as to reduce the CA's operational cost. To this end, the work derives a realistic estimate of the number of new revocations between two successive CRL generations as well as the size of CRL on a particular day.

Most previous analysis works on revocation schemes usually assume that a certificate is revoked at the half of its lifetime [190, 92]. In contrast, the work [99, 72] gives a realistic revocation model based on the empirical data of CRLs collected from VeriSign. They found that most of the certificate revocations occur during the early part of the certificate's lifetime. In fact, more than 30% of revocations occur within the *first two days* after certificates get issued. In addition, the percentage of revocations decreases as time elapses. This result thus invalidates the assumption about the CRL size taken by most earlier works [190, 92].

The work [99, 72] defines a probability density function (PDF) of certificate revocations over time. Suppose that there are α certificates issued at time X, with uniform issued age β. From time X to $X+\beta$, on average αb of the certificates will be revoked. In [99, 72], the basic time unit between two CRL releases (Δt) is assumed to be of one day. Function $R(t)$ is defined to represent *the revoked percentage*, that is the ratio of the number of revocations that occur in the time interval $[t, t+\Delta t]$, where $X<t<X+\beta$, to the total number of revocations in the time interval $[X, X+\beta]$. The work [99, 72] model the revocation distribution of certificates issued at a particular time with the following exponential PDF[6]:

$$R(t) = ke^{-kt} \tag{7.1}$$

where t is the time, and $k = 0.26$ as the function paramenter which has a good fit to the real observed data. Figure 7.1, reproduced from [99], shows the defined PDF and the actual plot from the empirical data.

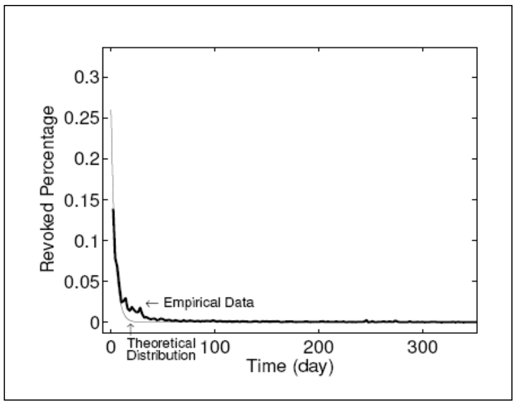

Figure 7.1: The fitted exponential PDF and empirical data for certificate revocations over time (from [99]).

[6]A Probability Density Function (PDF) is defined by taking $\lim_{\Delta t \to 0}$.

The model in [99, 72] derives a realistic estimate of the CRL data size by deriving the number of revoked entries on a particular day. Three scenarios are considered, namely: certificates issued at different times but with the same issued age β; certificates issued at different times and with different issued ages; and a more general case where the number of certificates generated at a particular time is not constant, but follows a Poisson distribution. The work [99, 72] derives an analytical model for the first and the second scenarios respectively, but use a simulation technique for the third. For our comparison framework in this chapter, we only consider the first scenario. If desired, the framework can be extended to analyze the second scenario in a rather straightforward manner.

The number of *new revocation requests* is derived in [99, 72] as follows. Suppose that v is any time in $(0, \beta]$. Function $f(v)$ is defined as the number of new certificate revocations between day v and day $v+\Delta t$ from all of the valid generations:[7]

$$f(v) = \alpha bR(v) + \alpha bR(v - \Delta t) + \alpha bR(v - 2\Delta t) + \ldots + \alpha bR(v - (n-1)\Delta t) \tag{7.2}$$

where n is the number of certificate generations in time period β, i.e. $n = \lceil \frac{\beta}{\Delta t} \rceil$. As mentioned, [99, 72] assumes $\Delta t = 1$ day. Similarly, the time interval between two certificate generations (ΔX) is also assumed to be 1 day. As a result, v in (7.2) is an integer. Thus, when v is in $(0, \beta]$, we have:

$$f(v) = \alpha bR(1) + \alpha bR(2) + \ldots + \alpha bR(v) = \alpha bke^{-k}\frac{1 - e^{-vk}}{1 - e^{-k}} \tag{7.3}$$

When v is in $(\beta, +\infty)$, which represents the steady-state condition, [99, 72] derives that:

$$f(v) = \alpha bR(1) + \alpha bR(2) + \ldots + \alpha bR(\beta) = \alpha bke^{-k}\frac{1 - e^{-\beta k}}{1 - e^{-k}} \tag{7.4}$$

The *size of CRL* is derived as follows. Let $F(v)$ be the valid cumulative number of revocations from time 1 to v. It is also the size of the CRL on that day. For v in $(0, \beta]$:

$$F(v) = \sum_{t=1}^{v} f(t) = \frac{\alpha bke^{-k}}{1 - e^{-k}}\left[v - \frac{e^{-k}}{1 - e^{-k}}\left(1 - e^{-vk}\right)\right] \tag{7.5}$$

For v in $(\beta, +\infty)$, [99, 72] derives:

$$F(v) = F(\beta) = \sum_{t=1}^{\beta} f(t) = \frac{\alpha bke^{-k}}{1 - e^{-k}}\left[\beta - \frac{e^{-k}}{1 - e^{-k}}\left(1 - e^{-\beta k}\right)\right] \tag{7.6}$$

From (7.5) and (7.6), it can be shown that the size of CRL increases from day 0 until day β. After that, in what we call *the steady-state condition*, the size of CRL becomes constant. Note that some revoked certificates may be expired after they reach their issued lifetimes, and accordingly removed from CRL.

We remark that there are a number of issues with respect to the derivations of $f(v)$ and $F(v)$ in [99, 72], which we discuss and address later in Section 7.4.2.

[7]The more recent version of the work [72] uses $N(v)$ instead of $f(v)$ to denote the function.

7.3 CREV Schemes for Lightweight Certificate Revocations

7.3.1 New Revocation Setting

Our CREV schemes differ from other previous revocation schemes in that the directory for *an individual certificate* is placed on the respective *principal's server*. The principal's server thus acts as a distribution point for its own CSI. As such, our schemes allows for the co-location of Web (transaction) server, where a principal's certificate is normally obtained by the verifiers, and the corresponding CSI's distribution point.

This (seemingly small) change has a number of important advantages:

- The CSI distribution servers are self-managed (i.e. independent from the CA), and deal only with *their own* respective verifiers.
- The servers also have a strong incentive to provide reliable and timely CSI access services for their own respective transactions. The new setting thus addresses the incentive problem on CSI management and dissemination [64]. Hence, it has economic advantages to those employing the verifier's ISP as directory [151, 150].
- Scalability issue is also naturally addressed since CSI accesses are now distributed to the respective principals' servers, which are expected to have sufficient capacity to meet the respective transaction volumes.

There are overheads associated with CSI transfer management between the CA (or its proxy) and EVCPs in the new setting. We show later that the overheads are manageable.

To work securely and efficiently under the proposed revocation setting, a revocation scheme should satisfy the following requirements:

(Req_1) **Verified principal's server:** The setting requires a principal to have an exclusively controlled domain name and the associated publicly-accessible server. Furthermore, the designation of the server should provide strong protection against impersonation attacks such as phishing attacks.

(Req_2) **Availability of principal's server:** As a CSI access point for a principal, the principal's server thus must be reliably available to the verifiers in order for latter to complete their online transactions with the principal.[8]

(Req_3) **Per-certificate CSI:** As a principal provides a CSI access service only to its own verifiers, a revocation scheme with per-certificate CSI is therefore more bandwidth and storage efficient.

(Req_4) **Positive status information:** The use of per-certificate CSI means that a revocation scheme should use a positive status representation to confirm the continued goodness ("*liveness*") of a certificate.

[8]Note that our CREV schemes can co-exist with other revocation schemes such as timely CRL. Hence, if a principal's server is temporarily unavailable, the verifier can resort to downloading the CRL in order to validate the status of a certificate.

7.3.2 CREV Overview and Assumptions

To address the Requirement Req_1, our schemes leverage on the availability of the Extended-Validation Certificates (EVCs) so as to ensure a verified domain name and the associated server. In theory, our CREV schemes could also work on the standard X.509 certificate with an included domain name. However, this falls short of providing strong assurance against attacks such as [102, 52]. By building upon upon the EVC, we thus leverage on a widely-accepted standardized premium certificate mechanism which provides the necessary assurance for Req_1.

Our schemes thus make use of an EVCP's server to acts as a CMAE for its own certificate. Figure 7.2 depicts the CSI communication flow between the parties involved.

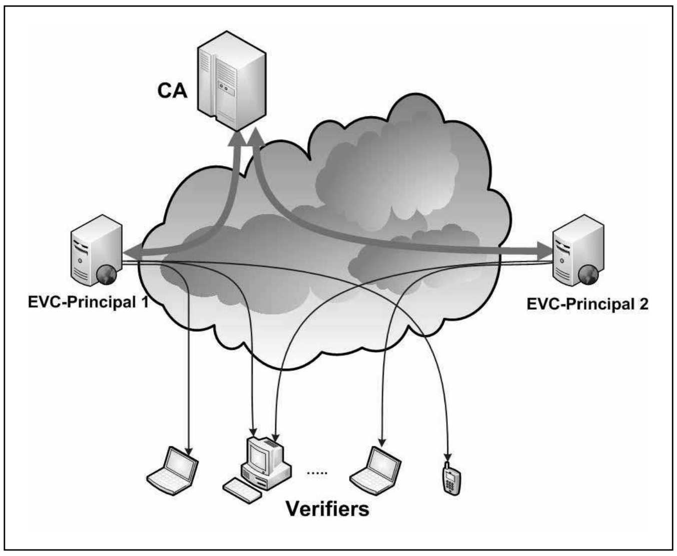

Figure 7.2: CSI communication flow in CREV schemes.

Two existing revocation schemes, CRS/NOVOMODO and OCSP, meet the requirements for per-certificate (Req_3) and positive status (Req_4) CSI. We enhance these two schemes to achieve the stated objectives of providing timely and lightweight revocation service. CREV-I improves CRS/NOVOMODO by setting up a *session-based hash-chaining* service between the CA and an EVCP. It aims to take advantage of the hash-chaining technique, but shorten the length of the hash chain for a faster verifier's operation and a reduced CA's storage requirement.[9] CREV-II takes advantage of available online schemes like OCSP. For each (EV) certificate, the CA produces only one OCSP Response per time interval. It also can make use of an optimization technique using group-based CSI [84], which is inspired by the notion of certificate-range in the CRT.

With respect to the Properties P_1–P_4 discussed in Section 7.1.1, our CREV schemes

[9]Here, we do not deal with amortization techniques such as those based on multidimensional hash chain [78]. Rather, we work at the protocol level to reduce the length of the used hash chains by taking advantage of the proposed new setting. The amortization techniques thus can still work together with CREV-I to make its costs even lower.

thus realize a revocation service with the following chosen design options:

- (D_1) Placement of directory: The directory is placed in the (server belonging to the) *principal* of an EV certificate.
- (D_2) Scope of released CSI: The released CSI provides assurance for *a single certificate* issued by the CA.
- (D_3) Positive or negative status representation: The status of a certificate is stated using a *positive* representation.
- (D_4) Use of linked CSIs for subsequent status updates: CREV-I makes use of a hash chain to provide lightweight subsequent status updates. In contrast, CREV-II does not use such a technique, since it is meant to allow the verifiers to employ the existing online status verification infrastructure such as OCSP.

Note that the Requirement Req_2 is met by our assumption that a principal's server has a strong incentive to provide reliable and timely CSI access service for its own transactions. In addition, we also make following assumptions between the EVCPs and the CA, which are reasonable under typical settings of an Internet transaction:

- Synchronized clocks are available to determine the current time on the part of entities involved. CRL, CRS/ NOVOMODO and OCSP also make this assumption.
- There exists an established Service Level Agreement (SLA) between the CA and an EVCP, with a reliable and secure communication channel.
- The number of verifiers is much more than the number of EVCPs. Moreover, the ratio of number of verifiers to the corresponding EVCPs is high.

To identify the availability of a CREV revocation scheme, we define the following extension to an EV Certificate:

$$\textit{CREV Extension} ::= \textit{CREV Scheme}, \textit{Transfer Mechanism} \tag{7.7}$$

The fields are as follows:

- *CREV Scheme*: gives the CREV scheme employed.
- *Transfer Mechanism*: defines the CSI transfer mechanism, e.g. HTTP.

7.3.3 CREV-I: Session-based Hash-Chaining Scheme

Despite their benefits, CRS and NOVOMODO can incur significant overheads when the hash chains employed are rather long. (Note that the length of the hash chain increases with higher timeliness guarantee.) The verifiers need to perform more hash computations, and the CA must allocate more storage for the hash chains (although amortization techniques can help). CREV-I improves CRS/NOVOMODO by setting up a *session-based* hash-chaining service between the CA and an EVCP. The idea is that each EVCP and the CA establish a secure session using a "3-way Session Establishment" protocol given below. Thus, CREV-I employs a hash chain, but it shortens the length of the hash chain for faster verifier's operation and reduced CA storage.

Similar to OCSP, the CA's availability for a request message may lead to the possibility of Denial of Service (DoS) attacks due to flood of requests to the CA, including replaying previously valid request messages.[10] To deal with this, we require every incoming $SessionRequest$ message (see Step 1) to be signed by the EVCP. Moreover, we also require the message to carry T, which is either:

- A timestamp of the current time based on network time protocols.
- A CA's nonce that is accessible by the verifiers from a CA's pre-defined URI. The nonce is regularly updated by the CA at a pre-determined short time interval.

In a published certificate, the CA includes:

- *CREV Extension*, which is defined in (7.7).
- *CREV-I Extension*, which contains: URI for the CA's nonce (URI_{CA_nonce}), URI for the *Session Reply* message ($URI_{hashchain\_session}$), and URI for the hash-chain token (URI_{hash_token}).

The following protocol is executed by the CA and EVCP to establish a hash chain session on a valid EVC. We define "*Session Request*" and "*Session Reply*" below to be syntactically similar to OCSP Request and OCSP Response respectively. In the protocol, we assume that $Serial\_No$ and CA's name (CA_ID) are sufficient to identify a unique certificate.[11] The notation $\langle M \rangle_{K_A^{-1}}$ denotes a message M which is signed using the private key of principal A (i.e. K_A^{-1}); whereas $nonce_X$ denotes a nonce from X.

1. $EVCP \rightarrow CA$: "*Session Request*"= $\langle SessReq, EVCP\_ID, CA\_ID, Serial\_No, T,$ $nonce_{EVCP}, SigAlgID \rangle_{K_{EVCP}^{-1}}$.

   where:

   $SessReq$ = header indicating a Session Request message;

   $EVCP\_ID$ = identity (i.e. domain name) of the EVCP;

   $Serial\_No$ = serial number of the EVC;

   T = timestamp or a CA's nonce accessible at URI_{CA_nonce};

   $SigAlgID$ = identification for the signing algorithm.

2. CA : If T or K_{EVCP}^{-1} is incorrect, then abort.

3. $CA \rightarrow EVCP$: "*Session Reply*"= $\langle SessReply, ReplyStatus, CA\_ID, EVCP\_ID,$ $nonce_{EVCP}, Serial\_No, CertStatus, HashAlgID, d, Y, N,$

[10]OCSP makes use of a nonce (from the requester) to bind a request and a response message. This measure is employed to protect the requester from possible replay attacks. However, there seems to be no mechanism in OCSP to prevent flood-of-requests attack to the responder (see also [124, Section 5]).

[11]OCSP [124] makes use of a serial number together with $HashAlgID$, $H(CA\_ID)$ and *H(CA's public-key)* due to the presumed lack of strong uniqueness guarantee for CA names.

$SessStart$, $SessExpiry$, $nonce_{CA}$, $SigAlgID\,\rangle_{K_{CA}^{-1}}$.

where:

$SessReply$ = header indicating a Session Reply message;

$ReplyStatus$ = status indicator for a successful session establishment;

$CertStatus$ = status of the EVC;

$HashAlgID$, d, Y, N = hash chain parameters (see Section 7.2.2);

$SessStart$ and $SessExpiry$ = the start and end times of the established session.

4. $EVCP$: If $nonce_{EVCP}$ or K_{CA}^{-1} incorrect, or $ReplyStatus$ is unsuccesful, then abort.

5. $EVCP \rightarrow CA$: "*Session ACK*"= $\langle SessACK$, $EVCP\_ID$, CA_ID, $nonce_{CA}$, $Serial\_No$, $SigAlgID\,\rangle_{K_{EVCP}^{-1}}$.

where:

$SessACK$ = header indicating a Session ACK message.

6. $EVCP$: Put *Session Reply* message from Step 3 at $URI_{hashchain\_session}$; Establish an association for hash chain updates with CA.

7. CA : If $nonce_{CA}$ and K_{EVCP}^{-1} is incorrect, then abort.
Start providing timely hash-chain token updates until the session expires, or the EV certificate is revoked.

The established session is good for time interval $t_{CREV\_I} = SessExpiry - SessStart$. For a valid certificate, on the i-th time interval (for $1 \leq i \leq \ell_{CREV\_I}$) after $SessStart$, the CA releases the latest hash chain token (V_i) to the EVCP as in CRS/NOVOMODO. The EVCP always puts the most recent V_i it receives from CA at URI_{hash_token}.

The verifier obtains a Session Reply from the specified $URI_{hashchain\_session}$, and V_i from URI_{hash_token}. It then validates K_{CA}^{-1} in the Session Reply, and then determines that the EVC is still valid if $H^i(V_i) = Y$, or that the EVC is revoked if $H(V_i) = N$.

The length of the hash chain in a CREV-I session is $\ell_{CREV\_I} = \frac{t_{CREV\_I}}{d} - 1$, which is much smaller than CRS/NOVOMODO. As a result, the workload for the verifier's repeated hash operations is much reduced. The CA now also stores and keep tracks of a much smaller (session-wide) hash chain. This is in contrast with the hash chain used in CRS/NOVOMODO, which is constructed for the whole certificate's lifetime.

A formal analysis of the session establishment protocol using MPKI-BAN Logic (the subject of Chapter 8) is given in Appendix E.

7.3.4 CREV-II: Session-based Online Status Scheme

CREV-II takes advantage of a CA's ability to support an online status notification service such as OCSP. Given OCSP as a standardized scheme for online status notification, here we simply assume the online status assurance in the form of OCSP Response. Unlike OCSP, however, the CA in CREV-II produces *only one OCSP Response* for an EVC principal per time interval (regardless of the number of verifiers). Compared to CREV-I, CREV-II has an advantage of providing a *CA's direct (signed) proof* of certificate goodness to EVCPs.[12] Thus, CREV-II may be more desirable than CREV-I if we want to take advantage of the existing OCSP infrastructure. Also notice that the CA in CREV-II provides the service only to EVCPs, which in turn make the CSI available to the respective verifiers. Hence, the new setting also allows for a workable economic model in which the CA charges EVCPs, rather than the verifiers, for the rendered CSI services.

To support CREV-II, the CA includes the following in a certificate it publishes:

- *CREV Extension*, which is defined in (7.7).
- *CREV-II Extension*, which contains: URI for CA's nonce (URI_{CA_nonce}) and URI for the latest OCSP Response (URI_{latest_OCSP}).

The following protocol establishes a CSI subscription session between CA and EVCP.

1. $EVCP \rightarrow CA$: "*Subscription Request*"$= \langle\, SubsRequest,\ EVCP\_ID,\ CA\_ID,\ Serial\_No,\ T,\ nonce_{EVCP},\ t_{EVCP},\ d_{EVCP},\ SigAlgID\,\rangle_{K^{-1}_{EVCP}}$.

   where:

   $SubsRequest$ = header indicating a Subscription Request message;

   T = either a timestamp or CA's nonce as in CREV-I;

   t_{EVCP} = EVCP's proposed lifetime of the established session;

   d_{EVCP} = EVCP's proposed time interval between two OCSP Responses.

2. CA : If T or K^{-1}_{EVCP} is incorrect, then abort.

3. $CA \rightarrow EVCP$: "*Subscription Reply*"$= \langle\, SubsReply,\ EstablishmentStatus,\ CA\_ID,\ EVCP\_ID,\ Serial\_No,\ nonce_{EVCP},\ CertStatus,\ d_{CA},\ SessStart,\ SessExpiry,\ nonce_{CA},\ SigAlgID\,\rangle_{K^{-1}_{CA}}$.

   where:

   $SubsReply$ = header indicating a Subscription Reply message;

   $EstablishmentStatus$ = status indicator for a successful subscription establishment;

   d_{CA} = selected time interval between two OCSP Responses.

   The session's lifetime (t_{CREV_II}) $= SessExpiry - SessStart$.

[12]In reporting a certificate's status, an online service is assumed in this thesis to provide its timeliest possible revocation guarantee as permitted by its policy.

4. $EVCP$: if $nonce_{EVCP}$ or K^{-1}_{CA} incorrect, or $EstablishmentStatus$ is unsuccesful, then abort.

5. $EVCP \rightarrow CA$: "*Subscription ACK*"= $\langle\, SubsACK,\ EVCP\_ID,\ CA\_ID,\ Serial\_No,\ nonce_{CA},\ SigAlgID\,\rangle_{K^{-1}_{EVCP}}$.
   where:
   $SubsACK$ = header indicating a Subscription ACK message.

6. $EVCP$: Establish an update association with CA.

7. CA : If $nonce_{CA}$ and K^{-1}_{EVCP} is incorrect, then abort.

After Step 7, the CA starts delivering (pushing) OCSP Response messages to the EVCP every d_{CA} time interval for ℓ_{CA} times, or until the certificate is revoked. Upon receipt of every OCSP Response from the CA, the EVCP puts it on URI_{latest_OCSP}.

Note that the periodically released OCSP Responses contain no requester's nonce. As such, the EVCP and the verifier must verify the freshness of an OCSP Response by checking the included values of `thisUpdate` and `nextUpdate` (as two fields in the standard OCSP Response) to be current. The CA must properly set a Response's validity period (`nextUpdate - thisUpdate`) to d_{CA}. Note that these OCSP Responses can be pre-produced by the CA.

If d_{CA} for multiple EVCPs is short, e.g. 1 minute, an optimization is applicable. The CA can issue a single OCSP Response that carries the statuses of *multiple* certificates by merging the generated OCSP Responses together. The validity periods of all the certificate statuses are set uniformly.

If the EVC serial number is sequential, we additionally can apply a *range optimization* based on the notion of certificate range in CRT [83], which is also employed in [84]. We use this optimization as follows: The CA keeps track of valid certificates as a set of *good certificate ranges*. Each good certificate range is defined as $[SN_a, SN_b]$ indicating that all certificates with serial number SN_i, where $SN_a \leq SN_i \leq SN_b$, are all valid. For an optimal range construction (resulting in the smallest number of ranges), we simply set $[SN_a, SN_b]$ to be the largest contiguous valid certificate range. If a certificate is revoked, the CA sends a single OCSP Response notifying the revoked status of that certificate. For a valid certificate with serial number SN_v, the CA sends an OCSP Response with a good certificate range where $SN_a \leq SN_v \leq SN_b$. Given that the majority of certificates are good, this optimization can significantly reduce the CA's signature operations as shown later in Section 7.4.4.

7.4 Analysis, Evaluation and Comparison of CREV Schemes

7.4.1 Security Analysis of CREV Schemes

The security of CREV schemes relies mainly on the underlying revocation schemes, namely CRS/NOVOMODO and OCSP. Provided that all the entities run with relatively synchronized clocks, the freshness of the released hash tokens (in CREV-I) and OCSP Responses (in CREV-II) can be established. The Session Reply message (in CREV-I) is signed by the CA. It thus behaves as a kind of "mini certificate" which carries information about the CA's hash chain parameters. The CREV-II scheme takes advantage of the OCSP Response model, where the verifier only needs to check the validity of the CA's signature and the Response's validity period.

We project that the CREV schemes are practical with near a real-time timeliness guarantee from ten to one minute. A ten minute guarantee may already be considered reasonably short-lived in many environments. A one minute guarantee could be even considered as being "indistinguishable" from a real-time service due to potential clock time differences among the involved entities [111]. The recency requirements in our schemes are thus set by the CA and EVCP, and not the verifier (as acceptor). As such, the schemes still fail to address an issue pointed out in [138]. Yet, with a near real-time timeliness guarantee, a verifier can proceed with the transactions only if it feels satisfied with the offered more fine-grained timeliness guarantee.

7.4.2 A Framework for Performance Analysis

Our performance analysis framework extends the model given in [99, 72]. A novelty of our framework is that it includes more real-world features and cost estimates.

Certification System Assumptions

Our analysis framework is intended to measuring the overheads of revocation schemes with a single CA during the *steady-state condition*, i.e. when certificate expiration is balanced by the new certificates added. This assumption is commonly adopted by many works [99, 72, 92, 190], which allows us to focus on the stable behavior of the revocation schemes and omit possible variable external factors. Since our objective is to provide a near real-time freshness guarantee, we make use of *minute* as our unit of time.

Some other assumptions commonly made when analyzing revocation schemes under the steady-state condition (e.g. taken in [99, 72, 92, 190]), which we also make here, are the following:

- The total number of principals and the corresponding valid certificates (N) is constant.

- Certificates have the same lifetime, which is β days $= \beta \times 1,440$ minutes.[13]
- There are $b \cdot N$ certificates revoked in the span of β days.
- The time interval between two successive CRL releases (Δt) is constant. The unit of Δt is minute(s).
- Certificate issuance takes place at a constant rate. The time interval between two successive certificate generations (ΔX) is the same as Δt.
- A new valid certificate is issued immediately to replace a revoked one. This can take place in practice by assuming that a principal always submits a revocation notice together with a replacement request to prevent any operational interruption.

Since we consider a certification system with uniform certificate lifetime β, the steady-state condition takes place after β days from the *first* certificate generation in the system, i.e. the time interval $(\beta, +\infty)$.

By considering a system where $\Delta X = \Delta t = 1$ minute as an example of our evaluation scenario, we thus cover the situation where the certificate issuance and the CRL release take place continuously.[14] As mentioned earlier, most earlier works [190, 92] make an assumption that a certificate is assumed to be revoked at half of its lifetime ($\frac{\beta}{2}$). Such an assumption is however crucial to the performance analysis of revocation schemes as it affects, among others, the estimate of the CRL size.

An Improved Model and Analysis

The work [99] (summarized in Section 7.2.3) is the first to suggest a realistic revocation model based on real empirical data of CRLs. Our analysis employs the same approach as [99, 72] but increases the realistic features as in the following aspects.

1. Revision of the CRL Size Estimate

The work [99, 72] always assume that $\Delta X = \Delta t = 1$ day. As such, there are $\alpha = \frac{N}{\beta}$ certificates issued on day X, which are valid between day X and day $X+\beta$. For certificates issued at time X, $R(t)$ is defined to represent the revoked percentage. It is modeled as an exponential PDF of $R(t) = ke^{-kt}$, where t is the time and the parameter $k = 0.26$.

The work [99, 72] calculates the CRL size estimate using the derivation shown in Eqs. 7.2–7.6. We note two issues with this derivation. Firstly, their calculations make use of $f(v)$ in [99] (or $N(v)$ in [72]) to obtain the number of new certificate revocations between day v and day $v+\Delta t$. The functions however use a *discrete summation* on the PDF which is only an approximation (see Eqs. 7.2–7.4). In our work, we derive the number of new certificate revocations using *integration*, which is a more proper method, on the

[13]The work [99, 72] also considers a scenario with certificates having several ages.

[14]It is possible to allow $\Delta X = c.\Delta t$ for some constant value c. Here, we assume a "continuous" issuance (i.e. $\Delta X = \Delta t$) as opposed to a batched one.

same PDF. This difference becomes important when ΔX and Δt is much less than one day. Simply applying the summation technique as in [99, 72] would derive the number of revoked entries that is *much larger* than that of the total revoked certificates $N \cdot b$ (see Section 7.4.4 for a sample figure based on two selected evaluation scenarios). Secondly, in the steady-state calculation (Eq. 7.4), $f(v)$ or $N(v)$ actually measures the number of revocations for days $[1, \beta+1]$ instead of $(0, \beta)$.

Thus, we reformulate $F(v)$ where $\Delta X = \Delta t = 1$ day by first defining $g(v)$ to replace $f(v)$ in [99] or $N(v)$ in [72]. Function $g(v)$ is defined as the number of new revocations between day $v - \Delta t$ and day v from all valid generations. When v is in $[0, \beta]$, we have:

$$g(v) = \sum_{i=1}^{v} \alpha b \int_{i-1}^{i} R(t)\, dt = \alpha b \int_{0}^{v} k e^{-kt}\, dt = \alpha b \left[- e^{-kt} \right]_0^v = \alpha b \left(1 - e^{-kv} \right). \tag{7.8}$$

For the steady-state condition, i.e. when $v \in (\beta, +\infty)$, we have:

$$g(\beta) = \alpha b \int_{0}^{\beta} k e^{-kt}\, dt = \alpha b \left[- e^{-kt} \right]_0^\beta = \alpha b \left(1 - e^{-k\beta} \right). \tag{7.9}$$

$F(v)$, representing the cumulative number of certificate revocations from day 0 to day v, with $v \in [1, \beta - 1]$ becomes:

$$F(v) = \sum_{i=1}^{v} g(i) = \sum_{i=1}^{v} \alpha b \int_{0}^{i} k e^{-kt}\, dt = \alpha b \left[v - e^{-k} \, \frac{1 - e^{-kv}}{1 - e^{-k}} \right]. \tag{7.10}$$

For $v \in [\beta, +\infty)$, which includes the steady-state condition, we thus have:

$$Rev\_Entries = F(\beta - 1) = \sum_{i=1}^{\beta-1} g(i) = \alpha b \left[(\beta - 1) - e^{-k} \, \frac{1 - e^{-k(\beta-1)}}{1 - e^{-k}} \right]. \tag{7.11}$$

Note that in Eq. 7.11, we make use of $F(\beta - 1)$ instead of $F(\beta)$ in Eq. 7.6 as in [99, 72]. This is since there are only $\beta - 1$ valid certificate generations, and not β. The generation on day v has no revoked certificates yet, and the generation on day $v - \beta$ is expired on that day, hence leaving only $\beta - 1$ generations from day $v - 1$ to $v - \beta + 1$.

We also find out that there is a simpler and more elegant way of calculating the CRL size on day v, which is described as follows. First, we define $h(i)$ as the total number of (non-expired) revoked certificates from certificates generated in i generation(s) *prior to* the current generation:

$$h(i) = \alpha b \int_{0}^{i} R(t)\, dt. \tag{7.12}$$

Function $h(i)$ thus represents the cumulative distribution function of $R(t)$. The number of entries in CRL during the steady-state condition (with $\beta - 1$ certificate generations counted) is:

$$Rev\_Entries = \sum_{i=1}^{\beta-1} h(i) = \sum_{i=1}^{\beta-1} \alpha b \int_{0}^{i} R(t)\, dt = \alpha b \left[(\beta - 1) - e^{-k} \, \frac{1 - e^{-k(\beta-1)}}{1 - e^{-k}} \right], \tag{7.13}$$

yielding the same result as Eq. 7.11. In our generalization below, we will make use of $h(i)$.

2. Generalization of CRL Size with Certificate Generation (ΔX) and CRL Issuance (Δt) in Minutes

In our performance evaluation, we set the revocation timeliness guarantee of 1 and 10 minutes. To deal with this, we need to generalize the calculation of the number of CRL entries where $\Delta X = \Delta t = \delta$ minutes.

Let M denote the number of minutes in a day, that is M=1,440. We then define a constant $\lambda = \frac{\delta}{M}$. The number of certificate issued per generation (α_g) is now:

$$\alpha_g = \frac{N\lambda}{\beta} = \alpha\lambda. \tag{7.14}$$

We generalize $h(v)$ into $h_g(v)$ as follows:

$$h_g(v) = \alpha_g b \int_0^i R(t)\, dt = \alpha_g b \int_0^i ke^{-kt} = \frac{N\lambda}{\beta}\, b\left(1 - e^{-kv}\right). \tag{7.15}$$

The number of revoked entries during the steady-state condition where $v \in (\beta, +\infty)$ with $\Delta X = \Delta t = \frac{\delta}{M}$ day is:

$$\begin{aligned} Rev\_Entries_g &= \sum_{i=1}^{\eta} h_g(\lambda i) = \sum_{i=1}^{\eta} \alpha_g b \int_0^{\lambda i} R(t)\, dt \\ &= \frac{N\lambda}{\beta}\, b\left[\left(\frac{\beta}{\lambda} - 1\right) - e^{-k\lambda}\,\frac{1 - e^{-k(\beta-\lambda)}}{1 - e^{-k\lambda}}\right]. \end{aligned} \tag{7.16}$$

where $\eta = \frac{\beta M}{\delta} - 1$ denotes the number of counted generations.

3. More Realistic Models for Query on Revoked Certificates

Determining the probabilities of whether a certificate being queried is either revoked or valid is a key factor in calculating the computational costs of a hash-chaining based scheme like CRS, NOVOMODO or CREV-I. This is since there is a great difference in cost between verifying a valid certificate and a revoked one. To derive such realistic probabilities, the framework needs to have some realistic assumptions for the probability of a revoked but not expired certificate to be queried.

We would expect that once a certificate has been revoked, the likelihood that the revoked certificate will be queried by a verifier should *decrease* over time. One reason for this would be that the revoked certificate is replaced by a new valid one, which we also assume in our framework. Hence, for example, a new certificate will be used by a web server whose certificate was recently revoked in order to secure its SSL-based transactions. Similarly, mobile codes that were signed by a revoked certificate are now signed using a new one. Some verifiers may still have a stale reference to the revoked certificate, but the numbers of queries issued by those verifiers would be expected to decrease over time.

The function $S(t)$ is defined as the probability that a certificate will still be queried in the time interval $[(t-1)\delta,\, t\delta\,]$ minutes after it is first known to be revoked.[15] Here, we take

[15]Unlike $R(t)$, $S(t)$ indicates the *probability value* at time t rather than a PDF. As such, in Eqs. 7.19 and 7.20 we make use of summation over $S(t)$ as opposed to integration in deriving the total number of queries on the revoked certificates.

into account the fact that certificates are used for different purposes, such as for online transactions and code/document signing. As such, one should also have different query models to suit these different purposes. We define the following two query models:[16]

Model A (Online-transaction certificates): This applies to certificates used for online transactions where a revoked certificate will quickly cease to be referred to. We define the following probability function for this model (shown in Figure 7.3):

$$S_A(t) = e^{-k_A \lambda t}. \tag{7.17}$$

As web servers for online-transactions can be updated very quickly upon a certificate revocation, we therefore choose a value of $k_A = 0.95$ in the evaluation of revocation schemes.

Model B (Code/document-signing certificates): This applies to certificates used to provide non-online services, such as the signing of documents/software that may be distributed in an off-line manner (e.g. using a physical media). In this type of certificates, a revoked certificate will still be referred to for a longer time period following its revocation. We model its probability function as:

$$S_B(t) = e^{-k_B \lambda t}. \tag{7.18}$$

We choose a value of $k_B = 0.01$ in the evaluation of revocation schemes.

Figure 7.3 shows the two probability functions with the chosen parameters.

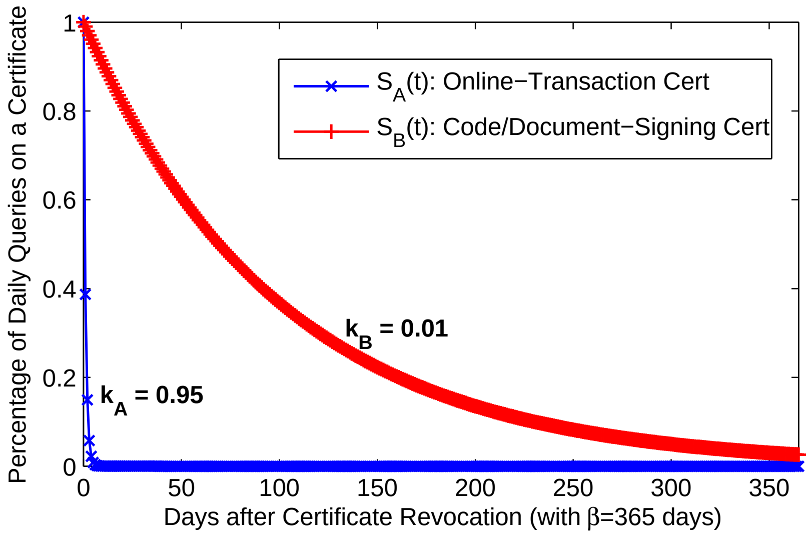

Figure 7.3: Two probability functions $S_A(t)$ and $S_B(t)$ are shown in the time interval $[0, \beta$-days) after the certificate's revocation, with $\beta = 365$ days. The function $S_A(t)$ is shown as an L-shaped curve close to the axes.

[16] A CA can have a different number of certificates classes. However, we assume that these classes can be mapped into the two defined query models.

4. Derivation of the Probabilities of Querying Revoked and Valid Certificates
Our objective here is to derive the probabilities of whether a query from a verifier is issued on a valid certificate (Pr_{valid}) and a revoked one (Pr_{rev}). We derive these two probabilities as follows.

First, the total number of daily queries issued by all verifiers is: $Q_{daily} = V \cdot Q_{Ver\_daily}$, where V is the number of verifiers and Q_{Ver_daily} is the average daily queries issued by each verifier to CA/CMAE.[17] Since we assume that queries on all the certificates in the system are uniformly distributed, we thus have the number of queries per certificate in a day: $q_{per\_cert\_daily} = Q_{daily}/N$.

Now, we derive $Q_{rev\_\beta\_days}$ which is the total number of queries on *all revoked* certificates (until their expirations) *throughout* β days under the steady-state condition. Since we have two query models on the revoked certificates (i.e. model A and B), we define $Q_{rev\_\beta\_days\_A}$ as follows:

$$Q_{rev\_\beta\_days\_A} = \sum_{i=1}^{\eta} \left[\Big(N_A \cdot b \cdot \int_{\lambda(i-1)}^{\lambda i} R(t)\, dt \Big) \cdot \Big(\sum_{j=1}^{\frac{\beta}{\lambda}-i} S_A(j) \cdot q_{per\_interval} \Big) \right] \tag{7.19}$$

where N_A denotes the number of certificates with query model A, $\eta = \frac{\beta M}{\delta} - 1$ denotes the number of counted certificate generations, $\lambda = \frac{\delta}{M}$, and $q_{per\_interval} = \lambda \cdot q_{per\_cert\_daily}$ denotes the number of queries on a certificate in the time interval of δ minutes. Similarly, we define $Q_{rev\_\beta\_days\_B}$ as follows:

$$Q_{rev\_\beta\_days\_B} = \sum_{i=1}^{\eta} \left[\Big(N_B \cdot b \cdot \int_{\lambda(i-1)}^{\lambda i} R(t)\, dt \Big) \cdot \Big(\sum_{j=1}^{\frac{\beta}{\lambda}-i} S_B(j) \cdot q_{per\_interval} \Big) \right] \tag{7.20}$$

where N_B denotes the number of certificates with query model B. Summing them up gives: $Q_{rev\_\beta\_days} = Q_{rev\_\beta\_days\_A} + Q_{rev\_\beta\_days\_B}$. The total number of queries on *all* non-expired certificates (both valid and revoked ones) for β days is $Q_{all\_\beta\_days} = Q_{daily} \cdot \beta$. Hence, the total number of queries on *all valid* certificates throughout β days (under the steady-state condition) is: $Q_{valid\_\beta\_days} = Q_{all\_\beta\_days} - Q_{rev\_\beta\_days}$. Finally, the probabilities of whether a status query is issued on a valid certificate (Pr_{valid}) and a revoked certificate (Pr_{rev}) are:

$$Pr_{rev} = \frac{Q_{rev\_\beta\_days}}{Q_{all\_\beta\_days}} \quad \text{and} \quad Pr_{valid} = \frac{Q_{valid\_\beta\_days}}{Q_{all\_\beta\_days}}. \tag{7.21}$$

5. Taking Realistic CSI Message Sizes into Account

Many previous analysis works [190, 92] take a simplified (or minimalistic) approach with respect to the CSI message sizes of standardized revocation schemes. In practice, how-

[17] Q_{daily} denotes the number of daily CSI look-ups received by the CA/CMAE. Given the short timeliness guarantees in our setting, we assume that all queries are always consulted with the CA/CMAE. Thus, no CSI cache is employed by a verifier. This is reasonable since we set the verifier's daily query (Q_{Ver_daily}) to 30, which is much less than the number of daily CSI updates (1,440 and 144). As a result, we can employ Q_{daily} for calculating query overheads in both the batch-updating and the online schemes.

ever, the CSI also include various auxiliary information. To achieve a realistic analysis, we follow the CSI message sizes in [133], which utilize a BER viewer for all ASN.1 standardized data structures. Furthermore, we also always assume SHA-1 for the hash algorithm and SHA-1 with RSA for the signature algorithm in all the examined schemes.

Performance Comparison Metrics

We use the following notations to denote the various costs incurred in a particular revocation scheme: Ovh_A= computation time needed by entity A (in seconds), Bw_{A-B} = network bandwidth needed from entity A to B (in MB), and $Stor_A$ = storage needed on A (in MB). The entities involved are: CA indicating the CA, Ver indicating a verifier, $CMAE$ indicating a CMAE as a designated repository, and $EVCP$ indicating an EVC principal's server.

To compare different schemes, we use the following metrics:

1. Certificate creation cost: Ovh_{CA}.
2. Update costs: Ovh_{CA}, Ovh_{CMAE} (Ovh_{EVCP}) and $Bw_{CA-CMAE}$ ($Bw_{CA-EVCP}$).
3. Query costs: Ovh_{CA}, Ovh_{CMAE} (Ovh_{EVCP}), $Bw_{CMAE-Ver}$ ($Bw_{EVCP-Ver}$) and Ovh_{Ver}.
4. Storage requirement at one point in time (during the steady-state condition): $Stor_{CA}$ and $Stor_{CMAE}$ ($Stor_{EVCP}$).
5. Timeliness: the revocation timeliness guarantee, which also represents the *window of vulnerability* of the revocation scheme.

For metrics (1) to (3), we measure the total *cost per day*. In order to have a more compact notation, we abuse the notation slightly and write $\forall A$ for all instances of entity A, and $\exists A$ for a single instance of A. Thus, for example, $D\_Bw_{CMAE-\exists Ver}$ denotes the daily bandwidth cost between a CMAE and a single verifier, whereas $D\_Bw_{CMAE-\forall Ver}$ is the daily bandwidth needed by a CMAE to all the verifiers. A summary of the notations used for the performance comparison is given in Table 7.2.

7.4.3 Performance Comparison

We derive the overheads of the following revocation schemes using our analysis framework: CRL (with a CMAE used), OCSP (with the CA as a Responder), CRS (with a CMAE used), and our two CREV schemes. For bandwidth calculation, we do not consider the cost due to the underlying network transfer mechanism(s), i.e. HTTP, TCP, or IP. For CRL and CRS which makes use of CMAE, we assume of one CMAE involved respectively. We use L_M to denote the length of message portion M.

Symbol	Description	Unit	Value(s)	
Parameters for Certification & Revocation System:				
N	No. of valid (non-revoked) and not-yet-expired certs	-	100,000	
β	Issued lifetime of a certificate	days	365	
b	Percentage of certificates revoked	-	$0.1 = 10\%$	
$\Delta X = \delta$	Time interval between two successive cert generations	mins	1	10
$\Delta t = \delta$	Time interval between two successive CRL releases	mins	1	10
M	Number of minutes in a day	-	1,440	
λ	Factor of δ/M	-	δ/M	
α_g	No. of certificates issued per certificate generation	-	$N\lambda/\beta$	
η	No. of cert generations counted to calculate CRL size	-	$\frac{\beta}{\lambda} - 1$	
$d = \delta$	Time interval for periodic hash-token release in CRS/NOVOMODO/CREV-I	mins	1	10
ℓ	Hash chain length which determines certificate lifetime in CRS/NOVOMODO/CREV-I	-	$\dfrac{\beta}{\lambda} - 1$	
U	Total number of CRL and hash-token releases per day	-	M/δ	
t_{CREV_I}	Lifetime of hash chain in a CREV-I session	hrs	3	12
t_{CREV_II}	Lifetime of a status subscription session in CREV-II	hrs	3	12
k	Parameter for realistic certificate revocation PDF	-	0.26 (Ref. [99, 72])	
k_A	Parameter for class-A query model on revoked certs	-	0.95	
k_B	Parameter for class-B query model on revoked certs	-	0.01	
N_A	No. of certificates with class-A query model	-	$0.8 \cdot N$	
N_B	No. of certificates with class-B query model	-	$0.2 \cdot N$	
V	Number of verifiers	-	30,000,000	
Q_{Ver_daily}	Average daily queries to CA/CMAE from a verifier	-	30	
Q_{daily}	Total average daily queries received by CA/CMAE	-	$V \cdot Q_{Ver\_daily}$	
$q_{per\_cert\_daily}$	Average daily queries on a certificate	-	Q_{daily}/N	
Basic Cryptographic Operation's Overheads and CSI Size:				
C_{sign}	Cost of a digital signature generation (RSA-1024)	ms	1.48	
C_{verify}	Cost of a digital signature verification (RSA-1024)	ms	0.07	
C_{hash}	Cost of computing a hash (SHA-1)	μs	0.40	
L_{hash}	Length of hash output (SHA-1)	bytes	20	
Revocation Scheme's Cost Factors:				
Bw_{A-B}	Bandwidth requirement from entity A to B	MB	-	
Ovh_A	Computation overheads on entity A	sec	-	
$Stor_A$	Storage requirement (due to CSI dissemination) on A	MB	-	
$D\_\langle Cost \rangle$	Total daily bandwidth/computation/storage costs	MB|sec	-	

Table 7.2: Notations used in the performance analysis. Value(s) column shows the selected values for the two investigated scenarios.

CRL (with a CMAE)

In our framework, the number of revoked (but non-expired) certificates where $\Delta X = \Delta t = \delta$ minutes is $Rev\_Entries_g$ (see Eq. 7.16). The CRL size can be defined as: $L_{CRL} = L_{CRL\_fields} + \lfloor Rev\_Entries_g \rfloor \cdot L_{CRL\_entry}$ where $L_{CRL\_fields} = 400$ bytes is the message length of the CRL header and signature, and $L_{CRL\_entry} = 39$ bytes is the length of each entry in the CRL data [133]. With $U = \frac{M}{\delta}$ as the total number of CRL updates in a day, the daily update costs are: $D\_Bw_{CA-CMAE} = U \cdot L_{CRL}$, $D\_Ovh_{CA} = U \cdot C_{sign}$, and $D\_Ovh_{CMAE} = U \cdot C_{verify}$.

Recall that Q_{Ver_daily} is the average number of queries issued by a verifier to the

CA/CMAE in a day, and Q_{daily} is the total queries received by the CA/CMAE in a day. The daily query costs of the CRL scheme are: $D\_Bw_{CMAE-\forall Ver}$ (for CMAE) $= Q_{daily} \cdot L_{CRL}$, $D\_Bw_{CMAE-\exists Ver}$ (for Ver)$= Q_{Ver\_daily} \cdot L_{CRL}$, $D\_Ovh_{CA} = 0$, $D\_Ovh_{CMAE} = 0$, and $D\_Ovh_{Ver} = Q_{Ver\_daily} \cdot C_{verify}$.

The storage requirements are: $Stor_{CA} = L_{CRL}$, and $Stor_{CMAE} = L_{CRL}$. The daily cost of certificate creation is: $D\_Ovh_{CA} = \frac{N}{\beta} \cdot C_{sign}$. Finally, the timeliness guarantee is δ minutes.

OCSP (with CA as Responder)

We analyze the overheads of OCSP where the CA functions as the OCSP Responder. Here we assume an OCSP usage mode where a nonce is used to bind an OCSP Response with the corresponding Request. There is no update cost between the CA and CMAE, since no CMAE is involved.

With $L_{OCSP\_Response} = 459$ bytes as the length of OCSP Response [133], the daily query costs (due to status reply) are: $D\_Bw_{CA-\forall Ver}$ (for CA)$= Q_{daily} \cdot L_{OCSP\_Response}$, $D\_Bw_{CA-\exists Ver}$ (for Ver)$= Q_{Ver\_daily} \cdot L_{OCSP\_Response}$, $D\_Ovh_{CA} = Q_{daily} \cdot C_{sign}$, and $D\_Ovh_{Ver} = Q_{Ver\_daily} \cdot C_{verify}$.

The storage requirement is: $Stor_{CA} = 0$, since the CA needs no additional data structure. The daily cost of certificate creation is: $D\_Ovh_{CA} = \frac{N}{\beta} \cdot C_{sign}$. The timeliness guarantee of OCSP can be close to zero when desired.

CRS (with a CMAE)

The CRS scheme is proposed to work with a CMAE [110]. Here, we set $d = \delta$ minute(s). Since the certificate lifetime is βM minutes, the length of the hash chain becomes $\ell = \frac{\beta M}{\delta} - 1$. Here, we assume that the CA stores the whole hash chain for all the valid certificates in its storage. An amortization technique such as [78] can be used to reduce its storage requirements, but at the cost of additional online processing for the CA.

We use $L_{CRS\_fields} = 161$ bytes to denote the length of the CA's timestamp and signature, and $L_{Serial\_No} = 7$ bytes to denote the length of a certificate's serial number [133]. The bandwidth cost for update between CA and CMAE is: $Bw_{CA-CMAE} = L_{CRS\_fields} + (N + \lfloor Rev\_Entries_g \rfloor) \cdot (L_{Serial\_No} + L_{hash})$. With $U = \frac{M}{\delta}$, the total daily update costs become: $D\_Bw_{CA-CMAE} = U \cdot Bw_{CA-CMAE}$, $D\_Ovh_{CA} = U \cdot C_{sign}$, and $D\_Ovh_{CMAE} = U \cdot C_{verify}$.

For queries, the overheads incurred on the verifier depends on the following conditions. If the certificate is revoked, then $Ovh_{Verifier\_rev} = C_{hash}$. If the certificate is valid, the overhead varies according to the number of hash operations required. The best case occurs when $i = 1$ or when the verifier previously has checked V_{i-1}, thus $Ovh_{Verifier\_valid\_best} = C_{hash}$. The worst case happens when $i = \ell$, in which the verifier needs to perform ℓ hash operations, resulting in $Ovh_{Verifier\_valid\_worst} = \ell \cdot C_{hash}$. The average number of hash computations is $\frac{\ell+1}{2}$, hence $Ovh_{Verifier\_valid\_avg} = \frac{\ell+1}{2} \cdot C_{hash}$. The verifier's average query cost which encompasses both the revoked and the valid cases is therefore: $Ovh_{Verifier\_Avg} = Pr_{rev} \cdot Ovh_{Verifier\_rev} + Pr_{valid} \cdot Ovh_{Verifier\_valid\_avg}$, with Pr_{rev} and Pr_{valid} defined in Eq. (7.21).

The corresponding daily query costs are: $D\_Bw_{CMAE-\forall Ver}$ (for CMAE)= $Q_{daily} \cdot L_{hash}$, $D\_Bw_{CMAE-\exists Ver}$ (for Ver)= $Q_{ver\_daily} \cdot L_{hash}$, $D\_Ovh_{CA} = 0$, $D\_Ovh_{CMAE} = 0$, and $D\_Ovh_{Verifier\_Avg} = Q_{ver\_daily} \cdot Ovh_{Verifier\_Avg}$.

For the storage costs, note that the CA can remove the subchains it has released. Hence, for a certificate issued i generations prior to the current time, the CA only needs to store $(\ell - i)$ hash tokens together with N_0. Thus, the storage requirements in CRS are: $Stor_{CA} = \sum_{i=1}^{\ell} \frac{N\lambda}{\beta} \cdot (i+1) \cdot L_{hash} + \lfloor Rev\_Entries_g \rfloor \cdot L_{hash} = \frac{N\lambda}{\beta} \cdot \frac{\ell^2+3\ell}{2} \cdot L_{hash} + \lfloor Rev\_Entries_g \rfloor \cdot L_{hash}$, and $Stor_{CMAE} = Bw_{CA-CMAE}$.

For each certificate, the total cost for generating hash chains (from Y_0 and N_0) is: $(\ell+1) \cdot C_{hash}$. Note that the hash generation portions can be pre-computed. The online overhead which we are concerned here is: $Ovh_{CA} = C_{Sign}$, and the corresponding daily cost is: $D\_Ovh_{CA} = \frac{N}{\beta} \cdot C_{Sign}$. The timeliness guarantee of CRS is $d = \delta$ minutes.

CREV-I

CREV-I establishes a short-term session of t_{CREV_I} minutes between the CA and an EVCP. To achieve the timeliness guarantee of δ minutes, we set the hash-chain update interval in CREV-I (d) to δ minutes. Assuming an uninterrupted service, each EVCP thus performs $S = \frac{M}{t_{CREV\_I}}$ session establishments daily. Hence, each EVCP receives $U = S \cdot \ell_{CREV\_I} = \frac{M}{\delta} - S$ hash-token updates in a day. We use $L_{CREV\_I\_Reply} = 605$ bytes to denote the length of *Session Reply* message, and $L_{CREV\_I\_Msgs} = 1{,}615$ bytes to denote the length of all messages in a session establishment of CREV-I.

The total daily update costs (comprising both the session establishments and the hash-token updates) are: $D\_Bw_{CA-\forall EVCP}$ (for CA) $= N \cdot (S \cdot L_{CREV\_I\_Msgs} + U \cdot L_{hash})$, $D\_Bw_{CA-\exists EVCP}$ (for EVCP)= $S \cdot L_{CREV\_I\_Msgs} + U \cdot L_{hash}$, $D\_Ovh_{CA} = N \cdot S \cdot (2 \cdot C_{verify} + C_{sign})$, and $D\_Ovh_{EVCP} = S \cdot (2 \cdot C_{sign} + C_{verify}) + U \cdot C_{hash}$.

The verifier's average query cost in CREV-I is: $Ovh_{Verifier\_Avg} = Pr_{rev} \cdot C_{hash} + Pr_{valid} \cdot (\frac{\ell_{CREV\_I}+1}{2} \cdot C_{hash} + C_{verify})$. The corresponding total daily costs due to query are: $D\_Bw_{EVCP-\forall Ver}$ (for EVCP) $= q_{per\_cert\_daily} \cdot (L_{hash} + L_{CREV\_I\_Reply})$, $D\_Bw_{EVCP-\exists Ver}$ (for Verifier) $= Q_{Ver\_daily} \cdot (L_{hash} + L_{CREV\_I\_Reply})$, $D\_Ovh_{CA} = 0$, $D\_Ovh_{EVCP} = 0$, and $D\_Ovh_{Verifier\_Avg} = Q_{Ver\_daily} \cdot Ovh_{Verifier\_Avg}$.

As in CRS, for a certificate whose hash tokens have ben released i times before, where $0 \leq i \leq \ell_{CREV\_I}$, the CA only needs to store $\ell_{CREV\_I} - i$ hash-tokens together with N_0. Hence, the CA's storage requirement is: $Stor_{CA} = \sum_{i=1}^{\ell_{CREV\_I}} \frac{N}{\ell_{CREV\_I}} \cdot (i+1) \cdot L_{hash} =$

$\frac{N}{\ell_{CREV\_I}} \cdot \frac{\ell^2_{CREV\_I}+3\ell_{CREV\_I}}{2} \cdot L_{hash}$. The storage needed in each EVCP is: $Stor_{EVCP} = L_T + L_{CREV\_I\_Reply} + L_{hash}$, where L_T denotes the length of the CA's nonce or timestamp for its replay protection (see Section 7.3.3).

For the CA's certificate creation, the (online) daily cost is: $D\_Ovh_{CA} = \frac{N}{\beta} \cdot C_{sign}$. The timeliness guarantee of CREV-I scheme is δ minutes.

CREV-II

CREV-II establishes a short-term session of $t_{CREV-II}$ minutes between the CA and an EVCP. Throughout an established session, the CA sends an OCSP Response message every $d_{CA}=\delta$ minutes. In a day, each EVCP thus performs $S = \frac{M}{t_{CREV\_II}}$ session establishments, and receives $U=S \cdot \ell_{CA}=\frac{M}{\delta}$ OCSP Response messages. We use $L_{CREV\_II\_Reply=}$ 550 bytes to denote the length of *Subscription Reply*, and $L_{CREV_II_Msgs} = 1{,}577$bytes to denote the length of all messages used in a session establishment of CREV-II.

The total daily update costs (comprising both the session establishments and the OCSP Response updates) are: $D\_Bw_{CA-\forall EVCP}$ (for CA) $= N \cdot (S \cdot L_{CREV\_II\_Msgs} + U \cdot L_{OCSP\_Response})$, $D\_Bw_{CA-\exists EVCP}$ (for EVCP) $= S \cdot L_{CREV\_II\_Msgs} + U \cdot L_{OCSP\_Response}$, $D\_Ovh_{CA} = N \cdot S \cdot (2 \cdot C_{verify} + C_{sign}) + N \cdot U \cdot C_{sign}$, and $D\_Ovh_{EVCP} = S \cdot (2 \cdot C_{sign} + C_{verify}) + U \cdot C_{verify}$. With the range optimization technique, the worst case scenario for the required number of group-based status updates occurs when each revoked certificate happens to be in between two valid ones. As such, we require *Rev_Entries* of OCSP Response messages. The average case requires $N_{range\_msgs} = \lceil \frac{Rev\_Entries}{2} \rceil$ messages. Let us use $L_{OCSP\_Response\_Range}$ to denote the length of OCSP Response message containing a range specification. The total daily update costs for CREV-II with range optimization are: D_Ovh_{CA} (with range optimization) $= N \cdot S \cdot (2 \cdot C_{verify} + C_{sign}) + N_{range\_msgs} \cdot U \cdot C_{sign}$, $D\_Bw_{CA-\forall EVCP}$ (for CA, with range optimization) $= N \cdot (S \cdot L_{CREV\_II\_Msgs} + U \cdot L_{OCSP\_Response\_Range})$, and $D\_Bw_{CA-\exists EVCP}$ (for EVCP, with range optimization) $= S \cdot L_{CREV\_II\_Msgs} + U \cdot L_{OCSP\_Response\_Range}$.

The total daily query costs are as follows: $D\_Bw_{EVCP-\forall Ver}$ (for EVCP) $= q_{per\_cert\_daily} \cdot L_{OCSP\_Response}$, $D\_Bw_{EVCP-\exists Ver}$ (for Ver) $= Q_{Ver\_daily} \cdot L_{OCSP\_Response}$, $D\_Ovh_{CA} = 0$, $D\_Ovh_{EVCP} = 0$, and $D\_Ovh_{Ver} = Q_{Ver\_daily} \cdot C_{verify}$. When range optimization technique is employed, we have: $D\_Bw_{EVCP-\forall Ver}$ (for EVCP, with range optimization) $= q_{per\_cert\_daily} \cdot L_{OCSP\_Response\_Range}$, $D\_Bw_{EVCP-\exists Ver}$ (for Ver, with range optimization) $= Q_{Ver\_daily} \cdot L_{OCSP\_Response\_Range}$,

The storage requirements are: $Stor_{CA} = 0$ (without range optimization) or $Stor_{CA} = N_{range\_msgs} \cdot L_{OCSP\_Response\_Range}$ (with range optimization); and $Stor_{EVCP} = L_T + L_{OCSP\_Response}$ (without range optimization) or $Stor_{EVCP} = L_T + L_{OCSP\_Response\_Range}$ (with range optimization). For the CA's certificate creation, the (online) cost is $D\_Ovh_{CA} = \frac{N}{\beta} \cdot C_{sign}$. The timeliness guarantee of CREV-II scheme is: $d_{CA} = \delta$ minutes.

7.4.4 Performance Evaluation

Evaluation Scenarios and Objectives

We conducted our evaluation on a certification system with $100,000$ certificates and a 10% revocation rate. The other parameter values for two evaluation scenarios (with the timeliness guarantee of 1 and 10 minutes respectively) are given in Table 7.2. Our main objective here is to have a quantitative comparison of the costs incurred by the various schemes under the same evaluation scenarios. For the calculation of Pr_{rev} and Pr_{valid} (Eq. 7.21), we assume that 80% of the certificates follow $S_A(t)$ query model on revoked certificates and the other 20% observe $S_B(t)$. We set that all hash values are produced using SHA-1, and signatures are created using RSA with a 1024-bit modulus. Based on the Crypto++ 5.6.0 Benchmarks (`http://www.cryptopp.com/benchmarks.html`) on Intel Core-2 PC with 1.83 GHz CPU running Windows Vista 32-bit, we have the overheads of the basic cryptographic operations as shown in Table 7.2.

Evaluation Results

The calculation results of several realistic performance factors under various timeliness guarantees (δ values), including $\delta = 10$ minutes and 1 minute, are shown in Table 7.3. As can be seen, with δ between 1 day and 1 minute, there are 9,880–9,894 entries in the CRL which represent ~98% of the total revoked certificates. The number of entries is significantly higher than using prior simplified model which were based on an assumption that revoked entries are kept in the CRL for $\frac{\beta}{2}$ days [190, 92] (only 5,000 entries in the CRL). So, even with N=$100,000$ and $\delta = 1$ day, the *difference* in CRL data size calculation is 190,720 bytes, which increases the burden of bandwidth-limited verifiers. We remark that simply applying function $f(v)$ or $N(v)$ in [99, 72] gives $Rev\_Entries \approx$ 5.7M (with $\delta = 1$ hour) and $\approx$ 200M (with $\delta = 10$ minutes) on 100,000 certificates and 10% revocation. This shows that the summation-based approximation can have significant error when δ is less than 1 day since the maximum number of entries in the CRL can only be 10,000.

Performance Factors	Timeliness Guarantee (δ)			
	1 day	**1 hour**	**10 mins**	**1 min**
$Rev\_Entries_g$	9880.33	9894.05	9894.53	9894.62
$CRL\,Size$ (KB)	376.68	377.21	377.21	377.21
Pr_{rev}	0.005442	0.005556	0.005561	0.005562
Pr_{valid}	0.994558	0.994444	0.994439	0.994438

Table 7.3: Calculation results of some performance factors under timeliness guarantees between 1 day and 1 minute.

Another important finding is that the probability of a verifier's query to be reported as valid is ∼99%. The probability is higher than 92% reported in [92] where it is simply assumed that $Pr_{valid} = N/(N + Rev\_Entries)$. The higher percentage is due to our more realistic exponential probability functions $S_A(t)$ and $S_B(t)$, which model the "decaying" effect of queries on the revoked certificates over time. This result is particularly important for hash-chaining based schemes since now the verifier is *almost certain* to perform the repeated hash operations to prove the validity (instead of the revocation) of a queried certificate.

Table 7.4 and Table 7.5 show the overheads of the examined revocation schemes under the selected certification scenarios.

7.5 Discussion

We discuss how CREV schemes compare with others on a certification scenario with 100,000 certificates and other parameter values as shown in Table 7.2. We first highlight the results from Table 7.4.

Given a CRL size of 377.21 KB for 100,000 principals, **the CRL scheme** incurs an enormous daily bandwidth of 3.22×10^8 MB between the CMAE and all verifiers ($D\_Bw_{CMAE-\forall Ver}$). Another real problem faced by CRL is that a verifier, who performs just 30 queries per day, needs to download 11.05 MB of daily CRL data which may not be reasonable for mobile devices.

In **OCSP**, a verifier receives only 0.013 MB of OCSP Responses daily. However, OCSP puts a large burden on the CA. Every day, the CA needs to deliver 3.93×10^5 MB of OCSP Response messages to all the verifiers, and spend 1.33×10^6 seconds (370 hours) of processing time to sign the Response messages.

CRS significantly reduces the bandwidth requirement from the CMAE to the verifiers. Each verifier also needs to access only ∼600 bytes of hash tokens daily. However, on average, a verifier needs to spend 10.45 ms verifying *a single* hash token (on an 1.83-GHz Intel Core-2 CPU). With the length of the hash chain of 52,559, the worst case of verifying a single hash token (belonging to a valid certificate) is 21.02 ms, which is much more expensive than verifying a digital signature (0.07 ms). As such, this overhead may be too costly for lightweight verifiers such as netbooks or mobile devices. Another problem is that CRS requires a huge storage on the CA (5.01×10^4 MB) when the CA stores all the hash chains (without amortization). Furthermore, the bandwidth requirement from the CMAE to all verifiers is also high (1.72×10^4 MB).

CREV-I offers an attractive trade-off between the CA's storage requirement and processing overheads while still maintaining a low verifier's computational (energy) and bandwidth requirements. The storage requirement on the CA is now down to 70.57 MB

Entity	**Daily Costs (U=Update, Q=Query)**	**Unit (/day)**	**CRL**	**OCSP**	**CRS**	**CREV-I**	**CREV-II**
CA	D_Ovh_{CA} (U+Q)	sec	0.21	$\mathbf{1.33 \times 10^6}$	0.21	324	2.16×10^4
	D_Ovh_{CA} (Cert Creation)	sec	0.41	0.41	0.41	0.41	0.41
	$Stor_{CA}$	MB	0.37	0	$\mathbf{5.01 \times 10^4}$	70.57	0
	$D\_Bw_{CA-CMAE}$ (U)	MB	53.05	-	423.80	-	-
	$D\_Bw_{CA-\forall EVCP}$ (U)	MB	-	-	-	578.88	6604.19
	$D\_Bw_{CA-\forall Ver}$ (Q)	MB	-	$\mathbf{3.93 \times 10^5}$	-	-	-
CMAE	D_Ovh_{CMAE} (U+Q)	sec	0.01	-	0.01	-	-
	$Stor_{CMAE}$	MB	0.37	-	2.94	-	-
	$D\_Bw_{CA-CMAE}$ (U)	MB	53.05	-	423.80	-	-
	$D\_Bw_{CMAE-\forall Ver}$ (Q)	MB	$\mathbf{3.32 \times 10^8}$	-	1.72×10^4	-	-
EVCP	D_Ovh_{EVCP} (U+Q)	sec	-	-	-	0.0061	0.07
	$Stor_{EVCP}$	MB	-	-	-	6.11×10^{-4}	4.53×10^{-4}
	$D\_Bw_{CA-\exists EVCP}$ (U)	MB	-	-	-	0.0058	0.066
	$D\_Bw_{EVCP-\forall Ver}$ (Q)	MB	-	-	-	14.03	3.94
Verifier	D_Ovh_{Ver} (Q)	sec	0.0021	0.0021	**0.31**	0.0025	0.0021
	$D\_Bw_{CA-\exists Ver}$ (Q)	MB	-	0.013	-	-	-
	$D\_Bw_{CMAE-\exists Ver}$ (Q)	MB	**11.05**	-	5.74×10^{-4}	-	-
	$D\_Bw_{EVCP-\exists Ver}$ (Q)	MB	-	-	-	0.047	0.013
Revocation Latency		mins	10	≈ 0	10	10	10

Table 7.4: Cost comparison of various schemes on a certification system with 100,000 certificates and $\delta = 10$ minutes.

Entity	Daily Costs (U=Update, Q=Query)	Unit (/day)	CRL	OCSP	CRS	CREV-I	CREV-II	CREV-II Range Opt
CA	D_Ovh_{CA} (U+Q)	sec	2.13	$\mathbf{1.33{\times}10^6}$	2.13	1296	$2.14{\times}10^5$	$1.18{\times}10^4$
	D_Ovh_{CA} (Cert Creation)	sec	0.41	0.41	0.41	0.41	0.41	0.41
	$Stor_{CA}$	MB	0.37	0	$\mathbf{5.01{\times}10^5}$	173.57	0	2251.27
	$D\_Bw_{CA-CMAE}$ (U)	MB	530.46	-	4238.01	-	-	-
	$D\_Bw_{CA-\forall EVCP}$ (U)	MB	-	-	-	3963.47	$6.42{\times}10^4$	$6.52{\times}10^4$
	$D\_Bw_{CA-\forall Ver}$ (Q)	MB	-	$\mathbf{3.94{\times}10^5}$	-	-	-	-
CMAE	D_Ovh_{CMAE} (U+Q)	sec	0.10	-	0.10	-	-	-
	$Stor_{CMAE}$	MB	0.37	-	2.94	-	-	-
	$D\_Bw_{CA-CMAE}$ (U)	MB	530.46	-	4238.01	-	-	-
	$D\_Bw_{CMAE-\forall Ver}$ (Q)	MB	$\mathbf{3.32{\times}10^8}$	-	$1.72{\times}10^4$	-	-	-
EVCP	D_Ovh_{EVCP} (U+Q)	sec	-	-	-	0.025	0.13	0.13
	$Stor_{EVCP}$	MB	-	-	-	$6.11{\times}10^{-4}$	$4.53{\times}10^{-4}$	$4.60{\times}10^{-4}$
	$D\_Bw_{CA-\exists EVCP}$ (U)	MB	-	-	-	0.040	0.64	0.65
	$D\_Bw_{EVCP-\forall Ver}$ (Q)	MB	-	-	-	14.03	3.94	4.00
Verifier	D_Ovh_{Ver} (Q)	sec	0.0021	0.0021	**3.14**	0.0032	0.0021	0.0021
	$D\_Bw_{CA-\exists Ver}$ (Q)	MB	-	0.013	-	-	-	-
	$D\_Bw_{CMAE-\exists Ver}$ (Q)	MB	**11.05**	-	$5.74{\times}10^{-4}$	-	-	-
	$D\_Bw_{EVCP-\exists Ver}$ (Q)	MB	-	-	-	0.047	0.013	0.013
Revocation Latency		mins	1	≈ 0	1	1	1	1

Table 7.5: Cost comparison of various schemes on a certification system with 100,000 certificates and $\delta = 1$ minute. Due to the short value of δ used, CREV-II can also operate with the range optimization technique.

from $5.01{\times}10^4$ MB earlier in CRS. With still 0.08 ms of the verifier's average processing time on a query, or 2.52 ms daily, the downsides of CREV-I are the CA's extra bandwidth and processing due to the establishments of sessions. The overall bandwidth of $D\_Bw_{CA-\forall EVCP}$ is however only 578.88 MB/day, which is still reasonable. The CA's daily processing time becomes 324 s ($\sim$5.4 minutes), which is significantly smaller compared to the 1.33×10^6 s or $\sim$370 hours in OCSP.

CREV-II has a unique advantage that it delivers a standardized OCSP Response message. The advantage over OCSP is that the bandwidth requirement from the CA to all EVCP servers is only 6604.19 MB/day, which is a very much smaller than that from the CA to all Verifiers in OCSP (3.93×10^5 MB/day). The CA's computational cost increases to $2.16{\times}10^4$ s or $\sim$6 hours daily, which is still quite affordable. CREV-II thus makes online status-based schemes much more viable to deploy.

We now discuss the results when $\delta = 1$ minute as shown in Table 7.5. Due to the higher timeliness guarantee, CRS now must deal with substantially longer hash chains. Its CA's storage requirement now jumps to $5.01{\times}10^5$ MB. More importantly, the daily Verifier's computation overheads now increases to 3.14 s from 0.31 s when $\delta = 10$ minutes. As expected, the CA's computation cost in CREV-II also increases. Now, it becomes $2.14{\times}10^5$ s or $\sim$60 hours daily (on an 1.83-GHz Intel Core-2 CPU).[18] Since $\delta = 1$ minute, CREV-II now can also operate with the range optimization technique by merging the generated OCSP Responses together. With this optimization technique, CREV-II manages to bring down the CA's processing cost to only $1.18{\times}10^4$ s or $\sim$3 hours daily. Thus, when $\delta = 1$ minute, the range optimization helps keep the CA's overhead in CREV-II reasonable, while OCSP already incurs a very high cost of 1.33×10^6 s or $\sim$370 hours.

In summary, both CREV-I and CREV-II thus offer a good trade-off between the costs incurred on CA, CMAE and EVCP, while maintaining fairly lightweight requirements on the verifier. Compared with other examined schemes, these two CREV schemes are more viable for deployment when a near real-time timeliness guarantee is needed. CREV-II incurs a higher overhead on the CA's processing compared to CREV-I due to a higher volume of signing operations. However, with the range-optimization technique, it can be kept still reasonable.

The main new requirement for our CREV schemes is that an EVCP now needs to be available and provide a reliable CSI access service. However, given the fact that an EVCP needs to be up and running to provide its own online services, this requirement is theefore not an issue. With a present trend of using *Content Delivery Network (CDN)* which are commercially available from various parties (e.g. Akamai Technologies and Amazon CloudFront), we envision that the EVCP servers are able to provide the uptime

[18]We remark that the CA functioning as an OCSP Responder usually makes use of a more powerful CPU specification.

requirement either by themselves or through the use of third party providers. Lastly, we also note that the CREV schemes can be deployed in conjunction with other standard revocation schemes such as CRL or OCSP, thus ameliorating the potential workload bottlenecks faced in the existing revocation schemes.

7.6 Chapter Summary

We have presented two lightweight, practical and inherently-distributed certificate revocation schemes, called CREV, based on the recently available Extended-Validation Certificate infrastructure. Based on the analysis using our realistic performance analysis framework, we have shown the practicality of our schemes when compared to several existing schemes under the scenarios of revocation timeliness guarantees between 1 and 10 minutes. The CREV-I and CREV-II schemes offer a good balance of incurred costs on the involved entities, while maintaining lightweight requirements on the verifier. With the range-optimization technique, even with a timeliness guarantee of 1 minute, CREV-II can keep the CA's computational overheads manageable. Given their practical setting-implementability and lightweight requirements, the proposed CREV schemes are thus very suitable for certificate revocation in numerous real-world scenarios. With their near real-time timeliness guarantee, the schemes can offer appealing alternative solutions to the problem of providing a timely yet lightweight revocation infrastructure, including to mobile devices where computation, bandwidth and energy usage need to be kept small.

Chapter 8

Extending BAN Logic for Reasoning with PKI-based Protocols

It is common nowadays for a running program on a host to interact with external programs over a public untrusted network like the Internet. In order to secure the communication as well as the access to any provided services, public-key cryptographic operations are often employed. PKI-based protocols, including those used in securing program distribution and certificate revocation management of the PPLC, must be shown to be secure. In view of this, although protocol analysis on PKI-based protocols does not constitute a step in the PPLC, it is however an important prerequisite for securing the PPLC. Furthermore, such analysis is usually not a focus of most secure software development methodologies such as [112]. We therefore address the need for establishing concise yet practical formal reasoning on PKI-based protocols in this chapter.

Although a network protocol specification may look incomplex, designing a correct specification that satisfies the intended security objectives is well recognized as difficult. Hence, a formal analysis is necessary to establish the security of the protocols. Despite the availability of numerous formal techniques, there is a challenge to devise one that can be handily utilized by many protocol designers to verify real-world protocols.

Among various authentication logics, BAN Logic [26, 27] (see also Section 2.6 for its background information) is one of the best known and most widely used techniques [107, 143, 108]. This may well explain the constant appearance of publications applying BAN Logic to protocols and security systems even till now [5, 24, 155, 188, 35, 33], with application domains as diverse as wireless network [188], mobile communication [33] and voting system [155]. BAN Logic however does not properly deal with the detailed aspects of PKI-based authentication, such as certificate processing, as commonly practiced nowadays. This is arguably because PKI was not well established yet when the logic was designed. The situation now is however very different since PKI becomes common, and is an infrastructure on which many real-world protocols critically rely. Given the ubiquity of PKI-based protocols and their importance to host security, there is a need to update BAN Logic to better reason with PKI-based protocols, yet remain practical to use.

This chapter explains our extension of BAN Logic, published in [159], which allows for a more concise reasoning with PKI-based protocols. In our work, we begin with the starting point of retaining the popularity of BAN Logic among protocol designers. Our extension is along the lines of the work by Gaarder and Snekkenes [51], but better captures the current aspects of PKI. We address various limitations of [51] in capturing many important concepts and practices of the modern PKI usage. We also apply our logic to verify the session establishment protocol in CREV scheme as outlined in Chapter 7.

The remainder of this chapter is organized as follows. We first survey the related works in Section 8.1. We then give a brief review of the previous extension of BAN Logic by Gaarder and Snekkenes in Section 8.2. Section 8.3 presents our new extended BAN Logic, whereas Section 8.4 gives insight on its usage in preventing flawed protocol designs. An example of the application of our logic is considered in Section 8.5. Section 8.6 discusses the results, and Section 8.7 finally concludes this chapter.

8.1 Related Work

There exist various works in the literature, such as [5, 51, 156, 103, 85, 71, 161], which apply formal methods to PKI. We briefly survey ones which extend authentication logics, particularly BAN Logic, to deal with public-key authentication. Our focus of comparison will be on certificate processing formalism, time durations, and rules on messages encrypted (signed) using public (private) keys.

As pointed out by many researchers, such as in [15], the original BAN Logic is known to have limitations in describing "serverless protocols". This may happen since the only way of promoting "once said" ($\mid\!\sim$) to "believe" ($\models$) is by use of the freshness property of a statement, which is typically in the form of a nonce or a timestamp. In a serverless protocol, such freshness guarantee however cannot be provided, because the server is not necessarily available at the time of communication. In PKI setting, the limitations have to do with accepting the validity of a certificate. A certificate is issued at a point in time to be valid afterwards within its specified validity time interval (unless it is revoked at some time within this time interval). To work around this problem, the original BAN Logic choose to ignore the initial handling of certificates by assuming that they have been previously distributed, checked, and accepted as valid. Aziz and Diffie, who applied BAN Logic in [15], alternatively assume a certificate to always be fresh. Hence, the required belief statements on the certificate's contents can somehow be derived.

In their work on formal verification of CCITT X.509 protocol [51], Gaarder and Snekkenes argued that important aspects of public-key authentication are lost when BAN Logic is used for PKI-based protocol verification. To amend this deficiency, they proposed enhancements to BAN Logic that take certificate checking into account as an integral part of the reasoning process. The extension defines the notion of *duration* to capture some time-related aspects. A principal can therefore claim that a formulae is, was, or will be good in a time interval. In [156], Stubblebine and Wright however argued that the assumptions used are too restrictive for reasoning about long-lived security associations. Additionally, there exist issues on synchronization and synchronization bounds. Nevertheless, the simplicity of the logic proposed in [51], while improving the ability to reason with PKI-based protocols, is appealing. Our work here focuses on reworking the logic to be more accurately in line with the current PKI practice.

The work of Syverson [161] also adds the element of time to a logic of authentication. It incorporates a temporal formalism into a semantic model of BAN Logic, which is developed in [2], using temporal notions of "all points in the run prior to the current one" and "at some point in the run prior to the current one." Here in our work, we adopt the duration model of [51] which is relatively easier to use, yet enables the analysis of subtle relationships in PKI-based protocols.

Stubblebine and Wright [156] also propose a logic extension for dealing with PKI. The logic supports the concept of synchronization, revocation and recency. In pursuing more expressiveness, it however becomes far more complex than the original BAN Logic. We view that the complexity is a drawback which could hinder its adoption and application in practice.

Parts of our extension logic appeared earlier in the author's Master's thesis [157], which described the basic ideas of the extension logic. The logic formulation in [157] however only covers the basic aspects and is not full fledged. More specifically, it does not cover: the stated sender constructs (constructs 8.15 and 8.16—see later), the related New message-meaning for (public-key) encrypted message Rule (Rule 8.17), as well as the analysis of encrypted signed message (Rule 8.18) and signed encrypted message (Rules 8.19–8.22). In addition, it also lacks a comparative analysis of the logic with other works incorporating time into BAN Logic as discussed earlier in this section. Also missing in [157] is the analysis of how the extension can help prevent BAN Logic's "imprecise proof" when applied to PKI-based protocols (elaborated later in Section 8.4). The reported results in this chapter are based on the work [159], which published during the author's Ph.D candidature.

8.2 The Extension by Gaarder-Snekkenes

The extension logic of Gaarder and Snekkenes [51], which we call here **GS-BAN**, keeps BAN Logic's secret-key aspects intact for easy application. Our proposed extension takes the same approach. Appendix C lists all the secret-key rules of (the original) BAN Logic which are relevant to our extension in this chapter. Below, we summarize the extension logic of GS-BAN, and pinpoint some problems with it.

8.2.1 GS-BAN Extension Summary

Constructs for Public-Key Formalism

The following logic constructs were defined for public-key authentication in [51] (we use some slight notational modifications on keys below):

- $\wp\kappa\,(P, K_P)$: Principal P has associated a good public key K_P ;
- $\Pi(K_P^{-1})$: Principal P has a good private key K_P^{-1} ;
- $\sigma(X, K_P^{-1})$: X signed with P's private key K_P^{-1} ;
- $\{X\}_{K_P}$: X encrypted under P's public key K_P .

In GS-BAN extension, it is assumed that a digital signature is always in *appendix mode*. (We also adopt the same assumption here.) Hence, the signature construct $\sigma(X, K_P^{-1})$ is actually a contracted form of the following:

$$\sigma(X, K_P^{-1}) = X, Sign(K_P^{-1}, hash(X)) \tag{8.1}$$

where: $Sign()$ is a signature function, and $hash()$ is a hash construction function.

Certificate and Its Idealization

As mentioned earlier, one of the extension's main contributions is the idealization of a certificate. It adopts the certificate based on the X.509 standard [75], whose basic structure can be described as follows:[1]

$$Cert_P = \sigma((N, I, \delta^P, P, K_P, A), K_I^{-1}) \tag{8.2}$$

where:

- N : unique serial number of the certificate;
- I : name of the issuer;
- δ^P : validity period of the certificate, which consists of t_1^P (not before) and t_2^P (not after);
- P : the distinguished name of the principal;
- K_P : the certified public key of P;
- A : identifier of the signature algorithm employed;
- K_I^{-1} : I's private key which is used to sign the certificate.

[1]To simplify the formalism, GS-BAN extension considers the certification path to be of length 1. We also take similar approach here.

Based on the certificate structure, GS-BAN gave a certificate the following idealization:

$$Cert_P = \sigma((\Theta(t_1^P, t_2^P), \wp\kappa\,(P, K_P)), K_I^{-1}). \tag{8.3}$$

In its idealized form, a certificate contains a newly defined *validity period* construct, which is also called "duration-stamp" in [51]. The construct $(\Theta(t_1, t_2), X)$ was specifically introduced to say that "statement X holds in the time interval $[t_1, t_2]$". A message might thus be tagged with the duration-stamp denoting the time interval during which its creator claims the message is good. In a certificate, the tagged message represents the *certificate statement* $C^P = \wp\kappa\,(P, K_P)$ in GS-BAN. Hence, the issuer I who uttered the duration-stamped certificate in (8.3) claims that $\wp\kappa\,(P, K_P)$ is good in the time interval of $[t_1^P, t_2^P]$.

Inference Rules

To reason with the certificate and public-key constructs, GS-BAN introduced the following inference rules:

- The Message-meaning (for public-key) Rule:

$$\frac{P \models \wp\kappa\,(Q, K_Q)\,, P \models \Pi(K_Q^{-1}), P \triangleleft \sigma(X, K_Q^{-1})}{P \models Q \mid\!\sim X} \tag{8.4}$$

- The See signed-message Rule:

$$\frac{P \triangleleft \sigma(X, K_Q^{-1})}{P \triangleleft X} \tag{8.5}$$

- The Certificate duration-stamp Rule:

$$\frac{P \models Q \mid\!\sim (\Theta(t_1^R, t_2^R), C^R), P \models Q \models \Delta(t_1^R, t_2^R)}{P \models Q \models C^R} \tag{8.6}$$

  In the above rule, Q acts as the issuer of a certificate for R, whose certificate statement is denoted by C^R. The rule thus provides a way of promoting *once said* to *believe* about a certificate statement. For this purpose, P needs to believe that Q holds a belief on **"good time interval"** $[t_1^R, t_2^R]$ (denoted by $\Delta(t_1^R, t_2^R)$) . In other words, P must believe that the validity period of $[t_1^R, t_2^R]$ still holds according to the current time in Q.

Message-Recipient Construct

GS-BAN also introduced a notion of "intended recipient" of a message. Its authors argued that such assurance needs to be explicitly stated as one of the goals in their X.509 protocol formalism. Moreover, they also claimed that the original BAN Logic provides no means of expressing it in the language available. Therefore, the construct "$\Re(X, P)$" was introduced to say that "P is the intended recipient of message X".

The construct is defined to appear in the following form:

$$P \to Q : X, \Re(X, Q) \tag{8.7}$$

which should be interpreted as "P sends to Q a particular message X together with a statement telling that Q is the intended recipient of X". This effort clearly represents a step forward in capturing the notion of *explicit principal naming* [1]. In doing so, however, GS-BAN requires the receiver to hold an assumption about the sender's jurisdiction over the message recipient statement (i.e. $Q \models P \Rightarrow \Re(X, Q)$), an approach which we will examine more closely below.

8.2.2 Problems and Limitations

Despite its improvements on BAN Logic, there exist some key aspects of GS-BAN extension [51] which we find rather unsettling:

- **Assumption on $\Pi(K_Q^{-1})$**: In verifying a protocol using GS-BAN, a (supplied) assumption needs to be added to a principal that he/she believes the goodness of the private key of *another principal* involved. That is, a formula such as "$P \models \Pi(K_Q^{-1})$" needs to be *vacuously held* at the start of the verification process. Such a stipulation is needed so that GS-BAN can process the Message-meaning Rule (8.4). We however view the inclusion of such an assumption as an unsound practice. Such a formula should never be supplied as an assumption, but must rather be logically derived from a certificate reasoning within the logic. The only assumption needed is the goodness of public and private keys of the CA in addition to the principal's own key pair. A principal then should be able to derive the goodness of the other principal's keys from the certificates issued by the CA. Furthermore, if a key is no longer considered good, revocation mechanisms like CRL [37] should be incorporated in the logic.

- **Assumption for message recipient construct**: One of the consequences of the Message-recipient construct in GS-BAN is that it requires a statement $\Re(X, P)$ to be made as a protocol goal for every message with the intended recipient construct. In our opinion, such a statement does not need to be stipulated as a formalism goal since we are mainly interested in deriving goals about the goodness of the generated session key(s). In our view, the intended recipient construct should instead be incorporated as a part of logic rules. Moreover, as mentioned above, the construct requires the receiver to hold an assumption about the sender's jurisdiction over the message recipient statement (i.e. $Q \models P \Rightarrow \Re(X, Q)$). In our new extension, by integrating the intended-recipient requirement into the New message-meaning Rule, we manage to eliminate the need for sender's jurisdiction, thus simplifying the reasoning on message processing.

- **Omission of certificate revocation process**: GS-BAN apparently has chosen to ignore the incorporation of the certificate revocation issue into its logic. Instead, it assumes that each principal always maintains the goodness of its private key. As a result, we need to believe that a certificate, once issued, is *always good* for the time-interval specified in its validity period. Considering the importance of certificate revocation in public-key authentication, its omission in the formalization represents a limitation of the extension.

8.3 MPKI-BAN: Extending BAN Logic to Deal with PKI

Motivated by the aforementioned drawbacks of GS-BAN extension [51], we propose the following extensions to BAN Logic which addresses various limitations of GS-BAN and also provides other substantial contributions. We call our new extension **MPKI-BAN** since it is aimed at dealing with modern PKI-based protocols.

Our approach of presenting the results in this chapter focuses on pragmatism. We focus on the logic definition and its usage application while leaving theoretical analysis of the logic, e.g. the logic's soundness and completeness with respect to some well-defined semantics, as a separate treatment beyond our work's scope. Our goal here is to expound an up-to-date yet accessible authentication logic which can be handily used by protocol designers who may not be experts in authentication logics or formal methods.

8.3.1 Revised Idealized Certificate

In MPKI-BAN, we idealize a certificate's statement as follows:[2]

$$Cert_P = \sigma(\Theta'(t_1^P, t_2^P, I, (\wp\kappa\,(P, K_P)\,, \Pi(K_P^{-1}))), K_I^{-1}) \tag{8.8}$$

To capture the need to ensure that a certificate is not revoked prior to deriving any beliefs on its contents, we define a "*validity period (duration-stamp) with revocation*" construct $\Theta'(t_1, t_2, I, X)$. This construct states that "statement X holds in the time interval $[t_1, t_2]$, *provided that* it is not revoked by the issuer I". We will redefine the certificate validation rule using this construct later in Section 8.3.5. We may keep the original validity period construct of GS-BAN $(\Theta(t_1, t_2), X)$, now restated as $\Theta(t_1, t_2, X)$, to represent a validity period *without revocation*. That is, the statement X will always be good in $[t_1, t_2]$ without any need to revalidate it with the issuer. For a certificate in the X.509-based PKI framework, however, we need to use the newly defined function $\Theta'()$ rather than $\Theta()$.

As can be seen, we now include $\Pi(K_P^{-1})$ into the idealized certificate's statement in contrast with the previous definition in (8.3). Hence, a valid certificate now assures that *both public and private keys* of a principal are good. With this modification, we thus eliminate the need to have an assumption about the goodness of the other principal's

[2]Note that the *complete* idealized certificate definition will include message-recipient construct as defined in (8.12).

private key as in GS-BAN. This modification is crucial as it helps formalize the close relationship between the certificate validity and the goodness a private key. The rule makes it clear that a belief on $\Pi(K_P^{-1})$ should be derived from a valid P's certificate. Consequently, should the private key of a principal ever be compromised, the principal must notify and ask the CA to revoke his/her certificate, a requirement that is consistent with current PKI practices.

8.3.2 New Use of Message-Recipient Construct

Different from GS-BAN, the message-recipient construct is defined in our extension to be part of signature construct (σ), which now appears in the following form:

$$\sigma(\Re(X, P), K_Q^{-1}). \tag{8.9}$$

Note that now we make statement $\Re(X, P)$ as a stronger statement than X, as it implies a statement X together with its recipient tag for P.

Unlike the use of the message-recipient construct in GS-BAN, we thus incorporate it into our New message-meaning Rule (defined below) as *one of its premises.* A principal thus needs to ensure the existence of a valid recipient tag in the signed message in order to proceed with the rule. With this, we also manage to eliminate the requirement of stating a message-recipient as a verification goal, as well as the requirement of introducing an assumption about the sender's jurisdiction over the message-recipient as in GS-BAN.

8.3.3 New Message-Meaning Rule for Private-Key Signed Message

In symmetric-key based authentication, the intended recipient of a message can usually be inferred from the shared secret-key employed in the encryption or the generation of Message Authentication Code (MAC). Unfortunately, this does not apply in private-key signed messages where the same private key is used to sign messages regardless of their intended recipient. Hence, in public-key authentication, there is a greater need to follow the "naming principle" [1].[3] Surprisingly, the same mistake due to disregarding this principle seems to be made time after time in many protocols (see e.g. [97]). Given this concern, we redefine the **New message-meaning for (private-key) signed message Rule** as follows:

$$\frac{P \models \wp\kappa\,(Q, K_Q)\,, P \models \Pi(K_Q^{-1}), P \triangleleft \sigma(\Re(X, P), K_Q^{-1})}{P \models Q \mid\!\sim X}. \tag{8.10}$$

[3]It is possible to argue that in a few situations, due to privacy considerations, protocol designers might aim to withhold identity information as long as possible, thus conflicting with the naming principle. We however focus here on general situations where ensuring secure authentication takes precedence over privacy concerns.

With (8.10), we thus integrate the message-recipient construct into the Message-meaning Rule. Our requirement for the third premise above is strongly motivated by the works of Meadows [107] and Boyd and Mathuria [25], which still managed to uncover a loophole in the Aziz-Diffie protocol [15] despite the application of BAN Logic to the proposed protocol. We will examine this later in Section 8.4 where we show how the new rule and some cautionary note on the application of BAN Logic can together help pinpoint the problem with the protocol and its imprecise proof.

8.3.4 All-Recipient See Rule

Having defined the New message-meaning for signed message Rule in (8.10) above, we further note that it is possible for a message to be actually intended for *all* principals in the protocol. A good example is a certificate, which is meant to be accepted by any principal as long as he/she trusts the issuer. We thus define a special principal name "*all*", and define the following **All-recipient see Rule**:

$$\frac{P \lhd \sigma(\Re(X, all), K_Q^{-1})}{P \lhd \sigma(\Re(X, P), K_Q^{-1})}. \tag{8.11}$$

8.3.5 Certificate and New Certificate-Validation Rule

In line with the use of *all recipient* definition above, a certificate is now to be idealized in MPKI-BAN as follows:

$$Cert_P = \sigma(\Re(\Theta'(t_1^P, t_2^P, I, (\wp\kappa\,(P, K_P)\,, \Pi(K_P^{-1}))), all), K_I^{-1}). \tag{8.12}$$

This certificate definition now correctly includes the message-recipient construct, and subsumes the previous interim definition given in (8.8).

To derive a belief on a certificate, this **New certificate-validation Rule** is used:

$$\frac{P \mid\equiv Q \mid\sim \Theta'(t_1^R, t_2^R, Q, C^R), P \mid\equiv Q \mid\equiv \Delta(t_1^R, t_2^R), P \mid\equiv Q \mid\equiv \Phi(C^R)}{P \mid\equiv Q \mid\equiv C^R}. \tag{8.13}$$

Here, Q acts as the certificate issuer. C^R denotes a certificate statement, consisting of $\wp\kappa\,(P, K_P)$ and $\Pi(K_P^{-1})$. This rule thus supersedes Rule (8.6) previously defined in GS-BAN. The third premise (i.e. $P \mid\equiv Q \mid\equiv \Phi(C^R)$) emphasizes the need for "*certificate revalidation step*" with the issuer Q as the principal stated in $\Theta'()$ construct. P must ensure this premise by checking that Q still believes that the uttered certificate statement remains valid (i.e. $Q \mid\equiv \Phi(C^R)$).[4] In the CRL model, this step is done by checking the absence of the certificate in question in Q's recent CRL.

[4]Note that although we put C^R as the parameter of $\Phi()$, in practice the matching is done based on the unique certificate's serial number N.

In the rule above, we note that the resulting belief statement (C^R) can be argued as an "unstable" statement [27]. That is, the statement is valid only at the time of validation but not necessarily thereafter, as the corresponding certificate might be revoked at some point of time in the future. A more elaborate logic might include a more general time-related reasoning as exemplified by the work [156]. However, in order to keep our extension simple, we avoid doing so. Instead, we limit such statements to beliefs on *the goodness of public and private key* of a principal, and not on actual statements communicated among the principals in the protocol. In fact, both BAN Logic [27] and GS-BAN [51] implicitly make a similar simplification too, because a secret key, in practice, will eventually cease to be valid due to its expiry or a possible security breach.

8.3.6 Duration-Stamp (without Revocation) Validation Rule

Having mentioned that we can keep a validity period without a revocation construct (see Section 8.3.1), we can define the following **Duration-stamp (without revocation) validation Rule**:

$$\frac{P \models Q \mid\sim \Theta(t_1, t_2, X), P \models Q \models \Delta(t_1, t_2)}{P \models Q \models X}. \tag{8.14}$$

This rule thus promotes *once said* to *believe* on a duration-stamped (without revocation) statement. Here, P needs to ensure that the validity period of $[t_1, t_2]$ still holds according to the current time in Q. This rule is similar to the certificate duration-stamp Rule (8.6) in GS-BAN. For validating a certificate, MPKI-BAN however makes use of the more comprehensive Rule (8.13), which also includes a revocation checking step.

8.3.7 Message-Sender Construct

Realizing the importance of the naming principle, we take another step to define a *message-sender* construct in MPKI-BAN. It aims to introduce the notion of a "stated sender" of a message. The message-sender construct $\mathcal{S}(X, Q)$ is defined to specifically say that "message X together with Q as the stated sender of the message". It can appear in the following two forms:

- Stated-sender within the encrypted message:
  This message-sender construct appears in the following form:
  $$\{\mathcal{S}(X, Q)\}_{K_P}. \tag{8.15}$$
  It occurs when the encryption is employed with an additional function of authentication.[5] Principal P, who receives the message from Q, ensures this construct by first decrypting the message, and then verifying that it contains Q as the sender ID together with correctly formatted message X. We define the corresponding Message-meaning for public-key encryption Rule in Section 8.3.8.

[5]It is thus important to make clear the role of encryption in a protocol specification (see e.g. [1]).

- Stated-sender outside of the encrypted message:

  This form appears as follows:
  $$\mathcal{S}(\{X\}_{K_P}, Q). \tag{8.16}$$
  It takes place when the encrypted message ($\{X\}_{K_P}$) comes in a message signed by Q. We discuss the usage of this construct and its related rule in Section 8.3.10.

8.3.8 New Message-Meaning Rule for Public-Key Encrypted Message

When receiving a private-key signed message, it is important to ensure the intended recipient of the message. This requirement is captured by the message-recipient construct. When dealing with a message encrypted with the public key of a recipient, it is the identity of the *sender* that matters.

We consider the two following commonly applicable scenarios where a public-key encrypted message portion appears in a message. The first is when the encryption is employed as a form of authentication in addition to providing confidentiality. This occurs with the whole message is encrypted. The second scenario occurs when an encrypted message portion is contained in a signed message. We deal with the first scenario here, whereas the second scenario is analyzed in Section 8.3.10 below.

Since we require a stated sender assurance on a public-key encrypted message, an already decrypted message must thus contain the sender's identity in a pre-defined location. The first type of the message-sender construct, $\{\mathcal{S}(X, Q)\}_{K_P}$ (8.15), applies in this scenario. That is, the identity of the sender is within the encrypted message. We thus define the following **New message-meaning for (public-key) encrypted message Rule**:
$$\frac{P \models \wp\kappa\,(P, K_P)\,, P \models \Pi(K_P^{-1}), P \lhd \{\mathcal{S}(X, Q)\}_{K_P}}{P \models Q \mid\!\sim X} \tag{8.17}$$

We later show in Section 8.4 how this rule could have helped deal with a loophole in Needham-Schroeder public-key authentication protocol, which was exploited in [97].

Note that it is possible that the decrypted message X is actually a signed message. This case is considered further below (Section 8.3.9).

8.3.9 Additional Message-Meaning Rule for Encrypted Signed Message

The message-meaning rule above (8.17) specifies how MPKI-BAN Logic deals with a public-key encrypted message. In the case where the decrypted message is actually a signed message, we thus encounter an *encrypted signed message*, i.e. a signed message which is sent encrypted.

For the already-decrypted message, we define the following **Message-meaning for (private-key) signed message –after decryption– Rule**:
$$\frac{P \models \wp\kappa\,(Q, K_Q)\,, P \models \Pi(K_Q^{-1}), P \mid\!\sim \sigma(\Re(X, P), K_Q^{-1})}{P \models Q \mid\!\sim X}. \tag{8.18}$$

8.3.10 Rule for Signed Encrypted Message

Here we address another subtle issue, namely the reasoning of *signed encrypted message*, that is a signed message that contains encrypted message portion(s). Although the commonly suggested practice is to perform signing before encrypting [7], it is not uncommon to find the reverse in some published protocols.[6]

Hence, after a message has been validated using the New message-meaning for signed message Rule (8.10), we need to specify how to process the encrypted message portion. Here, there are two possibility of the message-sender constructs as in (8.15) and (8.16):

- Stated-sender information is within the encrypted message ($\{\mathcal{S}(X,Q)\}_{K_P}$):
  Here, the sender identity is part of the encrypted message. We thus define the following **Message-meaning for encrypted message portion –after signature verification, with encrypted stated-sender and included freshness assurance– Rule**:
  $$\frac{P \models \wp\kappa\,(P, K_P)\,, P \models \Pi(K_P^{-1}), P \models Q \mid\!\sim \{\mathcal{S}(X,Q)\}_{K_P}}{P \models Q \mid\!\sim X}. \tag{8.19}$$

- Stated-sender information is outside of the encrypted message ($\mathcal{S}(\{X\}_{K_P}, Q)$):
  For this case, it is *insufficient* to simply take the presence of a sender ID in the (correctly signed) clear-text message portion. This is due to the "proof of authorship" problem of the encrypted message (see [7]). The problem is that we really cannot infer that the sender *actually knows* about X. To resolve this problem, we require the sender *to prove his knowledge of* X. On way to fulfill this requirement is by requiring the sender to generate the signature over the message including X. Thus, X is calculated in the signature generation although its (clear-text) value does not appear in the signed message. One example protocol performing this is a proposed enhancement in [25] to fix the broken Aziz-Diffie protocol [15], which is analyzed later in Section 8.4.2. In this way, the message-sender construct $\mathcal{S}(\{X\}_{K_P}, Q)$ can be satisfied. In validating the signature, the receiver thus needs to first decrypt $\{X\}_{K_M}$ into X, and check that the signature is correctly constructed using X as a part of the input message. We therefore can define the following **Message-meaning for encrypted message portion –after signature verification, with clear stated-sender and included freshness assurance– Rule**:
  $$\frac{P \models \wp\kappa\,(P, K_P)\,, P \models \Pi(K_P^{-1}), P \models Q \mid\!\sim \mathcal{S}(\{X\}_{K_P}, Q)}{P \models Q \mid\!\sim X}. \tag{8.20}$$

[6]Syverson [162] analyzed this ordering principle as suggested in [7], and questioned its applicability by putting forward examples when the principles are not applicable. Nevertheless, Syverson agreed that such principle is useful as a general guideline, although it should be critically used by the protocol designers.

Notice that without the message-sender assurance, the receiver P can only derive $P \models Q \mid\sim \{X\}_{K_P}$, and not $P \models Q \mid\sim X$. This difference is crucial, and disregarding it can lead to erroneous proof generations using the original BAN Logic.

In order for the consequences of (8.19) and (8.20) to be derived further using the Nonce verification Rule (R_2 in Appendix C), X needs to come with a freshness assurance so that $P \models \sharp(X)$ can be derived. There are two possible sources for such a freshness:

- The freshness assurance is inside the encrypted message:
  Here, statement X contains freshness assurance Y, which can be a nonce or a timestamp. As such, belief $P \models \sharp(Y)$ is derivable. Given the Freshness extension Rule (R_{10} in Appendix C), we therefore can derive the required belief $P \models \sharp(X)$. Rules (8.19) and (8.20) above are defined to deal with this situation.

- The freshness assurance is not within the encrypted message:
  In this case, the freshness assurance must come in the clear-text portion outside of the encrypted portion ($\{X\}_{K_P}$) in the signed message. To emphasize the need for a freshness assurance in this case, the third premise in Rules (8.19) and (8.20) must be applied on both the encrypted message portion *and* a "fresh" statement. Hence, for completeness, we also define the following two rules:

  – **Message-meaning for encrypted message portion –after signature verification, with encrypted stated-sender and external freshness assurance– Rule**:

  $$\frac{P \models \wp\kappa\,(P, K_P)\,, P \models \Pi(K_P^{-1}), P \models Q \mid\sim \{\mathcal{S}(X,Q)\}_{K_P}, Y}{P \models Q \mid\sim X, Y}. \tag{8.21}$$

  – **Message-meaning for encrypted message portion – after signature verification, with clear stated-sender and external freshness assurance– Rule**:

  $$\frac{P \models \wp\kappa\,(P, K_P)\,, P \models \Pi(K_P^{-1}), P \models Q \mid\sim \mathcal{S}(\{X\}_{K_P}, Q), Y}{P \models Q \mid\sim X, Y}. \tag{8.22}$$

  Statement Y in (8.21) and (8.21) is the statement in the previously-verified signed message, which also contains the freshness assurance (i.e. a timestamp or a nonce). Given $P \models \sharp(Y)$, we can therefore derive $P \models \sharp(X, Y)$ using the Freshness extension Rule (R_{10} in Appendix C). As a result, the belief on X can be derived.

8.3.11 Redefined Message-Meaning Rule for Keyed Hashed Message

For completeness, we redefine here a new construct for message authentication using secret-key based MAC:

$$\mu(X, K_{PQ}) = X, H(K_{PQ}, X). \tag{8.23}$$

Similar to signature construction in (8.1), MAC generation is performed by applying a selected keyed hash-construction function $H()$ to message X using K_{PQ}, and then appending the resulting hash value to X. The related **Message meaning for (secret-key) hashed message Rule** is defined as follows:

$$\frac{P \models Q \overset{K_{PQ}}{\longleftrightarrow} P, P \triangleleft \mu(X, K_{PQ})}{P \models Q \mid\sim X}. \tag{8.24}$$

Here we omit the requirements for the intended-recipient and stated-sender by assuming that the shared secret key is good and message X is unambiguously formatted.

8.3.12 Additional Rules for See Operator

For completeness, we also define the following new rules for see operator:

- The **See keyed-hashed message Rule**:

$$\frac{P \triangleleft \mu(X, K_{PQ})}{P \triangleleft X} \tag{8.25}$$

- The **See signed message Rule**:

$$\frac{P \triangleleft \sigma(X, K_Q^{-1})}{P \triangleleft X} \tag{8.26}$$

- The **See recipient-tagged message Rule**:

$$\frac{P \triangleleft \Re(X, P)}{P \triangleleft X} \tag{8.27}$$

For easy reference, all the newly added rules in MPKI-BAN are listed in Appendix D.

8.4 Using MPKI-BAN Logic

We now show how our MPKI-BAN could have helped prevent problems in vulnerable published protocols.

8.4.1 Needham-Schroeder Public-Key Authentication Protocol

In [97], Lowe published an attack on Needham-Schroeder public-key authentication protocol. The protocol proceeds as follows (Notation $\{X\}_K$ below denotes an encryption operation using key K. When K is a private key, it represents a signing operation.):

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_B, B\}_{K_S^{-1}}$
3. $A \rightarrow B : \{N_A, A\}_{K_B}$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \{K_A, A\}_{K_S^{-1}}$
6. $B \rightarrow A : \{N_A, N_B\}_{K_A}$
7. $A \rightarrow B : \{N_A\}_{K_B}$.

Despite the assumption that each principal has each other's public key correctly, Lowe managed to find an attack on the protocol. The problem with the protocol has to do with the encryption using public key of the recipient without clear identity of the sender. In order to prevent potential attacks, Lowe proposed the modification of message 6 into: $6'$. $B \rightarrow A : \{\boldsymbol{B}, N_A, N_B\}_{K_A}$.

In MPKI-BAN, with our New message-meaning for (public-key) encrypted message Rule (8.17), the derivation of the incorrect beliefs would be impossible. This is because the required statement $\{\mathcal{S}((N_A, N_B), \boldsymbol{B})\}_{K_A}$ is underivable from message 6. The proposed fix using message $6'$ adds the sender identity (i.e. B), as part of the encrypted message. Thus, the statement $\{\mathcal{S}((N_A, N_B), B)\}_{K_A}$ is thus derivable. This example highlights the value of integrating message-sender construct into Message-meaning Rule.

8.4.2 Aziz-Diffie Protocol

Now, let us analyze the protocol by Aziz and Diffie [15] which was still broken despite the use of the original BAN Logic by the authors to verify it.

The protocol uses public-key cryptography for securing the wireless link between a mobile (M) and a base (B). In the following, *alg_list* denotes a list of flags representing potential secret-key algorithms chosen by M. The flag *sel_alg* represents the particular algorithm selected by B from the list *alg_list*, and is to be employed to encrypt the subsequent data call. The protocol for providing the connection setup between M and B is as follows ([25]):

1. $M \rightarrow B: \; Cert(M),\, N_M,\, alg\_list$
2. $B \rightarrow M: \; Cert(B),\, \{X_B\}_{K_M},\, sel\_alg,\, \{hash(\{X_B\}_{K_M}, sel\_alg, N_M, alg\_list)\}_{K_B^{-1}}$
3. $M \rightarrow B: \; \{X_M\}_{K_B},\, \{hash(\{X_M\}_{K_B}, \{X_B\}_{K_M}\}_{K_M^{-1}}$

Here N_M is a nonce from M, whereas $Cert(M)$ and $Cert(B)$ are certificates of M and B respectively. X_B and X_M denote the particular session key values chosen by B and M, respectively. The final session key x is calculated as $X_M \oplus X_B$.

The protocol was verified in [15] using BAN Logic. Our analysis however reveals that the given proof in [15] apparently contains a flaw. The flaw is introduced in the error-prone idealization step of the formalism. The work [15] idealized message 2 (using the original BAN Logic) as:

$$\{\{\overset{K_B}{\longmapsto} B\}_{K_{Ca}^{-1}}, \boldsymbol{M} \overset{\boldsymbol{X_B}}{\longleftrightarrow} \boldsymbol{B}, N_M\}_{K_B^{-1}}. \tag{8.28}$$

The problem with this is that what can actually be derived from the message above is: $M \overset{\{X_B\}_{K_M}}{\longleftrightarrow} B$ or $\{M \overset{X_B}{\longleftrightarrow} B\}_{K_M}$, and not $M \overset{X_B}{\longleftrightarrow} B$. This formalism pitfall allowed [15] to incorrectly derive the desired goals despite the loophole in the protocol.

After the publication of [15], both [107] and [25] managed to mount an attack on the protocol. The attack outlined in [25] makes use of two parallel open sessions. Impersonating M in the first session, an attacker C is able to obtain $\{X_B\}_{K_M}$ from message 2. In the second session, C then replays $\{X_B\}_{K_M}$ to the initiating M when it plays a role as B. The work [25] correctly pointed out the source of the problem is that C can construct message 2 without the knowledge of X_B. To fix the protocol, [25] proposed the modifications of message 2 and 3 into:

$2'.\ B \rightarrow M:\ \ Cert(B),\ \boldsymbol{N}_B,\ \{X_B\}_{K_M},\ sel\_alg,\ \{hash(\boldsymbol{X}_B, \boldsymbol{M}, N_M, alg\_list)\}K_B^{-1}$
$3'.\ M \rightarrow B:\ \ \{X_M\}_{K_B},\ \{hash(\boldsymbol{X}_M, \boldsymbol{B}, \boldsymbol{N}_B\}K_M^{-1}$

Nonce $\boldsymbol{N}_B$ now provides the freshness assurance, taking the role of $\{X_B\}_{K_M}$ in the original protocol. Note that message $2'$ contains $\boldsymbol{M}$ in the signed hash's arguments. Likewise, message $3'$ now contains $\boldsymbol{B}$. Such inclusions are thus in line with our requirement of *message-recipient* in our New message-meaning for signed message Rule (8.10). Although M and B are not included in the clear portions of message $2'$ and $3'$ respectively, they are actually present from the message transfer context. As such, we should take their presence into account in the idealized protocol.

In message $2'$, the encrypted message portion $\{X_B\}_{K_M}$ is sent with a sender assurance outside of the encrypted message portion. The assurance is implicitly established by the sender's knowledge of X_B in the message's signature. Thus, this therefore belongs to the case of sender-assurance sent clear outside of the encrypted message ($\mathcal{S}(\{X_B\}_{K_M}, B)$), which is analyzed above in Section 8.3.10. Hence, we can process the statement using the message-meaning defined in (8.20). Similarly, message $3'$ employs the same technique on $\{X_M\}_{K_B}$. An alternative proposed solution is to simply use (explicit) enclosed sender-stated assurances. In this case, we can have $\{X_B, B\}_{K_M}$ in message $2'$, and $\{X_M, M\}_{K_B}$ in message $3'$.

8.5 Sample Application of MPKI-BAN Logic

Appendix E shows how we can apply MPKI-BAN Logic to formally analyze a PKI-based protocol. We have chosen the *3-way Session-Establishment* protocol of CREV-II which is outlined in Section 7.3.3. The protocol is chosen here so that we can establish the security assurance of the proposed session establishment of hash-chaining based revocation service. The idealized protocol, protocol goals, assumptions, and the proof that the protocol can achieve the goals are given in Appendix E.

8.6 Discussion

We have presented MPKI-BAN as an extension of BAN Logic to deal with PKI. It solves a number of issues in GS-BAN [51] whose solutions are vital to a more accurate reasoning with PKI-based protocols. In summary, our enhancements are as follows.

- We present an improved idealization of certificate, in which the assurance of a private key is also derived from certificate. This eliminates the need to have an assumption about the goodness of the other principal's private key as in GS-BAN.
- We define a New message-meaning for (private-key) signed message Rule, which contains the message-recipient construct. By doing so, we manage to eliminate the requirement for stating a message-recipient as a goal, as well as that for introducing an assumption about the sender's jurisdiction over the message-recipient construct as in GS-BAN.
- We also revise the Certificate-validation Rule, which now includes a third premise to highlight the need for a certificate revalidation step. In the CRL model, the new rule thus makes explicit of two requirements in validating a certificate: time synchronization with the issuer and checking with the issuer's recent CRL.
- We define the New message-meaning for (public-key) encrypted message Rule which now requires a message-sender construct. This modification is vital as it prevents a common pitfall in public-key protocol design, as clearly illustrated among others by Lowe's attack on Needham-Schroeder public-key authentication protocol [97].
- We additionally deal with the cases of encrypted signed message and signed encrypted message.

Although some of the modifications may look simple, they are however crucial for better reasoning with PKI-based protocols. Table 8.1 contrasts several constructs and rules from GS-BAN with those in our new MPKI-BAN.

8.7 Chapter Summary

We have presented MPKI-BAN, which makes BAN Logic more in line with concepts and practices in the modern PKI setting. MPKI-BAN removes various limitations of [51], which we refer to as GS-BAN, for more concise reasoning with a certificate-based public-key authentication. In particular, MPKI-BAN redresses the reasoning on the goodness of private keys, and takes certificate revocation into account. Furthermore, it also addresses common pitfalls in public-key based protocol design due to insufficient attention placed on the "intended recipient" and "stated sender" of a message. MPKI-BAN makes the recipient and the sender explicit, and these requirements are tightly coupled into the logic. In this way, it reduces the likelihood of allowing such flaws in the protocol and the

Construct/Rule	GS-BAN Extension [51]	Our MPKI-BAN Extension
Idealized certificate	$\sigma((\Theta(t_1^P, t_2^P), \wp\kappa\,(P, K_P)), K_I^{-1})$	$\sigma(\Re(\Theta'(t_1^P, t_2^P, I, (\wp\kappa\,(P, K_P)\,, \Pi(K_P^{-1}))), all), K_I^{-1})$
Certificate validation	$\frac{P \mid\equiv Q \mid\sim (\Theta(t_1^R, t_2^R), C^R), P \mid\equiv Q \mid\equiv \Delta(t_1^R, t_2^R)}{P \mid\equiv Q \mid\equiv C^R}$	$\frac{P \mid\equiv Q \mid\sim \Theta'(t_1^R, t_2^R, Q, C^R), P \mid\equiv Q \mid\equiv \Delta(t_1^R, t_2^R), P \mid\equiv Q \mid\equiv \Phi(C^R)}{P \mid\equiv Q \mid\equiv C^R}$
Usage of message-recipient	$\Re(X, P)$	$\sigma(\Re(X, P), K_Q^{-1})$
Usage of stated-sender	-	$\{\mathcal{S}(X, Q)\}_{K_P}$ or $\mathcal{S}(\{X\}_{K_P}, Q)$
Message-meaning for signed message	$\frac{P \mid\equiv \wp\kappa\,(Q, K_Q)\,, P \mid\equiv \Pi(K_Q^{-1}), P \triangleleft \sigma(X, K_Q^{-1})}{P \mid\equiv Q \mid\sim X}$	$\frac{P \mid\equiv \wp\kappa\,(Q, K_Q)\,, P \mid\equiv \Pi(K_Q^{-1}), P \triangleleft \sigma(\Re(X, P), K_Q^{-1})}{P \mid\equiv Q \mid\sim X}$
Message-meaning for signed message (after decryption)	-	$\frac{P \mid\equiv \wp\kappa\,(Q, K_Q)\,, P \mid\equiv \Pi(K_Q^{-1}), P \mid\sim \sigma(\Re(X, P), K_Q^{-1})}{P \mid\equiv Q \mid\sim X}$
Message-meaning for encrypted message	-	$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \triangleleft \{\mathcal{S}(X, Q)\}_{K_P}}{P \mid\equiv Q \mid\sim X}$
Message-meaning for encrypted message portion (after signature verification):		
with encrypted stated-sender and included freshness assurance	-	$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \mid\equiv Q \mid\sim \{\mathcal{S}(X, Q)\}_{K_P}}{P \mid\equiv Q \mid\sim X}$
with encrypted stated-sender and external freshness assurance	-	$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \mid\equiv Q \mid\sim \{\mathcal{S}(X, Q)\}_{K_P}, Y}{P \mid\equiv Q \mid\sim X, Y}$
with clear stated-sender and included freshness assurance	-	$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \mid\equiv Q \mid\sim \mathcal{S}(\{X\}_{K_P}, Q)}{P \mid\equiv Q \mid\sim X}$
with clear stated-sender and external freshness assurance	-	$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \mid\equiv Q \mid\sim \mathcal{S}(\{X\}_{K_P}, Q), Y}{P \mid\equiv Q \mid\sim X, Y}$

Table 8.1: Comparison of public-key constructs and rules from GS-BAN [51] and those from our MPKI-BAN.

corresponding formalism. In addition, our logic also deals with the cases of (public-key) signed encrypted message and encrypted signed message. Some examples on the usage of MPKI-BAN are given, and these examples demonstrate that MPKI-BAN can help prevent common mistakes in public-key protocol design and verification.

Given that BAN Logic is a widely-used logic, we envisage that MPKI-BAN extension can provide a practical and valuable tool without requiring the users to manipulate a substantially complex formalism. It is indeed our aim to make the formal analysis on PKI-based protocols become more handily accessible to a wider range of protocol designers, thus allowing more parties to improve the security of their protocols.

Chapter 9

Conclusion

This chapter concludes this thesis. We summarize the contributions of this thesis in Section 9.1, and discuss some future directions in Section 9.2.

9.1 Summary of the Thesis

Malware is a critical security threat today particularly to connected host systems. It has now evolved from occasional exploits to a global multi-million dollar criminal industry, and is threatening the Internet economy. In this thesis, we focus on protecting software, i.e. programs, to ensure that they run as intended without violating a host's security. We safeguard software from various attack vectors based on the Program Protection Life Cycle (PPLC) framework (see Figure 1.1). Our works reported in this thesis aim to mitigate malware threats by deploying additional security mechanisms on top of the OS as well as securing critical infrastructure which supports the host's security mechanisms. We have shown how the proposed measures can secure the PPLC stages by providing effective defense against numerous attack vectors while incurring acceptable performance overheads. Using BAN Logic's notation[1], Table 9.1 lists the beliefs established by the proposed measures, whereas Table 9.2 shows their relationship in achieving the objective assurance of "*good intended execution*" of a program on a host. We summarize the main conclusions from our results as follows.

Firstly, to protect a running program on a host particularly against zero-day attacks, we have conducted an in-depth security analysis on the Self-based (Stide) IDS model [67, 152, 153] and related system-call monitoring IDSs such as [146]. We have presented an efficient algorithm for automated mimicry attack construction on the Self-based IDS and its more precise graph-based variant. Using several real programs and exploits, we also have shown the practicality of the attack construction, with execution times of at most a few seconds. We also have shown how the construction method can be generalized

[1]Symbol '$\mid\equiv$' in a BAN Logic formula denotes *believes* operator. Section 2.6 provides some background on BAN Logic.

Our Proposed Mechanism(s)	Belief(s) Established
BinAuth (Chap. 5), Certificate revocation scheme (Chap. 7)	$H \models P$ (9.1) $H \models \text{good\_behavior\_of\_P}$ (9.2) $H \models \text{intact\_pathname\_of\_P}$ (9.3)
Vulnerability database (Chap. 6)	$H \models \text{no\_known\_vulnerability\_on\_P}$ (9.4)
Improved IDS (Chap. 3 & 4)	$H \models \text{unaltered\_execution\_of\_P}$ (9.5)

Table 9.1: Beliefs established by the proposed mechanisms.

Belief Derivations for "Good Intended Program Execution"
$\dfrac{H \models P,\ H \models \text{good\_behavior\_of\_P}}{H \models \text{trusted\_program\_P}}$ (9.6)
$\dfrac{H \models \text{trusted\_program\_P},\ H \models \text{intact\_pathname\_of\_P}}{H \models \text{good\_invocation\_of\_P}}$ (9.7)
$\dfrac{H \models \text{good\_invocation\_of\_P},\ H \models \text{no\_known\_vulnerability\_on\_P}}{H \models \text{good\_for\_execution\_of\_P}}$ (9.8)
$\dfrac{H \models \text{good\_for\_execution\_of\_P},\ H \models \text{unaltered\_execution\_of\_P}}{H \models \text{good\_intended\_execution\_of\_P}}$ (9.9)

Table 9.2: Belief interactions and derivations in achieving the desired "good intended program execution" belief.

into an approach to evaluate the robustness of an IDS against attacks targeting the IDS. The result allows us to find out how computationally expensive it is to perform a search to craft attacks on the IDS. The approach taken thus shows the feasibility of obtaining quantitative measurements on the IDS resistance against targeted attacks.

Due to threat from mimicry attacks, we also have proposed an improved IDS model by employing an abstraction technique on process credentials and system call arguments. The proposed technique can be easily combined with other system-call based IDS models. To avoid increasing the false positives, a user-supplied specification is used to abstract the arguments and credentials. This specification takes into account security semantics of the operations and their potential effects to host security. Using sample programs, we have shown that the robustness of the IDS is increased when the abstraction is used. We also have some evidence that the increase in detection accuracy does not lead to more false positives. Unlike other data-flow IDS models [91, 123, 167, 23], our enhancement highlights the virtue of incorporating a host's *security model* into the IDS to result in a more concise IDS.

By monitoring parameterized system calls of a running process, our improved IDS provides an effective approximation mechanism to ensure *unaltered execution of program P* (belief 9.5). The monitoring can be done in a fast and online fashion, and also allows for a possible prevention of suspected system-call invocations(s). This establishment of belief 9.5 allows us to derive the desired belief of *good intended execution of program P* as shown in (9.9). That is, given a belief that *program P is good for execution* (i.e. P is a good trusted program and is vulnerability-free for execution) established by Steps 1–3 of the PPLC as shown in (9.8), and a belief on *unaltered execution of program P* (belief 9.5) ensured by the IDS, a host can establish the desired belief that the good-behaving program P runs as intended by its trusted developer.

Secondly, to secure program invocations on a host, we have demonstrated the practicality of a mandatory, in-kernel, pre-execution binary authentication mechanism in Windows. We have presented a prototype, BinAuth, which exemplifies a lightweight binary integrity scheme and its integration into an OS. Using BinAuth, we have shown how a mandatory in-kernel authentication system can ensure that a host only executes programs from trusted sources, which were also unmodified while in transit and stored on the host's file system, with acceptable performance overheads even on a complex OS such as Windows. Practical binary authentication system such as BinAuth thus serves as a crucial enabler for providing more secure program execution environments on commodity OSes. We have also compared BinAuth with a variety of authentication systems [82, 9, 183, 175, 29, 19, 61, 142, 187] using our developed analysis framework, which inspects key design factors for a binary authentication system.

Using BAN Logic notation, we can state that BinAuth (together with the required PKI certificate revocation service) ensures the beliefs 9.1–9.3 listed in Table 9.1. Note that in addition to its content, program P also carries with itself a belief statement that it is a "*good program*".[2] That is, P is non-malicious in nature, and has no intention to violate any security policies of the target hosts (beyond the program's known functionalities) or any acceptable use policies. As such, we can infer both (9.6) and (9.7) to establish a belief of *good invocation of (trusted) program P* on host H.

Thirdly, we have proposed a framework for a machine-oriented vulnerability database, which allows for an automated vulnerability checking on a host using published advisories. We have demonstrated a proof-of-concept database, which shows effective integration of data from multiple sources and can be used directly by a vulnerability scanner. A corresponding scanner prototype has also been developed for Unix/Linux systems, although the basic principles are applicable to other OSes.

Our work was developed and published in [160] when vulnerability representation and automated processing were still not widely addressed by the security community. Given the present on-going standardization efforts on vulnerability processing by the U.S. Government sponsored organizations, such as CVE [116], OVAL [117], CPE [115]

[2]This belief is accepted by host H when it decides to install P into its system at time $t_{installed}$.

and SCAP [126], we are pleased to see that our proposed mechanisms are in line with these efforts. The developed system for automated vulnerability processing allows a host to establish an important belief that *program P has no known vulnerability* (belief 9.4). This assurance is important to prevent possible known-vulnerability attacks during the program's unpatched time interval. The establishment of belief 9.4 makes it possible to infer that *program P is good for execution* (as shown in 9.8).

Next, to provide an efficient and secure PKI-based program distribution and interaction, we have addressed the need for a lightweight and practical near real-time certificate revocation scheme. We have proposed two lightweight, practical and inherently-distributed certificate revocation schemes, called CREV, which are based on the recently available Extended-Validation (EV) certificate infrastructure. Our CREV schemes enhance CRS/NOVOMODO [110, 111] and OCSP [124] to support efficient revocation with near real-time timeliness guarantees (1–10 minutes). We also have developed a more realistic cost analysis framework for evaluating the certificate revocation schemes. Our framework enhances the model in [99, 72] by incorporating a number of novel aspects to result in a more realistic cost analysis and to properly deal with fine-grained certificate generation and CRL issuance time intervals. Using the developed framework, we have demonstrated that CREV schemes keep the overheads manageable on all the involved entities, while maintaining lightweight requirements on the verifier even under a timeliness guarantee of 1 minute. The proposed schemes also have an additional advantage in that the status queries are protected from external parties including the CA.

CREV schemes thus provide a host with an efficient and timely mechanism to validate certificates. Hence, they support a binary authentication system like BinAuth to establish the required beliefs on a digitally signed program P (beliefs 9.1 and 9.2). Note that without a strong assurance on the validity of the certificates used in digitally signed programs, the whole security of a host executing the programs can be easily compromised.

Finally, we have extended BAN Logic [26, 27] and a prior extension work by Gaarder and Snekkenes [51] in order to make BAN Logic more concise in reasoning on PKI-based protocols. In particular, our extension redresses the reasoning on the goodness of private keys, and considers certificate revocation as a part of certificate verification. We also address common pitfalls in public-key based protocol design due to insufficient attention placed on the "intended recipient" and the "stated sender" of a message. Our extension makes the recipient and the sender explicit, and these requirements are incorporated into the logic. In this way, our logic reduces the likelihood of allowing such vulnerabilities in the protocol and the corresponding formalism. In addition, our logic also deals with cases of (public-key) signed encrypted messages and encrypted signed messages. We also have shown the usage and the sample application of our logic to PKI-based protocols, including a session establishment protocol proposed in CREV-I scheme.

Altogether, we have shown how we can safeguard the four PPLC steps with efficient security measures. The proposed measures provide strong additional layers of protection to securing program execution, while only incurring acceptable performance overheads. In view of their benefits to host security, our proposals thus address timely and pressing problem of the increasing malware threats to today's connected hosts.

9.2 Future Work

We have identified the following directions to be pursued in the future.

Resource access monitoring on a program

One weakness of the PPLC model is that a host needs to trust a software developer that a piece of software behaves in a good manner and will not violate the host's security policies. One way to deal with this limitation is by conducting *resource access monitoring* on a program execution as an additional step of the PPLC. The host can generate the resource-access profile of a program by observing the program's normal usage. Alternatively, the profile can be supplied by the developer based on the expected program's behavior. In this way, a host H can limit (sandbox) a program execution within a set of approved operations. As a result, host H can believe that not only a program P runs as intended (i.e. "$H \models \text{good\_intended\_execution\_of\_P}$"), but also within a specified capabilities in terms of the host's resources accessed (i.e. "$H \models \text{within\_access\_specification\_execution\_of\_P}$").

Richer IDS model and application to other OSes

For our work on anomaly detector IDS, we can combine our gray-box IDS model with the white-box techniques if the program's source code is available [95]. Alternatively, binary analysis techniques [154] can be used to derive a more concise IDS model. Binary instrumentation technique can also be used to "randomize" normal system-call profile generations on a host. This would make the attacker's approximated normal profile different from the valid (randomized) one, rendering the former much less effective.

With regard to our IDS attack-construction framework, we also intend to apply it to more IDS models. We also would like to implement the PAC-based IDS for other OSes, such as Windows. Additionally, the PAC-based IDS model can also be tailored more for specific applications, such as web applications [63, 39].

More usable and robust binary authentication system

We can extend the BinAuth system to protect not only binary files, but also important non-binary files. Additionally, it can be extended to deal with various usage scenarios that require special treatment of binary authentication, such as during software development, software installations and software updates.

BinAuth does prevent untrusted binaries from being executed. It however does not prevent illegal additions or modifications to protected binaries. One possible work is to make BinAuth also monitor illegal modifications on protected binaries. Such monitoring, however, needs to be justified with respect to the incurred overheads. Given the availability of a secure booting infrastructure, such as Trusted Platform Module (TPM), we can also try to combine BinAuth with TPM in order to ensure the integrity of the overall protection chain, starting from the hardware up to the applications. Cryprographic keys and important files used by BinAuth can also be securely stored by TPM.

Richer vulnerability description model and improved implementation

To ensure a vulnerability-free host system, we would like to enhance the Movtraq framework by devising a richer vulnerability description model. The model should capture a vulnerability in a more precise way, yet still be amenable for automated processing. Another direction worth exploring is to standardize vulnerability checks on a target machine for different popular OSes. Such checks could also be parameterized to make them as generic as possible. We would like to make these checks represented as database entries, with each entry specifying: the description of how the check should be performed on a variety of OSes, associated database fields for the values of objects involved in the checking, and symbolic description of the pre-conditions and consequence.

Our Movtraq vulnerability database uses SQL for data access and transfer. To be in line with recent standardization efforts such as OVAL, we can redesign the database format to use an XML-based data representation. Other possible enhancements on Movtraq implementation could include convenient GUIs, non Perl-based scanner, and compatibility with various popular OSes.

Analysis of certificate revocation schemes

In the analysis of certificate revocation schemes using our realistic framework, we can calculate the overheads of various other revocation schemes to be compared with the CREV schemes. We can also further improve the developed framework by allowing certificates to have different lifetimes. Lastly, we can also conduct a simulation to support the results derived from our analytical based framework.

Formal analysis on PKI-based protocols

In proposing MPKI-BAN Logic, we take an approach that focuses on pragmatism. We center on the logic definition and its usage application while leaving theoretical analysis of the logic, such as the logic's soundness and completeness, as a separate treatment beyond our work's scope. Developing a semantic model for MPKI-BAN is an important future work. Additionally, we can try to apply public-key constructs and rules in MPKI-BAN to other authentication logics, including non modal-logic based formal analysis techniques.

Appendix A

Sample Configuration for Privilege and Argument Categorization

```
#====== Sample Configuration File =======#
# EUID Abstraction Section
# Format:  <categorized-euid>:<euid1>,<euid2>,...
0:0
1:2000,2001,2003
2:3000
100:*
-----------------------------------------
# EGID Abstraction Section
# Format:  <categorized-egid>:<egid1>,<egid2>,...
0:0
1:1,2,3,5
100:*
-----------------------------------------
# Argument Abstraction Section
# Format:  <cat-value> <syscall> <arg1> <arg2> <arg3> ...
1 open p=/etc/passwd o=O_WRONLY|o=O_RDRW *
2 open p=/etc/shadow o=O_WRONLY|o=O_RDRW *
3 open p=/etc/group o=O_WRONLY|o=O_RDRW *
4 open p=/proc/kmem o=O_WRONLY|o=O_RDRW *
5 open p=/etc/hosts.equiv o=O_WRONLY|o=O_RDRW *
6 open p=/etc/* o=O_WRONLY|o=O_RDRW *
7 open p=/etc/* o=O_RDONLY *
8 open p=/bin/* o=O_WRONLY|o=O_RDRW *
9 open p=/sbin/* o=O_WRONLY|o=O_RDRW *
10 open p=/boot/* o=O_WRONLY|o=O_RDRW *
11 open p=/dev/* o=O_WRONLY|o=O_RDRW *
12 open p=/usr/* o=O_WRONLY|o=O_RDRW *
13 open p=/lib/* o=O_WRONLY|o=O_RDRW *
14 open p=/var/* o=O_WRONLY|o=O_RDRW *
15 open p=/mnt/* o=O_WRONLY|o=O_RDRW *
16 open p=/proc/* o=O_WRONLY|o=O_RDRW *
17 open * o=O_WRONLY|o=O_RDRW *
18 open * * *
```

```
1 chmod p=/etc/{passwd,shadow,group,hosts.equiv}|p=/proc/kmem *
2 chmod * *
1 fchmod p=/etc/{passwd,shadow,group,hosts.equiv}|p=/proc/kmem *
2 fchmod * *
1 chown p=/etc/{passwd,shadow,group,hosts.equiv}|p=/proc/kmem *
2 chown * * *
1 fchown p=/etc/{passwd,shadow,group,hosts.equiv}|p=/proc/kmem *
2 fchown * * *
1 lchown p=/etc/{passwd,shadow,group,hosts.equiv}|p=/proc/kmem *
2 lchown * * *
1 rename p=/etc/{passwd,shadow,group} *
2 rename p=/proc/kmem *
3 rename p=/etc/hosts.equiv *
4 rename * p=/etc/{passwd,shadow,group}
5 rename * p=/proc/kmem
6 rename * p=/etc/hosts.equiv
7 rename * *
1 link p=/etc/{passwd,shadow,group} *
2 link p=/proc/kmem *
3 link p=/etc/hosts.equiv *
4 link * p=/etc/{passwd,shadow,group}
5 link * p=/proc/kmem
6 link * p=/etc/hosts.equiv
7 link * *
1 unlink p=/etc/{passwd,shadow,group,hosts.equiv}|p=/proc/kmem
2 unlink *
1 symlink p=/etc/{passwd,shadow,group} *
2 symlink p=/proc/kmem *
3 symlink p=/etc/hosts.equiv *
4 symlink * p=/etc/{passwd,shadow,group}
5 symlink * p=/proc/kmem
6 symlink * p=/etc/hosts.equiv
7 symlink * *
1 mount * p=/etc * * *
2 mount * p=/{bin,sbin} * * *
3 mount * p=/boot * * *
4 mount * p=/usr * * *
5 mount * p=/lib * * *
6 mount * p=/proc * * *
7 mount * * * * *
1 mknod p=/etc/{passwd,shadow,group,hosts.equiv}|p=/proc/kmem * *
2 mknod * * *
1 init_module * * *
1 execve p=/bin/{sh,csh,ksh} * *
2 execve * * *
-----------------------------------------
# Illegal Transition Section
# Format:  <syscall> <cat-values> [<cat-euid>,<cat-egid>]*
open [1..6,8-11,13-15] 0,* *,0
{chmod,fchmod,chown,fchown,lchown,mknod,unlink,init_module,execve} 1 0,* *,0
{rename,link,symlink,mount} [1..6] 0,* *,0
-----------------------------------------
```

Appendix B

Database Entities in Movtraq Vulnerability Database

Movtraq vulnerability database (see Section 6.3) defines seven entities (tables) to store the information of a vulnerability entry. Their key fields, as seen from an integration and machine processing perspective, are listed below. The fields labeled by '*' make use of the vulnerability description expressions described in Section 6.4. Those labeled by '†' can use the $Software\_ID$ (see Section 5.6) or the CPE [115] format.

Vulnerability Entity — records the main information about each vulnerability: Vul_ID, Vulnerability name, CVE_ID, other corresponding IDs (e.g. CERT_ID, BugTraq_ID), a textual description for the vulnerability, and fields relating to Component Specification Entity, Environment Specifications Entity, and Exploit Entity.

Component Specification Entity — records the information of a component factor: Comp_ID, vulnerability consequences*, fields relating to Operating System Entity, Application Entity, and Service Entity.

Operating System Entity — records a component factor originating from the OS: OS_ID, OS name, vulnerable versions†, hardware type information†.

Application Entity — records a component factor due to an application: application_ID, name of application, vulnerable versions†, hardware type information†.

Service Entity — records a component factor due to a service (a service could be a daemon): service_ID, name of service, vulnerable versions†, hardware type information†, protocols, and port numbers.

Environment Specifications Entity — records the information of environmental factors affecting a vulnerability: Env_ID, existence of required user, application, service, and file object*, vulnerable kernel versions of the running OS†, remote exploitation flag.

Exploit Entity — records the details of any available exploits: Exp_ID, exploit information (URL, filename, description, etc.), privileges needed to execute the exploit*, consequences of using the exploit*.

Note that some fields sharing a similar function can occur a few times in different contexts. *Consequences* field, for example, occurs both in Component Specification Entity and Exploit Entity. The former describes the consequences of the specified vulnerability, whereas the latter lists those from using a specific available exploit.

Appendix C

Relevant Rules of BAN Logic

Below are the rules of the original BAN Logic [26, 27] which are relevant to our MPKI-BAN extension explained in Chapter 8. We follow the notation and description of BAN Logic as in [27].

- (R_1) **Message-meaning (for secret-key encryption) Rule**:
  (If P believes that the key K_{PQ} is shared with Q, and sees a message X encrypted under K_{PQ}, then P believes that Q once said X.)

$$\frac{P \models Q \overset{K_{PQ}}{\longleftrightarrow} P, P \triangleleft \{X\}_{K_{PQ}}}{P \models Q \mid\!\sim X}$$

- (R_2) **Nonce verification Rule**:
  (This checks that a message is recent, and hence that the sender still believes in it.)

$$\frac{P \models \sharp(X), P \models Q \mid\!\sim X}{P \models Q \models X}$$

- (R_3) **Jurisdiction Rule**:
  (If P believes that Q has jurisdiction over X, then P trusts Q on the truth of X.)

$$\frac{P \models Q \Rightarrow X, P \models Q \models X}{P \models X}$$

- (R_4) **And-introduction Rule**:
  (If P believes each individual statement separately, then P believes a set of statements.)

$$\frac{P \models X, P \models Y}{P \models (X, Y)}$$

- (R_5) **And-elimination Rule**:
  (If P believes a set of statements, then P believes each individual statement separately.)

$$\frac{P \models (X, Y)}{P \models (X)}$$

- (R_6) **Believe and-elimination Rule**:
  (And-elimination Rule that applies to a belief held by other principal.)

$$\frac{P \mid\equiv Q \mid\equiv (X, Y)}{P \mid\equiv Q \mid\equiv X}$$

- (R_7) **Said and-elimination Rule**:
  (And-elimination Rule that applies to the $\mid\sim$ operator. Note that And-introduction Rule does not apply to $\mid\sim$.)

$$\frac{P \mid\equiv Q \mid\sim (X, Y)}{P \mid\equiv Q \mid\sim X}$$

- (R_8) **See components Rule**:
  (If P sees a formula, then P also sees its components, provided he knows the necessary keys.)

$$\frac{P \triangleleft (X, Y)}{P \triangleleft X}$$

- (R_9) **See encrypted message (for secret-key) Rule**:
  (If P sees an message encrypted with a secret key K_{PQ}, and that he knows the key K_{PQ}, then P also sees its message.)

$$\frac{P \mid\equiv Q \overset{K_{PQ}}{\longleftrightarrow} P, P \triangleleft \{X\}_{K_{PQ}}}{P \triangleleft X}$$

- (R_{10}) **Freshness extension Rule**:
  (If one part of a formula is known to be fresh, then the entire formula is also fresh.)

$$\frac{P \mid\equiv \sharp(X)}{P \mid\equiv \sharp(X, Y)}$$

- (R_{11}) **Key-symmetry (for secret-key) Rule**:
  (A secret key is used between a pair of principals in either direction.)

$$\frac{P \mid\equiv R \overset{K_{RR'}}{\longleftrightarrow} R'}{P \mid\equiv R' \overset{K_{R'R}}{\longleftrightarrow} R}$$

- (R_{12}) **Believe key-symmetry (for secret-key) Rule**:
  (Key-symmetry Rule that applies to a belief held by other principal.)

$$\frac{P \mid\equiv Q \mid\equiv R \overset{K_{RR'}}{\longleftrightarrow} R'}{P \mid\equiv Q \mid\equiv R' \overset{K_{R'R}}{\longleftrightarrow} R}$$

Appendix D

New Rules of MPKI-BAN Logic

Below are the newly defined rules in our MPKI-BAN Logic elaborated in Chapter 8.

- (R_{13}) **New message-meaning for (private-key) signed message Rule**:

$$\frac{P \models \wp\kappa\,(Q, K_Q)\,, P \models \Pi(K_Q^{-1}), P \triangleleft \sigma(\Re(X, P), K_Q^{-1})}{P \models Q \mid\!\sim X}$$

- (R_{14}) **All-recipient see Rule**:

$$\frac{P \triangleleft \sigma(\Re(X, all), K_Q^{-1})}{P \triangleleft \sigma(\Re(X, P), K_Q^{-1})}$$

- (R_{15}) **New certificate-validation Rule**:

$$\frac{P \models Q \mid\!\sim \Theta'(t_1^R, t_2^R, Q, C^R), P \models Q \models \Delta(t_1^R, t_2^R), P \models Q \models \Phi(C^R)}{P \models Q \models C^R}$$

  with $C^R = \wp\kappa\,(P, K_P)\,, \Pi(K_P^{-1})$.

- (R_{16}) **Duration-stamp (without revocation) validation Rule**:

$$\frac{P \models Q \mid\!\sim \Theta(t_1, t_2, X), P \models Q \models \Delta(t_1, t_2)}{P \models Q \models X}$$

- (R_{17}) **New message-meaning for (public-key) encrypted message Rule**:

$$\frac{P \models \wp\kappa\,(P, K_P)\,, P \models \Pi(K_P^{-1}), P \triangleleft \{\mathcal{S}(X, Q)\}_{K_P}}{P \models Q \mid\!\sim X}$$

- (R_{18}) **Message-meaning for (private-key) signed message –after decryption– Rule**:

$$\frac{P \models \wp\kappa\,(Q, K_Q)\,, P \models \Pi(K_Q^{-1}), P \mid\!\sim \sigma(\Re(X, P), K_Q^{-1})}{P \models Q \mid\!\sim X}$$

- (R_{19}) **Message-meaning for encrypted message portion –after signature verification, with encrypted stated-sender and included freshness assurance– Rule**:

$$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \mid\equiv Q \mid\sim \{\mathcal{S}(X,Q)\}_{K_P}}{P \mid\equiv Q \mid\sim X}$$

- (R_{20}) **Message-meaning for encrypted message portion –after signature verification, with encrypted stated-sender and external freshness assurance– Rule**:

$$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \mid\equiv Q \mid\sim \{\mathcal{S}(X,Q)\}_{K_P}, Y}{P \mid\equiv Q \mid\sim X, Y}$$

- (R_{21}) **Message-meaning for encrypted message portion –after signature verification, with clear stated-sender and included freshness assurance– Rule**:

$$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \mid\equiv Q \mid\sim \mathcal{S}(\{X\}_{K_P}, Q)}{P \mid\equiv Q \mid\sim X}$$

- (R_{22}) **Message-meaning for encrypted message portion –after signature verification, with clear stated-sender and external freshness assurance– Rule**:

$$\frac{P \mid\equiv \wp\kappa\,(P, K_P)\,, P \mid\equiv \Pi(K_P^{-1}), P \mid\equiv Q \mid\sim \mathcal{S}(\{X\}_{K_P}, Q), Y}{P \mid\equiv Q \mid\sim X, Y}$$

- (R_{23}) **Message meaning for (secret-key) hashed message Rule**:

$$\frac{P \mid\equiv Q \overset{K_{PQ}}{\longleftrightarrow} P, P \triangleleft \mu(X, K_{PQ})}{P \mid\equiv Q \mid\sim X}$$

- (R_{24}) **See keyed-hashed message Rule**:

$$\frac{P \triangleleft \mu(X, K_{PQ})}{P \triangleleft X}$$

- (R_{25}) **See signed message Rule**:

$$\frac{P \triangleleft \sigma(X, K_Q^{-1})}{P \triangleleft X}$$

- (R_{26}) **See recipient-tagged message Rule**:

$$\frac{P \triangleleft \Re(X, P)}{P \triangleleft X}$$

Appendix E

Sample Application of MPKI-BAN Logic

This appendix shows how we can apply MPKI-BAN Logic (elaborated in Chapter 8) to formally analyze a PKI-based protocol. We have chosen the *3-way Session-Establishment* protocol of CREV-I outlined in Section 7.3.3. The protocol is chosen here so that we can establish the security assurance of the proposed session establishment of a hash-chaining based revocation service.

For ease of reference, we show the protocol again below (only the message transfers are shown). Recall that the notation $\langle M \rangle_{K_A^{-1}}$ denotes a message M which is signed using the private key of principal A (i.e. K_A^{-1}); and $nonce_X$ denotes a nonce from X.

1. $EVCP \rightarrow CA$: "*Session Request*"$= \langle SessReq, EVCP\_ID, CA\_ID, Serial\_No, T,$
   $nonce_{EVCP}, SigAlgID \rangle_{K_{EVCP}^{-1}}$.

   where:

   $SessReq$ = header indicating a Session Request message;

   $EVCP\_ID$ = identity (i.e. domain name) of the EVCP;

   CA_ID = identity (i.e. name) of the CA;

   $Serial\_No$ = serial number of the EVC;

   T = timestamp or a CA's nonce accessible at URI_{CA_nonce};

   $SigAlgID$ = identification for the signing algorithm.

2. $CA \rightarrow EVCP$: "*Session Reply*"$= \langle SessReply, ReplyStatus, CA\_ID, EVCP\_ID,$
   $nonce_{EVCP}, Serial\_No, CertStatus, HashAlgID, d, Y, N, SessStart,$
   $SessExpiry, nonce_{CA}, SigAlgID \rangle_{K_{CA}^{-1}}$.

   where:

   $SessReply$ = header indicating a Session Reply message;

   $ReplyStatus$ = status indicator for a successful session establishment;

   $CertStatus$ = status of the EVC;

   $HashAlgID$, d, Y, N = hash chain parameters (see Section 7.2.2);

   $SessStart$ and $SessExpiry$ = the start and end times of the established session.

3. $EVCP \rightarrow CA$: "*Session ACK*"$= \langle SessACK,\ EVCP\_ID,\ CA\_ID,\ nonce_{CA},\ Serial\_No,$
$SigAlgID\rangle_{K^{-1}_{EVCP}}$.

where:

$SessACK$ = header indicating a Session ACK message.

E.1 Idealized Protocol

We can idealize the messages in the protocol above as follows.

1. $EVCP \rightarrow CA$: $\sigma(\Re\ ((Sess\_Req, Serial\_No, T, nonce_{EVCP}), CA)\ , K^{-1}_{EVCP})$

2. $CA \rightarrow EVCP$: $\sigma\big(\Re\,((Sess\_Reply, nonce_{EVCP}, Hash\_Chain, nonce_{CA}), EVCP), K^{-1}_{CA}\big)$
with: $Hash\_Chain = (ReplytStatus,\ Serial\_No,\ CertStatus,\ SessStart,\ SessExpiry,$
$HashAlgID,\ d,\ Y,\ N)$

3. $EVCP \rightarrow CA$: $\sigma(\Re\ ((Sess\_ACK, nonce\_CA, EVCP \models Hash\_Chain), CA)\ , K^{-1}_{EVCP})$

The message-recipient construct ($\Re(X, receiver)$) is idealized in all messages above. This is because each message always contains the ID of the receiver, which is also taken into consideration in generating the respective message's digital signature.

We specifically define $Hash\_Chain$ as the message portion containing all the parameters specifying a hash-chaining based (e.g. CRS/NOVOMODO) revocation system. $Hash\_Chain$ is to be used by the verifiers as reference token information, together with the CA's latest hash token, in ensuring the validity of the EVCP's certificate.

Note that the protocol above is between an EVCP and the CA. Thus, unlike a typical protocol where the principals establish a trust on each other through a (common) CA, here it is the CA with whom the EVCP deals directly. Since all principals are assumed to have believed the CA's public and private keys, we do not logically infer the following EVCP's beliefs: $EVCP \models \wp\kappa\,(CA, K_{CA})$ and $EVCP \models \Pi(K^{-1}_{CA})$. Rather, they are assumed to be true at the start of the protocol run. Conversely, since the CA is the authoritative entity on the goodness of the EVCP's public and private key pair, the CA determines by itself the goodness of the keys (i.e. $CA \models \wp\kappa\,(EVCP, K_{EVCP})$ and $CA \models \Pi(K^{-1}_{EVCP})$).

E.2 Initial-State Assumptions

We list all the assumptions below:

- EVCP believes correct and good public and private keys of CA:
  α_1: $EVCP \models \wp\kappa\,(CA, K_{CA})$
  α_2: $EVCP \models \Pi(K^{-1}_{CA})$
- CA believes the goodness of EVCP's public and private keys (based on its certification record):
  α_3: $CA \models \wp\kappa\,(EVCP, K_{EVCP})$
  α_4: $CA \models \Pi(K^{-1}_{EVCP})$

- Each principal believes the freshness of its own nonce(s) or timestamp:
  α_5: $CA \models \sharp(T)$
  α_6: $EVCP \models \sharp(nonce_{EVCP})$
  α_7: $CA \models \sharp(nonce_{CA})$
- CA believes that EVCP has a jurisdiction over the Session Request message portion:
  α_8: $CA \models EVCP \Rrightarrow SessReq, Serial\_No, nonce_{EVCP}$
- EVCP believes that CA has a jurisdiction over $SessReply$ and $Hash\_Chain$:
  α_9: $EVCP \models CA \Rrightarrow SessReply, Hash\_Chain$
- CA believes that EVCP has a jurisdiction over $SessACK$ and EVCP's belief on $Hash\_Chain$:
  α_{10}: $CA \models EVCP \Rrightarrow SessACK, (EVCP \models Hash\_Chain)$

E.3 Protocol Goals

The CREV-I session establishment protocol does not aim to establish a session key to secure the subsequent communication session. Rather, it aims to establish a mutual authentication between the CA and EVCP, and to allow EVCP to securely obtain the CA's $Hash\_Chain$. In addition, the CA needs to establish a belief that EVCP has believed its generated $Hash\_Chain$.

We list the goals as follows:

- CA believes $SessReq, Serial\_No, nonce_{EVCP}$ (sent in EVCP's Session Request message):
  G_1: $CA \models SessReq, Serial\_No, nonce_{EVCP}$
- EVCP believes $SessReply, Hash\_Chain, nonce_{CA}$ (sent in CA's Session Reply message):
  G_2: $EVCP \models SessReply, Hash\_Chain, nonce_{CA}$
- CA believes $SessACK$ together with EVCP's belief on $Hash\_Chain$ (sent in EVCP's Session ACK message):
  G_3: $CA \models SessACK, (EVC \models Hash\_Chain)$

E.4 The Proof

We prove that the proposed protocol achieve its stated goals as follows. Note that the rule numbering here follows that of Appendix D.
After **message 1**, we have:

$$CA \triangleleft\ \sigma(\,\Re\,((SessReq, Serial\_No, T, nonce_{EVCP}), CA)\,, K^{-1}_{EVCP}) \tag{E.1}$$

Using New message-meaning for signed message Rule (R_{13}) on α_3, α_4, and (E.1):

$$CA \models EVCP \mid\sim SessReq, Serial\_No, T, nonce_{EVCP} \tag{E.2}$$

Using Freshness extension Rule (R_{10}) on α_5:

$$CA \models \sharp(SessReq, Serial\_No, T, nonce_{EVCP}) \tag{E.3}$$

Using Nonce verification Rule (R_2) on (E.3) and (E.2):

$$CA \models EVCP \models SessReq, Serial\_No, T, nonce_{EVCP} \tag{E.4}$$

Using And-elimination Rule (R_5) on (E.4):

$$CA \mid\equiv EVCP \mid\equiv SessReq, Serial\_No, nonce_{EVCP} \tag{E.5}$$

Using Jurisdiction Rule (R_3) on α_8 and (E.5):

$$\boxed{G_1 : CA \mid\equiv SessReq, Serial\_No, nonce_{EVCP}} \tag{E.6}$$

After **message 2**, we have:

$$EVCP \triangleleft\ \sigma(\Re((SessReply, nonce_{EVCP}, Hash\_Chain, nonce_{CA}), EVCP), K_{CA}^{-1}) \tag{E.7}$$

Using New message-meaning for signed message Rule (R_{13}) on α_1, α_2, and (E.7):

$$EVCP \mid\equiv CA \mid\sim SessReply, nonce_{EVCP}, Hash\_Chain, nonce_{CA} \tag{E.8}$$

Using Freshness extension Rule (R_{10}) on α_6:

$$EVCP \mid\equiv \sharp(SessReply, nonce_{EVCP}, Hash\_Chain, nonce_{CA}) \tag{E.9}$$

Using Nonce verification Rule (R_2) on (E.9) and (E.8):

$$EVCP \mid\equiv CA \mid\equiv SessReply, nonce_{EVCP}, Hash\_Chain, nonce_{CA} \tag{E.10}$$

Using And-elimination Rule (R_5) on (E.10):

$$EVCP \mid\equiv CA \mid\equiv SessReply, Hash\_Chain, nonce_{CA} \tag{E.11}$$

Using Jurisdiction Rule (R_3) on α_9 and (E.11):

$$\boxed{G_2 : EVCP \mid\equiv SessReply, Hash\_Chain, nonce_{CA}} \tag{E.12}$$

After **message 3**, we have:

$$CA \triangleleft \sigma(\Re\ (SessACK, nonce_{CA}, (EVCP \mid\equiv Hash\_Chain), CA), K_{EVCP}^{-1}) \tag{E.13}$$

Using New message-meaning for signed message Rule (R_{13}) on α_3, α_4, and (E.13):

$$CA \mid\equiv EVCP \mid\sim SessACK, nonce_{CA}, (EVCP \mid\equiv Hash\_Chain) \tag{E.14}$$

Using Freshness extension Rule (R_{10}) on α_7:

$$CA \mid\equiv \sharp(SessACK, nonce_{CA}, EVCP \mid\equiv Hash\_Chain) \tag{E.15}$$

Using Nonce verification Rule (R_2) on (E.15) and (E.14):

$$CA \mid\equiv EVCP \mid\equiv SessACK, nonce_{CA}, (EVCP \mid\equiv Hash\_Chain) \tag{E.16}$$

Using And-elimination Rule (R_5) on (E.16):

$$CA \mid\equiv EVCP \mid\equiv SessACK, (EVCP \mid\equiv Hash\_Chain) \tag{E.17}$$

Using Jurisdiction Rule (R_3) on α_{10} and (E.17):

$$\boxed{G_3 : CA \mid\equiv SessACK, (EVCP \mid\equiv Hash\_Chain)} \tag{E.18}$$

E.5 Discussion

We have given a proof of the proposed session establishment protocol in CREV-I by using MPKI-BAN Logic. Some interesting points to note from the proof are:

- We can see that the three stated goals (G_1–G_3) are all achievable. Using the protocol, EVCP and the CA can authenticate each other in order to set up a new session. In particular, EVCP can derive a belief on $Hash\_Chain$ which is sent together with CA's nonce in a valid Session Reply message (i.e. $G_2 : SessReply, (EVCP \models Hash\_Chain), nonce_{CA})$. Moreover, the CA can also establish a belief that EVCP believes $Hash\_Chain$, which is sent in a valid Session ACK message (i.e. $G_3 : CA \models SessACK, (EVCP \models Hash\_Chain))$. Capturing these facts is important because only after establishing these beliefs, the CA then starts sending its periodical hash tokens to EVCP.

- Although the CA believes that EVCP believes $Hash\_Chain$ (i.e. G_3), EVCP however has no knowledge that the CA has established G_3. In other words, the following belief is not derivable: $EVCP \models CA \models (EVCP \models Hash\_Chain)$. When proposing the protocol, we assume that the channel between EVCP and the CA is reliable. Hence, EVCP simply assumes that the CA can derive G_3 upon receipt of the message 3. In a lossy channel, mechanisms to deal with possible message losses are thus required.

- It is possible to omit the use of CA's nonce ($nonce_{CA}$) in Session Reply and Session ACK messages. Instead one may chose to use $SessStart$ (representing the timestamp where the established session starts to be valid) as the freshness assurance. That is, $SessStart$, which is generated and sent in the message 2 by the CA, functions as a nonce to be returned in the message 3 by EVCP. The CA however now needs to keep track of the latest Session ACK message that it sends to each EVCP. This is required in order to ensure that $Serial\_No+SessStart$ is unique. The use of $SessStart$ as a CA's nonce can work provided that $SessStart$ timestamp is sufficiently fine-grained. Otherwise, the nonce space is rather limited, and may result in a potential "oracle attack" on the generation of the CA's (signed) Session Reply message. Our use of $nonce_{CA}$ functions as a challenge to EVCP, as well as acting as a "salt" to increase the message space of the Session Reply message so as to reduce the risk of an oracle attack.

- Our formalism using MPKI-BAN Logic on the protocol thus demonstrates the Logic's benefits in highlighting the implicit assumptions of the protocol as well as formulating the desired protocol goals. The Logic can subsequently help protocol designers to reason more systematically on the protocol by showing how the protocol can establish its objectives.

Appendix F

List of Author's Published and Submitted Work

The following are published and submitted works by the author during the author's Ph.D. candidature.

Published Works:

- Sufatrio, Roland H. C. Yap and Liming Zhong, "A Machine-Oriented Integrated Vulnerability Database for Automated Vulnerability Detection and Processing", In *Proceedings of the 18th USENIX Large Installation System Administration*, pp. 47–58, 2004.
- Sufatrio and Roland H. C. Yap, "Improving Host-based IDS with Argument Abstraction to Prevent Mimicry Attacks", In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID).* pp. 146–164, 2005.
- Rajiv Ramnath, Sufatrio, Roland H. C. Yap, and Wu Yongzheng, "WinResMon: A Tool for Discovering Software Dependencies, Configuration, and Requirements in Microsoft Windows", In *Proceedings of the 20th USENIX Large Installation System Administration*, pp. 175-186, 2006.
- Felix Halim, Rajiv Ramnath, Sufatrio, Yongzheng Wu, and Roland H. C. Yap, "A Lightweight Binary Authentication System for Windows", In *Proceedings of the Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM)*, pp. 295–310, Springer, 2008.
- Sufatrio and Roland H. C. Yap, "Extending BAN Logic for Reasoning with Modern PKI-based Protocols", In *Proceedings of the IFIP International Workshop on Network and System Security 2008 (NSS)*, pp. 190–197, 2008.
- Yongzheng Wu, Sufatrio, Roland H. C. Yap, Rajiv Ramnath and Felix Halim, "Establishing Software Integrity Trust: A Survey and Lightweight Authentication System for Windows, Book Chapter, in *Trust Modeling and Management in Digital Environments: From Social Concept to System Development*, Information Science Reference, 2010.

Submitted Work:

- Sufatrio and Roland H. C. Yap, "Practical and Near Real-Time Certificate Revocation", 2010.

Bibliography

[1] Abadi, M., and Needham, R. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering 22*, 1 (1996), 6–15.

[2] Abadi, M., and Tuttle, M. R. A semantics for a logic of authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC)* (1991), pp. 201–216.

[3] Adams, C., Cain, P., Pinkas, D., and Zuccherato, R. *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP).* IETF RFC 3161, 2001.

[4] Adams, C., and Lloyd, S. *Understanding PKI: Concepts, Standards, and Deployment Considerations*, 2nd ed. Addison-Wesley Professional, 2002.

[5] Agray, N., van Der Hoek, W., and de Vink, E. On BAN Logics for industrial security protocols. In *Proceedings of the 2nd International Workshop of Central and Eastern Europe on Multi-Agent Systems* (2002), pp. 29–36.

[6] Aiello, W., Lodha, S., and Ostrovsky, R. Fast digital identity revocation. In *Proceedings of the 18th Annual International Cryptology Conference (CRYPTO)* (1998), pp. 137–152.

[7] Anderson, R., and Needham, R. Robustness principles for public key protocols. In *Proceedings of the 15th Annual International Cryptology Conference (CRYPTO)* (1995), pp. 236–247.

[8] Anti Phishing Working Group (APWG). Phishing activity trends report 2nd half 2008. Retrieved on October 22, 2010, from `http://www.antiphishing.org/reports/apwg_report_H2_2008.pdf`, 2009.

[9] Apvrille, A., Gordon, D., Hallyn, S., Pourzandi, M., and Roy, V. Digsig: Runtime authentication of binaries at kernel level. In *Proceedings of the 18th USENIX Large Installation System Administration Conference* (2004), pp. 59–66.

[10] Arbaugh, W. A. *Chaining Layered Integrity Checks.* PhD thesis, University of Pennsylvania, 1999.

[11] Arboi, M. The NASL2 reference manual. Retrieved on October 22, 2010, from `http://www.nessus.org/doc/nasl2_reference.pdf`.

[12] Arnold, E. R. The trouble with Tripwire. Retrieved on October 22, 2010, from `http://www.securityfocus.com/infocus/1398`, 2001.

[13] Axelsson, S. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security 3*, 3 (2000), 186–205.

[14] Axelsson, S. Intrusion detection systems: A taxomomy and survey. Tech. Rep. TR 99-15, Chalmers University of Technology, 2000.

[15] Aziz, A., and Diffie, W. Privacy and authentication for wireless local area networks. *IEEE Personal Communication 1*, 1 (1994), 25–31.

[16] Bace, R., and Mell, P. Intrusion Detection Systems. Tech. Rep. Special Publication on Intrusion Detection Systems, National Institute of Standards and Technology (NIST), 2001.

[17] Baldwin, R. W. Rule based analysis of computer security. Tech. Rep. MIT/LCS/TR-401, Massachusetts Institute of Technology, 1988.

[18] Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer, and Communication Security (ASIACCS)* (2006), pp. 16–25.

[19] Beattie, S., Black, A., Cowan, C., Pu, C., and Yang, L. *CryptoMark: Locking the stable door ahead of the trojan horse.* White Paper. WireX Communications Inc., 2000.

[20] Bellovin, S. M. Computer security—an end state? *Communications of the ACM 44*, 3 (2001), 131–132.

[21] Berbecaru, D. MBS-OCSP: An OCSP based certificate revocation system for wireless environments. In *Proceedings of the 4th IEEE International Symposium on Signal Processing and Information Technology* (2004), pp. 267–272.

[22] Bernaschi, M., Gabrielli, E., and Mancini, L. V. REMUS: A security-enhanced operating system. *ACM Transactions on Information and System Security 5*, 1 (2002), 36–61.

[23] Bhatkar, S., Chaturvedi, A., and Sekar, R. Dataflow anomaly detection. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy* (2006), pp. 48–62.

[24] Bicakci, K., and Baykal, N. One-time passwords: Security analysis using BAN Logic and integrating with smartcard authentication. In *Proceedings of the 18th International Symposium on Computer and Information Sciences* (2003), pp. 794–801.

[25] Boyd, C., and Mathuria, A. Key establishment protocols for secure mobile communications: A selective survey. In *Proceedings of the 3rd Australasian Conference on Information Security and Privacy (ACISP)* (1998), pp. 344–355.

[26] Burrows, M., Abadi, M., and Needham, R. A logic of authentication. *Proceedings of the Royal Society 426*, 1871 (1989).

[27] Burrows, M., Abadi, M., and Needham, R. A logic of authentication, revised. Tech. Rep. SRC Technical Report 39, Digital Systems Research Centre, 1990.

[28] CA/Browser Forum. Guidelines for the issuance and management of Extended Validation Certificates, version 1.2. Retrieved on October 22, 2010, from http://www.cabforum.org/Guidelines_v1_2.pdf, 2009.

[29] Catuogno, L., and Visconti, I. An architecture for kernel-level verification of executables at run time. *The Computer Journal 47*, 5 (2004), 511–526.

[30] CERT Coordination Center. CERT statistics (historical): Cataloged vulnerabilities. Retrieved on October 22, 2010, from http://www.cert.org/stats/cert_stats.html.

[31] CERT Coordination Center. CERT/CC overview incident and vulnerability trends. Retrieved on October 22, 2010, from ftp://ftp.upc.es/pub/cert/cert_advisories/www.cert.org/present/cert-overview-trends/module-2.pdf.

[32] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM Computing Surveys 41*, 3 (2009), 1–58.

[33] Chang, C., Pan, H., and Jia, H. A secure short message communication protocol. *International Journal of Automation and Computing 5*, 2 (2008), 202–207.

[34] Cheminod, M., Bertolotti, I., Durante, L., Maggi, P., Pozza, D., Sisto, R., and Valenzano, A. Detecting chains of vulnerabilities in industrial networks. *IEEE Transactions on Industrial Informatics 5*, 2 (2009), 181–193.

[35] Chen, L., Zhang, G., and Li, X. Efficient identity authentication protocol and its formal analysis. In *Proceedings of the 2007 International Conference on Computational Intelligence and Security Workshops* (2007), pp. 712–716.

[36] Chen, S., Xu, J., Sezer, E. C., Gauriar, P., and Iyer, R. K. Non-control-data attacks are realistic threats. In *Proceedings of the 14th USENIX Security Symposium* (2005), pp. 177–192.

[37] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and Polk, W. *Internet X.509 Public Key Infrastructure certificate and Certificate Revocation List (CRL) profile.* IETF RFC 5280, 2008.

[38] Cooper, D. A. A closer look at revocation and key compromise in Public Key Infrastructures. In *Proceedings of the 21st National Information Systems Security Conference* (1998), pp. 555–565.

[39] Criscione, C., and Zanero, S. Masibty: An anomaly based intrusion prevention system for Web applications. In *Proceedings of the 2009 Black Hat Europe* (2009).

[40] Debar, H., and Viinikka, J. Intrusion detection: Introduction to intrusion detection and security information management. In *Foundations of Security Analysis and Design III (FOSAD 2004/2005)* (2005), pp. 207–236.

[41] Deraison, R., and Gula, R. Blended security assessments: Combining active, passive and host assessment techniques. Tech. rep., Tenable Security, 2009.

[42] Dierks, T., and Rescorla, E. *The Transport Layer Security (TLS) protocol version 1.2.* IETF RFC 5246, 2008.

[43] Digistamp, Inc. Frequently asked questions – digital signatures. Retrieved on October 22, 2010, from `http://www.digistamp.com/FAQsig.htm#tsSig`.

[44] Eastlake, D., and Hansen, T. *US Secure Hash Algorithms (SHA and HMAC-SHA).* IETF RFC 4634, 2006.

[45] F-Secure. F-Secure reports amount of malware grew by 100% during 2007. Retrieved on October 22, 2010, from `http://www.f-secure.com/en_EMEA/about-us/pressroom/news/2007/fs_news_20071204_1_eng.html`, 2007.

[46] Farmer, D., and Spafford, E. H. The COPS security checker system. In *Proceedings of the Summer 1990 USENIX Conference* (1990), pp. 165–170.

[47] Feng, H. H., Giffin, J. T., Huang, Y., Jha, S., Lee, W., and Miller, B. P. Formalizing sensitivity in static analysis for intrusion detection. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy* (2004), pp. 194–208.

[48] Feng, H. H., Kolesnikov, O. M., Fogla, P., Lee, W., and Gong, W. Anomaly detection using call stack information. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy* (2003), pp. 62–75.

[49] Foreman, P. *Vulnerability Management.* CRC Press, 2010.

[50] Forrest, S., Hofmeyr, S., and Somayaji, A. The evolution of system-call monitoring. In *Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC)* (2008), pp. 418–430.

[51] Gaarder, K., and Snekkenes, E. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology 3*, 2 (1991), 81–98.

[52] Gabrilovich, E., and Gontmakher, A. The homograph attack. *Communications of the ACM 45*, 2 (2002), 128–128.

[53] Gao, D., Reiter, M. K., and Song, D. On gray-box program tracking for anomaly detection. In *Proceedings of the 13th USENIX Security Symposium* (2004), pp. 103–118.

[54] Garfinkel, S., and Spafford, G. *Practical Unix Security*, 2nd ed. O'Reilly and Associate, 1996.

[55] Gemini Security Solutions, Inc. Long term digital signatures. Retrieved on October 22, 2010, from `http://geminisecurity.com/wp-content/uploads/2009/01/long-term-digital-signatures.pdf`.

[56] GeoTrust, Inc. True Credentials for code signing certificate practice statement. Retrieved on October 22, 2010, from `http://www.geotrust.com/resources/cps/pdfs/tc_code_signing_CPS_v.1.1.pdf`, 2004.

[57] Giffin, J. T., Jha, S., and Miller, B. P. Efficient context-sensitive intrusion detection. In *Proceedings of the 11th Network and Distributed System Security Symposium* (2004).

[58] Giffin, J. T., Jha, S., and Miller., B. P. Automated discovery of mimicry attacks. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)* (2006), pp. 41–60.

[59] Gligor, V. D., Kailar, R., Stubblebine, S., and Gong, L. Logics for cryptographic protocols - virtues and limitations. In *Proceedings of the 4th IEEE Computer Security Foundations Workshop* (1991), pp. 219–226.

[60] Goyal, V. Certificate revocation using fine grained certificate space partitioning. In *Proceedings of the 11th International Conference on Financial Cryptography and Data Security* (2007), pp. 247–259.

[61] Grimes, R. Authenticode. Retrieved on October 22, 2010, from `http://technet.microsoft.com/en-us/library/cc750035.aspx`.

[62] Gritzalis, S., Spinellis, D., and Georgiadis, P. Security protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification. *Computer Communications 22*, 8 (1999), 697–709.

[63] Guha, A., Krishnamurthi, S., and Jim, T. Using static analysis for Ajax intrusion detection. In *Proceedings of the 18th International Conference on World Wide Web* (2009), pp. 561–570.

[64] Gutmann, P. PKI: It's not dead, just resting. *Computer 35*, 8 (2002), 41–49.

[65] Haber, S., and Stornetta, W. S. How to time-stamp a digital document. *Journal of Cryptology 3*, 2 (1991), 99–111.

[66] Halim, F., Ramnath, R., Sufatrio, Wu, Y., and Yap, R. H. C. A lightweight binary authentication system for Windows. In *Proceedings of the Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM). IFIP International Federation for Information Processing - Trust Management II, Vol. 263/2008* (2008), Springer, pp. 295–310.

[67] Hofmeyr, S. A., Forrest, S., and Somayaji, A. Intrusion detection using sequences of system calls. *Journal of Computer Security 6*, 3 (1998), 151–180.

[68] Hoglund, G., and McGraw, G. *Exploiting Software: How to Break Code*. Addison-Wesley Professional, 2004.

[69] Holgers, T., Watson, D. E., and Gribble, S. D. Cutting through the confusion: A measurement study of homograph attacks. In *Proceedings of the 2006 USENIX Annual Technical Conference* (2006), pp. 261–266.

[70] Howard, J. Kuangplus: A general computer vulnerability checker. Master's thesis, Australian Defence Force Academy, 1999.

[71] Howell, J., and Kotz, D. A formal semantics for SPKI. In *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS)* (2000), pp. 140–158.

[72] Hu, N., Tayi, G. K., Ma, C., and Li, Y. Certificate revocation release policies. *Journal of Computer Security 17*, 2 (2009), 127–157.

[73] Iliadis, J., Gritzalis, S., Spinellis, D., de Cock, D., Preneel, B., and Gritzalis, D. Towards a framework for evaluating certificate status information mechanisms. *Computer Communications 26*, 16 (2003), 1839–1850.

[74] Inoue, H., and Somayaji, A. Lookahead pairs and full sequences: A tale of two anomaly detection methods. In *Proceedings of the 2nd Annual Symposium on Information Assurance* (2007), pp. 9–19.

[75] ITU-T Recommendation X.509. Information Technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks, 2000.

[76] Jackson, C., and Barth, A. Beware of finer-grained origins. In *Proceedings of the Web 2.0 Security and Privacy 2008* (2008).

[77] Jackson, C., Simon, D. R., Tan, D. S., and Barth, A. An evaluation of Extended Validation and picture-in-picture phishing attacks. In *Proceedings of the Usable Security 2007* (2007), pp. 281–293.

[78] Jakobsson, M. Fractal hash sequence representation and traversal. In *Proceedings of the 2002 IEEE International Symposium on Information Theory* (2002), pp. 437–444.

[79] Just, M., and van Oorschot, P. C. Addressing the problem of undetected signature key compromise. In *Proceedings of the Network and Distributed System Security Symposium* (1999).

[80] Kemmerer, R. A., and Vigna, G. Intrusion detection: A brief history and overview. *Computer 35*, 4 (2002), 27–30.

[81] Kessler, V., and Wedel, G. AUTLOG - an advanced logic of authentication. In *Proceedings of 7th IEEE Computer Security Foundations Workshop* (1994), pp. 90–99.

[82] Kim, G. H., and Spafford, E. H. The design and implementation of Tripwire: A file system integrity checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security* (1994), pp. 18–29.

[83] Kocher, P. C. On certificate revocation and validation. In *Proceedings of the 2nd International Conference on Financial Cryptography* (1998), pp. 172–177.

[84] Koga, S., Ryou, J.-C., and Sakurai, K. Pre-production methods of a response to certificates with the common status - design and theoretical evaluation. In *Proceedings of the 1st European PKI Workshop Research and Applications (EuroPKI)* (2004), pp. 85–97.

[85] Kohlas, R., and Maurer, U. Reasoning about public-key certification: On bindings between entities and public keys. *Journal on Selected Areas in Communications 18* (2000), 551–560.

[86] Krawczyk, H., Bellare, M., and Canetti, R. *HMAC: Keyed-hashing for message authentication.* IETF RFC 2104, 1997.

[87] Krsul, I. V. *Software Vulnerability Analysis.* PhD thesis, Purdue University, 1998.

[88] Kruegel, C., Kirda, E., Mutz, D., Robertson, W., and Vigna, G. Automating mimicry attacks using static binary analysis. In *Proceedings of the 14th USENIX Security Symposium* (2005), pp. 161–176.

[89] Kruegel, C., Valeur, F., and Vigna, G. *Intrusion Detection and Correlation: Challenges and Solutions.* Springer, 2005.

[90] Kruegel, C., and Vigna, G. Anomaly detection of Web-based attacks. In *Proceedings of the 10th ACM Conference on Computer and Communication Security* (2003), pp. 251–261.

[91] Krugel, C., Mutz, D., Valeur, F., and Vigna, G. On the detection of anomalous system call arguments. In *Proceedings of the 8th European Symposium on Research in Computer Security (ESORICS)* (2003), pp. 326–343.

[92] Lim, T.-L., and Lakshminarayanan, A. On the performance of certificate validation schemes based on pre-computed responses. In *Proceedings of the 50th Annual IEEE Global Telecommunications Conference (GLOBECOM)* (2007), pp. 182–187.

[93] Lioy, A., Marian, M., Moltchanova, N., and Pala, M. PKI past, present and future. *International Journal of Information Security 5* (2006), 18–29.

[94] Lipson, H. F. Tracking and tracing cyber-attacks: Technical challenges and global policy issues. Retrieved on October 22, 2010, from `http://www.cert.org/archive/pdf/02sr009.pdf`, 2002.

[95] Liu, Z., Bridges, S. M., and Vaughn, R. B. Combining static analysis and dynamic learning to build accurate intrusion detection models. In *Proceedings of the 3rd IEEE International Workshop on Information Assurance* (2005), pp. 164–177.

[96] Lopez, J., Oppliger, R., and Pernul, G. Why have Public Key Infrastructures failed so far? *Internet Research 15*, 5 (2005), 544–556.

[97] Lowe, G. Some new attacks upon security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop* (1996), pp. 162–169.

[98] Lundin, E., and Jonsson, E. Survey of research in the intrusion detection area. Tech. Rep. 02-04, Chalmers University of Technology, 2002.

[99] Ma, C., Hu, N., and Li, Y. On the release of CRLs in Public Key Infrastructure. In *Proceedings of the 15th USENIX Security Symposium* (2006), pp. 17–28.

[100] Maggi, P., Pozza, D., and Sisto, R. Vulnerability modelling for the analysis of network attaks. In *Proceedings of the 3rd International Conference on Dependability of Computer Systems* (2008), pp. 15–22.

[101] Mao, W., and Boyd, C. On a limitations of BAN Logic. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques –EUROCRYPT* (1993), pp. 240–247.

[102] Marlinspike, M. New tricks for defeating SSL in practice. In *Proceedings of the 2009 Black Hat DC* (2009).

[103] Maurer, U. Modelling a Public-Key Infrastructure. In *Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS)* (1996), pp. 325–350.

[104] Maurer, U. Intrinsic limitations of digital signatures and how to cope with them. In *Proceedings of the 6th Information Security Conference – ISC '03* (2003), pp. 180–192.

[105] Maxion, R. A. Masquerade detection using enriched command lines. In *Proceedings of the 2003 International Conference on Dependable Systems & Networks* (2003), pp. 5–14.

[106] McDaniel, P., and Rubin, A. A response to 'Can we eliminate Certificate Revocation Lists?'. In *Proceedings of the 4th International Conference on Financial Cryptography* (2000), pp. 245–258.

[107] Meadows, C. A. Formal verification of cryptographic protocols: A survey. In *Proceedings of the 4th International Conference on the Theory and Application of Cryptology – ASIACRYPT* (1994), pp. 133–150.

[108] Meadows, C. A. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communications 21*, 1 (2003), 44–54.

[109] Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. *Handbook of Applied Cryptography*. CRC Press, 1996.

[110] Micali, S. Efficient certificate revocation. Tech. Rep. MIT-LCS-TM-542b, Massachusetts Institute of Technology, 1996.

[111] Micali, S. NOVOMODO: Scalable certificate validation and simplified PKI management. In *Proceedings of the 1st Annual PKI Research Workshop* (2002), pp. 15–25.

[112] Microsoft Corporation. Microsoft Security Development Lifecycle (SDL) - version 5.0. Retrieved on October 22, 2010, from `http://www.microsoft.com/downloads/en/details.aspx?FamilyID=7d8e6144-8276-4a62-a4c8-7af77c06b7ac&displaylang=en`, 2010.

[113] Microsoft Developer Network. Signtool. Retrieved on October 22, 2010, from `http://msdn.microsoft.com/en-us/library/aa387764(VS.85).aspx`.

[114] Miller, C. The legitimate vulnerability market: Inside the secretive world of 0-day exploit sales. In *Proceedings of the 2007 Workshop on the Economics of Information Security* (2007).

[115] MITRE Corporation. Common Platform Enumeration (CPE). Retrieved on October 22, 2010, from `http://cpe.mitre.org`.

[116] MITRE Corporation. Common Vulnerabilities and Exposures (CVE). Retrieved on October 22, 2010, from `http://cve.mitre.org`.

[117] MITRE Corporation. Open Vulnerability and Assessment Language (OVAL). Retrieved on October 22, 2010, from `http://oval.mitre.org`.

[118] MITRE Corporation. OVAL Interpreter. Retrieved on October 22, 2010, from `http://oval.mitre.org/language/interpreter.html`.

[119] MITRE Corporation. OVAL Repository. Retrieved on October 22, 2010, from `http://oval.mitre.org/repository`.

[120] MITRE Corporation. An introduction to the OVAL Language, version 5.0. Retrieved on October 22, 2010, from `http://oval.mitre.org/oval/documents/docs-06/an_introduction_to_the_oval_language.pdf`, 2006.

[121] Motara, Y., and Irwin, B. In-kernel cryptographic executable verification. In *Proceedings of IFIP International Conference on Digital Forensics* (2005), pp. 303–313.

[122] Munoz, J. L., Forn, J., Esparza, O., and Soriano, B. M. Using OCSP to secure certificate-using transactions in m-commerce. In *Proceedings of the 1st International Conference on Applied Cryptography and Network Security* (2003), pp. 280–292.

[123] Mutz, D., Valeur, F., Kruegel, C., and Vigna, G. Anomalous system call detection. *ACM Transactions on Information and System Security 9* (2006), 61–93.

[124] Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. IETF RFC 2560, 1999.

[125] Naor, M., and Nissim, K. Certificate revocation and certificate update. In *Proceedings of the 7th USENIX Security Symposium* (1998), pp. 217–228.

[126] National Institute of Standards and Technology (NIST). Security Content Automation Protocol (SCAP). Retrieved on October 22, 2010, from `http://scap.nist.gov`.

[127] Nessett, D. M. A critique of the Burrows, Abadi, and Needham Logic. *ACM Operating Systems Review 24*, 2 (1990), 35–38.

[128] NESSUS. Retrieved on October 22, 2010, from `http://www.nessus.org`.

[129] Nielsen, R., and Hamilton, B. A. Observations from the deployment of a large scale PKI. In *Proceedings of the 4th Annual PKI R&D Workshop* (2005).

[130] One, A. Smashing the stack for fun and profit. *Phrack 7*, 49 (1996).

[131] Organisation for Economic Co-operation and Development (OECD). Malicious software (malware): A security threat to Internet economy, Ministerial Background Report, DISTI/ICCP/Reg(2007)5/Final. Retrieved on October 22, 2010, from `http://www.oecd.org/dataoecd/53/34/40724457.pdf`, 2008.

[132] Patcha, A., and Park, J.-M. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks 51*, 12 (2007), 3448–3470.

[133] Perlines Hormann, T., Wrona, K., and Holtmanns, S. Evaluation of certificate validation mechanisms. *Computer Communications 29*, 3 (2006), 291–305.

[134] Pevzner, P. A. L-tuple DNA sequencing: Computer analysis. *Journal of Biomolecular Structure and Dynamics 7* (1989), 63–74.

[135] Provos, N. Improving host security with system call policies. In *Proceedings of the 12th USENIX Security Symposium* (2003), pp. 257–272.

[136] Public Cooperative Vulnerability Database. Retrieved on October 22, 2010, `https://cirdb.cerias.purdue.edu/coopvdb/public/`.

[137] Rescorla, E. Security holes... Who cares? In *Proceedings of the 12th USENIX Security Symposium* (2003), pp. 75–90.

[138] Rivest, R. Can we eliminate Certificate Revocation Lists? In *Proceedings of the 2nd International Conference on Financial Cryptography* (1998), pp. 178–183.

[139] Russinovich, M. Sigcheck v1.65. Retrieved on October 22, 2010, from `http://technet.microsoft.com/en-us/sysinternals/bb897441.aspx`.

[140] Scarfone, K., and Mell, P. Guide to Intrusion Detection and Prevention Systems (IDPS). Tech. Rep. Special Publication 800-94, National Institute of Standards and Technology (NIST), 2007.

[141] Scheibelhofer, K. PKI without revocation checking. In *Proceedings of the 4th Annual PKI R&D Workshop* (2005).

[142] Schmid, M., Hill, F., Ghosh, A., and Bloch, J. Preventing the execution of unauthorized Win32 applications. In *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX)* (2001), pp. 175–183.

[143] Schneier, B. *Applied cryptography: Protocols, algorithms, and source code in C*, 2nd ed. Wiley, New York, 1996.

[144] Security Administrator Tool for Analyzing Networks (SATAN). Retrieved on October 22, 2010, from `http://www.porcupine.org/satan`.

[145] Securityfocus BugTraq. Retrieved on October 22, 2010, from `http://www.securityfocus.com/archive/1`.

[146] Sekar, R., Bendre, M., Dhurjati, D., and Bollineni, P. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy* (2001), pp. 144–155.

[147] Sharma, A., Martin, J. R., Anand, N., Cukier, M., Sanders, W. H., and S, W. H. Ferret: A host vulnerability checking tool. In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing* (2004), pp. 389–394.

[148] Sherif, J. S., and Dearmond, T. G. Intrusion detection: Systems and models. In *Proceedings of the 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (2002), pp. 115–133.

[149] Sobey, J., Biddle, R., van Oorschot, P. C., and Patrick, A. S. Exploring user reactions to new browser cues for Extended Validation certificates. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS)* (2008), pp. 411–427.

[150] Solworth, J. A. Beacon certificate push revocation. In *Proceedings of the 2nd ACM Workshop on Computer Security Architecture* (2008), pp. 59–66.

[151] Solworth, J. A. Instant revocation. In *Proceedings of the 5th European PKI workshop on Public Key Infrastructure: Theory and Practice* (2008), pp. 31–48.

[152] Somayaji, A., and Forrest, S. Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symposium* (2000), pp. 185–197.

[153] Somayaji, A. B. *Operating system stability and security through process homeostasis.* PhD thesis, The University of New Mexico, 2002.

[154] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G., Liang, Z., Newsome, J., Poosankam, P., and Saxena, P. BitBlaze: A new approach to computer security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security* (2008), pp. 1–25.

[155] Storer, T., Martin, U., and Duncan, I. BAN Logic analysis of the UK postal voting system. Tech. rep., University of St. Andrews, 2003.

[156] Stubblebine, S., and Wright, R. An authentication logic with formal semantics supporting synchronization, revocation, and recency. *IEEE Transactions on Software Engineering 28*, 3 (2002), 256–285.

[157] Sufatrio. Authentication schemes for secure mobile Internet services. Master's thesis, National University of Singapore, 2001.

[158] Sufatrio, and Yap, R. H. C. Improving host-based IDS with argument abstraction to prevent mimicry attacks. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID)* (2005), pp. 146–164.

[159] Sufatrio, and Yap, R. H. C. Extending BAN Logic for reasoning with modern PKI-based protocols. In *Proceedings of the IFIP International Workshop on Network and System Security 2008 (NSS)* (2008), pp. 190–197.

[160] Sufatrio, Yap, R. H. C., and Zhong, L. A machine-oriented integrated vulnerability database for automated vulnerability detection and processing. In *Proceedings of the 18th USENIX Large Installation System Administration* (2004), pp. 47–58.

[161] Syverson, P. F. Adding time to a logic of authentication. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS)* (1993), pp. 97–101.

[162] Syverson, P. F. Limitations on design principles for public key protocols. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy* (1996), pp. 62–73.

[163] Syverson, P. F., and Cervesato, I. The logic of authentication protocols. In *Proceedings of the Foundations of Security Analysis and Design (FOSAD)* (2001), pp. 63–136.

[164] Tan, K. M. C., Killourhy, K. S., and Maxion, R. A. Undermining an anomaly-based Intrusion Detection System using common exploits. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID)* (2002), pp. 54–73.

[165] Tan, K. M. C., and Maxion, R. A. 'Why 6?' Defining the operational limits of Stide, an anomaly-based intrusion detector. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy* (2002), pp. 188–202.

[166] Tan, K. M. C., and Maxion, R. A. Determining the operational limits of an anomaly-based intrusion detector. *IEEE Journal on Selected Areas in Communications: Special Issue on Design and Analysis Techniques for Security Assurance 21*, 1 (2003), 96–110.

[167] Tandon, G., and Chan, P. K. On the learning of system call attributes for host-based anomaly detection. *International Journal on Artificial Intelligence Tools 15*, 6 (2006), 875–892.

[168] Tec-Ed. Extended Validation and VeriSign brand, white paper. Retrieved on October 22, 2010, from `http://www.verisign.com/static/040655.pdf`, 2007.

[169] Thawte, Inc. Thawte code signing certificate agreement. Retrieved on October 22, 2010, from http://www.thawte.com/assets/documents/guides/pdf/develcertsign.pdf.

[170] The National Vulnerability Database. Retrieved on October 22, 2010, http://nvd.nist.gov.

[171] The Open Source Vulnerability Database. Retrieved on October 22, 2010, http://osvdb.org/search/advsearch.

[172] Toomey, W., and Howard, J. Kuangplus: Automating vulnerability detection. In *Proceedings of the AUUG2K Conference* (2000), pp. 163–174.

[173] Trusted Computing Group. Retrieved on October 22, 2010, from http://www.trustedcomputinggroup.org.

[174] US-CERT Vulnerability Notes Database. Retrieved on October 22, 2010, from http://www.kb.cert.org/vuls.

[175] van Doorn, L., Ballintijn, G., and Arbaugh, W. A. Signed executables for Linux. Tech. Rep. CS-TR-4256, University of Maryland, 2001.

[176] van Oorschot, P. An alternate explanation of two BAN-Logic 'failures'. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques – EUROCRYPT* (1993), pp. 443–447.

[177] VeriSign, Inc. Verisign certification practice statement version 3.8.1. Retrieved on October 22, 2010, from http://www.verisign.com/repository/CPSv3.8.1_final.pdf, 2009.

[178] Vishik, C., Johnson, S., and Hoffman, D. Infrastructure for trusted environment: In search of a solution. In *Proceedings of the ISSE/SECURE Securing Electronic Business Processes* (2007), pp. 219–227.

[179] Wagner, D., and Dean, D. Intrusion detection via static analysis. In *Proceedings of 2001 IEEE Symposium on Security and Privacy* (2001), pp. 156–168.

[180] Wagner, D., and Soto, P. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)* (2002), pp. 255–264.

[181] Wang, X., and Yu, H. How to break MD5 and other hash functions. In *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT* (2005), pp. 19–35.

[182] Warrender, C., Forrest, S., and Pearlmutter, B. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy* (1999), pp. 133–145.

[183] Williams, M. A. Anti-trojan and trojan detection with in-kernel digital signature testing of executables. NetXSecure NZ Ltd. Retrieved on October 22, 2010, from http://www.netxsecure.net/downloads/sigexec.pdf, 2002.

[184] Windows Software Update Services. Retrieved on October 22, 2010, from http://www.microsoft.com/windowsserversystem/sus/default.mspx.

[185] Windows Update. Retrieved on October 22, 2010, from http://www.microsoft.com/windows/downloads/windowsupdate/learn/default.mspx.

[186] Wu, Y., Sufatrio, Yap, R. H. C., Ramnath, R., and Halim, F. Establishing software integrity trust: A survey and lightweight authentication system for Windows. Book chapter. In *Trust Modeling and Management in Digital Environments: From Social Concept to System Development*, Z. Yan, Ed. Information Science Reference, 2010, ch. 4.

[187] Wurster, G., and van Oorschot, P. Self-signed executables: Restricting replacement of program binaries by malware. In *Proceedings of the 2nd USENIX Workshop on Hot Topics in Security* (2007), pp. 1–5.

[188] Xu, S., and Huang, C.-T. Attacks on PKM protocols of IEEE 802.16 and its later versions. In *Proceedings of the 3rd International Symposium on Wireless Communication Systems* (2006), pp. 185–189.

[189] Zerkle, D., and Levitt, K. Netkuang: A multi-host configuration vulnerability checker. In *Proceedings of the 6th Conference on USENIX Security Symposium* (1996).

[190] Zheng, P. Tradeoffs in certificate revocation schemes. *ACM Computer Communication Review 33*, 2 (2003), 103–112.

[191] Zhou, J., and Deng, R. On the validity of digital signatures. *Computer Communication Review 30*, 2 (2000), 29–34.

[192] Zhou, J., and Lam, K.-Y. Securing digital signatures for non-repudiation. *Computer Communications 22*, 8 (1999), 710–716.