Terms of use

Instead of distributing this XeTeX template with dummy content, which really doesn't make a lot of sense if one needs to see all the features in action, I've chosen to distribute the source of my PhD dissertation.

Every single bit of what I distribute is under the Attribution-NonCommercial-ShareAlike 3.0 Unported license. So, before using these files, please follow this URL:

`http://creativecommons.org/licenses/by-nc-sa/3.0/`.

Fonts and packages

You will need to install some free packages and special fonts if you want to unleash all the power of XeTeX . This document was typeset using the XeTeX typesetting system created by the Non-Roman Script Initiative and the memoir class created by Peter Wilson. The body text is set 10pt with Adobe Caslon Pro. Other fonts include `Envy Code R`, Optima Regular and. Most of the drawings are typeset using the TikZ/PGF packages by Till Tantau.

Help!

To build this document run `xelatex thesis`, then `bibtex thesis` and then `xelatex thesis` twice and enjoy the `thesis.pdf`. For further help refer to `http://www.ctan.org/tex-archive/macros/latex/contrib/memoir/memman.pdf` and `http://faq.tug.org`. Googling about LaTeX often makes you fail `http://mirror.ctan.org/info/l2tabu/english/l2tabuen.pdf`.

How to remove this note

To remove this note, open `thesis.tex` and comment line 146.

–Federico, fmaggi@elet.polimi.it

Politecnico di Milano
Dipartimento di Elettronica e Informazione
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

Integrated Detection of Anomalous Behavior of Computer Infrastructures

Doctoral Dissertation of:
Federico Maggi

Advisor:
Prof. Stefano Zanero

Tutor:
Prof. Letizia Tanca

Supervisor of the Doctoral Program:
Prof. Patrizio Colaneri

2009 - XXII

Preface

This thesis embraces all the efforts that I put during the last three years as a PhD student at Politecnico di Milano. I have been working under the supervision of Prof. S. Zanero and Prof. G. Serazzi, who is also the leader of the research group I am part of. In this time frame I had the wonderful opportunity of being "initiated" to research, which radically changed the way I look at things: I found my natural *"thinking outside the box"* attitude — that was probably well-hidden under a thick layer of lack-of-opportunities, I took part of very interesting joint works — among which the year I spent at the Computer Security Laboratory at UC Santa Barbara is at the first place, and I discovered the Zen of my life.

My research is all about *computers* and every other technology possibly related to them. Clearly, the way I look at computers has changed a bit since when I was seven. Still, I can remember me, typing on that Commodore 64 in front of a tube TV screen, trying to get that d—n routine written in Basic to work. I was just playing, obviously, but when I recently found a picture of me in front of that screen...it all became clear.

So, although my attempt of writing a program to authenticate myself was a little bit naive — being limited to a print instruction up to that point apart, of course — I thought *"maybe I am not in the wrong place, and the fact that my research is still about security is a good sign"*!

Many years later, this work comes to life. There is a humongous amount of people that, directly or indirectly, have contributed to my research and, in particular, to this work. Since my first step into the lab, I will not, ever, be thankful enough to Stefano, who, despite my skepticism, convinced me to submit that application for the PhD program. For trusting me since the very first moment I am thankful to Prof. G. Serazzi as well, who has been always supportive. For hosting and supporting my research abroad I thank Prof. G. Vigna, Prof. C. Kruegel, and Prof. R. Kemmerer. Also, I wish to thank Prof. M. Matteucci for the great collaboration, Prof. I. Epifani for her insightful suggestions and Prof. H. Bos for the detailed review and the constructive comments.

On the colleagues-side of this acknowledgments I put all the fellows of Room 157, Guido, the crew of the seclab and, in particular, Wil with whom I shared all the pain of paper writing between Sept

'08 and Jun '09.

On the friends-side of this list Lorenzo and Simona go first, for being our family.

I have tried to translate in simple words the infinite gratitude I have and will always have to Valentina and my parents for being my fixed point in my life. Obviously, I failed.

Federico Maggi
Milano
September 2009

Abstract

This dissertation details our research on anomaly detection techniques, that are central to several classic security-related tasks such as network monitoring, but it also have broader applications such as program behavior characterization or malware classification. In particular, we worked on anomaly detection from three different perspective, with the common goal of recognizing awkward activity on computer infrastructures. In fact, a computer system has several weak spots that must be protected to avoid attackers to take advantage of them. We focused on protecting the operating system, central to any computer, to avoid malicious code to subvert its normal activity. Secondly, we concentrated on protecting the web applications, which can be considered the modern, shared operating systems; because of their immense popularity, they have indeed become the most targeted entry point to violate a system. Last, we experimented with novel techniques with the aim of identifying related events (e.g., alerts reported by intrusion detection systems) to build new and more compact knowledge to detect malicious activity on large-scale systems.

Our contributions regarding host-based protection systems focus on characterizing a process' behavior through the system calls invoked into the kernel. In particular, we engineered and carefully tested different versions of a multi-model detection system using both stochastic and deterministic models to capture the features of the system calls during normal operation of the operating system. Besides demonstrating the effectiveness of our approaches, we confirmed that the use of finite-state, deterministic models allow to detect deviations from the process' control flow with the highest accuracy; however, our contribution combine this effectiveness with advanced models for the system calls' arguments resulting in a significantly decreased number of false alarms.

Our contributions regarding web-based protection systems focus on advanced training procedures to enable learning systems to perform well even in presence of changes in the web application source code — particularly frequent in the Web 2.0 era. We also addressed data scarcity issues that is a real problem when deploying an anomaly detector to protect a new, never-used-before application. Both these issues dramatically decrease the detection capabilities of an intrusion detection

system but can be effectively mitigated by adopting the techniques we propose.

Last, we investigated the use of different stochastic and fuzzy models to perform automatic alert correlation, which is as post processing step to intrusion detection. We proposed a fuzzy model that formally defines the errors that inevitably occur if time-based alert aggregation (i.e., two alerts are considered correlated if they are close in time) is used. This model allow to account for measurements errors and avoid false correlations due to delays, for instance, or incorrect parameter settings. In addition, we defined a model to describe the alert generation as a stochastic process and experimented with non-parametric statistical tests to define robust, zero-configuration correlation systems.

The aforementioned tools have been tested over different datasets — that are thoroughly documented in this document — and lead to interesting results.

Sommario

Questa tesi descrive in dettaglio la nostra ricerca sulle tecniche di anomaly detection. Tali tecniche sono fondamentali per risolvere problemi classici legati alla sicurezza, come per esempio il monitoraggio di una rete, ma hanno anche applicazioni di più ampio spettro come l'analisi del comportamento di un processo in un sistema o la classificazione di malware. In particolare, il nostro lavoro si concentra su tre prospettive differenti, con lo scopo comune di rilevare attività sospette in un sistema informatico. Difatti, un sistema informatico ha diversi punti deboli che devono essere protetti per evitare che un aggressore possa approfittarne. Ci siamo concentrati sulla protezione del sistema operativo, presente in qualsiasi computer, per evitare che un programma possa alterarne il funzionamento. In secondo luogo ci siamo concentrati sulla protezione delle applicazioni web, che possono essere considerate il moderno sistema operativo globale; infatti, la loro immensa popolarità ha fatto sì che diventassero il bersaglio preferito per violare un sistema. Infine, abbiamo sperimentato nuove tecniche per identificare relazioni tra eventi (e.g., alert riportati da sistemi di intrusion detection) con lo scopo di costruire nuova conoscenza per poter rilevare attività sospette su sistemi di larga-scala.

Riguardo ai sistemi di anomaly detection host-based ci siamo focalizzati sulla caratterizzazione del comportamento dei processi basandoci sul flusso di system call invocate nel kernel. In particolare, abbiamo ingegnerizzato e valutato accuratamente diverse versioni di un sistema di anomaly detection multi-modello che utilizza sia modelli stocastici che modelli deterministici per catturare le caratteristiche delle system call durante il funzionamento normale del sistema operativo. Oltre ad aver dimostrato l'efficacia dei nostri approcci, abbiamo confermato che l'utilizzo di modelli deterministici a stati finiti permettono di rilevare con estrema accuratezza quando un processo devia significativamente dal normale control flow; tuttavia, l'approccio che proponiamo combina tale efficacia con modelli stocastici avanzati per modellizzare gli argomenti delle system call per diminuire significativamente il numero di falsi allarmi.

Riguardo alla protezione delle applicazioni web ci siamo focalizzati su procedure avanzate di addestramento. Lo scopo è permettere ai sistemi basati su apprendimento non supervisionato di funzionare correttamente anche in presenza di

cambiamenti nel codice delle applicazioni web — fenomeno particolarmente frequente nell'era del Web 2.0. Abbiamo anche affrontato le problematiche dovute alla scarisità di dati di addestramento, un ostacolo più che realistico specialmente se l'applicazione da proteggere non è mai stata utilizzata prima. Entrambe le problematiche hanno come conseguenza un drammatico abbassamento delle capacità di detection degli strumenti ma possono essere efficacemente mitigate adottando le tecniche che proponiamo.

Infine abbiamo investigato l'utilizzo di diversi modelli, sia stocastici che fuzzy, per la correlazione di allarmi automatica, fase successiva alla rilevazione di intrusioni. Abbiamo proposto un modello fuzzy che definisce formalmente gli errori che inevitabilmente avvengono quando si adottano algoritmi di correlazione basati sulla distanza nel tempo (i.e., due allarmi sono considerati correlati se sono stati riportati più o meno nello stesso istante di tempo). Questo modello permette di tener conto anche di errori di misurazione ed evitare decisioni scorrete nel caso di ritardi di propagazione. Inoltre, abbiamo definito un modello che descrive la generazione di allarmi come un processo stocastico e abbiamo sperimentato con dei test non parametrici per definire dei criteri di correlazione robusti e che non richiedono configurazione.

Contents

Introduction 1

Network connected devices such as personal computers, mobile phones, or gaming consoles are nowadays enjoying immense popularity. In parallel, the Web and the humongous amount of services it offers have certainly became the most ubiquitous tools of all the times. Facebook counts more than 250 millions active users of which 65 millions are using it on mobile devices; not to mention that more than 1 billion photos are uploaded to the site *each month* [Facebook, 2009]. And this is just one, popular website. One year ago, Google estimated that the approximate number of unique *Uniform Resource Locators* (URLs) is 1 trillion [Alpert and Hajaj, 2008], while YouTube has stocked more than 70 million videos as of March 2008, with 112,486,327 views just on the most popular video as of January 2009 [Singer, 2009]. And people from all over the world inundate the Web with more than 3 million tweets *per day*. Not only the Web 2.0 has became predominant; in fact, thinking that on December 1990 the Internet was made of *one* site and today it counts more than 100 million sites is just astonishing [Zakon, 2006].

The Internet and the Web are huge [Miniwatts Marketing Grp., 2009]. The relevant fact, however, is that they both became the most advanced workplace. Almost every industry connected its own network to the Internet and relies on these infrastructures for a vast majority of transactions; most of the time monetary transactions. As an example, every year Google looses approximately 110 millions of US Dollars in ignored ads because of the *"I'm feeling lucky"* button. The scary part is that, during their daily work activities, people typically pay poor or no attention at all to the risks that derive from exchanging any kind of information over such a complex, interconnected infrastructure. This is demonstrated by the effectiveness of social engineering [Mitnick, 2002] scams carried over the Internet or the phone [Granger, 2001]. Recall that 76% of the phishing is related to finance. Now, compare this landscape to what the most famous security quote states.

> "The only truly secure computer is one buried in concrete, with the power turned off and the network cable cut".
>
> —*Anonymous*

In fact, the Internet is all but a safe place [Ofer Shezaf and Jeremiah Grossman and Robert Auger, 2009], with more than 1,250 *known* data breaches between 2005 and 2009 [Clearinghouse, 2009] and an estimate of 263,470,869 records stolen by intruders. One may wonder why the advance of research in computer security and the increased awareness of governments and public institutions are still not capable of avoiding such incidents. Besides the fact that the aforementioned numbers would be order of magnitude higher in absence of countermeasures, todays' security issues are, basically, caused by the combination of two phenomena: the high amount of software vulnerabilities and the effectiveness of todays' exploitation strategy.

software flaws — (un)surprisingly, software is affected by vulnerabilities. Incidentally, tools that have to do with the Web, namely, browsers and 3rd-party extensions, and web applications, are the most vulnerable ones. For instance, in 2008, Secunia reported around 115 security vulnerabilities for Mozilla Firefox, 366 for Internet Explorer's ActiveX [Secunia, 2008]. Office suites and e-mail clients, that are certainly the must-have-installed tool on every workstation, hold the second position [The SANS Institute, 2005].

massification of attacks — in parallel to the explosion of the Web 2.0, attackers and the underground economy have quickly learned that a sweep of exploits run against *every* reachable host have

> more chances to find a vulnerable target and, thus, is much more profitable compared to a single effort to break into a high-value, well-protected machine.

These circumstances have initiated a vicious circle that provides the attackers with a very large pool of vulnerable targets. Vulnerable client hosts are compromised to ensure virtually unlimited bandwidth and computational resources to attackers, while server side applications are violated to host malicious code used to infect client visitors. And so forth. An old fashioned attacker would have violated a single site using all the resources available, stolen data and sold it to the underground market. Instead, a modern attacker adopts a "vampire" approach and exploit client-side software vulnerabilities to take (remote) control of million hosts. In the past the diffusion of malicious code such as viruses was sustained by sharing of infected, cracked software through floppy or compact disks; nowadays, the Web offers unlimited, public storage to attackers that deploy their exploit on compromised websites.

Thus, not only the type of vulnerabilities has changed, posing virtually every interconnected device at risk. The exploitation strategy created new types of threats that take advantage of classic malicious code patterns but in a new, extensive, and tremendously effective way.

1.1 Todays' Security Threats

Every year, new threats are discovered and attacker take advantage of them until effective countermeasures are found. Then, new threats are discovered, and so forth. Symantec quantifies the amount of new malicious code threats to be 1,656,227 as of 2008 [Turner et al., 2009], 624,267 one year earlier and only 20,547 in 2002. Thus, countermeasures must advance at least with the same grow rate. In addition:

> [...] the current threat landscape — such as the increasing complexity and sophistication of attacks, the evolution of attackers and attack patterns, and malicious activities being pushed to emerging countries — show not just the benefits of, but also the need for increased cooperation among security companies, governments, aca-

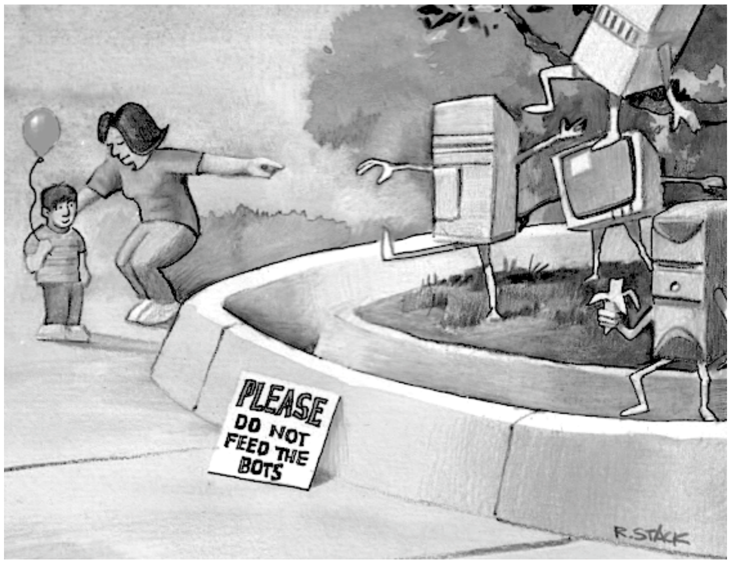


Figure 1.1: Illustration taken from [Holz, 2005] and ©2005 IEEE. Authorized license limited to Politecnico di Milano.

> demics, and other organizations and individuals to combat these changes [Turner et al., 2009].

Todays' underground economy run a very proficient market: everyone can buy credit card information for as low as \$0.06–\$30, full identities for just \$0.70–\$60 or rent a scam hosting solution for \$3–\$40 per week plus \$2-\$20 for the design [Turner et al., 2009].

The main underlying technology actually employs a classic type of software called *bot* (jargon for *robot*), which is not malicious *per sé*, but is used to remotely control a network of compromised hosts, called *botnet* [Holz, 2005]. Remote commands can be of any type and typically include launching an attack, starting a phishing or spam campaign, or even updating to the latest version of the bot software by downloading the binary code from a host controlled by the attackers (usually called *bot master*) [Stone-Gross et al., 2009]. The exchange good has now become the botnet infrastructure itself rather than the data that can be stolen or the spam that can be sent. These are mere outputs of todays' most popular service offered for rent by the underground economy.

1.1.1 The Role of Intrusion Detection

The aforementioned, dramatic big picture may lead to think that the malicious software will eventually proliferate at every host of the Internet and no effective remediation exists. However, a more careful analysis reveals that, despite the complexity of this scenario, the problems that must be solved by a security infrastructure can be decomposed into relatively simple tasks that, surprisingly, may already have a solution. Let us look at an example.

Example 1.1.1 This is how a sample exploitation can be structured:

injection — a malicious request is sent to the vulnerable web application with the goal of corrupting all the responses sent to legitimate clients from that moment on. For instance, more than one releases of the popular WordPress blog application are vulnerable to injection attacks[1] that allow an attacker to permanently include arbitrary content to the pages. Typically, such an arbitrary content is malicious code (e.g., JavaScript, VBSCrip, ActionScript, ActiveX) that, every time a legitimate user requests the infected page, executes on the client host.

infection — Assuming that the compromised site is frequently accessed — this might be the realistic case of the WordPress-powered ZDNet news blog[2] — a significant amount of clients visit it. Due to the high popularity of vulnerable browsers and plug-ins, the client may run Internet Explorer — that is the most popular — or an outdated release of Firefox on Windows. This create the perfect circumstances for the malicious page to successfully execute. In the best case, it may download a virus or a generic malware from a website under control of the attacker, so infecting the machine. In the worst case, this code may also exploit specific browser vulnerabilities and execute in privileged mode.

control & use — The malicious code just download installs and hides itself onto the victim's computer, which has just joined a bot-

[1]http://secunia.com/advisories/23595
[2]http://wordpress.org/showcase/zdnet/

net. As part of it, the client host can be remotely controlled by the attackers who can, for instance, rent it, use its bandwidth and computational power along with other computers to run a distributed *Denial of Service* (DoS) attack. Also, the host can be used to automatically perform the same attacks described above against other vulnerable web applications. And so forth.

This simple yet quite realistic example shows the various kinds of malicious activity that are generated during a typical drive-by exploitation. It also shows its requirements and assumptions that must hold to guarantee success. More precisely, we can recognize:

network activity — clearly, the whole interaction relies on a network connection over the Internet: the *HyperText Transfer Protocol* (HTTP) connections used, for instance, to download the malicious code as well as to launch the injection attack used to compromise the web server.

host activity — similarly to every other type of attack against an application, when the client-side code executes, the browser (or one of its extension plug-ins) is forced to behave improperly. If the malicious code executes till completion the attack succeeds and the host is infected. This happens only if the platform, operating system, and browser all match the requirements assumed by the exploit designer. For instance, the attack may succeed on Windows and not on Mac OS X, although the vulnerable version of, say, Firefox is the same on both the hosts.

HTTP traffic — in order to exploit the vulnerability of the web application, the attacking client must generate malicious HTTP requests. For instance, in the case of an *Structured Query Language* (SQL) injection — that is the second most common vulnerability in a web application — instead of a regular

```
GET /index.php?username=myuser
```

the web server might be forced to process a

```
GET /index.php?username=' OR 'x'='x'--\&content=<script
src="evil.com/code.js">
```

that causes the `index.php` page to behave improperly.

It is now clear that protection mechanisms that analyze the network traffic, the activity of the client's operating system, the web server's HTTP logs, or any combination of the three, have chances of recognizing that something malicious is happening in the network. For instance, if the *Internet Service Provider* (ISP) network adopt Snort, a lightweight *Intrusion Detection System* (IDS) that analyzes the network traffic for known attack patterns, could block all the packets marked as suspicious. This would prevent, for instance, the SQL injection to reach the web application. A similar protection level can be achieved by using other tools such as ModSecurity [Ristic, 2008]. One of the problems that may arise with these classic, widely adopted solutions is if a zero day attack is used. A zero day attack or threat exploits a vulnerability that is unknown to the public, undisclosed to the software vendor, or a fix is not available; thus, protection mechanisms that merely blacklist known malicious activity immediately become ineffective. In a similar vein, if the client is protected by an anti-virus, the infection phase can be blocked. However, this countermeasure is once again successful only if the anti-virus is capable of recognizing the malicious code, which assumes that the code is known to be malicious.

Ideally, an effective and comprehensive countermeasure can be achieved if all the protection tools involved (e.g., client-side, server-side, network-side) can collaborate together. For instance, if a website is publicly reported to be malicious, a client-side protection tool should block all the content downloaded from that particular website. This is only a simple example.

Thus, countermeasures against todays' threats already exist but are subject to at least two drawbacks:

- they offer protection only against known threats. To be effective we must assume that all the hostile traffic can be enumerated, which is clearly an impossible task.

> Why is "Enumerating Badness" a dumb idea? It's a dumb idea because sometime around 1992 the amount of Badness in the Internet began to vastly outweigh the amount of Goodness. For every harmless, legitimate, application, there are dozens or hundreds of pieces of malware, worm tests, exploits, or viral code. Examine a typical antivirus package and

> you'll see it knows about 75,000+ viruses that might infect your machine. Compare that to the legitimate 30 or so apps that I've installed on my machine, and you can see it's rather dumb to try to track 75,000 pieces of Badness when even a simpleton could track 30 pieces of Goodness [Ranum, 2005].

- they lack of cooperation, which is crucial to detect global and slow attacks.

This said, we conclude that classic approaches such as dynamic and static code analysis and IDS already offer good protection but industry and research should move toward methods that require little or no knowledge. In this work, we indeed focus on the so called anomaly-based approaches, i.e., those that attempt to recognize the threats by detecting any variation from a system's normal operation, rather than looking for signs of known-to-be-malicious activity.

1.2 Original Contributions

Our main research area is *Intrusion Detection* (ID). In particular, we focus on anomaly-based approaches to detect malicious activities. Since todays' threats are complex, a single point of inspection is not effective. A more comprehensive monitoring system is more desirable to protect both the network, the applications running on a certain host, and the web applications (that are particularly exposed due to the immense popularity of the Web). Our contributions focus on the mitigation of both host-based and web-based attacks, along with two techniques to correlate alerts from hybrid sensors.

1.2.1 Host-based Anomaly Detection

Typical malicious processes can be detected by modeling the characteristics (e.g., type of arguments, sequences) of the system calls executed by the kernel, and by flagging unexpected deviations as attacks. Regarding this type of approaches, our contributions focus on hybrid models to accurately characterize the behavior of a binary application. In particular:

- we enhanced, re-engineered, and evaluated a novel tool for modeling the normal activity of the Linux 2.6 kernel. Compared to other existing solutions, our system shows better detection capabilities and good contextualization of the alerts reported. These results are detailed in Section 3.2.
- We engineered and evaluated an IDS to demonstrate that the combined use of (1) deterministic models to characterize a process' control flow and (2) stochastic models to capture normal features of the data flow, lead to better detection accuracy. Compared to the existing deterministic and stochastic approaches separately, our system shows better accuracy, with almost zero false positives. These results are detailed in Section 3.3.
- We adapted our techniques for forensics investigation. By running experiments on real-world data and attacks, we show that our system is able to detect hidden tamper evidence although sophisticated anti-forensics tools (e.g., userland process execution) have been used. These results are detailed in Section 3.4.

1.2.2 Web-based Anomaly Detection

Attempts of compromising a web application can be detected by modeling the characteristics (e.g., parameter values, character distributions, session content) of the HTTP messages exchanged between servers and clients during normal operation. This approach can detect virtually any attempt of tampering with HTTP messages, which is assumed to be evidence of attack. In this research field, our contributions focus on training data scarcity issues along with the problems that arise when an application changes its legit behavior. In particular:

- we contributed to the development of a system that learns the legit behavior of a web application. Such a behavior is defined by means of features extracted from 1) HTTP requests, 2) HTTP responses, 3) SQL queries to the underlying database, if any. Each feature is extracted and learned by using different models, some of which are improvements over well-known approaches and some others are original. The main contribution of this work is the *combination* of database query models with HTTP-based models. The resulting system has been validated through preliminary experiments that shown very high accuracy. These results are detailed in Section 4.1.2.

- we developed a technique to automatically detect legit changes in web applications with the goal of suppressing the large amount of false detections due to code upgrades, frequent in todays' web applications. We run experiments on real-world data to show that our simple but very effective approach accurately predict changes in web applications and can distinguish good *vs.* malicious changes (i.e., attacks). These results are detailed in Section 4.3.

- We designed and evaluated a machine learning technique to aggregate IDS models with the goal of ensuring good detection accuracy even in case of scarce training data available. Our approach relies on clustering techniques and nearest-neighbor search to look-up well-trained models used to replace undertrained ones that are prone to overfitting and thus false detections. Experiments on real-world data have shown that almost every false alert due to overfitting is avoided with as low as 32-64 training samples per model. These results are described in Section 4.2.

Although these techniques have been developed on top of a web-based anomaly detector, they are sufficiently generic to be easily adapted to other systems using learning approaches.

1.2.3 Alert Correlation

IDS alerts are usually post-processed to generate compact reports and eliminate redundant, meaningless, or false detections. In this research field, our contributions focus on unsupervised techniques applied to aggregate and correlate alert events with the goal of reducing the effort of the security officer. In particular:

- We developed and tested an approach that accounts for the common measurement errors (e.g., delays and uncertainties) that occur in the alert generation process. Our approach exploits fuzzy metrics both to model errors and to construct an alert aggregation criterion based on distance in time. This technique has been show to be more robust compared to classic time-distance based aggregation metrics. These results are detailed in Section 5.1.

- We designed and tested a prototype that models the alert generation process as a stochastic process. This setting allowed us to construct a simple, non-parametric hypothesis test that can detect whether two alert streams are correlated or not. Besides its simplicity, the advantage of our approach is to not requiring any parameter. These results are described in Section 5.3.

The aforementioned results have been published in the proceedings of international conferences and international journals.

1.3 Document Structure

This document is structured as follows. Chapter 2 introduces the ID, that is the topic of our research. In particular, Chapter 2 rigorously describes all the basic components that are necessary to define the ID task and an IDSs. The reader with knowledge on this subject may skip the first part of the chapter and focus on Section 2.2 and 2.3 that include a comprehensive review of the most relevant and influential modern approaches on network-, host-, web-based ID techniques, along with a separate overview of the alert correlation approaches.

As described in Section 1.2, the description of our contributions is structured into three chapters. Chapter 3 focuses on host-based techniques, Chapter 4 regards web-based anomaly detection, while Chapter 5 described two techniques that allow to recognize relations between alerts reported by network- and host-based systems. Reading Section 2.2.1 is recommended before reading Chapter 5.

The reader interested in protection techniques for the operating system can skim through Section 2.2.2 and then read Chapter 3. The reader with interests on web-based protection techniques can read Section 2.2.3 and then Chapter 4. Similarly, the reader interested in alert correlation systems can skim through Section 2.2.1 and 2.2.2 and then read Chapter 5.

Detecting Malicious Activity 2

Malicious activity is a generic term that refers to automated or manual attempts of compromising the security of a computer infrastructure. Examples of malicious activity include the output generated (or its effect on a system) by the execution of simple script kiddies, viruses, DoS attacks, exploits of *Cross-Site Scripting* (XSS) or SQL injection vulnerabilities, and so forth. This chapter describes the research tools and methodologies available to detect and mitigate malicious activity on a network, a single host, a web-server and the combination of the three.

First, the background concepts and the ID problem are described in this chapter along with a taxonomic description of the most relevant aspects of an IDS. Secondly, a detailed survey of the selected state-of-the-art anomaly detection approaches is presented with the help of further classification dimensions. In addition, the problem of alert correlation, that is an orthogonal topic, is described and the most relevant, recent research approaches are overviewed. Last but not least, the problem of evaluating an IDS is presented to provide the essential terminology and criteria to understand the effectiveness of both the reviewed literature and our original contributions.

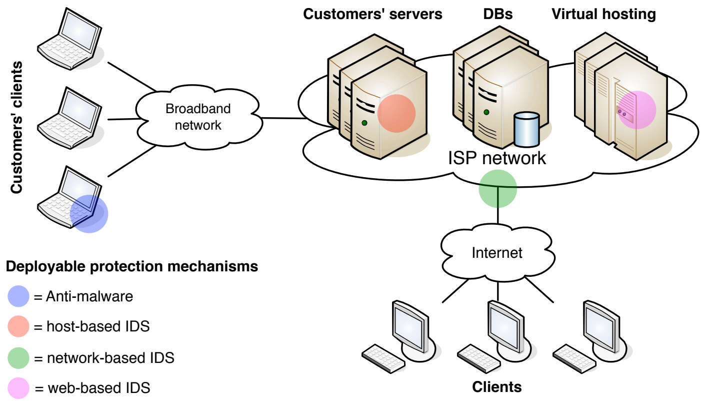


Figure 2.1: Generic and comprehensive deployment scenario for IDSs.

2.1 Intrusion Detection

ID is the practice of analyzing a system to find malicious activity. This section defines this concept more precisely by means of simple but rigorous definitions. The context of such definitions is the generic scenario of an Internet site, for instance, the network of an ISP. An example is depicted in Figure 2.1. An Internet site —and, in general, the Internet itself— is the state-of-the-art computer infrastructure. In fact, it is a network that adopts almost any kind of known computer technology (e.g., protocols, standards, containment measures), it runs a rich variety of servers such as HTTP, *File Transfer Protocol* (FTP), *Secure SHell* (SSH), *Virtual Private Network* (VPN) to support a broad spectrum of sophisticated applications and services (e.g., web applications, e-commerce, applications, the Facebook Platform, Google Applications). In addition, a generic Internet site receives and process traffic from virtually any user connected to the Net and thus represents the perfect research testbed for IDS.

Definition 2.1.1 (System) A *system* is the domain of interest for security.

Note that a system can be a physical or a logical one. For instance, a physical system could be a host (e.g., the *DataBase* (DB) server or one of the client machines shown in the figure), a whole network (e.g., the ISP network shown in the figure); a logical system can be an application, a service, such as one of the web services run in a virtual machine deployed in the ISP network. While running, each system produces *activity*, that we define as follows.

Definition 2.1.2 (System activity) A system *activity* is any data generated while the system is working. Such activity is formalized a sequence of events $\mathbb{I} = [I_1, I_2, I_i, \dots, I_N]$.

For instance, each of the clients of Figure 2.1 produces system logs: in this case $\mathbb{I}$ would contain an I_i for each log entry. A human readable representation follows.

```
chatWithContact:(null)] got a nil targetContact.  Aug 18 00:29:40
[0x0-0x1b01b].org.mozilla.firefox[0]: NPP_Initialize called Aug 18
00:29:40 [0x0-0x1b01b].org.mozilla.firefox[0]: 2009-08-18 00:29:40.039
firefox-bin[254:10b] NPP_New(instance=0x169e0178,mode=2,argc=5) Aug 18
00:29:40 [0x0-0x1b01b].org.mozilla.firefox[0]: 2009-08-18 00:29:40.052
firefox-bin[254:10b] NPP_NewStream end=396239
```

Similarly, the web servers generates HTTP requests and responses: in this case $\mathbb{I}$ would contain an I_i for each HTTP message. Its human readable representation follows.

```
/media//images/favicon.ico HTTP/1.0” 200 1150 ”-” ”Mozilla/5.0
(Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.0.10)
Gecko/2009042315 Firefox/3.0.10 Ubiquity/0.1.4” 128.111.48.4
[20/May/2009:15:26:44 -0700] ”POST /report/ HTTP/1.0” 200 19171
”http://www.phonephishing.info/report/” ”Mozilla/5.0 (Macintosh; U;
Intel Mac OS X 10.5; en-US; rv:1.9.0.10) Gecko/2009042315
Firefox/3.0.10 Ubiquity/0.1.4” 128.111.48.4 [20/May/2009:15:26:44
-0700] ”GET /media//css/main.css HTTP/1.0” 200 5525
”http://www.phonephishing.info/report/” ”Mozilla/5.0 (Macintosh; U;
Intel Mac OS X 10.5; en-US; rv:1.9.0.10) Gecko/2009042315
Firefox/3.0.10 Ubiquity/0.1.4” 128.111.48.4 [20/May/2009:15:26:44
-0700] ”GET /media//css/roundbox.css HTTP/1.0” 200 731
”http://www.phonephishing.info/media//css/main.css” ”Mozilla/5.0
(Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.0.10)
Gecko/2009042315 Firefox/3.0.10 Ubiquity/0.1.4”
```

Other examples of system activity are described in Section 2.1.3.

In this document, we often used the term *normal behavior* referring to a set of characteristics (e.g., the distribution of the characters of string parameters, the mean and standard deviation of the values of integer parameters) extracted from the system activity gathered during normal operation (i.e., without being compromised). Moreover, in the remainder of this document, we need other definitions.

Definition 2.1.3 (Activity Profile) The *activity profile* (or activity model) $c_{\mathbb{I}}$ is a set of models

$$c_{\mathbb{I}} = \langle m^{(1)}, \dots, m^{(u)}, \dots, m^{(U)} \rangle$$

generated by extracting features from the system activity $\mathbb{I}$.

This definition will be used in Section 2.1.3.1 and Example 2.1.1 and 2.1.2 describe an instance of $m^{(u)}$. An example of a real-world profile is described in Example 4.1.1. We can now define the:

Definition 2.1.4 (System Behavior) The *system behavior* is the set of features (or models), along with their numeric values, extracted by (or contained in) the activity profile.

In particular, we will use this term as a synonym of *normal* system behavior, referring to the system behavior during normal operation.

Given the high-accessibility of the Internet, publicly available systems such as web servers, web applications, DB servers, are constantly at risk. In particular, they can be compromised with the goal of stealing valuable information, deploying infection kits or running phishing and spam campaigns. These are all examples of *intrusions*. More formally.

Definition 2.1.5 (Intrusion) An *intrusion* is the automated or manual act of violating one or more security paradigms (i.e., confidentiality, integrity, availability) of a system. Intrusions are formalized as a sequence of events:

$$\mathbb{O} = [O_1, O_2, O_i, \dots, O_M] \subseteq \mathbb{I}$$

Typically, when an intrusion takes place, a system behaves unexpectedly and, as a consequence, its activity differs than during normal operation. This is because an attack or the execution of malware code often exploit vulnerabilities to bring the system into states it was not designed for. For this reason, the activity that the system generates is called *malicious activity*; often, this term is also used to indicate the attack or the malware code execution itself. An example of O_i is the following log entry: it shows evidence of a XSS attack that will make a vulnerable page to display the arbitrary content supplied as part of the GET request (while the page was not intentionally designed to this purpose).

```
/report/add/comment/<DIV
STYLE=”background-image:\0075\0072\006C\0028’\006a
      \0061\0076\0061\0073\0063\0072\0069\0070\0074\003a\0061\006c
      \0065\0072\0074\0028.1027\0058.1053\0053\0027\0029’\0029”>/
HTTP/1.0” 200 731 ”http://www.phonephishing.info/report/add/”
”Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.0.10)
Gecko/2009042315 Firefox/3.0.10 Ubiquity/0.1.4”
```

Note 2.1.1 We adopt a simplified representation of intrusions with respect to a dataset $\mathbb{D}$ (i.e., both normal and malicious): with $\mathbb{O} \subseteq \mathbb{D}$ we indicate that the activity $\mathbb{I}$ contains malicious events $\mathbb{O}$; however, strictly speaking, intrusions events and activity events can be of completely different types, thus the ”$\subseteq$” relation may not be defined in a strict mathematical sense.

If a system is well-designed, any intrusion attempts always leave some traces in the system's activity. These traces are called *tamper evidence* and are essential to perform *intrusion detection*.

Definition 2.1.6 (Intrusion Detection) Intrusion *detection* is the separation of intrusions from normal activity through the analysis of the activity of a system, while the system is running. Intrusions are marked as *alerts*, which are formalized as a sequence of event $\mathbb{A} = [A_1, A_2, A_i, \ldots, A_L] \subseteq \mathbb{O}$.

Similarly, $\mathbb{A} \subseteq \mathbb{O}$ must not be interpreted in a strict sense: it is just a notation to indicate that for each intrusion, an alert may or may not exist. Note that ID can be also performed by manually inspecting a system activity. This is clearly a tedious and unefficient task, thus research and industrial interests are focused on automatic approaches.

Definition 2.1.7 (Intrusion Detection System) An *intrusion detection system* is an automatic tool that performs the ID task.

Given the above definitions, an abstract model of an IDS is shown in Figure 2.2.

$$\begin{tikzcd}
\mathbb{I}, \mathbb{O} \arrow[r] & \boxed{\text{IDS}} \arrow[r] & \mathbb{A}
\end{tikzcd}$$

Figure 2.2: Abstract I/O model of an IDS.

Although each IDS relies on its own data model and format to represent $\mathbb{A}$, the *Internet Engineering Task Force* (IETF) proposed *Intrusion Detection Message Exchange Format* (IDMEF) [Debar et al., 2006] as a common format for reporting alert streams generated by different IDS.

2.1.1 Evaluation

Evaluating an IDS means running it on collected data $\mathbb{D} = \mathbb{I} \cup \mathbb{O}$ that resembles real-world scenarios. This means that such data includes both intrusions $\mathbb{O}$ and normal activity $\mathbb{I}$, i.e., $|\mathbb{I}|, |\mathbb{O}| > 0$ and $\mathbb{I} \cap \mathbb{O} = \varnothing$, i.e., $\mathbb{I}$ must include no malicious activity other than $\mathbb{O}$. The system is run in a controlled environment to collect $\mathbb{A}$ along with performance indicators for comparison with other systems. This section presents the basic criteria used to evaluate modern IDSs.

More precisely, to perform an evaluation experiment correctly a fundamental hypothesis must hold: the set $\mathbb{O}$ must be known and perfectly distinguishable from $\mathbb{I}$. In other words, this means that, $\mathbb{D}$ must be labeled with a *truth file*, i.e., a list of all the events known to be malicious. This allows to treat the ID problem as a classic classification problem, for which a set of well-established evaluation metrics are available. In particular, we are interested at calculating the following sets.

Definition 2.1.8 (*True Positives* (TPs)) The set of *true positives* is $TP := \{A_i \in \mathbb{A} \mid \exists O_j : f(A_i) = O_j\}$.

Where $f : \mathbb{A} \mapsto \mathbb{O}$ is a generic function that, given an alert A_i finds the corresponding intrusion O_j by parsing the truth file. The TP set is basically the set of alerts that are fired because a real intrusion has taken place. The perfect IDS is such that $TP \equiv \mathbb{O}$.

Definition 2.1.9 (*True Positives* (TPs)) The set of *false positives* is

$$FP := \{A_i \in \mathbb{A} \mid \nexists O_j : f(A_i) = O_j\}.$$

On the other hand, the alerts in FP are incorrect because no real intrusion can be found in the observed activity. The perfect IDS is such that $FP = \varnothing$.

Definition 2.1.10 (*True Negatives* (TNs)) The set of *true negatives* is

$$TN := \{I_j \in \mathbb{I} \mid \nexists A_i : f(A_i) = I_j\}.$$

Note that the set of TN does not contain alerts. Basically, it is the set of correctly unreported alerts. The perfect IDS is such that $TN \equiv \mathbb{I}$.

Definition 2.1.11 (*False Negatives* (FNs)) The set of *false negatives* is

$$FN := \{O_j \in \mathbb{O} \mid \nexists A_i : f(A_i) = O_j\}.$$

Similarly to TN, FN does not contain alerts. Basically, it is the set of incorrectly unreported alerts. The perfect IDS is such that $FN = \varnothing$. Note that, $TP + TN + FP + TN = 1$ must hold.

In this and other documents, the term *false alert* refers to $FN \cup FP$. Given the aforementioned sets, aggregated measures can be calculated.

Definition 2.1.12 (*Detection Rate* (DR)) The *detection rate*, or *true positive rate*, is defined as:

$$DR := \frac{TP}{TP + FN}.$$

The perfect IDS is such that $DR = 1$. Thus, the DR measures the detection capabilities of the system, that is, the amount of malicious events correctly classified and reported as alerts. On the other side, the *False Positive Rate* (FPR) is defined as follows.

Definition 2.1.13 (FPR) The *false positive rate* is defined as:

$$FPR := \frac{FP}{FP + TN}.$$

The perfect IDS is such that $FPR = 0$. Thus, the FPR measures the inaccuracies of the system, that is, the amount of legit events incorrectly classified and reported as alerts. There are also other metrics such as the *accuracy* and the *precision* which are often used to evaluate

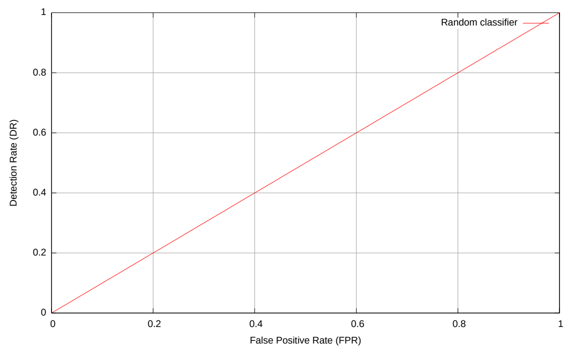


Figure 2.3: The ROC space.

information retrieval systems; however, these metrics are not popular for IDS evaluation.

The ROC analysis, originally adopted to measure transmission quality of radar systems, is often used to produce a compact and easy to understand evaluation of classification systems. Even though there is no standardized procedure to evaluate an IDS, the research community agrees on the use of ROC curves to compare the detection capabilities and quality of an IDS. A ROC curve is the plot of $DR = DR(FPR)$ and is obtained by tuning the IDS to trade off *False Positives* (FPs) against true positives. Without going into the details, each point of a ROC curve correspond to a fixed amount of DR and FPR calculated under certain conditions (e.g., sensitivity parameters). By modifying its configuration the quality of the classification changes and other points of the ROC are determined. The ROC space is plotted in Figure 2.3 along with the performances of a random classifiers, characterized by $DR = FPR$. The perfect, ideal IDS can increase its DR from 0 to 1 with $FPR = 0$: however, it must hold that $FPR \rightarrow 1 \Rightarrow DR \rightarrow 1$.

2.1.2 Alert Correlation

The problem of intrusion detection is challenging in todays' complex networks. In fact, it is common to have more than one IDS deployed, monitoring different segments and different aspects of the whole infrastructure (e.g., hosts, applications, network, etc.). The amount of

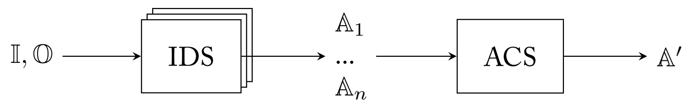


FIGURE 2.4: Abstract I/O model of an IDS with an alert correlation system.

alerts reported by a network of IDSs running in a complex computer infrastructure is larger, by several orders of magnitude, than what was common in the smaller networks monitored years ago. In such a context, network administrators are loaded by several alerts and long security reports often containing a non-negligible amount of FPs. Thus, the creation of a clean, compact, and unified view of the security status of the network is needed. This process is commonly known as alert correlation [Valeur et al., 2004] and it is currently one of the most difficult challenges of this research field. More precisely.

Definition 2.1.14 (Alert Correlation) The *alert correlation* is the identification of relations among alerts

$$A_1, A_2, A_i, \dots, A_L \in \mathbb{A}_1 \cup \mathbb{A}_2 \cup \cdots \cup \mathbb{A}_K$$

to generate a unique, more compact and comprehensive sequence of alerts $\mathbb{A}' = [A'_1, A'_2, \dots, A'_P]$.

A desirable property is that $\mathbb{A}'$ should be as complete as $\mathbb{A}_1 \cup \mathbb{A}_2 \cup \cdots \cup \mathbb{A}_K$ without introducing errors such as alerts that do not correspond to a real intrusion. As for the ID, alert correlation can be a manual, and tedious, task. Clearly, automatic alert correlation systems are more attractive and can be considered a complement of a modern IDS.

Definition 2.1.15 (Alert Correlation System) An *alert correlation system* is an automatic tool that performs the alert correlation task.

The overall IDS abstract model is complemented with an alert correlation engine as shown in Figure 2.4.

2.1.3 Taxonomic Dimensions

The first comprehensive taxonomy of IDSs has been proposed in [Debar et al., 1999] and revised in [Debar et al., 2000]. Another good survey appeared in [Axelsson, 2000a].

Compared to the classic survey found in the literature, this section complements the basic taxonomic dimensions by focusing on *modern* techniques. In particular, todays' intrusion detection approaches can be categorized by means of the specific modeling techniques appeared in recent research.

It is important to note that the taxonomic dimensions that are hereby suggested are not exhaustive, thus certain IDSs may not fit into it (e.g., [Costa et al., 2005; Portokalidis et al., 2006; Newsome and Song, 2005]). On the other hand, an exhaustive and detailed taxonomy would be difficult to read. To overcome this difficulty, in this section we describe a high-level, technique-agnostic taxonomy based on the dimensions summarized in Table 2.1; in each sub-section of Section 2.2, which focus on anomaly-based models, we expand the taxonomic dimensions by listing and accurately detailing further classification criteria.

2.1.3.1 Type of model

IDSs must be divided into two opposite groups: misuse-based *vs.* anomaly-based. The former create models of the *malicious* activity while the latter create models of *normal* activity. Misuse-based models look for *patterns* of malicious activity; anomaly-based models look for *unexpected* activity. In some sense, IDSs can either "blacklist" or "whitelist" the observed activity.

Typically, the first type of models consists in a database of all the known attacks. Besides requiring frequent updates —which is just a technical difficulty and can be easily automated— misuse-based systems assumes the feasibility of enumerating *all* the malicious activity.

Despite the limitation of being inherently incomplete, misuse-based systems widely adopted in the real-world [Roesch, 1999, 2009]. This is mainly due to their simplicity (i.e., attack models are triggered by means of pattern matching algorithms) and accuracy (i.e., they generate virtually no false alerts because an attack signature can either match or not).

Anomaly-based approaches are more complex because creating a

Feature	Misuse-based	Anomaly-based
Modeled activity:	Malicious	Normal
Detection method:	Matching	Deviation
Threats detected:	Known	Any
False negatives:	High	Low
False positives:	Low	High
Maintenance cost:	High	Low
Attack desc.:	Accurate	Absent
System design:	Easy	Difficult

Table 2.1: Duality between misuse- and anomaly-based intrusion detection techniques. Note that, an anomaly-based IDS can detect "Any" threat, under the assumption that an attack always generates a deviation in the modeled activity.

specification of normal activity is obviously a difficult task. For this reason, there are no well-established and widely-adopted techniques; instead, misuse models are as sophisticated as a pattern matching problem. In fact, the research on anomaly-based systems is very active.

These systems are effective only under the assumption that malicious activity, such as an attack or malware being executed, *always* produces sufficient deviations from normal activity such that models are triggered, i.e., anomalies. This clearly has the positive side-effect of requiring zero knowledge on the malicious activity, which makes these systems particularly interesting. The negative side-effect is their tendency of producing a significant amount of false alerts (the notion of "significant amount of false alerts" will be discussed in detail in Section 2.4.2).

Obviously, an IDS can benefit of both the techniques by, for instance, enumerating all the known attacks and using anomaly-based heuristics to prompt for suspicious activity. This practice is often adopted by modern anti-viruses.

It is important to remark that, differently from the other taxonomic dimensions, the distinction between misuse- and anomaly-based approaches is fundamental: they are based on opposite hypotheses and yield to completely different system designs and results. Their duality is highlighted in Table 2.1.

Example 2.1.1 (Misuse *vs.* Anomaly) A misuse-based system M and an anomaly-based system A process the same log containing a full dump of the system calls invoked by the kernel of an audited machine. Log entries are in the form:

```
<function_name>(<arg1_value>, <arg2_value>, ...)
```

The system M has the following simple attack model[1]:

```
if (function_name == ”read”) {
  /* ... */ if (match(decode(arg3_value), ”a{4}b{4}c{4}d{4}e{4}\
      f{4}...x{4}3RH˜TY7{33}QZjAXP0A0AkAAQ2AB2BB0BBAB\
      XP8ABuJIXkweaHrJwpf02pQzePMhyzWwSuQnioXPOHuBxKn\
      aQlkOjpJHIvKOYokObPPwRN1uqt5PA...” ))
  fire_alert(”VLC bug 35500 is being exploited!”); /* ... */
}
```

The simple attack signature looks for a pattern generated from the exploit. If the content of the buffer (i.e., `arg3_value`) that stores the malicious file matches the given pattern then an alert is fired. On the other hand, the system A has the following model, based on the sample character distribution of each file's content. Such frequencies are calculated during normal operation of the application.

```
/* ... */ cd['<'] = {0.1, 0.11} cd['a'] = {0.01, 0.2} cd['b'] =
{0.13, 0.23} /* ... */

b = decode(arg3_value);

if ( !(cd['c'][0] < count('c', b) < cd['c'][1]) ||\
     !(cd['<'][0] < count('<', b) < cd['<'][1]) ||\
     ... || ...)  fire_alert(”Anomalous content detected!”);
/* ... */
```

Obviously, more sophisticated models can be designed. The purpose of this example is that of highlighting the main differences between the two approaches.

A generalization of the aforementioned examples allows us to better define an anomaly-based IDS.

[1]Generated from the real world exploit `http://milw0rm.com/exploits/9303`

Definition 2.1.16 (Anomaly-based IDS) An *anomaly-based IDS* is a type of IDS that generate alerts $\mathbb{A}$ by relying on normal activity profiles (Definition 2.1.3).

2.1.3.2 System activity

IDSs can be classified based on the type of the activity they monitor. The classic literature distinguishes between network-based and host-based systems; *Network-based Intrusion Detection System* (NIDS) and *Host-based Intrusion Detection System* (HIDS), respectively. The former inspect network traffic (i.e., raw bytes sniffed from the network adapters), and the latter inspect the activity of the operating system. The scope of a network-based system is as large as the broadcast domain of the monitored network. On the other hand, the scope host-based systems is limited to the single host.

Network-based IDSs have the advantage of having a large scope, while host-based ones have the advantage of being fed with detailed information about the host they run on (e.g., process information, *Central Processing Unit* (CPU) load, number of active processes, number of users). This information is often unavailable to network-based systems and can be useful to refine a decision regarding suspicious activity. For example, by inspecting both the network traffic and the kernel activity, an IDS can filter the alerts regarding the Apache web server version 2.0.0 on all the hosts running version 2.0.2. On the other hand, network-based systems are centralized and are much more easy to manage and deploy. However, NIDS are limited to the inspection of unencrypted payload, while HIDS may have it decrypted by the application layer. For example, a NIDS cannot detect malicious HTTPS traffic.

The network stack is standardized (see Note 2.1.2), thus the definition of network-based IDS is precise. On the other hand, because of the immense variety of operating system implementations, a clear definition of "host data" lacks. Existing host-based IDSs analyze audit log files in several formats, other systems keep track of the commands issued by the users through the console. Some efforts have been made to propose standard formats for host data: *Basic Security Module* (BSM) and its modern re-implementation called OpenBSM [Watson and Salamon, 2006; Watson, 2006] are probably the most used by the research community as they allow developers to gather the full dump of the system calls before execution in the kernel.

Note 2.1.2 Although the network stack implementation may vary from system to system (e.g., Windows and Cisco platforms have different implementation of *Trasmission Control Protocol* (TCP)), it is important to underline that the notion of IP, TCP, HTTP *packet* is well defined in a system-agnostic way, while the notion of *operating system activity* is rather vague and by no means standardized.

Example 2.1.1 describes a sample host-based misuse detection system that inspects the arguments of the system calls. A similar, but more sophisticated, example based on network traffic is Snort [Roesch, 1999, 2009].

Note 2.1.3 (Inspection layer) Network-based systems can be further categorized by their protocol inspection capabilities, with respect to the network stack. webanomaly [Kruegel et al., 2005; Robertson, 2009] is network-based, in the sense that runs in promiscuous mode and inspects network data. On the other hand, it also HTTP-based, in the sense that decodes the payload and reconstructs the HTTP messages (i.e., request and response) to detect attacks against web applications.

2.1.3.3 Model construction method

Regardless of their type, misuse- or anomaly-based, models can be specified either manually or automatically. However, for their nature, misuse models are often manually written because they are based on the exhaustive enumeration of known malicious activity. Typically, these models are called attack signatures; the largest repository of manually generated misuse signatures is released by the Sourcefire Vulnerability Research Team^{TM} [Sourcefire, 2009]. Misuse signatures can be generated automatically: for instance, in [Singh et al., 2004] a method to build misuse models of worms is described. Similarly, low-interaction honeypots often uses malware emulation to automatically generate signatures. Two recently proposed techniques are [Portokalidis et al., 2006; Portokalidis and Bos, 2007].

On the other hand, anomaly-based models are more suitable for automatic generation. Most of the anomaly-based approaches in the literature focus on unsupervised learning mechanisms to construct models that precisely capture the normal activity observed. Manually specified models are typically more accurate and are less prone to FPs, although automatic techniques are clearly more desirable.

Example 2.1.2 (Learning character distributions) In Example 2.1.1, the described system A adopts a learning based character distribution model for strings. Without going into the details, the idea described in [Mutz et al., 2006] observes string arguments and estimate the characters' distribution over the *American Standard for Information Interxchange* (ASCII) set. More practically, the model is a histogram $H(c), \forall c \in 0, 255$, where $H(c)$ is the normalized frequency of character c. During detection, a χ^2 test is used to decide whether or not a certain sting is deemed anomalous.

Beside the obvious advantage of being resilient against evasion, this model requires no human intervention.

2.2 Relevant Anomaly Detection Techniques

Our research focuses on anomaly detection. In this section, the selected state of the art approaches are reviewed with particular attention to network-, host- and web-based techniques, along with the most influential approaches in the recent literature alert correlation. This section provides the reader with the basic concepts to understand our contributions.

2.2.1 Network-based techniques

Our research does not include network-based IDSs. Our contributions in alert correlation, however, leverage both network- and host-based techniques, thus a brief overview of the latest (i.e., proposed between 2001 and 2006) network-based detection approaches is provided in this section. We remark that all the techniques included in the following are based on TCP/IP, meaning that models of normal activity are constructed by inspecting the decoded network frames up to the TCP layer.

In Table 2.2 the selected approaches are marked with bullets to highlight their specific characteristics. Such characteristics are based on our analysis and experience, thus, other classifications may be possible. They are defined as follows:

Time refers to the use of *timestamp* information, extracted from network packets, to model normal packets. For example, normal packets may be modeled by their minimum and maximum inter-arrival time.

Header means that the TCP header is decoded and the fields are modeled. For example, normal packets may be modeled by the observed ports range.

Payload refers to the use of the payload, either at *Internet Protocol* (IP) or TCP layer. For example, normal packets may be modeled by the most frequent byte in the observed payloads.

Stochastic means that stochastic techniques are exploited to create models. For example, the model of normal packets may be constructed by estimating the sample mean and variance of certain features (e.g., port number, content length).

Deterministic means that certain features are modeled following a deterministic approach. For example, normal packets may be only those containing a specified set of values for the *Time To Live* (TTL) field.

Clustering refers to the use of clustering (and subsequent classification) techniques. For instance, payload byte vectors may be compressed using a *Self Organizing Map* (SOM) where class of different packets will stimulate neighbor nodes.

Note that, since recent research have experimented with several techniques and algorithms, mixed approaches exist and often lead to better results.

In [Mahoney and Chan, 2001] a mostly deterministic, simple detection technique is presented. During training, each field of the header of the TCP packets are extracted and tokenized into 4 bytes bins (for memory efficiency reasons). The tokenized values are clustered by means of their values and every time a new value is observed the clustering is updated. The detection approach is deterministic, since a packet is classified as anomalous if the values of its header do not match any of the clusters. Besides the fact of being completely unsupervised, this techniques detects between 50% and 75% of the probe and DoS attacks, respectively, in *Intrusion Detection eVALuation* (IDEVAL) 1999 [Lippmann et al., 2000]. Slight performance issues and a rate of 10 FPs per day (i.e., roughly, 1 false alert every 2 hours) are the only disadvantage of the approach.

The approach described in [Kruegel et al., 2002] reconstructs the payload stream for each service (i.e., port); this avoids to evade the detection mechanism by using packet fragmentation. In addition to this, a basic application inspection is performed to distinguish among the different types of request of the specific service, e.g., for HTTP the service type could be GET, POST, HEAD. The sample mean and variance of the content length are also calculated and the distribution of the bytes (interpreted as ASCII characters) found in the payload is estimated using simple histograms. The anomaly score used for detection aggregates information regarding the type of service, expected content length and payload distribution. With a low performance overhead and low FPR, this system is capable of detecting anomalous interactions with the application layer. One critique

Approach	Time	Header	Payload	Stochastic	Determ.	Clustering
[Mahoney and Chan, 2001]		●				●
[Kruegel et al., 2002]		●	●	●		
[Sekar et al., 2002]		●		●	●	
[Ramadas, 2003]			●			●
[Mahoney and Chan, 2003b]	●		●		●	
[Zanero and Savaresi, 2004]		●	●			●
[Wang and Stolfo, 2004]			●	●		
[Zanero, 2005b]		●	●			●
[Bolzoni et al., 2006]		●	●			●
[Wang et al., 2006]			●	●		

Table 2.2: Taxonomy of the selected state of the art approaches for network-based anomaly detection.

is that the system has not been tested on several applications other than *Domain Name System* (DNS).

Probably inspired by the misuse-based, finite-state technique described in [Vigna and Kemmerer, 1999], in [Sekar et al., 2002] the authors describe a system to learn the TCP specification. The basic finite state model is extended with a network of stochastic properties (e.g., the frequency of certain transitions, the most common value of a state attribute, the distribution of the values found in the fields of the IP packets) among states and transitions. Such properties are estimated during training and exploited at detection to implement smoother thresholds that ensure as low as 5.5 false alerts per day. On the other hand, the deterministic nature of the finite state machine detects attacks with a 100% DR.

Learning Rules for Anomaly Detection (LERAD), the system described in [Mahoney and Chan, 2003b] is an optimized rule mining algorithm that works well on data with tokenized domains such as the fields of TCP or HTTP packets. Although the idea implemented in LERAD can be applied at any protocol layer, it has been tested on TCP and HTTP but no more than the 64% of the attack in the testing dataset were detected. Even if the FPR is acceptable (i.e., 10 alerts per day) its limited detection capabilities worsen if real-world data is used instead of synthetic datasets such as IDEVAL. In [Tandon and Chan, 2003] the LERAD algorithm (Learning Rules for Anomaly Detection) is used to mine rules expressing "normal" values of arguments, normal sequences of system calls, or both. No relationship is learned among the values of different arguments; sequences and argument values are handled separately; the evaluation is quite poor however, and uses non-standard metrics.

Unsupervised learning techniques to mine pattern from payload of packets has been shown to be an effective approach. Both the network-based approaches described so far and other proposals [Labib and Vemuri, 2002; Mahoney, 2003] had to cope with data represented using a high number of dimensions (e.g., a vector with 1460 dimensions, that is the maximum number of bytes in the TCP payload). While the majority of the proposals circumvent the issue by ignoring the payload, the aforementioned issue is brilliantly solved in [Ramadas, 2003] and extended in *Unsupervised Learning IDS with 2-Stages Engine* (ULISSE) [Zanero and Savaresi, 2004; Zanero, 2005b] by exploiting the clustering capabilities of a SOM [Kohonen, 2000] with a faster algorithm [Zanero, 2005a], specifically designed for high-dimensional data. The payload of TCP packets is indeed compressed into the bi-dimensional grid representing the SOM, organized in such a way that class of packets can be quickly extracted. The approach relies on, and confirms, the assumption that the traffic belongs to a relatively small number of services and protocols that can be mapped onto a small number of clusters. Network packet are modeled as a multivariate time-series, where the variables include the packet class into the SOM plus some other features extracted from the header. At detection, a fast discounting learning algorithm for outlier detection [Yamanishi et al., 2004] is used to detect anomalous packets. Although the system has not been released yet, the prototype is able to reach a 66.7% DR with as few as 0.03% FPs. In comparison, one of the prototype dealing with payloads available in literature [Wang et al., 2005], the best overall result leads to the detection of 58.7% of the attacks, with a FPR that is between 0.1% and 1%. The main weakness of this approach is that it works at the granularity of the packets and thus might be prone to simple evasion attempts (e.g., by splitting an attack onto several malicious packets, interleaved with long sequence of legit packets). Inspired by [Wang et al., 2005] and [Zanero and Savaresi, 2004], [Bolzoni et al., 2006] has been proposed.

The approach presented in [Wang et al., 2005] differs from the one described in [Zanero and Savaresi, 2004; Zanero, 2005b] even though the underlying key idea is rather similar: byte frequency distribution. Both the two approaches, and also [Kruegel et al., 2002], exploit the distribution of byte values found in the network packets to produce some sort of "normality" signatures. [Wang et al., 2005] utilizes a simple clustering algorithm to aggregate similar packets and produce a smoother and more abstract signature, [Zanero and Savaresi, 2004; Zanero, 2005b] introduces the use of SOM to accomplish the task of finding classes of normal packets.

An extension to [Wang and Stolfo, 2004], which uses 1-grams, is described in [Wang et al., 2006] that uses higher-order, randomized n-grams to mitigate mimicry attacks. In addition, the newer approach does not estimate the frequency distribution of the n-grams, which causes many FPs if n increases. Instead, it adopts a filtering technique to compress the n-grams into memory efficient arrays of bits. This decreased the FPR of about two orders of magnitude.

2.2.2 Host-based techniques

A survey of the latest (i.e., proposed between 2000 and 2009) host-based detection approaches is provided in this section. Most of the techniques leverage system call invoked by the kernel to create models of normal behavior of processes.

In Table 2.3 the selected approaches are marked with bullets to highlight their specific characteristics. Such characteristics are based on our analysis and experience, thus, other classifications may be possible. They are defined as follows:

Syscall refers, in general, to the use of system calls to characterize normal host activity. For example, a process may be modeled by the stream of system calls invoked during normal operation.

Stochastic means that stochastic techniques are exploited to create models. For example, the model of normal processes may be constructed by estimating the sample mean and variance of certain features (e.g., length of the `open`'s `path` argument).

Deterministic means that certain features are modeled following a deterministic approach. For example, normal processes may be only those that use a fixed number, say 12, of file descriptors.

Comprehensive approaches are those that have been extensively developed and, in general, incorporate a rich set of features, beyond the proof-of-concept.

Context refers to the use of context information in general. For example, the normal behavior of processes can be modeled also by means of the number of the environmental variables utilized or by the sequence of system calls invoked.

Data means that the data flow is taken into account. For example, normal processes may be modeled by the set of values of the system call arguments.

Forensics means that the approach has been also evaluated for off-line, forensics analysis.

Our contributions are included in Table 2.3 and are detailed in Chapter 3. Note that, since recent research have experimented with

Approach	Syscall	Determ.	Stochastic	Comprehen.	Context	Data	Forensics
[Lee and Stolfo, 2000]			●	●			
[Sekar et al., 2001]	●	●					
[Wagner and Dean, 2001]	●	●					
[Tandon and Chan, 2003]	●	●			●		
[Kruegel et al., 2003a]	●		●	●	●	●	
[Zanero, 2004]	●		●		●		
[Giffin et al., 2005]	●	●			●	●	
[Mutz et al., 2006]	●		●	●		●	
[Bhatkar et al., 2006]	●	●			●	●	
[Mutz et al., 2007]	●		●	●		●	
[Fetzer and Suesskraut, 2008]	●	●			●	●	
[Maggi et al., 2008]	●		●		●	●	●
[Maggi et al., 2009a]	●		●	●	●	●	
[Frossi et al., 2009]	●	●	●		●	●	

Table 2.3: Taxonomy of the selected state of the art approaches for host-based anomaly detection. Our contributions are highlighted.

several techniques and algorithms, mixed approaches exist and often lead to better results.

Host-based anomaly detection has been part of intrusion detection since its very inception: it already appears in the seminal work [Anderson, 1980]. However, the definition of a set of statistical characterization techniques for events, variables and counters such as the CPU load and the usage of certain commands is due to [Denning, 1987]. The first mention of intrusion detection through the analysis of the sequence of syscalls from system processes is in [Forrest et al., 1996], where "normal sequences" of system calls are considered. A similar idea was presented earlier in [Ko et al., 1994], which proposes a misuse-based idea by manually describe the canonical sequence of calls of each and every program, something evidently impossible in practice.

In [Lee and Stolfo, 2000] a set of models based on data mining techniques is proposed. In principle, the models are agnostic with respect to the type of raw event collected, which can be user activity (e.g., login time, CPU load), network packets (e.g., data collected with tcpdump), or operating system activity (e.g., system call traces collected with OpenBSM). Events are processed using automatic classification algorithms to assign labels drawn from a finite set. In addition, frequent episodes are extracted and, finally, association rules among events are mined. Such there algorithms are combined together at detection time. Events marked with wrong labels, unexpectedly frequent episodes or rule violations will all trigger alerts.

Alternatively, other authors proposed to use static analysis, as opposed to dynamic learning, to profile a program's normal behavior. The technique described in [Wagner and Dean, 2001] combines the benefits of dynamic and static analysis. In particular, three models are proposed to automatically derive a specification of the application behavior: call-graph, context-free grammars (or non-deterministic pushdown automata), and digraphs. All the models' building blocks are system calls. The call-graph is statically constructed and then simulated, while the program is running, to resolve non-deterministic paths. In some sense, the context-free grammar model —called abstract stack— is the evolution of the call-graph model as it allows to keep track of the state (i.e., call stack). The digraph —actually k-graph— model is keeps track of k-long sequences of system calls from an arbitrary point of the execution. Despite is simplicity, which ensures a negligible performance overhead with respect to the others, this model achieves the best detection precision.

In [Tandon and Chan, 2003] the LERAD algorithm (Learning Rules for Anomaly Detection) is described. Basically, it is a learning system to mine rules expressing normal values of arguments, normal sequences of system calls, or both. In particular, the basic algorithm learns rules in the form $A = a, B = b, \cdots \Rightarrow X \in \{x_1, x_2, \dots\}$ where uppercase letters indicate parameter names (e.g., `path`, `flags`) while lowercase symbols indicate their corresponding values. For some reason, the rule-learning algorithm first extracts random pairs from the training set to generate a first set of rules. After this, two optimization steps are run to remove rules with low coverage and those prone to generate FPs (according to a validation dataset). A system call is deemed anomalous if no matching rule is found. A similar learning and detection algorithm is run among sequences of system calls. The main advantage of the described approach is that no relationship is learned among the values of different arguments of the same system call. Beside the unrealistic assumption regarding the availability of a labeled validation dataset, another side issue is that the evaluation is quite poor and uses non-standard metrics.

LibAnomaly [Kruegel et al., 2003a] is a library to implement stochastic, self-learning, host-based anomaly detection systems. A generic anomaly detection model is trained using a number of system calls from a training set. At detection time, a likelihood rating is returned by the model for each new, unseen system call (i.e., the probability of it being generated by the model). A confidence rating can be computed at training for any model, by determining how well it fits its training set; this value can be used at runtime to provide additional information on the reliability of the model. When data is available, by using cross-validation, an overfitting rating can also be optionally computed.

LibAnomaly includes four basic models. The string length model computes, from the strings seen in the training phase, the sample mean μ and variance σ^2 of their lengths. In the detection phase, given l, the length of the observed string, the likelihood p of the input string length with respect to the values observed in training is equal to one if $l < \mu + \sigma$ and $\frac{\sigma^2}{(l-\mu)^2}$ otherwise. As mentioned in the Example 2.1.2, the character distribution model analyzes the discrete probability distribution of characters in a string. At training time, the so called ideal character distribution is estimated: each string is considered as a set of characters, which are inserted into an histogram, in decreasing order of occurrence, with a classical rank order/frequency representation. During the training phase, a compact representation of mean and variance of the frequency for each rank is computed. For detection, a χ^2 Pearson test returns the likelihood that the observed string histogram comes from the learned model. The structural inference model encodes the syntax of strings. These are simplified before the analysis, using a set of reduction rules, and then used to generate a probabilistic grammar by means of a Markov model induced by exploiting a Bayesian merging procedure, as described in [Stolcke and Omohundro, 1994c, 1993b, 1994b]. The token search model is applied to arguments which contain flags or modes. During detection, if the field has been flagged as a token, the input is compared against the stored values list. If it matches a former input, the model returns 1 (i.e., not anomalous), else it returns 0 (i.e., anomalous).

In [Zanero, 2004] a general Bayesian framework for encoding the behavior of users is proposed. The approach is based on hints drawn from the quantitative methods of ethology and behavioral sciences. The behavior of a user interacting with a text-based console is encoded as a *Hidden Markov Model* (HMM). The observation set includes the commands (e.g., `ls`, `vim`, `cd`, `du`) encountered during training. Thus, the system learns the user behavior in terms of the model structure (e.g., number of states) and parameters (e.g., transition matrix, emission probabilities). At detection, unexpected or out of context commands are detected as violations (i.e., lower value) of the learned probabilities. One of the major drawbacks of this system is its applicability to real-world scenarios: in fact, todays' host-based threats perform more sophisticated and stealthy operations than invoking commands.

In [Giffin et al., 2005] an improved version of [Wagner and Dean, 2001] is presented. It is based on the analysis of the binaries and incorporates the execution environment as a model constraint. More precisely, the environment is defined as the portion of input known at process load time and fixed until it exits. In addition, the technique deals with dynamically-linked libraries and is capable of constructing the data-flow analysis even across different shared-objects.

An extended version of LibAnomaly is described in [Mutz et al., 2006]. The basic detection models are essentially the same. In addition, the authors exploit Bayesian networks instead of naive thresholds to classify system calls according to each model output. This

```
source_dir = dir; target_file = file;
      out = open(target_file , WR);
push(source_dir); while ((dir_name =
      pop()) != NULL) {
  d = opendir(dir_name); foreach (
        dir_entry ∈ d) {
    if (isdirectory(dir_entry))
      push(dir_entry); else {
      in = open(dir_entry , RD); read(in
            , buf); write(out, buf);
      close(in);
    } }
} close(out); return 0;
}
```

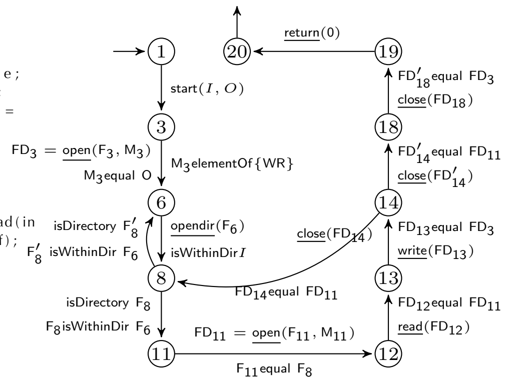


Figure 2.5: A data flow example with both unary and binary relations.

results in an improvement in the DRs. Some of our work described in Section 3 is based upon this and the original version of LibAnomaly.

Data-flow analysis has been also recently exploited in [Bhatkar et al., 2006], where an anomaly detection framework is developed. Basically, it builds an *Finite State Automaton* (FSA) model of each monitored program, on top of which it creates a network of relations (called *properties*) among the system call *arguments* encountered during training. Such a network of properties is the main difference with respect to other FSA based IDSs. Instead of a pure *control flow* check, which focuses on the behavior of the software in terms of sequences of system calls, it also performs a so called *data flow* check on the internal variables of the program along their existing cycles. This approach has really interesting properties, among which the fact that not being stochastic useful properties can be demonstrated in terms of detection assurance. On the other hand, though, the set of relationships that can be learned is limited (whereas the relations encoded by means of the stochastic models we describe in Section 3.2.3 are not decided a priori and thus virtually infinite). The relations are all deterministic, which leads to a brittle detection model potentially prone to FPs. Finally, it does not discover any type of relationship between different arguments of the same call.

This knowledge is exploited in terms of *unary* and *binary* relationships. For instance, if an open system call always uses the same filename at the same point, a unary property can be derived. Similarly, relationships among two arguments are supported, by inference over the observed sequences of system calls, creating constraints for the detection phase. Unary relationships include equal (the value of a given argument is always constant), elementOf (an argument can take a limited set of values), subsetOf (a generalization of elementOf, indicating that an argument can take multiple values, all of which drawn from a set), range (specifies boundaries for numeric arguments), isWithinDir (a file argument is always contained within a specified directory), hasExtension (file extensions). Binary relationships include: equal (equality between system call operands), isWithinDir (file located in a specified directory; contains is the opposite), hasSameDirAs, hasSameBaseAs, hasSameExtensionAs (two arguments have a common directory, base directory or extension, respectively).

The behavior of each application is logged by storing the *Process IDentifier* (PID), the *Program Counter* (PC), along with the system calls invoked, their arguments and returned value. The use of the PC to identify the states in the FSA stands out as an important difference from other approaches. The PC of each system call is determined through *stack unwinding* (i.e., going back through the activation records of the process stack until a valid PC is found). The technique obviously handles process cloning and forking.

The learning algorithm is rather simple: each time a new value is found, it is checked against all the known values of the same type. Relations are inferred for each execution of the monitored program and then pruned on a set intersection basis. For instance, if relations R_1 and R_2 are learned from an execution trace T_1 but R_1 only is satisfied in trace T_2, the resulting model will not contain R_2. Such a process is obviously prone to FPs if the training phase is not exhaustive, because invalid relations would be kept instead of being discarded. Figure 2.5 shows an example (due to [Bhatkar et al., 2006]) of the final result of this process. During detection, missing transitions or violations of properties are flagged as alerts. The detection engine keeps track of the execution over the learned FSA, comparing transitions and relations with what happens, and raising an alert if an edge is missing or a constraint is violated.

This FSA approach is promising and has interesting features especially in terms of detection capabilities. On the other hand, it only takes into account relationships between different types of arguments. Also, the set of properties is limited to pre-defined ones and totally deterministic. This leads to a possibly incomplete detection model potentially prone to false alerts. In Section 3.3 we detail how our approach improves the original implementation.

Another approach based on the analysis of system calls is [Fetzer and Suesskraut, 2008]. In principle, the system is similar to the behavior-based techniques we mentioned before. However, the authors have tried to overcome two limitations of the learning-based approaches which, typically, have high FPRs and require a quite ample training set. This last issue is mitigated by adopting a completely different approach: instead of requiring training, the system administrator is required to specify a set of small whitelist-like models of the desired behavior of a certain application. At runtime, these models are evolved and adapted to the particular context the protected application runs into; in particular, the system exploits taint analysis to update a system call model on-demand. This system can offer very high levels of protection but the effort required to specify the initial model may not be so trivial; however, the effort may be worth for mission-critical applications on which customized hardening would be needed anyways.

2.2.3 Web-based techniques

A survey of the latest (i.e., proposed between 2003 and 2009) host-based detection approaches is provided in this section. All the techniques included in the following are based on HTTP, meaning that models of normal activity are constructed either by inspecting the decoded network frames up to the HTTP layer, or by acting as reverse HTTP proxies.

In Table 2.4 the selected approaches are marked with bullets to highlight their specific characteristics. Such characteristics are based on our analysis and experience, thus, other classifications may be possible. They are defined as follows:

Adaptive refers to the capability of self-adapting to variations in the normal behavior.

Stochastic means that stochastic techniques are exploited to create models. For example, the model of normal HTTP requests may be constructed by estimating the sample mean and variance of certain features (e.g., length of the string parameters contained in a POST request).

Deterministic means that certain features are modeled following a deterministic approach. For example, normal HTTP sessions may be only those that are generated by a certain finite state machine.

Comprehensive approaches are those that have been extensively developed and, in general, incorporate a rich set of features, beyond the proof-of-concept.

Response indicates that HTTP responses are modeled along with HTTP requests. For instance, normal HTTP responses may be modeled with the average number of `<script />` nodes found in the response body.

Session indicates that the concept of web application session is taken into account. For example, normal HTTP interactions may be modeled with the sequences of paths corresponding to a stream of HTTP requests.

Data indicates that parameters (i.e., GET and POST variables) contained into HTTP requests are modeled. For example, normal HTTP request may be modeled as the cardinality of string parameters in each request.

Our contributions are included in Table 2.4 and detailed in Chapter 4. Note that, since recent research have experimented with several techniques and algorithms, mixed approaches exist and often lead to better results.

Anomaly-based detectors specifically designed to protect web applications are relatively recent. They have been first proposed in [Cho and Cha, 2004], where a system to detect anomalies in web application sessions is described. Like most of the approaches in the literature, this technique assumes that malicious activity expresses itself in the parameters found into HTTP requests. In the case of this tool, such data is parsed from the access logs. Using Bayesian technique to assign a probability score to the k-sequences ($k = 3$ in the experiments) of requested resources (e.g., `/path/to/page`), the system can spot out unexpected sessions. Even though this approach has been

Approach	Adaptive	Stochastic	Deterministic	Comprehen.	Response	Session	Data
[Cho and Cha, 2004]		●				●	
[Kruegel et al., 2005]		●		●			●
[Ingham et al., 2007]			●			●	●
[Criscione et al., 2009]		●		●	●	●	●
[Song et al., 2009]		●					●
[Maggi et al., 2009c]	●		●		●	●	●
[Robertson et al., 2009]	●	●					●

Table 2.4: Taxonomy of the selected state of the art approaches for web-based anomaly detection. Our contributions are highlighted.

poorly evaluated, it proposed the basic ideas on which the current research is still based.

The first technique to accurately model the normal behavior of web application parameters is described in [Kruegel et al., 2005]. This approach is implemented in webanomaly, a tool that can be deployed in real-world scenarios. In some sense, webanomaly [Robertson, 2009] is the adaptation of LibAnomaly models to capture the normal features of the interaction between client and server-side applications through the HTTP protocol. Instead of modeling a system call and its arguments, the same models are mapped onto resources and their parameters (e.g., `?p=1&show=false`). Obviously, parameters are the focus of the analysis which employs string lenght models, token finder models, and so forth; in addition, sessions features are captured as well in terms of sequences of resources. In recent versions of the tools, webanomaly incorporated models of HTTP responses similar to those described in [Criscione et al., 2009] (see Section 4.1.2). Besides the features shared with [Kruegel et al., 2005], the approach models the *Document Object Model* (DOM) to enhance the detection capabilities against SQL injection and XSS attacks. In Section 4.3 an approach that exploit HTTP responses to *detect changes* and update *other* anomaly models is described.

The approach described in [Ingham et al., 2007] learns deterministic models, FSA, of HTTP requests. The idea of extracting resources and parameters is similar to that described in [Kruegel et al., 2005]. However, instead of adopting sophisticated models such as HMM to encode strings' grammar, this system applies drastic reductions to the parameters values. For instance, dates are mapped to $\{0, 1\}$ where 1 indicates that the format of the date is known, 0 otherwise; filenames are replaced with either a length or the extension, if the file-type is known; and so forth. For each request, the output of this step is a list of tokens that represent the states of the FSA. Transitions are labeled with the same tokens by processing them in chronological order (i.e., as they appear in the request). In some sense, this approach can be considered a porting to the web domain of the techniques used to model process behavior by means of FSA [Wagner and Dean, 2001; Bhatkar et al., 2006].

A tool to protect against code-injection attacks has been recently proposed in [Song et al., 2009]. The approach exploits a mixture of Markov chains to model legitimate payloads at the HTTP layer. The computational complexity of n-grams with large n is solved using Markov chain factorization, making the system algorithmically efficient.

2.3 Relevant Alert Correlation Techniques

A survey of the latest (i.e., proposed between 2001 and 2009) alert correlation approaches is provided in this section.

In Table 2.5 the selected approaches are marked with bullets to highlight their specific characteristics. Such characteristics are based on our analysis and experience, thus, other classifications may be possible. They are defined as follows:

Formal means that formal methods are used to specify rigorous models used in the correlation process. For instance, the relations among alerts are specified by means of well defined first-order logic formulae.

Verification means that the success of each alert is taken into account. For instance, all the alerts related to attacks against port 80 are discarded if the target system does not run HTTP services.

Stochastic means that stochastic techniques are exploited to create models. For example, statistic hypothesis tests may be used to decide correlation among stream of alerts.

Comprehensive approaches are those that have been extensively developed and, in general, incorporate a rich set of features, beyond the proof-of-concept.

Time refers to the use of *timestamp* information extracted from alerts. For example, alerts streams may be modeled as time series.

Impact refers to techniques that take into account the impact (e.g., the cost) of handling alerts. For instance, alerts regarding non-business critical machines are marked low priority.

Clustering refers to the use of clustering (and subsequent classification) techniques. For instance, similar alerts can be grouped together by exploiting custom distance metrics.

Our contributions are included in Table 2.5 and are detailed in Chapter 5. Note that, since recent research have experimented with several techniques and algorithms, mixed approaches exist and often lead to better results.

Approach	Formal	Verif.	Comprehen.	Time	Stochastic	Clustering	Impact
[Debar and Wespi, 2001]	●						
[Valdes and Skinner, 2001]					●		
[Porras et al., 2002]							●
[Cuppens and Miage, 2002]			●			●	
[Morin et al., 2002]	●						
[Julisch and Dacier, 2002]						●	
[Qin and Lee, 2003]				●	●	●	
[Valeur et al., 2004]		●	●	●			
[Kruegel and Robertson, 2004]		●					
[Viinikka et al., 2006]				●	●		
[Maggi and Zanero, 2007]				●	●		
[Frias-Martinez et al., 2008]					●	●	
[Maggi et al., 2009b]				●			

Table 2.5: Taxonomy of the selected state of the art approaches for alert correlation. Our contributions are highlighted.

A deterministic intrusion detection technique adapted for alert correlation is shown in [Eckmann et al., 2000]. The use of finite state automata enables for complex scenario descriptions, but it requires known scenarios signatures. It is also unsuitable for pure anomaly detectors which cannot differentiate among different types of events. Similar approaches, with similar strengths and shortcomings but different formalisms, have been tried with the specification of pre- and post-conditions of the attacks [Templeton and Levitt, 2000], sometimes along with time-distance criteria [Ning et al., 2004]. It is possible to mine scenario rules directly from data, either in a supervised [Dain and Cunningham, 2001] or unsupervised [Julisch and Dacier, 2002] fashion.

Statistical techniques have been also proposed, for instance E-MERALD implements an alert correlation engine based on probabilistic distances [Valdes and Skinner, 2001] which relies on a set of similarity metrics between alerts to fuse "near" alerts together. Unfortunately, its performance depends on appropriate choice of weighting parameters.

The best examples of algorithms that do not require such features are based on time-series analysis and modeling. For instance, [Viinikka et al., 2006] is based on the construction of time-series by counting the number of alerts occurring into sampling intervals; the exploitation of trend and periodicity removal algorithms allows to filter out predictable components, leaving *real* alerts only as the output. More than a correlation approach, this is a false-positive and noise-suppression approach, though.

In [Qin and Lee, 2003] an interesting algorithm for alert correlation which seems suitable also for anomaly-based alerts is proposed. Alerts with the same feature set are grouped into collections of time-sorted items belonging to the same "type" (following the concept of type of [Viinikka et al., 2006]). Subsequently, frequency time series are built, using a fixed size sliding-window: the result is a time-series for each collection of alerts. The prototype then exploits the *Granger Causality Test* (GCT) [Thurman and Fisher, 1998], a statistical hypothesis test capable of discovering causality relationships between two time series when they are originated by linear, stationary processes. The GCT gives a stochastic measure, called *Granger Causality Index* (GCI), of how much of the history of one time series (the supposed cause) is needed to "explain" the evolution of the other one (the supposed consequence, or target). The GCT is based on the estimation of two models: the first is an *Auto Regressive* (AR) model, in which future samples of the target are modeled as influenced only by past samples of the target itself; the second is an *Auto Regressive Moving Average eXogenous* (ARMAX) model, which also takes into account the supposed cause time series as an exogenous component. A statistical F-test built upon the model estimation errors selects the best-fitting model: if the ARMAX fits better, the cause effectively influences the target.

In [Qin and Lee, 2003] the unsupervised identification of causal relationships between events is performed by repeating the above procedure for each couple of time-series. The advantage of the approach is that it does not require prior knowledge (even if it may use attack probability values, if available, for an optional prioritization phase). However, in Section 5 we show that the GCT fails however in recognizing "meaningful" relationships between IDEVAL attacks.

Techniques based on the reduction of FPs in anomaly detection systems has also been studied in [Frias-Martinez et al., 2008]. Similar behavioral profiles for individual hosts are grouped together using a k-means clustering algorithm. However, the distance metric used was not explicitly defined. Coarse network statistics such as the average number of hosts contacted per hour, the average number of packets exchanged per hour, and the average length of packets exchanged per hour are all examples of metrics used to generate behavior profiles. A voting scheme is used to generate alerts, in which alert-triggering events are evaluated against profiles from other members of that cluster. Events that are deemed anomalous by all members generate alerts.

Last, as will be briefly explained in Chapter 5, the alert correlation task may involve alert verification, i.e., before reporting an alert, a procedure is run to *verify* whether or not the attack actually left some traces or, in general, had some effect. For example, this may involve checking whether a certain TCP port, say, 80, is open; if not, all alerts regarding attacks against the protected HTTP server may be safely discarded, thus reducing the FPR. Although an alert correlation system would benefit from such techniques, we do not review them, since our focus is on the actual *correlation* phase, i.e., recognizing *related* events, rather than pruning uninteresting events. The interested reader may refer to a recent work [Bolzoni et al., 2007] that, beside describing an implementation of a novel verification system, introduces the alert verification problem and provides a comprehensive review of the related work.

2.4 Evaluation Issues and Challenges

The evaluation of IDSs is *per sé* an open, and very difficult to solve, research problem. Besides two attempts of proposing a rigorous methodology for IDS evaluation [Puketza et al., 1996, 1997], there are no standard guidelines to perform this task. This problem is magnified by the lack of a reliable source of test data, that is a well-known issue. However, building a dataset to conduct repeatable experiments is clearly a very difficult task because it is nearly impossible to reproduce the activity of a real-world computer infrastructure. In particular:

- for *privacy reasons*, researchers cannot audit an infrastructure and collect arbitrary data; in the best case, the use of obfuscation and anonymization of the payload (e.g., character substitution on text-based application protocol) to protect the privacy, produces unrealistic content that will make inspection techniques adopted by many IDS to fail. For instance, changes in the syntax and lexicon of strings have negative consequences for models such as the character distribution estimator described in Example 2.1.2.

- System activity collected from real-world infrastructures inevitably contain intrusions and not all of them are known in advance (i.e., 0-day attacks); this makes the generation of a *truth file* a very difficult task. To workaround this problem, it is a common practice to filter out known attacks using misuse-based systems such as Snort and use the alert log as the truth file. Unfortunately, this implicitly assumes that the chosen misuse-based system is the baseline of the evaluation (i.e., the best tool). On the other hand, anomaly-based systems are supposed to detect unknown attacks; thus, using the alert log as the truth file makes the evaluation of such systems completely meaningless.

- In the past, two attempts have been made to simulate the user activity on a comprehensive computer infrastructure, i.e., a military computer network, to collect clean, background audit data to train IDSs based on unsupervised learning techniques. Even though this ensures that the traffic contain no intrusions, the approach has been shown to have at least two types of shortcomings. The first is described in Section 2.4.1. The second problem is that, the attacks included (labeled in advance) do once again represent only known attacks since the exploits are taken from public repositories. Thus, IDSs cannot be tested against realistic intrusions such as custom attacks against proprietary and never-exploited-before systems.

Attempts to partially solve these issues can be divided into two groups. Some approaches proposes automated mechanisms to *generating* testing datasets, while other concentrate on the *methodologies* used to perform the evaluation task. Notably, in [Cretu et al., 2008b] a traffic sanitization procedure is proposed in order to clean the background traffic; however, it is not clear to what extent this method is substantially different from running an arbitrary anomaly-based IDS to filter out attacks. Examples of publicly available, but obsolete or unreliable, datasets are [Lippmann et al., 1999, 2000; Potter, 2006; Swartz, 2009; S. and D., 1999]: as exemplified by the critique described in the next section, however, all these dataset are either unusable because of their lack of a *truth file* or are extremely biased with respect to the real world. Among the methodological approaches, [Vigna et al., 2004] proposes a tool to automatically test the effectiveness of evasion techniques based on mutations of the attacks. Instead, [Lee and Xiang, 2001] proposes alternative metrics to evaluate the detection capabilities, [Sharif et al., 2007] focuses on the evaluation issues in the case of host-based IDSs and defines some criteria to correctly calculate their accuracy.

Recently, probably inspired by [Potter, 2006], the research is moving toward mechanisms to instrument large computer security exercises [Augustine et al., 2006] and contests such as the DEFCON *Capture The Flag* (CTF)[2] with the goal of collecting datasets to test IDS. The most up-to-date example is [Sangster et al., 2009], which describes the efforts made to collect the public dataset available at [Sangster, 2009]. This dataset have the advantage of containing a rich variety of attacks including 0-days and custom, unpublished exploits. However, because of the aforementioned reasons, this approach fails once again on the labeling phase since Snort is used to generate the truth file. A side question is to what extent the background traffic

[2]Details available at `www.defcon.org`

represent the real activity on the Internet. In fact, since the competition was run in the controlled environment of a private network, the users tend to behave in a different way, not to mention that most of the participants are experienced computer users.

2.4.1 Regularities in audit data of IDEVAL

IDEVAL is basically the only dataset of this kind which is freely available along with truth files; in particular we used the 1999 dataset [Lippmann et al., 2000]. These data are artificially generated and contain both network and host auditing data. A common question is how realistic these data are. Many authors already analyzed the network data of the 1999 dataset, finding many shortcomings [McHugh, 2000; Mahoney and Chan, 2003a]. Our own analysis, published in [Maggi et al., 2009a], of the 1999 host-based auditing data revealed that this part of the dataset is all but immune from problems. The first problem is that in the training datasets there are too few execution instances for each software, in order to properly model its behavior, as can be seen in Table 3.5. Out of (just) 6 programs present, for two (`fdformat` and `eject`), only a handful of executions is available, making training unrealistically simple.

The number of system calls used is also extremely limited, making execution flows very plain. Additionally, most of these executions are similar, not covering the full range of possible execution paths of the programs (thus causing overfitting of any anomaly model). For instance, in Figure 2.6 we have plotted the frequency of the length (in system calls) of the various executions of `telnetd` on the training data. The natural clustering of the data in a few groups clearly shows how the executions of the program are sequentially generated with some script, and suffer of a lack of generality.

System calls arguments show the same lack of variability: in all the training dataset, all the arguments of the system calls related to `telnetd` belong to the following set:

```
fork, .so.1, utmp, wtmp, initpipe, exec, netconfig,
        service_door, :zero, logindmux, pts
```

The application layer contains many flaws, too. For instance, the FTP operations (30 sessions on the whole) use a very limited subset of file (on average 2 per session), and are performed always by the same users on the same files, for a limitation of the synthetic

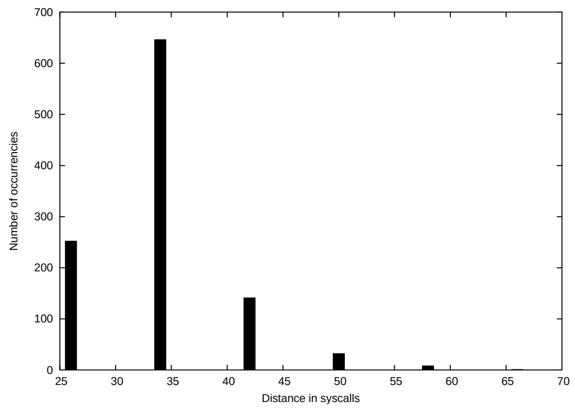


FIGURE 2.6: `telnetd`: distribution of the number of other system calls among two `execve` system calls (i.e., distance between two consecutive `execve`).

generator of these operations. In addition, during training, no uploads or idle sessions were performed. Command executions are also highly predictable: for instance, one script always execute a cycle composed of `cat`, `mail`, `mail` again, and at times `lynx`, sometimes repeated twice. The same happens (but in a random order) for `rm`, `sh`, `ps` and `ls`. In addition, a number of processes have evidently crafted names (e.g. `logout` is sometimes renamed `lockout` or `log0ut`); the same thing happens with path names, which are sometimes different (e.g. `/usr/bin/lynx` or `/opt/local/bin/lynx`), but an analysis shows that they are the same programs (perhaps symbolic links generated to create noise over the data). The combination of the two creates interesting results such as `/etc/loKout` or `/opt/local/bin/l0gout`. In a number of cases, processes `lynx`, `mail` and `q` have duplicate executions with identical PID and timestamps, and with different paths and/or different arguments; this is evidently an inexplicable flaw of the dataset. We also found many program executions to be curiously meaningless. In fact, the BSM traces of some processes contain just `execve` calls, and this happens for 28% of the programs in the testing portion dataset (especially for those with a crafted name, like `loKout`). It is obvious that testing an host-based IDS with one-syscall-long sequences does not make a lot of sense, not to talk about the relevance of training against such sequences.

An additional problem is that since 1999, when this dataset was created, everything changed: the usage of network protocols, the protocols themselves, the operating systems and applications used. For instance, all the machines involved are Solaris version 2.5.1 hosts, which are evidently ancient nowadays. The attacks are similarly outdated: the only attack technique used are buffer overflows, and all the instances are detectable in the `execve` system call arguments. Nowadays attackers and attack type are much more complex than this, operating at various layers of the network and application stack, with a wide range of techniques and scenarios that were just not imaginable in 1999.

To give an idea of this, we were able to create a detector which finds all the buffer overflow attacks without any FP: a simple script which flags as anomalous any argument longer than 500 characters can do this (because all the overflows occur in the parsing of the command line, which is part of the parameters of the `execve` system call which originates the process). This is obviously unrealistic.

Because of the aforementioned lack of alternatives, most existing researches use IDEVAL. This is a crucial factor: any bias or error in the dataset has influenced, and will influence in the future, the very basic research on this topic.

2.4.2 The base-rate fallacy

The ID is a classification task, thus can be formalized as a Bayesian classification problem. In particular FPR and DR can be defined as probabilities. More precisely, let us define the following event predicates:

- $O =$"Intrusion", $\neg O =$"Non-intrusion";
- $A =$"Alert reported", $\neg A =$"No alert reported".

Then, we can re-write DR and FPR as follows.

- $DR = P(A \mid O)$, i.e., the probability to classify an intrusive event as an actual intrusion.
- $FPR = P(A \mid \neg O)$, i.e., the probability to classify a legit event as an intrusion.

Given the above definition and the Bayes' theorem, two measures that are more interesting than DR and FPR can be calculated.

Definition 2.4.1 (Bayesian DR) The *Bayesian DR* [Axelsson, 2000b] is defined as:

$$P(O \mid A) = \frac{P(O) \cdot P(A \mid O)}{P(O) \cdot P(A \mid O) + P(\neg O) \cdot P(A \mid \neg O)}.$$

$P(O)$ is the *base-rate* of intrusions, i.e., the probability for an intrusion to take place, regardless of the presence of an IDS. The Bayesian DR not only quantifies the probability for an alert to indicate an intrusion, it also take into account how frequently an intrusion really happens. As pointed out by [Axelsson, 2000b], in this equation the $FPR = P(A \mid \neg O)$ is strictly dominated by $P(\neg O)$. In fact, in the real world, $P(O)$ is very low (e.g., 10^{-5}, given two intrusions per day, 1,000,000 audit records per day and 10 records per intrusion) and thus $P(O) \rightarrow 1$. This phenomenon, called *base-rate fallacy*, magnifies the presence of FP. In fact, even in the unrealistic case of $DR = 1$, a very low $FPR = 10^{-5}$ quickly drops the DR to 0.0066, which is three orders of magnitude below 1.

Besides its impact on the evaluation of an IDS, the base-rate fallacy has a practical impact. When inspecting the alerts log, the security officer would indeed tend to safely ignore most of the alerts because the past alarms have been shown to be incorrect.

2.5 Concluding Remarks

In this chapter we first introduced the basic concepts and definitions of ID, including the most relevant issues that arise during experimental evaluation. ID techniques play a fundamental role to recognize malicious activity in todays' scenario. In particular, learning-based anomaly detection techniques have been shown to be particularly interesting since, basically, they implement a black-box IDS which is easy to deploy and requires no or scarce maintenance efforts. These systems, however, are not with their drawbacks. From the industrial point of view, the amount of FP generated by anomaly-based systems is not negligible; further exacerbating this problem is the base-rate fallacy summarized in Section 2.4.2. In fact, if one considers the popularity of real attacks, even a minuscule FPR is magnified and instantly becomes a cost for the organization. From the research point of view, the lack of a well-established methodology to evaluate IDS is an issue; in addition to this, the generation of a reliable testing dataset is an open research problem. Todays' evaluation is limited to two, major datasets: IDEVAL, which, besides being deprecated, contains several regularities that make the evaluation extremely biased. An alternative is the modern *Cyber Defense eXercise* (CDX) 2009 labeled dataset collected during a security competition; this is clearly better than IDEVAL if ignoring the fact that the labeling phase is consists in running Snort. This inherently assumes Snort as the evaluation baseline of every IDS.

Secondly, focusing on anomaly-based techniques, we reviewed the most recent state-of-the-art IDSs to protect a host application and a web server. We also overviewed a few approaches to capture and analyze the network traffic as a stream of packets. This topic is included in the reviewed literature because the contributions in Chapter 5 work on alerts generated by host- and network-based systems. Network-based IDS are popular as they are easy to deploy and can protect a wide range of machine (i.e., the whole network); however, it has been shown how these systems can be evaded by exploiting the lack of local knowledge on the single host. The need of both global and local knowledge about a network is probably the main motivation in favor of alert correlation systems. The most advanced network-based anomaly detectors inspect packet up to the TCP layer and exploit payload clustering and classification techniques to recognize anomalous traffic.

On the other hand, host-based systems have been shown to be effective at detecting malicious processes on a single computer. Almost all the reviewed approaches analyze the system calls intercepted in the operating system kernel. Some tools known to work well for network-based systems have been utilized in this field as well: for instance, clustering and Bayesian classification techniques of network packets has been adapted by several approaches to cluster similar system calls and flag process with low Bayesian probability as malicious, respectively. Stochastic and deterministic techniques have been shown to be very effective. In particular, as we detail in Section 3.3, deterministic models such as automata are well suited to capture a process' control flow. On the other hand, stochastic techniques such as character distribution or Gaussian intervals have been show to correctly model the data flow (e.g., the content of system call arguments) with low FP.

Web-based ID approaches have been overviewed as well. Although this topic is relatively new, web-based approaches are enjoying immense popularity. In fact, the tremendous ubiquity of the Web has become a high-profit opportunity for the underground criminals to spread malware by violating vulnerable, popular websites. The research community have immediately recognized the relevance of this problem. As a consequence, the industry started to adopt web-based protection tools that have became remarkably accurate and, basically, ready to be deployed in real-world environments.

Finally, by comparing Table 2.2, 2.3, and 2.4 *vs.* 2.5 it can be immediately noticed how new this problem is. In fact, during the last decade a common line for network-based techniques can be traced (i.e., exploiting payload classification). The same holds for both host-based (i.e., the use of hybrid deterministic/stochastic techniques on system call sequences and arguments) and web-based (i.e., the use of ensemble of stochastic models on HTTP parameters) techniques, but not for alert correlation approaches. They indeed explore the use of different techniques, but no well-established ideas can be recognized, yet.

Host-based Anomaly Detection 3

A host-based anomaly detector builds profiles of the system activity by observing data collected on a single host (e.g., a personal computer, a database server or a web server). By adopting learning techniques, a host-based IDS can automatically capture the normal behavior of the host and flag significant deviations.

In this chapter we describe in details two contributions we proposed to mitigate malicious activity in a POSIX-like operating system. In both these works, our tools analyze the activity at the granularity of the system call (e.g., `open`, `read`, `chmod`) and monitor each running process in parallel using a multi-threaded scheduling. Our approaches integrate the most advanced techniques that, according to the recent literature, have been shown to be effective. We also propose novel techniques, such as clustering of system calls to classify system calls and Markov models to capture each process' behavior, and their refinements to further lower the FPR. All the systems described in this chapter have been evaluated using both IDEVAL and a more realistic dataset that also includes new attacks (details on the generation of such a dataset are described). In addition, we describe our tests on a dataset we created through the implementation of an innovative technique of anti-forensics, and we show that our approach yields promising results in terms of detection. The goal of this extensive set of experiments is to use our prototype to circumvent definitive anti-forensics tools. Basically, we demonstrate how our tool can detect stealthy in-memory injections of executable code, and in-memory execution of binaries (the so-called "userland exec" technique, which we re-implement in a reliable way).

3.1 Preliminaries

In order to avoid the shortcomings mentioned in Section 2.4, besides the use of IDEVAL for comparison purposes with SyscallAnomaly (which was tested on that dataset), we generated an additional experimental dataset for other frequently used console applications. We chose different buffer overflow exploits that allow to execute arbitrary code.

More precisely, in the case of `mcweject 0.9`, the vulnerability [National Vulnerability Database, 2007a] is a very simple stack overflow, caused by improper bounds checking. By passing a long argument on the command line, an aggressor can execute arbitrary code on the system with root privileges. There is a public exploit for the vulnerability [Harry, 2007] which we modified slightly to suit our purposes and execute our own payload. The attack against `bsdtar` is based on a publicly disclosed vulnerability in the PAX handling functions of `libarchive 2.2.3` and earlier [National Vulnerability Database, 2007b], where a function in file `archive_read_support_format_tar.c` does not properly compute the length of a buffer when processing a malformed PAX archive extension header (i.e., it does not check the length of the header as stored in a header field), resulting in a heap overflow which allows code injection through the creation of a malformed PAX archive which is subsequently extracted by an unsuspecting user on the target machine. In this case, we developed our own exploit, as none was available online, probably due to the fact that this is a heap overflow and requires a slightly more sophisticated exploitation vector. In particular, the heap overflow allows to overwrite a pointer to a structure which contains a pointer to a function which is called soon after the overflow. So, our exploit overwrites this pointer, redirecting it to the injected buffer. In the buffer we craft a clone of the structure, which contains a pointer to the shellcode in place of the correct function pointer.

Our testing platform runs a vanilla installation of FreeBSD 6.2 on a x86 machine; the kernel has been recompiled enabling the appropriate auditing modules. Since our systems, and other host-based anomaly detectors [Kruegel et al., 2003a; Mutz et al., 2006], accept input in the BSM format, the OpenBSM [Watson and Salamon, 2006] auditing tools collection has been used for collecting audit trails. We have audited vulnerable releases of `eject` and `bsdtar`, namely: `mcweject 0.9` (which is an alternative to the BSD `eject`) and the version of `bsdtar` which is distributed with FreeBSD 6.2. During the generation process, the audit trails keep changing, along with the simulated user behavior. It is important to underline that normal users would never use really random names for their files and directories, they usually prefer to use words from their tongue plus a limited set of characters (e.g., ., -, _) for concatenating them. Therefore, we rely on a large dictionary of words for generating file names.

The `eject` executable has a small set of command line option and a very plain execution flow. For the simulation of a legitimate user, we simply chose different permutations of flags and different devices. For this executable, we manually generated 10 executions, which are remarkably similar (as expected).

Creating a dataset of normal activity for the `bsdtar` program is more challenging. It has a large set of command line options, and in general is more complex than `eject`. While the latter is generally called with an argument of `/dev/*`, the former can be invoked with any argument string, for instance `bsdtar cf myarchive.tar /first/path /second/random/path` is a perfectly legitimate command line. Using a procedure similar to the one used for creating the IDEVAL dataset, and in fact used also in [Sekar et al., 2001], we prepared a shell script which embeds pseudo-random behaviors of an average user who creates or extracts archives. To avoid the regularities found in IDEVAL, to simulate user activity, the script randomly creates random-sized, random-content files inside a snapshot of a real-world desktop file-system. In the case of the simulation of super-user executions, these files are scattered around the system; in the case of a regular user, they are into that user's own home directory. Once the file-system has been populated, the tool randomly walks through the directory tree and randomly creates TAR archives. Similarly, found archives are randomly expanded. The randomization takes also into account the different use of flags made by users: for instance, some users prefer to uncompress an archive using `tar xf archive.tar`, many others still use the dash `tar -xf archive.tar`, and so on.

In addition to the aforementioned datasets, we used attacks against `sing`, `mt-daapd`, `proftpd`, `sudo`, and `BitchX`. To generate clean training data we followed similar randomization and scripting mechanisms described previously.

Specifically, `sing` is affected by CVE-2007-6211, a vulnerability which allows to write arbitrary text on arbitrary files by exploiting a combination of parameters. This attack is meaningful because it does not alter the control flow, but just the data flow, with an `open` which writes on unusual files. Training datasets contain traces of regular usage of the program invoked with large sets of command line options.

`mt-daapd` is affected by a format string vulnerability (CVE-2007-5825) in `ws_addarg()`. It allows remote execution of arbitrary code by including the format specifiers in the username or password portion of the base64-encoded data on the `Authorization: Basic` HTTP header sent to `/xml-rpc`. The `mod_ctrls` module of `proftpd` let local attackers to fully control the integer `reqarglen` (CVE-2006-6563) and exploit a stack overflow to gain root privileges.

`sudo` does not properly sanitize data supplied through `SHELLOPTS` and `PS4` environment variables, which are passed on to the invoked program (CVE-2005-2959). This leads to the execution of arbitrary commands as privileged user, and it can be exploited by users who have been granted limited superuser privileges. The training set includes a number of execution of programs commonly run through sudo (e.g., `passwd`, `adduser`, editing of `/etc/` files) by various users with different, limited superuser privileges, along with benign traces similar to the attacks, invoked using several permutations of option flags.

`BitchX` is affected by CVE-2007-3360, which allows a remote attacker to execute arbitrary commands by overfilling a hash table and injecting an EXEC hook function which receives and executes shell commands. Moreover, failed exploit attempts can cause DoS. The training set includes several *Internet Relay Chat* (IRC) client sessions and a legal IRC session to a server having the same address of the malicious one.

Note 3.1.1 First, it is important to underline that the scripts that we prepared to set up the experiments are only meant to generate the system calls that are generated when a particular executable is stimulated with different command line options and different inputs. By no means we claim that such scripts can emulate a user's overall activity. Although the generation of large dataset for IDS evaluation goes beyond the scope of our work, these scripts attempt to reflect the way a regular user invokes `tar` or `eject`; this was possible because they are both simple programs which require a limited number of command line options. Clearly, generating such a dataset for more sophisticated applications (e.g., a browser, a highly-interactive graphic tool) would be much more difficult.

Secondly, we recall that these scripts have been set up *only* for collecting clean data. Such data collection is not needed by our system when running, since the user data would be already available.

In [Bhatkar et al., 2006] a real web and SSH server logs were used for testing. While this approach yields interesting results, we did not follow it for three reasons. Firstly, in our country various legal concerns limit what can be logged on real-world servers. In second place, HTTP and SSH are complex programs where understanding what is correctly identified and what is not would be difficult (as opposed to simply counting correct and false alerts). Finally, such a dataset would not be reliable because of the possibility of the presence of real attacks inside the collected logs (in addition to the attacks inserted manually for testing).

3.2 Malicious System Calls Detection

In this section we describe our contribution regarding anomaly detection of host-based attacks by exploiting unsupervised system calls arguments and sequences [Maggi et al., 2009a]. Analyzing both the theoretical foundations described in [Kruegel et al., 2003a; Mutz et al., 2006], and the results of our tests, we proposed an alternative system, which improves some of the ideas of SyscallAnomaly (the IDS developed on top of LibAnomaly) along with clustering, Markov based modeling, and behavior identification. The approach is implemented in a prototype called *Syscall Sequence Arguments Anomaly Detection Engine* (S^2A^2DE), written in *American National Standard Institute* (ANSI) C. A set of anomaly detection models for the individual parameters of the calls is defined. Then, a clustering process which helps to better fit models to system call arguments, and creates inter-relations among different arguments of a system call is defined by means of ad-hoc distance metrics. Finally, we exploit Markov models to encode the monitored process' normal behavior. The resulting system needs no prior knowledge input; it has a good FPR, and it is also able to correctly contextualize alarms, giving the security officer more information to understand whether a TP or FP happened, and to detect variations over the entire execution flow, as opposed to punctual variations over individual instances.

S^2A^2DE uses the sequence as well as the parameters of the system calls executed by a process to identify anomalous behaviors. As detailed in Section 2.2.2, the use of system calls as anomaly indicators is well established in literature (e.g. in [Forrest et al., 1996; Cabrera et al., 2001; Hofmeyr et al., 1998; Somayaji and Forrest, 2000; Cohen, 1995; Lee and Stolfo, 1998; Ourston et al., 2003; Jha et al., 2001; Michael and Ghosh, 2002; Sekar et al., 2001; Wagner and Dean, 2001; Giffin et al., 2005; Yeung and Ding, 2003]), usually without handling their parameters (with the notable exceptions of [Kruegel et al., 2003a; Tandon and Chan, 2003; Bhatkar et al., 2006]). S^2A^2DE is an improvement to the existing tool by means of four key novel contributions:

- we build and carefully test anomaly detection models for system call parameters, in a similar way to [Kruegel et al., 2003a];
- we introduce the concept of *clustering* arguments in order to automatically infer different ways to use the same system call;

this leads to more precise models of normality on the arguments;

- the same concept of clustering also creates correlations among the different parameters of a same system call, which is not present in any form in [Kruegel et al., 2003a; Tandon and Chan, 2003; Bhatkar et al., 2006];

- the traditional detector based on deviations from previously learned Markov models is complemented with the concept of clustering; the sequence of system calls is transformed into a sequence of labels (i.e., classes of calls): this is conceptually different than what has been done in other works (such as [Tandon and Chan, 2003]), where sequences of events and single events by themselves are both taken into account but in an orthogonal way.

The resulting model is also able to correctly contextualize alarms, providing the user with more information to understand what caused any FP, and to detect variations over the execution *flow*, as opposed to variations over *single* system call. We also discuss in depth how we performed the implementation and the evaluation of the prototype, trying to identify and overcome the pitfalls associated with the usage of the IDEVAL dataset.

3.2.1 Analysis of SyscallAnomaly

In order to replicate the original tests of SyscallAnomaly, we used the host-based auditing data in BSM format contained in the IDEVAL dataset. For now, it is sufficient to note that we used the BSM audit logs from the system named `pascal.eyrie.af.mil`, which runs a Solaris 2.5.1 operating system. Before describing the dataset used, we briefly summarize the models used by SyscallAnomaly: string length model, the character distribution model, the structural inference model and the token search model.

string length model — computes, from the strings seen in the training phase, the sample mean μ and variance σ^2 of their lengths. In the detection phase, given l, the length of the observed string, the likelihood p of the input string length with respect to the values observed in training is equal to one if $l < \mu + \sigma$ and $\frac{\sigma^2}{(l-\mu)^2}$ otherwise.

character distribution model — analyzes the discrete probability distribution of characters in a string. Each string is considered as a set of characters, which are inserted into an histogram, in decreasing order of occurrence, with a classical rank order/frequency representation. For detection, a χ^2 Pearson test returns the likelihood that the observed string histogram comes from the learned model.

structural inference model — learns the structure of strings, that are first simplified using the following translation rules: `[A-Z]` $\rightarrow$ `A`, `[a-z]` $\rightarrow$ `a`, `[0-9]` $\rightarrow$ `0`. Finally, multiple occurrences of the same character are simplified. For instance, `/usr/lib/libc-.so` is translated into `/aaa/aaa/aaaa.aa`, and further compressed into `/a/a/a.a`. A probabilistic grammar encoded in a *Hidden Markov Model* (HMM) is generated by exploiting a Bayesian merging procedure, as described in [Stolcke and Omohundro, 1993b, 1994c,b].

token search model — applied to arguments which contain flags or modes. This model uses a statistical test to determine whether or not an argument contains a finite set of values. The core idea (drawn from [Lee et al., 2002]) is that if the set is finite, then the number of different arguments in the training set will grow in a much slower way than the total number of samples. This is tested using a Kolgomorov-Smirnov non parametric test. If the field contains a set of tokens, the set of values observed during training is stored. During detection, if the field has been flagged as a token, the input is compared against the stored values list. If it matches a former input, the model returns 1, else it returns 0, without regard to the relative frequency of the tokens in the training data.

A confidence rating is computed at training for each model, by determining how well it fits its training set; this value is meant to be used at runtime to provide additional information on the reliability of the model.

Returning to the analysis, we used a dataset that contains 25 buffer overflow attacks against 4 different applications: `eject`, `fd-`

Program	SyscallAnomaly	S^2A^2DE
`fdformat`	0	1 (4)
`eject`	0	1 (6)
`ps`	0	2 (10)
`ftpd`	14	2 (45)
`telnetd`	17	2 (198)
`sendmail`	8	4 (97)

Table 3.1: Comparison of SyscallAnomaly (results taken from [Kruegel et al., 2003a]) and S^2A^2DE in terms of number FPs on the IDEVAL dataset. For S^2A^2DE , the amount of system calls flagged as anomalous is reported between brackets.

`format`, `ps`, and `ffbconfig` (not tested). We used data from weeks 1 and 3 for training, and data from weeks 4 and 5 for testing the detection phase. However, it must be noted that some attacks are not directly detectable through system call analysis. The most interesting attacks for testing SyscallAnomaly are the ones in which an attacker exploits a vulnerability in a local or remote service to allow an intruder to obtain or escalate privileges.

In addition to the programs named above, we ran SyscallAnomaly also on three other programs, namely `ftpd`, `sendmail` and `telnetd`, which are known not to be subject to any attack in the dataset, in order to better evaluate the FPR of the system. In Table 3.1 we compare our results with the released version of SyscallAnomaly [Kruegel et al., 2009] to reproduce the results reported in [Kruegel et al., 2003a].

As can be seen, our results are different from those reported in [Kruegel et al., 2003a], but the discrepancy can be explained by a number of factors:

- the version of SyscallAnomaly and LibAnomaly available online could be different from or older than the one used for the published tests;
- several parameters can be tuned in SyscallAnomaly, and a different tuning could produce different results;
- part of the data in the IDEVAL dataset under consideration are corrupted or malformed;

- in [Kruegel et al., 2003a] it is unclear if the number of FPs is based on the number of executions erroneously flagged as anomalous, or on the number of anomalous syscalls detected.

These discrepancies make a direct comparison difficult, but our numbers confirm that SyscallAnomaly performs well overall as a detector. However, FPs and detected anomalies are interesting to study, in order to better understand how and where SyscallAnomaly fails, and to improve it. Therefore we analyzed in depth a number of executions. Just to give a brief example, let us consider `eject`: it is a very plain, short program, used to eject removable media on UNIX-like systems: it has a very simple and predictable execution flow, and thus it is straightforward to characterize: dynamic libraries are loaded, the device `vol@0:volctl` is accessed, and finally, the device `unnamed_floppy` is accessed.

The dataset contains only one kind of attack against `eject`: it is a buffer overflow with command execution (see Table 3.2). The exploit is evident in the `execve` system call, since the buffer overflow is exploited from the command line. Many of the models in SyscallAnomaly are able to detect this problem: the character distribution model, for instance, performs quite well. The anomaly value turns out to be 1.316, much higher than the threshold (0.0012). The string length and structural inference models flag this anomaly as well, but interestingly they are mostly ignored since their confidence value is too low. The confidence value for the token model is 0, which in SyscallAnomaly convention means that the field is not recognized as a token. This is actually a shortcoming of the association of models with parameters in SyscallAnomaly, because the "filename" argument is not really a token.

A FP happens when a removable unit, unseen during training, is opened (see Table 3.2). The structural inference model is the culprit of the false alert, since the name structure is different from the previous one for the presence of an underscore. As we will see later on, the extreme brittleness of the transformation and simplification model is the main weakness of the Structural Inference model.

Another alert happens in the opening of a localization file (Table 3.3), which triggers the string length model and creates an anomalous distribution of characters; moreover, the presence of numbers, underscores and capitals creates a structure that is flagged as anomalous by the structural inference model. The anomaly in the token

True positive (`execve`)		
file	`/usr/bin/eject`	
argv	`eject\0x20\0x20\0x20\0x20[...]`	
	Models	*Prob. (Conf.)*
file	Token Search	0.999999 (0)
	String Length	10^{-6} (0)
argv	Character Distribution	0.005 (0.928)
	Structural Inference	10^{-6} (0.025)
	Total Score (Threshold)	1.316 (0.0012)

False positive (`open`)		
file	`/vol/dev/rdiskette0/c0t6d0/volume_1`	
flags	`crw-rw-rw-`	
	Models	*Prob. (Conf.)*
	String Length	0.667 (0.005)
path	Character Distribution	0.99 (0.995)
	Structural Inference	10^{-6} (1)
	Token Search	0.999 (1)
	Total Score (Threshold)	8.186 (1.454)

Table 3.2: A true positive and a FP on `eject`.

search model is due to the fact that the open mode (`-r-xr-xr-x`) is not present in any of the training files. This is not an attack, but is the consequence of the buffer overflow attack, and as such is counted as a true positive. However, it is more likely to be a lucky, random side effect.

Without getting into similar details for all the other programs we analyzed (details which can be found in Section 2.4.1), let us summarize our findings. `ps` is a jack-of-all-trades program to monitor process execution, and as such is much more articulated in its options and execution flow than any of the previously analyzed executables. However, the sequence of system calls does not vary dramatically depending on the user specified options: besides library loading, the

True positive (open)		
path	`/usr/lib/locale/iso_8859_1/[...]`	
flags	`-r-xr-xr-x`	
	Models	*Prob. (Conf.)*
	String Length	0.0096 (0.005)
	Character Distribution	0.005 (0.995)
	Structural Inference	10^{-6} (0.986)
	Token Search	10^{-6} (1)
	Total Score (Threshold)	8.186 (1.454)

Table 3.3: True positive on `fdformat` while opening a localization file.

program opens `/tmp/ps_data` and the files containing process information in `/proc`. Also in this case, attacks are buffer overflows on a command-line parameter. In this case, as was the case for `fdformat`, a correlated event is also detected, the opening of file `/tmp/foo` instead of file `/tmp/ps_data`. Both the Token Search model and the Structural Inference model flag an anomaly, because the opening mode is unseen before, and because the presence of an underscore in `/tmp/ps_data` makes it structurally different from `/tmp/foo`. However, if we modify the exploit to use `/tmp/foo_data`, the Structural Inference model goes quiet. A FP happens when `ps` is executed with options `lux`, because the structural inference model finds this usage of parameters very different from `-lux` (with a dash), and therefore strongly believes this to be an attack. Another FP happens when a zone file is opened, because during training no files in `zoneinfo` were opened. In this case it is very evident that the detection of the opening of the `/tmp/foo` file is more of another random side effect than a detection, and in fact the model which correctly identifies it also creates FPs for many other instances. In the case of `in.ftpd`, a common FTP server, a variety of commands could be expected. However, because of the shortcomings of the IDEVAL dataset detailed in Section 2.4.1, the system call flow is fairly regular. After access to libraries and configuration files, the logon events are recorded into system log files, and a `vfork` call is then executed to create a child process to actually serve the client requests. In this case, the FPs mostly happen because of the opening of files never accessed during training, or with "unusual modes".

sendmail is a really complex program, with complex execution flows that include opening libraries and configuration files, accessing the mail queue (/var/spool/mqueue), transmitting data through the network and/or saving mails on disk. Temporary files are used, and the setuid call is also used, with an argument set to the recipient of the message (for delivery to local users).

A FP happens for instance when sendmail uses UID 2133 to deliver a message. In training that particular UID was not used, so the model flags it as anomalous. Since this can happen in the normal behavior of the system, it is evidently a generic problem with the modeling of UIDs as it is done in LibAnomaly. Operations in /var/-mail are flagged as anomalous because the file names are similar to /var/mail/emonca000Sh, and thus the alternation of lower and upper case characters and numbers easily triggers the Structural Inference model.

We outlined different cases of failure of SyscallAnomaly. But what are the underlying reasons for these failures? The *structural inference model* is probably the weakest overall. It is too sensitive against non alphanumeric characters, since they are not altered nor compressed: therefore, it reacts strongly against slight modifications that involve these characters. This becomes visible when libraries with variable names are opened, as it is evident in the FPs generated on the ps program. On the other hand, the compressions and simplifications introduced are excessive, and cancel out any interesting feature: for instance, the strings /tmp/tempfilename and /etc/shadow are indistinguishable by the model. Another very surprising thing, as we already noticed, is the choice of ignoring the probability values in the Markov model, turning it into a binary value (0 if the string cannot be generated, 1 otherwise). This assumes an excessive weight in the total probability value, easily causing a false alarm.

To verify our intuition, we re-ran the tests excluding the Structural Inference model: the DR is unchanged, while the FPR strongly diminishes, as shown in Table 3.4 (once again, both in terms of global number of alerts, and of flagged system calls). Therefore, the Structural Inference model is not contributing to detection; instead it is just causing a growth in the anomaly scores which could lead to an increased number of false positives. The case of telnetd is particularly striking: excluding the Structural Inference model makes all the false positives disappear.

Executable	False Positives (Syscalls flagged)	
	With the HMM	Without the HMM
`fdformat`	1 (4)	1(4)
`eject`	1 (6)	1 (3)
`ps`	2 (10)	1 (6)
`ftpd`	2 (45)	2 (45)
`telnetd`	2 (198)	0 (0)
`sendmail`	4 (97)	4 (97)

Table 3.4: Behavior of SyscallAnomaly with and without the Structural Inference Model

The *character distribution model* is much more reliable, and contributes positively to detection. However, it is not accurate about the particular distribution of each character, and this can lead to possible mimicry attacks. For instance, executing `ps -[x)` has a very high probability, because it is indistinguishable from the usual form of the command `ps -axu`.

The *token search model* has various flaws. First of all, it is not probabilistic, as it does not consider the relative probability of the different values. Therefore, a token with 1000 occurrences is considered just as likely as one with a single occurrence in the whole training set. This makes the training phase not robust against the presence of outliers or attacks in the training dataset. Additionally, since the model is applied only to fields that have been determined beforehand to contain a token, the statistical test is not useful: in fact, in all our experiments, it never had a single negative result. It is also noteworthy that the actual implementation of this test in [Kruegel et al., 2009] differs from what is documented in [Kruegel et al., 2003a; Mutz et al., 2006; Lee et al., 2002].

Finally, the *string length model* works very well, even if this is in part due to the artifacts in the dataset, as we describe in Section 2.4.1.

3.2.2 Improving SyscallAnomaly

We can identify and propose three key improvements over SyscallAnomaly. Firstly, we redesign improved models for anomaly detection on arguments, focusing on their reliability. Over these improved

Executable	% of `open`	Executions
`fdformat`	92.42%	5
`eject`	93.23%	7
`ps`	93.62%	105
`telnetd`	91.10%	65
`ftpd`	95.66%	1082
`sendmail`	86.49%	827

Table 3.5: Percentage of `open` syscalls and number of executions (per program) in the IDEVAL dataset.

models, we introduce a clustering phase to create correlations among the various models on different arguments of the same system call: basically, we divide the set of the invocations of a single system call into subsets which have arguments with an higher similarity. This idea arises from the consideration that some system calls do not exhibit a single normal behavior, but a plurality of behaviors (ways of use) in different portions of a program. For instance, as we will see in the next sections, an `open` system call can have a very different set of arguments when used to load a shared library or a user-supplied file. Therefore, the clustering step aims to capture relationships among the values of various arguments (e.g. to create correlations among some filenames and specific opening modes). In this way we can achieve better characterization.

Finally, we introduce a sequence-based correlation model through a Markov chain. This enables the system to detect deviations in the control flow of a program, as well as abnormalities in each individual call, making evident the whole anomalous context that arises as a consequence, not just the single point of an attack.

The combination of these improvements solves the problems we outlined in the previous sections, and the resulting prototype thus outperforms SyscallAnomaly, achieving also a better generality.

3.2.2.1 Distance metrics for system calls clustering

We applied a hierarchical agglomerative clustering algorithm [Han and Kamber, 2000] to find, for each system call, subclusters of invocation with similar arguments; we are interested in creating models on these clusters, and not on the general system call, in order to better capture normality and deviations. This is important because, as can be seen from Table 3.5, in the IDEVAL dataset the single system call `open` constitutes up to 95% of the calls performed by a process. Indeed, `open` (and probably `read`) is one of the most used system calls on UNIX-like systems, since it opens a file or device in the file system and creates a handle (descriptor) for further use. `open` has three parameters: the file path, a set of flags indicating the type of operation (e.g. read-only, read-write, append, create if non existing, etc.), and an optional opening mode, which specifies the permissions to set in case the file is created. Only by careful aggregation over these parameters we may divide each "polyfunctional" system call into "subgroups" that are specific to a single function.

We used a single-linkage, bottom-up agglomerative technique. Conceptually, such an algorithm initially assigns each of the N input elements to a different cluster, and computes an $N \times N$ distance matrix D. Distances among clusters are computed as the *minimum* distance between an element of the first cluster and an element of the second cluster. The algorithm progressively joins the elements i and j such that $D(i, j) = \min(D)$. D is updated by substituting i and j rows and columns with the row and column of the distances between the newly joined cluster and the remaining ones. The minimum distance between two different clusters $d_{stop,min}$ is used as a stop criterion, in order to prevent the clustering process from lumping all of the system calls together; moreover, a lower bound $d_{stop,num}$ for the number of final clusters is used as a stop criterion as well. If any of the stop criteria is satisfied, the process is stopped. The time complexity of a naive implementation is roughly $O(N^2)$. This would be too heavy, in both time and memory. Besides introducing various tricks to speed up our code and reduce memory occupation (as suggested in [Golub and Loan, 1996]), we introduced an heuristic to reduce the average number of steps required by the algorithm. Basically, at each step, instead of joining just the elements at minimum distance d_{min}, also all the elements that are at a distance $d < \beta d_{min}$ from both the elements at minimum distance are joined, where β is a parameter of the algorithm. In this way, groups of elements that are very close together are joined in a single step, making the algorithm (on average) much faster, even if worst-case complexity is unaffected. Table 3.6 indicates the results measured in the case of $d_{stop,min} = 1$ and $d_{stop,num} = 10$; we want to recall that these timings, seemingly very

Exec.	#elements	Naïve	Optimized
`fdformat`	11	0.14” (0.12”)	0.014” (0.002”)
`eject`	13	0.24” (0.13”)	0.019” (0.003”)
`ps`	880	19’52” (37”)	7” (5”)
`sendmail`	3450	∞	7’19” (6’30”)

Table 3.6: Execution times with and without the heuristic (and in parenthesis, values obtained by performance tweaks)

high, refer to the training phase and not to the run time phase.

The core step in creating a good clustering is of course the definition of the *distance* among different sets of arguments. We proceed by comparing corresponding arguments in the calls, and for each couple of arguments a we compute the following.

Definition 3.2.1 (Arguments Distance) The *argument distance* between two instances i and j of the same system call with respect to argument a is defined as:

$$d_a(i,j) := \begin{cases} K_a + \alpha_a \delta_a(i,j) & \text{if the elements are different} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where K_a is a fixed quantity which creates a step between different elements, while the second term is the real distance between the arguments $\delta_a(i,j)$, normalized by a parameter α_a. Note that, the above formula is a template: the index a denotes that such variables are parametric with respect to the type of argument; how $K_{(\cdot)}$, $\alpha_{(\cdot)}$, and $\delta_{(\cdot)}$ are computed will be detailed below for each type of argument. The distance total between two occurrences i and j system calls is defined as follows.

Definition 3.2.2 (Total Distance) The *total distance* between two instances i and j of the same system call is defined as:

$$D(i,j) := \sum_{a \in A} d_a(i,j) \tag{3.2}$$

where A is the set of system call arguments.

Hierarchical clustering, however, creates a problem for the detection phase, since there is no concept analogous to the centroid of partitioning algorithms that can be used for classifying new inputs, and we cannot obviously afford to re-cluster everything after each one. Thus, we need to generate, from each cluster, a representative model that can be used to cluster (or classify) further inputs. This is a well known problem which needs a creative solution. For each type of argument, we decided to develop a stochastic model that can be used to this end. These models should be able to associate a *probability* to inputs, i.e. to generate a *probability density function* that can be used to state the probability with which the input belongs to the model. As we will see, in most cases, this will be in the form of a discrete probability, but more complex models such as HMMs will also be used. Moreover, a concept of distance must be defined between each model and the input. The model should be able to incorporate new candidates during training, and to slowly adapt in order to represent the whole cluster. It is important to note that it is not strictly necessary for the candidate model and its distance functions to be the same used for clustering purposes. It is also important to note that the clustering could be influenced by the presence of outliers (such as attacks) in the training set. This could lead to the formation of small clusters of anomalous call instances. As we will see in Section 3.2.3, this does not inficiate the ability of the overall system to detect anomalies.

As previously stated, at least four different types of arguments are passed to system calls: path names and file names, discrete numeric values, arguments passed to programs for execution, users and group identifiers (`UID`s and `GID`s). For each type of argument, we designed a representative model and an appropriate distance function.

File name arguments Path names and file names are very frequently used in system calls. They are complex structures, rich of useful information, and therefore difficult to model properly. A first interesting information is commonality of the *path*, since files residing in the same branch of the file system are usually more similar than the ones in different branches. Usually, inside a path, the *first* and the *last* directory carry the most significance. If the filename has a similar *structure* to other filenames, this is indicative: for instance common *prefixes* in the filename, such as the prefix `lib`, or common suffixes such as the extensions.

For the clustering phase, we chose to re-use a very simple model already present in SyscallAnomaly, the directory tree depth. This is easy to compute, and experimentally leads to fairly good results even if very simple. Thus, in Equation 3.1 we set δ_a to be the distance in depth. For instance, let $K_{path} = 5$ and $\alpha_{path} = 1$; comparing `/usr/lib/libc.so` and `/etc/passwd` we obtain $d_a = 5 + 1 \cdot 1 = 6$, while comparing `/usr/lib/libc.so` and `/usr/lib/libelf.so.1` we obtain $d_a = 0$.

After clustering has been done, we represent the path name of the files of a cluster with a probabilistic tree which contains all the directories involved with a probability weight for each. For instance, if a cluster contains: `/usr/lib/libc.so`, `/usr/lib/libelf.so`, or `/usr/local/lib/libintl.so`, the generated tree will be as in Figure 3.1.

File names are usually too variable, in the context of a single cluster, to allow a meaningful model to be always created. However, we chose to set up a system-wide threshold below which the filenames are so regular that they can be considered a model, and thus any other filename can be considered an anomaly. The probability returned by the model is therefore $P_T = P_t \cdot P_f$, where P_t is the probability that the path has been generated by the probabilistic tree and P_f is set to 1 if the filename model is not significant (or if it is significant and the filename belongs to the learned set), and to 0 if the model is significant and the filename is outside the set.

Note 3.2.1 (Configuration parameters) According to the experiments described in Section 3.2.5, the best detection accuracy is achieved with the default value of $K_{path} = 1$ and $\alpha_{path} = 1$. In particular, α_{path} is just a normalization parameter and, in most of the cases, does not require to be changed. K_{path} influences the quality of the clustering and thus may need to be changed if some particular, pathological false positives need to be eliminated in a specific setting.

Discrete numeric values Flags, opening modes, etc. are usually chosen from a limited set. Therefore we can store all of them along with a discrete probability. Since in this case two values can only be "equal" or "different", we set up a binary distance model for clustering.

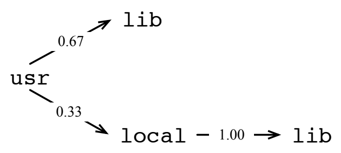


FIGURE 3.1: Probabilistic tree example.

Definition 3.2.3 (Discrete numeric args. distance) Let $x = i_{disc}$, $y = j_{disc}$ be the values of the arguments of type *disc* (i.e., discrete numeric argument) of two instances i and j of the same system call. The *distance* is defined as:

$$d_{disc}(i,j) := \begin{cases} K_{disc} & \text{if } x \neq y \\ 0 & \text{if } x = y, \end{cases}$$

where K_{disc} is a configuration parameter.

Models fusion and incorporation of new elements are straightforward, as well as the generation of probability for a new input to belong to the model.

Note 3.2.2 (Configuration parameters) According to the experiments described in Section 3.2.5, the best detection accuracy is achieved with the default value of $K_{disc} = 1$. This parameter influences the quality of the clustering and thus may need to be changed if some particular, pathological false positives need to be eliminated in a specific setting.

Execution argument We also noticed that execution argument (i.e. the arguments passed to the `execve` system call) are difficult to model, but we found the length to be an extremely effective indicator of similarity of use. Therefore we set up a binary distance model.

Definition 3.2.4 (Execution arguments distance) Let $x = i_{disc}$ and $y = j_{disc}$ be the string values of the arguments of type *disc* (i.e., execution argument) of two instances i and j of the same system call. The *distance* is defined as:

$$d_{arg}(i,j) := \begin{cases} K_{arg} & \text{if } |x| \neq |y| \\ 0 & \text{if } |x| = |y|, \end{cases}$$

where K_{arg} is a configuration parameter and $|x|$ is the length of string x.

Note 3.2.3 (Configuration parameters) According to the experiments described in Section 3.2.5, the best detection accuracy is achieved with the default value of $K_{arg} = 1$. This parameter influences the quality of the clustering and thus may need to be changed if some particular, pathological false positives need to be eliminated in a specific setting.

In this way, arguments with the same length are clustered together. For each cluster, we compute the minimum and maximum value of the length of arguments. Fusion of models and incorporation of new elements are straightforward. The probability for a new input to belong to the model is 1 if its length belongs to the interval, and 0 otherwise.

Token arguments Many arguments express `UID`s or `GID`s, so we developed an ad-hoc model for *users and groups* identifiers. Our reasoning is that all these discrete values have three different meanings: `UID` 0 is reserved to the super-user, low values usually are for system special users, while real users have `UID`s and `GID`s above a threshold (usually 1000). So, we divided the input space in these three groups, and computed the distance for clustering using the following definition.

Definition 3.2.5 (UID/GID argument distance) Let $x = i_{disc}$ and $y = j_{disc}$ be the values of the arguments of type *disc* (i.e., `GID/UID` argument) of two instances i and j of the same system call.

$$d_{uid}(i,j) := \begin{cases} K_{uid} & \text{if } g(x) = g(y) \\ 0 & \text{if } g(x) \neq g(y) \end{cases},$$

where K_{uid} is a configuration parameter, the function $g : \mathbb{N} \mapsto \{groups\}$ and $\{groups\}$ is the set of user groups.

Since `UID`s are limited in number, they are preserved for testing, without associating a discrete probability to them. Fusion of models and incorporation of new elements are straightforward. The probability for a new input to belong to the model is 1 if the UID belongs to the learned set, and 0 otherwise.

Note 3.2.4 (Configuration parameters) According to the experiments described in Section 3.2.5, the best detection accuracy is achieved with the default value of $K_{uid} = 1$. This parameter influences the quality of the clustering and thus may need to be changed if some particular, pathological false positives need to be eliminated in a specific setting.

In Table 3.7 we summarize the association of the models described above with the arguments of each of the system calls we take into account. This model based clustering is somehow error prone since we would expect obtained centroids to be more general and thus somehow to interfere when clustering either new or old instances. To double check this possible issue we follow a simple process:

1. creation of clusters on the training dataset;
2. generation of models from each cluster;
3. use of models to classify the original dataset into clusters, and check that inputs are correctly assigned to the same cluster they contributed to create.

This is done both for checking the representativeness of the models, and to double-check that the different distances computed make sense and separate between different clusters.

As Table 3.8 shows, for each program in the IDEVAL dataset (considering the representative open system call), the percentage of inputs correctly classified, and a confidence value, computed as the average "probability to belong" computed for each element w.r.t. the cluster it helped to build. The results are almost perfect, as expected, with a lower value for the ftpd program, which has a wider variability in file names.

3.2.3 Capturing process behavior

To take into account the execution context of each system call, we use a first order Markov chain to represent the program flow. The model states represent the system calls, or better they represent the various clusters of each system call, as detected during the clustering process. For instance, if we detected three clusters in the `open` system call, and two in the `execve` system call, then the model will be constituted by

System call	Model used for the arguments	
`open`	`pathname`	Path
	`flags`	Discrete Numeric
	`mode`	Discrete Numeric
`execve`	`filename`	Path
	`argv`	Execution Argument
`setuid`	`uid`	User/Group
`setgid`	`gid`	User/Group
`setreuid`	`ruid`	User/Group
`setregid`	`euid`	User/Group
`setresuid, setresgid`	`ruid`	User/Group
	`euid`	User/Group
	`suid`	User/Group
`symlink, link`	`oldpath`	Path
`rename`	`newpath`	Path
`mount`	`source`	Path
	`target`	Path
	`flags`	Discrete Numeric
`umount`	`target`	Path
	`flags`	Path
`exit`	`status`	Discrete Numeric
`chown`	`path`	Path
`lchown`	`group`	User/Group
	`owner`	User/Group
`chmod, mkdir`	`path`	Path
`creat`	`mode`	Discrete Numeric
`mknode`	`pathname`	Path
	`mode`	Discrete Numeric
	`dev`	Discrete Numeric
`unlink, rmdir`	`pathname`	Path

Table 3.7: Association of models to system call arguments in our prototype

Executable	#elements	% correct	Conf.
`fdformat`	10	100%	1
`eject`	12	100%	1
`ps`	525	100%	1
`telnetd`	38	100%	0.954
`ftpd`	69	97.1%	0.675
`sendmail`	3211	100%	0.996

Table 3.8: Cluster validation process.

five states: $\texttt{open}_1$, $\texttt{open}_2$, $\texttt{open}_3$, $\texttt{execve}_1$, $\texttt{execve}_2$. Each transition will reflect the probability of passing from one of these groups to another through the program. As we already observed in Section 2.2.2, this approach was investigated in former literature, but never in conjunction with the handling of parameters and with a clustering approach.

A time domain process demonstrates a Markov property if the conditional probability density of the current event, given all present and past events, depends only on the K most recent events. K is known as the *order* of the underlying model. Usually, models of order $K = 1$ are considered, because they are simpler to analyze mathematically. Higher-order models can usually be approximated with first order models, but approaches for using high-order Markov models in an efficient manner have also been proposed, even in the intrusion detection field [Ju and Vardi, 2001]. A good estimate of the order can be extracted from data using the criteria defined in [Merhav et al., 1989]; for normal Markov models, a χ^2-test for first against second order dependency can be used [Haccou and Meelis, 1992], but also an information criterion such as *Bayesian Information Criterion* (BIC) or *Minimum Description Length* (MDL) can be used.

3.2.3.1 Training Phase

Each execution of the program in the training set is considered as a sequence of observations. Using the output of the clustering process, each system call is classified into the correct cluster, by computing the probability value for each model and choosing the cluster whose models give out the maximum composite probability along all known models: $\max(\prod_{i \in \mathrm{M}} P_i)$. The probabilities of the Markov model are then straightforward to compute. The final results can be similar to what is shown in 3.2. On a first sight, this could resemble a simple Markov chain, however it should be noticed that each of the state of this Markov model could be mapped into the set of possible cluster elements associated to it. From this perspective, it could be seen as a very specific HMM where states are fully observable through an uniform emission probability over the associated syscalls. By simply turning the clustering algorithm into one with overlapping clusters we could obtain a proper HMM description of the user behavior as it would be if we decide to further merge states together. This is actually our ongoing work toward full HMM behavior modeling with the aim, through overlapping syscalls clustering, of improving the performance of the classical Baum-Welch algorithm and solving the issue of HMM hidden space cardinality selection [Rabiner, 1989]. From our experiments, in the case of the simple traces like the ones found in the IDEVAL dataset, the used model suffice so we are working on more complex scenarios (i.e., new real datasets) to improve in a more grounded way the proposed algorithm.

This type of model is resistant to the presence of a limited number of outliers (e.g. abruptly terminated executions, or attacks) in the training set, because the resulting transition probabilities will drop near zero. For the same reason, it is also resistant to the presence of any cluster of anomalous invocations created by the clustering phase. Therefore, the presence of a minority of attacks in the training set will not adversely affect the learning phase, which in turn does not require then an attack-free training set.

The final result can be similar to what is shown in Figure 3.2. In future extensions, the Markov model could then be simplified by merging some states together: this would transform it in a Hidden Markov Model, where the state does not correspond directly to a system call cluster, but rather to a probability distribution of possible outcomes. However, the use of HMMs complicates both training and detection phases [Rabiner, 1989]. From our experiments, in the case of the simple traces like ones found in the IDEVAL dataset, this step is unnecessary.

3.2.3.2 Detection Phase

For detection, three distinct probabilities can be computed for each executed system call: the probability of the *execution* sequence inside

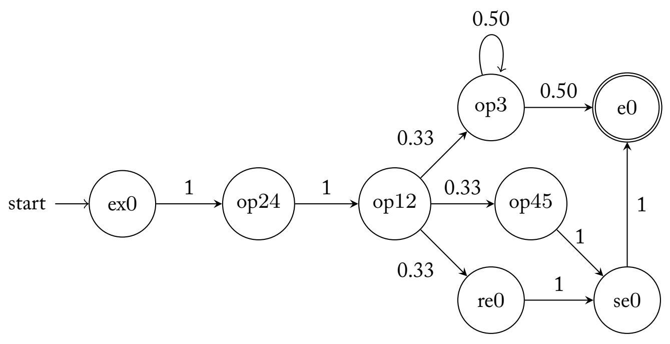


FIGURE 3.2: Example of Markov model.

the Markov model up to now, P_s; the probability of the *system call* to belong to the best-matching cluster, P_c; the *last transition* probability in the Markov model, P_m.

The latter two probabilities can be combined into a single "punctual" probability of the single system call, $P_p = P_c \cdot P_m$, keeping a separate value for the "sequence" probability P_s. In order to set appropriate threshold values, we use the training data, compute the lowest probability over all the dataset for that single program (both for the sequence probability and for the punctual probability), and set this (eventually modified by a tolerance value) as the anomaly threshold. The tolerance can be tuned to trade off DR for FPR.

During detection, each system call is considered in the context of the process. The cluster models are once again used to classify each system call into the correct cluster as explained above: therefore ($P_c = \max(\prod_{i \in \mathrm{M}} P_i)$). P_s and P_m are computed from the Markov model, and require our system to keep track of the current state for each running process. If either P_s or $P_p = P_c \cdot P_m$ are lower than the anomaly threshold, the process is flagged as malicious.

Note 3.2.5 (Probability Scaling) Given an l-long sequence of system calls, its sequence probability is $P_s(l) = \prod_{i=0}^{l} P_p(i)$ where $P_p(i) \in [0,1]$ is the punctual probability of the i-th system call in the sequence. Therefore, it is self-evident that $\lim_{l \to +\infty} P_s(l) = 0$. Experimentally, we observed that the sequence probability quickly decreases to zero, even for short sequences (on the IDEVAL dataset, we found that $P_s(l) \simeq 0$ for $l \geq 25$). This leads to a high number of false positives, since many sequences are assigned probabilities close to zero (thus, always lower than any threshold value).

To overcome this shortcoming, we implemented two probability scaling functions, both based on the geometric mean. As a first attempt, we computed $P_s(l) = \sqrt[l]{\prod_{i=1}^{l} P_p(i)}$, but in this case

$$\mathrm{P}\left[\lim_{l\to+\infty} P_s(l) = e^{-1}\right] = 1.$$

Proof 3.2.1 *Let* $\mathrm{G}(P_p(1), \ldots, P_p(l)) = \sqrt[l]{\prod_{i=1}^{l} P_p(i)}$ *the geometric mean of the sample* $P_p(1), \ldots, P_p(l)$*. If we assume that the sample is generated by a uniform distribution (i.e.,* $P_p(1), \ldots, P_p(l) \sim \mathcal{U}(0,1)$*) then* $\log P_p(i) \sim \mathcal{E}(\beta = 1)\ \forall i = 1, \ldots, l$*: this can be proven by observing that the Cumulative Density Function* (CDF) *[Pestman, 1998] of* $\log X$ *(with* $X = P_p \sim \mathcal{U}(0,1)$*) equals the CDF of an exponentially distributed variable with* $\beta = 1$.

The arithmetic mean $\mathrm{A}(\cdot)$ *of the sample*

$$-\log(P_p(1)), \ldots, -\log(P_p(l))$$

converges (in probability) to $\beta = 1$ *for* $l \to +\infty$*, that is:*

$$\mathrm{P}\left[\lim_{l\to+\infty} \frac{1}{l}\sum_{i=1}^{l} -\log P_p(i) = -\beta = -1\right] = 1$$

because of the strong law of large numbers. Being the geometric mean:

$$\mathrm{G}(P_p(1), \ldots, P_p(l)) = \left(\prod_{i=1}^{l} P_p(i)\right)^{\frac{1}{l}} = e^{\frac{1}{l}\sum_{i=1}^{l} -\log P_p(i)}$$

we have

$$\mathrm{P}\left[\lim_{l\to+\infty} \left(e^{\frac{1}{l}\sum_{i=1}^{l} -\log P_p(i)}\right) = e^{-\beta} = e^{-1}\right] = 1$$

■

This is not our desired result, so we modified this formula to introduce a sort of "forgetting factor": $P_s(l) = \sqrt[2l]{\prod_{i=1}^{l} P_p(i)^i}$. In this case, we can prove that $\mathrm{P}\left[\lim_{l\to+\infty} P_s(l) = 0\right] = 1$.

Proof 3.2.2 *The convergence of $P_s(l)$ to zero can be proven by observing that:*

$$\begin{aligned}
P_s(l) &= \left(\sqrt[l]{\prod_{i=1}^{l} P_p(i)^i}\right)^{\frac{1}{2}} = \left(e^{\frac{1}{l}\sum_{i=1}^{l} i\cdot\log P_p(i)}\right)^{\frac{1}{2}} = \\
&= \left(e^{\frac{1}{l}\sum_{j=0}^{l}\left(\sum_{i=1}^{l-j}\log P_p(i)\right)}\right)^{\frac{1}{2}} = \\
&= \left(e^{\sum_{j=0}^{l}\left(\frac{1}{l}\sum_{i=1}^{l}\log P_p(i)-\frac{1}{l}\sum_{i=l-j+1}^{l}\log P_p(i)\right)}\right)^{\frac{1}{2}}
\end{aligned}$$

Because of the previous proof, we can write that:

$$\mathrm{P}\left[\lim_{l\to+\infty}\frac{1}{l}\sum_{i=1}^{l}\log P_p(i) = -1\right] = 1$$

We can further observe that, being $\log P_p(i) < 0$*:*

$$\forall j, l > 0 : \sum_{i=1}^{l}\log P_p(i) < \sum_{i=l-j+1}^{l}\log P_p(i)$$

therefore, the exponent is a sum of infinite negative quantities lesser than 0, *leading us to the result that,* in probability

$$\begin{aligned}
&\lim_{l\to+\infty}\left(e^{\sum_{j=0}^{l}\left(\frac{1}{l}\sum_{i=1}^{l}\log P_p(i)-\frac{1}{l}\sum_{i=l-j+1}^{l}\log P_p(i)\right)}\right)^{\frac{1}{2}} = \\
&\lim_{x\to-\infty} e^x = 0
\end{aligned}$$

■

Even if this second variant once again makes $P_s(l) \to 0$ (in probability), our experiments have shown that this effect is much *slower* than in the original formula: $P_s(l) \simeq 0$ for $l \geq 300$ (vs. $l \geq 25$ of the previous version), as shown in Figure 3.3. In fact, this scaling function also leads to much better results in terms of FPR (see Section 3.2.5).

Instead of using probabilities — that are sensitive to the sequence length — a possible alternative, which we are currently exploring, is the exploitation of distance metrics between Markov models [Stolcke and Omohundro, 1993b, 1994c,b] to define robust criteria to

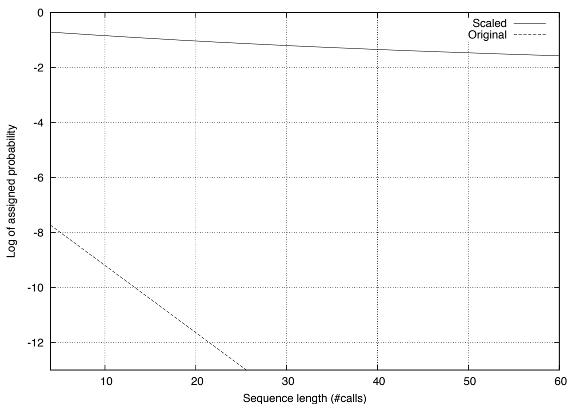


Figure 3.3: Measured sequence probability (log) vs. sequence length, comparing the original calculation and the second variant of scaling.

compare new and learned sequence models. Basically, the idea is to create and continuously update a Markov model associated to the program instance being monitored, and to check how much such a model differs from the ones the system has learned for the same program. This approach is complementary to the one proposed above, since it requires long sequences to get a proper Markov model. So, the use of both criteria (sequence likelihood in short activations, and model comparison in longer ones) could lead to a reduction of false positives on the sequence model.

3.2.4 Prototype implementation

We implemented the above described system into a two-stage, highly configurable, modular architecture written in ANSI C. The high-level structure is depicted in Figure 3.4: the system is plugged into the Linux `auditd`. The Cluster Manager is in charge of the clustering phase while the Markov Model Manager implements Markov modeling features. Both the modules are used in both training phase and detection phase, as detailed below.

The Cluster Manager module implements abstract clustering procedures along with abstract representation and storage of generated

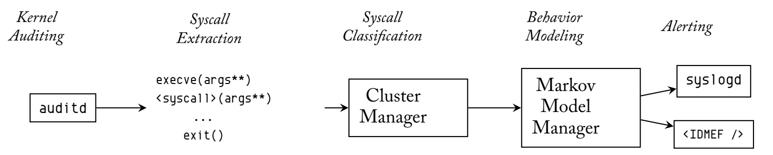


FIGURE 3.4: The high-level structure of our prototype.

clusters. The Markov Model Manager is conceptually similar to Cluster Manager: it has a basic Markov chains implementation, along with ancillary modules for model handling and storage.

During training, Cluster Manager is invoked to create the clusters on a given system call trail while Markov Model Manager infers the Markov model according to the clustering. At running time, for each system call, the Cluster Manager is invoked to find the appropriate cluster; the Markov Model Manager is also used to keep track of the current behavior of the monitored process: if significant deviations are found alerts are fired and logged accordingly.

The system can output to both standard output, `syslog` facilities and IDMEF files. Both the clustering phase and the behavioral analysis are multi-threaded and intermediate results of both procedures can be dumped in *eXtensible Markup Language* (XML).

3.2.5 Experimental Results

In this section, we both compare the detection accuracy of our proposal and analyze the performances of the running prototype we developed. Because of the known issues of IDEVAL (plus our findings summarized in Section 2.4.1), we also collected fresh training data and new attacks to further prove that our proposal is promising in terms of accuracy.

As we detailed in Section 3.2.2.1, our system can be tuned to avoid overfitting; in the current implementation, such parameters can be specified for *each* system call, thus in the following we report the bounds of variations instead of listing *all* the single values: $d_{stop,num} \in \{1, 2, 3\}$, $d_{stop,min} = \{6, 10, 20, 60\}$.

3.2.5.1 Detection accuracy

For the reasons outlined above and in Section 2.4.1, as well for the uncertainty outlined in Section 3.2.1, we did not rely on purely numerical results on DR or FPR. Instead, we compared the results obtained by our software with the results of SyscallAnomaly in the terms of a set of case studies, comparing them singularly. What turned out is that our software has two main advantages over LibAnomaly:

- a better contextualization of anomalies, which lets the system detect whether a single syscall has been altered, or if a sequence of calls became anomalous consequently to a suspicious attack;
- a strong characterization of subgroups with closer and more reliable sub-models.

As an example of the first advantage, let us analyze again the program `fdformat`, which was already analyzed in Section 3.2.1. As can be seen from Table 3.9, our system correctly flags `execve` as anomalous (due to an excessive length of input). It can be seen that P_m is 1 (the system call is the one we expected), but the models of the syscall are not matching, generating a very low P_c. The localization file opening is also flagged as anomalous for two reasons: scarce affinity with the model (because of the strange filename), and also erroneous transition between the open subgroups `open2` and `open10`. In the case of such an anomalous transition, thresholds are shown as "undefined" as this transition has never been observed in training. The attack effect (`chmod` and the change of permissions on `/export/home-/elmoc/.cshrc`) and various intervening syscalls are also flagged as anomalous because the transition has never been observed ($P_m = 0$); while reviewing logs, this also helps us in understanding whether or not the buffer overflow attack has succeeded. A similar observation can be done on the execution of `chmod` on `/etc/shadow` ensuing an attack on `eject`.

In the case of `ps`, our system flags the `execve` system call, as usual, for excessive length of input. File `/tmp/foo` is also detected as anomalous argument for `open`. In `LibAnomaly`, this happened just because of the presence of an underscore, and was easy to bypass. In our case, `/tmp/foo` is compared against a sub-cluster of `open` which contains only the `/tmp/ps_data`, and therefore will flag as anomalous, with a very high confidence, any other name, even if structurally similar. A

HMM State	execve0 (START $\Rightarrow$ execve0)
filename	/usr/bin/fdformat
argv	fdformat\0x20\0x20\0x20\0x20[...]
P_c	0.1
P_m	1
P_p (thresh.)	0.1 (1)
HMM State	open10 (open2 $\Rightarrow$ open10)
pathname	/usr/lib/locale/iso_8859_1/[...]
flags	-r-xr-xr-x
mode	33133
P_c	$5 \cdot 10^{-4}$
P_m	0
P_p (thresh.)	0 (*undefined*)
HMM State	open11 (open10 $\Rightarrow$ open11)
pathname	/devices/pseudo/vol@0:volctl
flags	crw-rw-rw-
mode	8630
P_c	1
P_m	0
P_p (thresh.)	0 (*undefined*)
HMM State	chmod (open11 $\Rightarrow$ chmod)
pathname	/devices/pseudo/vol@0:volctl
flags	crw-rw-rw-
mode	8630
P_c	0.1
P_m	0
P_p (thresh.)	0 (*undefined*)
HMM State	exit0 (chmod $\Rightarrow$ exit0)
status	0
P_c	1
P_m	0
P_p (thresh.)	0 (*undefined*)

Table 3.9: fdformat: attack and consequences

	DR	FPR	
Granularity:	Sequence	Sequence	Call
Markov model	`bsdtar`		
Y	100%	1.6%	0.1%
N	88%	1.6%	0.1%
	`eject`		
Y	100%	0%	0%
N	0%	0%	0%

Table 3.10: DRs and FPRs on two test programs, with (Y) and without (N) Markov models.

sequence of `chmod` syscalls executed inside directory `/home/secret` as a result of the attacks are also flagged as anomalous program flows.

Limiting the scope to the detection accuracy of our system, we performed several experiments with both `eject` and `bsdtar`, and we summarize the results in Table 3.10. The prototype has been trained with ten different execution of `eject` and more than a hundred executions of `bsdtar`. We then audited eight instances of the activity of `eject` under attack, while for `bsdtar` we logged seven malicious executions. We report DRs and FPRs with (Y) and without (N) the use of Markov models, and we compute FPRs using cross-validation through the data set (i.e., by training the algorithm on a subset of the dataset and subsequently testing the other part of the dataset). Note that, to better analyze the false positives, we accounted for both false positive sequences (Seq.) and false positive system calls (Call).

In both cases, using the complete algorithm yield a 100% DR with a very low FPR. In the case of `eject`, the exploit is detected in the very beginning: since a very long argument is passed to the `execve`, this triggers the argument model. The detection of the shellcode we injected exploiting the buffer overflow in `bsdtar` is identified by the `open` of the unexpected (special) file `/dev/tty`. Note that, the use of thresholds calculated on the overall Markov model allows us to achieve a 100% DR in the case of `eject`; without the Markov model, the attack wouldn't be detected at all.

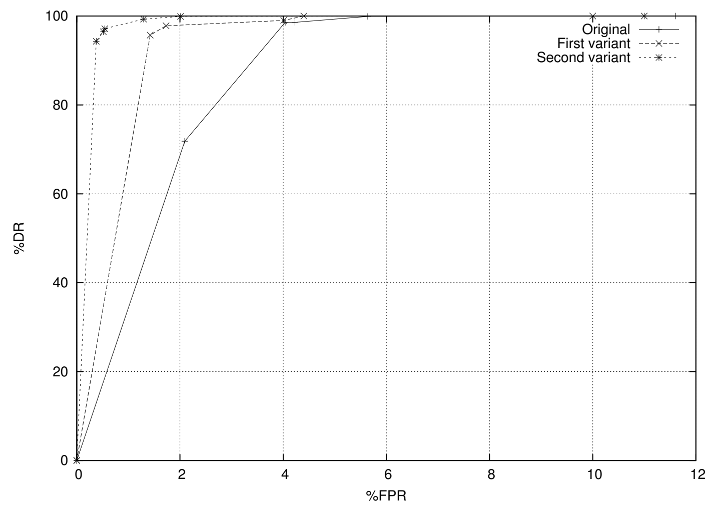


Figure 3.5: Comparison of the effect on detection of different probability scaling functions.

It is very difficult to compare our results directly with the other similar systems we identified in Section 2.2.2. In [Tandon and Chan, 2003] the evaluation is performed on the *Defense Advanced Research Projects Agency* (DARPA) dataset, but DRs and FPRs are not given (the number of detections and false alarms is not normalized), so a direct comparison is difficult. Moreover, detection is computed using an arbitrary time window, and false alerts are instead given in "alerts per day". It is correspondingly difficult to compare against the results in [Bhatkar et al., 2006], as the evaluation is ran over a dataset which is not disclosed, using two programs that are very different from the ones we use, and using a handful of exploits chosen by the authors. Different scalings of the false positives and DRs also make a comparison impossible to draw.

As a side result, we tested the detection accuracy of the two scaling functions we proposed for computing the sequence probability P_s. As shown in Figure 3.5, the first and the second variant both show lower FPR w.r.t. to the original, unscaled version.

Note 3.2.6 (Configuration parameters) Although we performed all the experiments with different values of $d_{stop,min}$ and $d_{stop,num}$, we report only the best detection accuracy, achieved with the default settings: $d_{stop,min} = 10$, $d_{stop,num} = 3$.

These parameters influence the quality of the clustering and thus may need to be changed if some particular, pathological false positives need to be eliminated in a specific setting. In this case, human intervention is required; in particular, it is required to run the tool on a small dataset collected on a live system and check the clustering for outliers. If clusters containing too many outliers exist, then $d_{stop,min}$ may be increased. $d_{stop,num}$ should be increased only if outliers cannot be eliminated.

3.2.5.2 Performance measurements

An IDS should not introduce significant performance overheads in terms of the time required to classify events as malicious (or not). An IDS based on the analysis of system calls has to intercept and process every single syscall invoked on the operating system by userspace applications; for this reason, the fastest a system call is processed, the best. We profiled the code of our system with `gprof` and `valgrind` for CPU and memory requirements. We ran the IDS on data drawn from the IDEVAL 1999 dataset (which is sufficient for performance measurements, as in this case we are only interested in the throughput and not in realistic DRs).

In Table 3.11 we reported the measurement of performance on the five working days of the first week of the dataset for training, and of the fourth week for testing. The throughput X varies during training between 6120 and 10228 syscalls per second. The clustering phase is the bottleneck in most cases, while the Markov model construction is generally faster. Due to the clustering step, the training phase is memory consuming: in the worst case, we recorded a memory usage of about 700 MB. The performance observed in the detection phase is of course even more important: in this case, it varies between 12395 and 22266 syscalls/sec. Considering that the kernel of a typical machine running services such as HTTP/FTP on average executes system calls in the order of thousands per second (e.g., around 2000 system calls per second for `wu-ftpd` [Mutz et al., 2006]), the overhead introduced by our IDS is noticeable but does not severely impact system operations.

3.3 Mixing Deterministic and Stochastic Models

In this section, we describe an improvement to the syscall-based anomaly detection system described in Section 3.2, which incorporates both the deterministic models of what we called FSA-DF, detailed in [Bhatkar et al., 2006] (that we analyzed in detail in Section 2.2.2), and the stochastic models of S^2A^2DE . In [Frossi et al., 2009] we propose a number of modifications, described in the following, that significantly improve the performance of both the original approaches. In this section we begin by comparing FSA-DF with S^2A^2DE and analyzing their respective performance in terms of detection accuracy. Then, we outline the major shortcomings of the two systems, and propose various changes in the models that can address them. In particular, we kept the deterministic control flow model of FSA-DF and substituted some deterministic dataflow relations with their stochastic equivalent, using the technique described in Section 3.3.1.3. This allowed us to maintain an accurate characterization of the process control flow and, at the same time, to avoid many FP due to the deterministic relations among arguments. More precisely, the original learning algorithm of FSA-DF can be schematized as follow:

```
∀c = $\langle syscall_{i-1}, syscall_i \rangle$
make_state(c, PC)
learn_relations(c)
  equal
  elementOf
  subsetOf
  range
  hasExtension
  isWithinDir
  contains
  hasSameDirAs
  hasSameBaseAs
  hasSameExtensionAs
```

We noticed that simple relations such as equal, elementOf, contains were not suitable for strings. In particular, they raise too many FP also in obvious cases like "`/tmp/php1553`" *vs.* "`/tmp/php9022`" where the deterministic `equal` does not hold, but a smoother predicate could easily detect the common prefix, and thus group the two strings in the same class. This problem is clearly extended to the other two relations that are based on string confrontation as well. To mitigate these side-effects, we augmented the learning algorithm as follows:

```
learn_string_domain($syscall_i$)
$\forall c = \langle syscall_{i-1}, syscall_i \rangle$
make_state(c, PC)
learn_relations(c)
  save_model
  subsetOf
  range
  hasExtension
  isWithinDir
  hasSameDirAs
  hasSameBaseAs
  hasSameExtensionAs
```

where the pseudo-function learn_string_domain equals to the training of the SOM as described in Section 3.3.1.3, and save_model plays the role the three relations mentioned above. Basically, it stores the *Best Matching Unit* (BMU) corresponding to the string arguments of the current system call. The BMU is retrieved by querying the SOM previously trained. In addition to this, we complemented the resulting system with two new models to cope with DoS attacks (see Section 3.3.1.2) and the presence of outlier in the training dataset (see Section 3.3.1.1).

We show how targeted modifications of their anomaly models, as opposed to the redesign of the global system, can noticeably improve the overall detection accuracy. Finally, the impact of these modifications are discussed by comparing the performance of the two original implementations with two modified versions complemented with our models.

3.3.1 Enhanced Detection Models

The improvements we made focus on *path* and *execution* arguments. A new *string length* model is added exploiting a Gaussian interval as detailed in Section 3.3.1.1. The new *edge frequency* model described in Section 3.3.1.2 have been added to detect DoS attacks. Also, in Section 3.3.1.3 we describe how we exploited SOM to model the similarity among *path arguments*. The resulting system, Hybrid IDS incorporates the models of FSA-DF and S^2A^2DE along with the aforementioned enhancements.

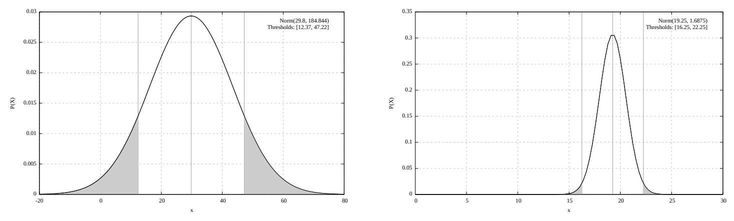


FIGURE 3.6: Sample estimated Gaussian intervals for string length. Training data of `sudo` (left) and `ftp` (right) was used. $\mathcal{N}(29.8, 184.844)$, thresholds [12.37, 47.22] (left) and $\mathcal{N}(19.25, 1.6875)$, thresholds [16.25, 22.25] (right).

3.3.1.1 Arguments Length Using Gaussian Intervals

The model for system call execution arguments implemented in S^{2}A^{2}DE takes into account the minimum and maximum length of the parameters found during training, and checks whether each string parameter falls into this range (model probability 1) or not (model probability 0). This technique allows to detect common attempts of buffer overflow through the command line, for instance, as well as various other command line exploits. However, such criteria do not model "how different" two arguments are to each others; a smoother function is more desirable. Furthermore, the frequency of each argument in the training set is not taken into account at all. Last but not least, the model is not resilient to the presence of attacks in the training set; just one occurrence of a malicious string would increase the length of the maximum interval allowing argument of almost every length.

The improved version of the interval model uses a Gaussian distribution for modeling the argument length $X_{args} = |args|$, estimated from the data in terms of sample mean and sample variance. The anomaly threshold is a percentile T_{args} centered on the mean. Arguments which length is *outside* the stochastic interval are flagged as anomalous. This model is resilient to the presence of outliers in the dataset. The Gaussian distribution has been chosen since is the natural stochastic extension of a range interval for the length. An example is shown in Figure 3.6.

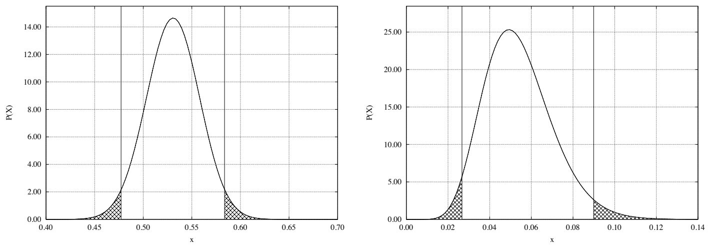
Model Validation During detection the model self-assesses its precision by calculating the kurtosis measure [Joanes and Gill, 1998], defined as $\gamma_X = \frac{\mathrm{E}^4(X)}{\mathrm{Var(X)}^2}$. Thin tailed distributions with a low peak around the mean exhibit $\gamma_X < 0$ while positive values are typical of fat tailed distributions with an acute peak. We used $\hat{\gamma}_X = \frac{\mu_{X,4}}{\sigma_X^4} - 3$ to estimate γ_X. Thus, if $\gamma_{X_{args}} < 0$ means that the sample is spread on a big interval, while positive the values indicates a less "fuzzy" set of values. It is indeed straightforward that highly negative values indicates not significant estimations as the interval would include almost all lengths. In this case, the model falls back to a simple interval.

3.3.1.2 DoS Detection Using Edge Traversal Frequency

DoS attacks which force the process to get stuck in a legal section of the normal control flow could be detected by S^{2}A^{2}DE as violations of the Markov model, but not by FSA-DF. On the other hand, the statistical models implemented in S^{2}A^{2}DE are more robust but have higher *False Negative Rates* (FNRs) than the deterministic detection implemented in FSA-DF. However, as already stated in Section 3.2, the cumulative probability of the traversed edges works well only with execution traces of similar and fixed length, otherwise even the rescaled score decreases to zero, generating false positives on long traces.

To solve these issues a stochastic model of the edge frequency traversal is used. For each trace of the training set, our algorithm counts the number of edge traversals (i.e., Markov model edge or FSA edge). The number is then normalized w.r.t. all the edges obtaining frequencies. Each edge is then associated to the sample $X_{edge} = x_1, x_2, \ldots$. We show that the random samples X_{edge} is well estimated using a Beta distribution. Figure 3.7 shows sample plots of this model estimated using the `mt-daapd` training set; the quantiles associated to the thresholds are computed and shown as well. As we did for the Gaussian model, the detection thresholds are defined at configuration time as a percentile T_{edge} centered on the mean (Figure 3.7). We chose the Beta for its high flexibility; a Gaussian is unsuitable to model skewed phenomena.

Model Validation Our implementation is optimized to avoid overfitting and meaningless estimations. A model is valid only if the



(a) (b)

Figure 3.7: Two different estimations of the edge frequency distribution. Namely, Beta(178.445, 157.866) with thresholds [0.477199, 0.583649] (left) and Beta(10.3529,181.647) with thresholds [0.0266882, 0.0899057] (right).

training set includes a significant (e.g., $|\min_i\{x_i\} - \max_i\{x_i\}| \geq \delta x_{min} = 0.04$) amount ($N_{edge}^{min} = 6$) of paths. Otherwise it construct a simpler frequency range model. The model exhibits the side effect of discarding the extreme values found in training and leads to erroneous decisions. More precisely, if the sample is $X_{edge} = 1, 1, \ldots, 0.9, 1$, the right boundary will never be exactly 1, and therefore legal values will be discarded. To solve this issue, the quantiles close to 1 are approximated to 1 according to a configuration parameter $\bar{X}_{cut}$. For instance, if $\bar{X}_{cut} = 3$ the quantile $F_X(\cdot) = 0.99\underline{9}$ is approximated to 1.

3.3.1.3 Path Similarity Using Self Organizing Maps

Path argument models are already implemented in S^{2}A^{2}DE and FSA-DF. Several, general-purpose string comparison techniques have been proposed so far, especially in the field of database systems and data cleansing [Elmagarmid et al., 2007]. We propose a solution based on Symbol-SOMs [Somervuo, 2004] to define an accurate distance metric between paths. Symbol SOM implements a smooth similarity measure otherwise unachievable using common, crisp distance functions among strings (e.g., edit distance).

The technique exploits SOMs, which are unsupervised neural algorithms. A SOM produces a compressed, multidimensional representation (usually a bi-dimensional *map*) of the input space by preserving the main topological properties. It is initialized randomly, and then adapted via a competitive and cooperative learning process. At each cycle, a new input is compared to the known models, and the BMU node is selected. The BMU and its neighborhood models are then updated to make them better resemble future inputs.

We use the technique described in [Kohonen and Somervuo, 1998] to map *strings* onto SOMs. Formally, let

$$S_t = [s_t(1) \cdots s_t(L)]$$

denote the t-th string over the alphabet $\mathcal{A}$ of size $|\mathcal{A}|$. Each symbol $s_t(i), i = 1 \ldots L$, is then encoded into a vector $\underline{s}_t(i)$ of size $|\mathcal{A}|$ initialized with zeroes except at the w-th position which corresponds to the index of the encoded symbol (e.g., $s_t(i) =$ 'b' would be $\underline{s}_t(i) = [0\ 1\ 0\ 0 \cdots 0]^T$, $w = 2$). Thus, each string S_t is represented with sequence of L vectors like $\underline{s}_t(i)$, i.e. a $L \times |\mathcal{A}|$-matrix: $\underline{\underline{S}}_t$.

Let $\underline{\underline{S}}_t$ and $\underline{\underline{M}}_k$ denote two vector-encoded strings, where $\underline{\underline{M}}_k$ is the model associated with SOM node k. The distance between the two strings is $D'(S_t, M_k) = D(\underline{\underline{S}}_t, \underline{\underline{M}}_k)$. $D(\cdot,\cdot)$ is also defined in the case of $L_{S_t} = |S_t| \neq |M_k| = L_{M_k}$ relying on dynamic time warping techniques to find the best alignment between the two sequences before computing the distance. Without going into details, the algorithm [Somervuo, 2004] aligns the two sequences $\underline{s}_t(i) \in \underline{\underline{S}}_t, \underline{m}_k(j) \in \underline{\underline{M}}_k$ using a mapping $[\underline{s}_t(i), \underline{m}_k(j)] \mapsto [\underline{s}_t(i(p)), \underline{m}_k(j(p))]$ defined through the warping function $F : [i, j] \mapsto [i(p), j(p)]$:

$$F = [[i(1), j(1)], \ldots, [i(p), j(p)], \ldots, [i(P), j(P)]].$$

The distance function D is defined over the warping alignment of size P, $D(\underline{\underline{S}}_t, \underline{\underline{M}}_k) = \sum_{p=1}^{P} d(i, j)$, which is $P = L_{S_t} = L_{M_k}$ if the two strings have equal lengths. More precisely:

$$d(i, j) = d(i(p), j(p)) ||\underline{s}_t(i(p)) - \underline{m}_k(j(p))||.$$

The distance is defined upon $g_{i,j} = g(i, j)$, the variable which stores the cumulative distance in each trellis point $(i, j) = (i(p), i(p))$. The trellis is first initialized to 0 in $(0, 0)$, to $+\infty$ for both $(0, \cdot)$ and $(\cdot, 0)$, otherwise:

$$\begin{array}{c|cccccccc|}
 & & \mathsf{/} & \mathsf{b} & \mathsf{i} & \mathsf{n} & \mathsf{/} & \mathsf{s} & \mathsf{h} \\
 & 0 & +\infty & +\infty & +\infty & +\infty & +\infty & +\infty & +\infty \\
\mathsf{/} & +\infty & 0 & 1.414 & 2.828 & 4.242 & 4.242 & 5.656 & 7.071 \\
\mathsf{v} & +\infty & 1.414 & 1.414 & 2.828 & 4.242 & 5.656 & 5.656 & 7.071 \\
\mathsf{a} & +\infty & 2.828 & 2.828 & 2.828 & 4.242 & 5.656 & 7.071 & 7.071 \\
\mathsf{r} & +\infty & 4.242 & 5.656 & 5.656 & 5.656 & 4.242 & 5.656 & 7.071 \\
\mathsf{/} & +\infty & 4.242 & 4.242 & 4.242 & 4.242 & 5.656 & 7.071 & 8.485 \\
\mathsf{l} & +\infty & 5.656 & 5.656 & 7.071 & 7.071 & 5.656 & 5.656 & 7.071 \\
\mathsf{o} & +\infty & 7.071 & 7.071 & 7.071 & 8.485 & 7.071 & 7.071 & 7.071 \\
\mathsf{g} & +\infty & 8.485 & 8.485 & 8.485 & 8.485 & 8.485 & 8.485 & \mathbf{8.485}
\end{array}$$

Figure 3.8: Distance computation example.

$$g(i,j) = \min \begin{cases} g(i,j-1) + d(i,j) \\ g(i-1,j-1) + d(i,j) \\ g(i-1,j) + d(i,j) \end{cases}$$

Note that $i \in [1, L_{S_t}]$ and $j \in [1, L_{M_k}]$ thus the total distance is $D(\underline{\underline{S}}_t, \underline{\underline{M}}_k) = g(L_{S_t}, L_{M_k})$. A simple example of distance computation is show in Figure 3.8 ($\mathcal{A}$ is the English alphabet plus extra characters). The overall distance is $D'(S_t, M_k) = 8.485$. We used a symmetric Gaussian neighborhood function h whose center is located at the BMU $c(t)$. More precisely

$$h(k, c(t), t) = \alpha(t) e^{-\frac{d(c(t),k)}{2\sigma^2(t)}}$$

where $\alpha(t)$ controls the learning rate and $\sigma(t)$ is the actual width of the neighborhood function. The SOM algorithm uses *two* training cycles. During (1) *adaptation* the map is more flexible, while during (2) *tuning* the learning rate $\alpha_{(\cdot)}$ and the width of the neighborhood $\sigma_{(\cdot)}$ are decreased. On each phase such parameters are denoted as $\alpha_1, \alpha_2, \sigma_1, \sigma_2$.

Symbol SOMs are "plugged" into FSA-DF by associating each transition with the *set* of BMUs learned during training. At detection, an alert occurs whenever a path argument falls into neighborhood of a non-existing BMU. Similarly, in the case of S^{2}A^{2}DE , the neighborhood function is used to decide whether the string is anomalous or not, according to a proper threshold which is the minimum value of the neighborhood function encountered during training, for each node.

3.3.2 Experimental Results

In this section we describe our efforts to cope with the lack of reliable testing datasets for intrusion detections. The testing methodology is here detailed along with the experiments we designed. Both detection accuracy and performance overhead are subjects of our tests.

In order to evaluate and highlight the impact of each specific model, we performed targeted tests rather than reporting general DRs and FPRs only. Also, we ensured that all possible alerts types are inspected (i.e., true/false positive/negative). In particular, for each IDS, we included one *legal* trace in which file operations are performed on files *never* seen during training but with a similar name (e.g., training on `/tmp/log`, testing on `/tmp/log2`); secondly, we inserted a trace which mimics an attack.

3.3.2.1 Comparison of Detection Accuracy

The detection accuracy of Hybrid IDS (H), FSA-DF (F) and S^{2}A^{2}DE (S) is here analyzed and compared. Both training parameters and detection results are summarized in Table 3.12. The parameters used to train the SOM are fixed except for $\sigma_1(t)$: $\alpha_1(t) = 0.5 \div 0.01$, $\sigma_2(t) = 3$ and $\alpha_2(t) = 0.1 \div 0.01$. Percentiles for both X_{args} and X_{edge} are detailed. The "paths/cycle%" (paths per cycle) row indicates the amount of paths arguments used for training the SOM. The settings for clustering stage of S^{2}A^{2}DE are constant: minimum number of clusters (3, or 2 in the case of the `open`); maximum merging distance (6, or 10 in the case of the `open`); the "null" and the "don't care" probability values are fixed at 0.1 and 10, respectively, while 10 is the maximum number of leaf clusters. In order to give a better understanding of how each prototype works, we analyzed by hand the detection results on each target application.

`sing`: Hybrid IDS is not tricked by the false positive mimic trace inserted. The Symbol SOM model recognizes the similarity of `/tmp/log3` with the other paths inserted in the training. Instead, both FSA-DF and S^{2}A^{2}DE raise false alarms; the former has never seen the path during training while the latter

Training throughput				
Ses.	#Calls	#Progr.	t (Clust., HMM) $[s]$	X $[call/s]$
1	97644	111	12.056 (7.683, 3.268)	8099
2	34931	67	3.415 (1.692, 1.356)	10228
3	41133	129	6.721 (3.579, 2.677)	6120
4	50239	152	7.198 (3.019, 3.578)	6979
5	38291	115	4.503 (2.219, 1.849)	8503
Avg. processing time			$1.2910 \cdot 10^{-4}$	
Detection throughput				
Ses.	#Calls	#Progr.	t $[s]$	X $[call/s]$
1	109160	149	6.722	16239
2	160565	186	12.953	12395
3	103605	143	4.653	22266
4	115334	107	5.212	22128
5	112242	147	5.674	19781
Avg. processing time			$5.6581 \cdot 10^{-5}$	

Table 3.11: Training and detection throughput X. On the same dataset, the average processing time with S^{2}A^{2}DE disabled is around 0.08741084, thus, in the worst case, S^{2}A^{2}DE introduces a 0.1476% overhead (estimated).

	`sing`	`mt-daapd`	`proftpd`	`sudo`	`BitchX`
SOM	15×15	15×15	15×15	15×15	10×10
Traces	18	18	18	18	14
Syscalls	5808	194879	64640	52034	103148
Paths	2700	2700	23632	1316	14921
Paths/cycle%	2	2	1	8	1

Table 3.12: Parameters used to train the IDSs. Values includes the number of traces used, the amount of paths encountered and the number of paths per cycle.

recognizes the string in the tree path model but an alarm is raised because of threshold violation. S^2A^2DE recognizes the attack containing the longer subsequent invocations of `mmap2`; FSA-DF also raises a violation in the file name because it has never been trained against `/etc/passwd` nor `/etc/shadow`; and Hybrid IDS is triggered because the paths are placed in a different SOM region w.r.t. the training.

`mt-daapd`: The legit traces violate the binary and unary relations causing several false alarms on FSA-DF. On the other hand, the smoother path similarity model allows Hybrid IDS and S^2A^2DE to pass the test with no false positives. The changes in the control flow caused by the attacks are recognized by all the IDSs. In particular, the DoS attack (special-crafted request sent fifty times) triggers an anomaly in the edge frequency model.

`proftpd`: The legit trace is correctly handled by all the IDSs as well as the anomalous root shell that causes unexpected calls (`setuid`, `setgid` and `execve`) to be invoked. However, FSA-DF flags more than 1000 benign system calls as anomalous because of temporary files path not present in the training.

`sudo`: Legit traces are correctly recognized by all the engines and attacks are detected without errors. S^2A^2DE fires an alert because of a missing edge in the Markov model (i.e., the unexpected execution of `chown root:root script` and `chmod +s script`). Also, the absence of the `script` string in the training triggers a unary relation violation in FSA-DF and a SOM violation in Hybrid IDS. The traces which mimic the attack are erroneously flagged as anomalous, because the system call sequences are *strictly* similar to the attack.

`BitchX`: The exploit is easily detected by all the IDSs as a control flow violation through extra `execve` system calls are invoked to execute injected commands. Furthermore, the Hybrid IDS anomaly engine is triggered by three edge frequency violations due to paths passed to the FSA when the attack is performed which are different w.r.t. the expected ones.

3.3.2.2 Specific Comparison of SOM-S^2A^2DE and S^2A^2DE

We also specifically tested how the introduction of a Symbol SOM improves over the original probabilistic tree used for modeling the path arguments in S^2A^2DE . Training parameters are reported in Table 3.14 while results are summarized in Table 3.15, the FPR decreases in the second test. However, the first test exhibits a lower FNR as detailed in the following.

The `mcweject` utility is affected by a stack overflow CVE-2007-1719 caused by improper bounds checking. Root privileges can be gained if `mcweject` is `setuid`. Launching the exploit is as easy as `eject -t illegal_payload`, but we performed it through the userland execution technique we describe in Section 3.4.1 to make it more stealthy avoiding the `execve` that obviously triggers an alert in the S^2A^2DE for a missing edge in the Markov chain. Instead, we are interested in comparing the string models only. SOM-S^2A^2DE detects it with no issues because of the use of different “types” of paths in the `opens`.

An erroneous computation of a buffer length is exploited to execute code via a specially crafted PAX archives passed to `bsdtar` (CVE-2007-3641). A heap overflow allows to overwrite a structure pointer containing itself another pointer to a function called right after the overflow. The custom exploit [Maggi et al., 2008] basically redirects that pointer to the injected shellcode. Both the original string model and the Symbol SOM models detect the attack when an unexpected special file (i.e., `/dev/tty`) is opened. However, the original model raises many false positives when significantly different paths are encountered. This situation is instead handled with no false positives by the smooth Symbol SOM model.

Since this dataset has been originally prepared for [Maggi et al., 2008, 2009a], details on its generation are described in Section 3.1.

3.3.2.3 Performance Evaluation and Complexity Discussion

We performed both empirical measurements and theoretical analysis of the performance of the various proposed prototypes. Detection speed results are summarized in Table 3.16. The datasets for detection accuracy were reused: we selected the five test applications on which the IDSs performed worst. Hybrid IDS is slow because the BMU algorithm for the symbol SOM is invoked for each system call with a path argument (`opens` are quite frequent), slowing down the

	`sing`	`mt-daapd`	`profdtpd`	`sudo`	`BitchX`
Traces	22	18	21	22	15
Syscalls	1528	9832	18114	3157	107784
S^2A^2DE	10.0%	0%	0%	10.0%	0.0%
FSA-DS	5.0%	16.7%	28%	15.0%	0.0%
Hybrid IDS	0.0%	0%	0%	10.0%	0.0%

Table 3.13: Comparison of the FPR of S^2A^2DE vs. FSA-DF vs. Hybrid IDS. Values include the number of traces used. Accurate description of the impact of each *individual* model is in Section 3.3.2.1

	`mcweject`	`bsdtar`
SOM	15×15	15×15
Traces	10	240
Syscalls	84	12983
Paths	48	3477
Paths/cycle%	50	2

Table 3.14: Parameters used to train the IDSs. Values includes the number of traces used, the amount of paths encountered and the number of paths per cycle.

	`mcweject`	`bsdtar`
Traces	12	2
Syscalls	75	102
S^2A^2DE	0.0%	8.7%
SOM-S^2A^2DE	0.0%	0.0%

Table 3.15: Comparison of the FPR of S^2A^2DE vs. SOM-S^2A^2DE . Values include the number of traces used.

	`sing`	`sudo`	`BitchX`	`mcweject`	`bsdtar`	
System calls	3470	15308	12319	97	705	Speed
S^2A^2DE	0.4	0.8	1.9	0.1	0.1	8463
FSA-DF	1.3	1.5	1.2	-	-	7713
Hybrid IDS	29	5.8	27.7	-	-	1067
SOM-S^2A^2DE	-	-	-	8.8	19	25

Table 3.16: Detection performance measured in "seconds per system call". The average speed is measured in system calls per second (last column).

detection phase. Also, we recall that the current prototype relies on a system call interceptor based on `ptrace` which *introduces high run-time overheads*, as shown in [Bhatkar et al., 2006]. To obtain better performance, an in-kernel interceptor could be used. The theoretical performance of each engine can be estimated by analyzing the bottleneck algorithm.

3.3.2.4 Complexity of FSA-DF

During training, the bottleneck is the binary relation learning algorithm. $T_F^{train} = O(S \cdot M + N)$, where M is the total number of system calls, $S = |Q|$ is the number of states of the automaton, and N is the sum of the length of all the string arguments in the training set. At detection $T_{FSA-DF}^{det} = O(M + N)$.

Assuming that each system call has $O(1)$ arguments, the training algorithm is invoked $O(M)$ times. The time complexity of each i-th iteration is $Y_i + |X_i|$, where Y_i is the time required to compute all the unary and binary relations and $|X_i|$ indicates the time required to process the $i - th$ system call X. Thus, the overall complexity is bounded by $\sum_{i=1}^{M} Y + |X_i| = M \cdot Y + \sum_{i=1}^{M} |X_i|$. The second factor $\sum_{i=1}^{M} |X_i|$ can be simplified to N because strings are represented as a tree; it can be shown [Bhatkar et al., 2006] that the total time required to keep the longest common prefix information is bounded by the total length of all input strings. Furthermore, Y is bounded by the number of unique arguments, which in turn is bounded by S; thus, $T_F^{train} = O(S \cdot M + N)$. This also prove the time complexity of the detection algorithm which, at each state and for each input, requires unary and binary checks to be performed; thus, its cost is bounded by $M + N$. ■

3.3.2.5 Complexity of Hybrid IDS

In the training phase, the bottleneck is the Symbol SOM creation time: $T_H^{train} = O(C \cdot D \cdot (L^2 + L))$, where C is the number of learning cycles, D is the number of nodes, and L is the maximum length of an input string. At detection time $T_H^{det} = O(M \cdot D \cdot L^2)$.

T_H^{train} depends on both the number of training cycles, the BMU algorithm and node updating. The input is randomized at each training session and a constant amount of paths is used, thus the input size is $O(1)$. The BMU algorithm depends on both the SOM size and the distance computation, bounded by $L_{input} \cdot L_{node} = L^2$, where L_{input} and L_{node} are the *lengths* of the input string and the node string, respectively. More precisely, the distance between strings is performed by comparing all the vectors representing, respectively, each character of the input string and each character of the node string. The char-by-char comparison is performed in $O(1)$ because the size of each character vector is fixed. Thus, the distance computation is bounded by $L^2 \simeq L_{input} \cdot L_{node}$. The node updating algorithm depends on both the number of nodes D, the length of the node string L_{node} and the training cycles C, hence each cycle requires $O(D \cdot (L^2 + L))$, where L is the length of the longest string. The creation of the FSA is similar to the FSA-DF training, except for the computation of the relations between strings which time is no longer $O(N)$ but it is bounded by $M \cdot D \cdot L^2$ (i.e., the time required to find the Best Matching Unit for one string). Thus, according to *Proof 1*, this phase requires $O(S \cdot M + M \cdot D \cdot L^2) < O(C \cdot D \cdot (L^2 + L))$. The detection time T_H^{det} is bounded by the BMU algorithm, that is $O(M \cdot D \cdot L^2)$. ■

The clustering phase of S^{2}A^{2}DE is $O(N^2)$ while with SOM-S^{2}A^{2}DE it grows to $O(N^2 L^2)$.

In the worst case, the clustering algorithm used in [Maggi et al., 2009a] is known to be $O(N^2)$, where N is the number of system calls: the distance function is $O(1)$ and the distance matrix is searched for the two closest clusters. In the case of SOM-S^{2}A^{2}DE , the distance function is instead $O(L^2)$ as it requires one run of the BMU algorithm. ■

3.4 Forensics Use of Anomaly Detection Techniques

Anti-forensics is the practice of circumventing classical forensics analysis procedures, making them unreliable or impossible. In this section we describe how machine learning algorithms and anomaly detection techniques can be exploited to cope with a wide class of definitive anti-forensics techniques. In [Maggi et al., 2008] we tested S^2A^2DE , described in Section 3.2 including the improvements detailed in Section 3.3, on a dataset we created through the implementation of an innovative technique of anti-forensics, and we show that our approach yields promising results in terms of detection.

Computer forensics is usually defined as the process of applying scientific, repeatable analysis processes to data and computer systems, with the objective of producing evidence that can be used in an investigation or in a court of law. More in general, it is the set of techniques that can be applied to understand if, and how, a system has been used or abused to commit mischief [Mohay et al., 2003]. The increasing use of forensic techniques has led to the development of anti-forensic techniques that can make this process difficult, or impossible [Garfinkel, 2007; Berghel, 2007; Harris, 2006].

Anti-forensics techniques can be divided into at least two groups, depending on their target. If the *identification* phase is targeted, we have *transient* anti-forensics techniques, which make the acquired evidence difficult to analyze with a specific tool or procedure, but not impossible to analyze in general. If instead the *acquisition* phase is targeted, we have the more effective class of *definitive* anti-forensics techniques, which effectively deny once and forever any access to the evidence. In this case, the evidence may be destroyed by the attacker, or may simply not exist on the media. This is the case of in-memory injection techniques that are described in the following.

In particular, we propose the use of machine learning algorithms and anomaly detectors to circumvent such techniques. Note that, referring once again to the aforementioned definition of computer forensics in [Mohay et al., 2003], we focused only on detecting *if* a system has been compromised. In fact, if definitive anti-forensics techniques can make it impossible to detect *how* the system has been exploited. We illustrate a prototype of anomaly detector which analyzes the sequence and the arguments of system calls to detect intrusions. We also use this prototype to detect in-memory injections of executable code, and in-memory execution of binaries (the so-called "userland exec" technique, which we re-implement in a reliable way). This creates a usable audit trail, without needing to resort to complex memory dump and analysis operations [Burdach, 2009; Ring and Cole, 2004].

3.4.1 Problem statement

Anti-forensics is defined by symmetry on the traditional definition of computer forensics: it is the set of techniques that an attacker may employ to make it difficult, or impossible, to apply scientific analysis processes to the computer systems he penetrates, in order to gather evidence [Garfinkel, 2007; Berghel, 2007; Harris, 2006]. The final objective of anti-forensics is to reduce the quantity and spoil the quality [Grugq, 2005] of the evidence that can be retrieved by an investigation and subsequently used in a court of law.

Following the widely accepted partition of forensics [Pollitt, 1995] in *acquisition*, *identification*, *evaluation*, and *presentation*, the only two phases where technology can be critically sabotaged are both acquisition and identification. Therefore, we can define anti-forensics as follows.

Definition 3.4.1 (Anti-forensics) *Anti-forensics* is a combination of all the methods that make acquisition, preservation and analysis of computer-generated and computer-stored data difficult, unreliable or meaningless for law enforcement and investigation purposes.

Even if more complex taxonomies have been proposed [Harris, 2006], we can use the traditional partition of the forensic process to distinguish among two types of anti-forensics:

Transient anti-forensics when the *identification* phase is targeted, making the acquired evidence difficult to analyze with a specific tool or procedure, but not impossible to analyze in general.

Definitive anti-forensics when the *acquisition* phase is targeted, ruining the evidence or making it impossible to acquire.

Examples of transient anti-forensics techniques are the fuzzing and abuse of file-systems in order to create malfunctions or to exploit vulnerabilities of the tools used by the analyst, or the use of log analysis tools vulnerabilities to hide or modify certain information [Foster and Liu, 2005; Grugq, 2005]. In other cases, entire file-systems have been hidden inside the metadata of other file-systems [Grugq, 2005], but techniques have been developed to cope with such attempts [Piper et al., 2006]. Other examples are the use of steganography [Johnson and Jajodia, 1998], or the modification of file metadata in order to make filetype not discoverable. In these cases the evidence is not completely unrecoverable, but it may escape any quick or superficial examination of the media: a common problem today, where investigators are overwhelmed with cases and usually under-trained, and therefore overly reliant on tools.

Definitive anti-forensics, on the other hand, effectively denies access to the evidence. The attackers may encrypt it, or securely delete it from file-systems (this process is sometimes called "counter-forensics") with varying degrees of success [Geiger, 2005; Garfinkel and Shelat, 2003]. Access times may be rearranged to hide the time correlation that is usually exploited by analysts to reconstruct the events timeline. The final anti-forensics methodology is not to leave a trail: for instance, modern attack tools (commercial or open source) such as Metasploit [Metasploit, 2009], Mosdef or Core IMPACT [Core Security Technologies, 2009] focus on pivoting and in-memory injection of code: in this case, nothing or almost nothing is written on disk, and therefore information on the attack will be lost as soon as it is powered down, which is usually standard operating procedure on compromised machines. These techniques are also known as "disk-avoiding" procedures.

Memory dump and analysis operations have been advocated in response to this, and tools are being built to cope with the complex task of the reliable acquisition [Burdach, 2009; Schatz, 2007] and analysis [Burdach, 2009; Ring and Cole, 2004; Nick L. Petroni et al., 2006] of a modern system's memory. However, even in the case that the memory can be acquired and examined, if the process injected and launched has already terminated, once more, no trace will be found of the attack: these techniques are much more useful against in-memory resident backdoors and rootkits, which by definition are persistent.

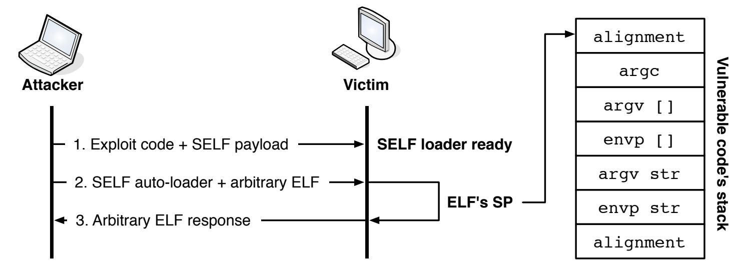


Figure 3.9: An illustration of the in-memory execution technique.

3.4.2 Experimental Results

Even in this evaluation, we used the dataset described in Section 3.1. However, a slight modification of the generation mechanism was needed. More precisely, in the tests conducted we used a modified version of SELF [Pluf and Ripe, 2005], which we improved in order to reliably run under FreeBSD 6.2 and ported to a form which could be executed through code injection (i.e., to shellcode format). SELF implements a technique known as *userland exec*: it modifies any statically linked *Executable Linux Format* (ELF) binary and, by building a specially-crafted stack, it allows an attacker to load and run that ELF in the memory space of a target process without calling the kernel and, more importantly, without leaving any trace on the hard disk of the attacked machine. This is done through a two-stage attack where a shellcode is injected in the vulnerable program, and then retrieves a modified ELF from a remote machine, and subsequently injects it into the memory space of the running target process, as shown schematically in Figure 3.9.

Eight experiments with both `eject` and `bsdtar` were performed. Our anomaly detector was first trained with ten different execution of `eject` and more than a hundred executions of `bsdtar`. We also audited eight instances of the activity of `eject` under attack, while for `bsdtar` we logged seven malicious executions. We repeated the tests both with a simple shellcode which opens a root shell (a simple `execve` of `/bin/sh`) and with our implementation of the userland exec technique.

The overall results are summarized in Table 3.17. Let us consider the effectiveness of the detection of the attacks themselves. The

Executable	Regular shellcode		Userland exec	
	FPR	DR	FPR	DR
`eject`	0%	75%	0%	100%
`bsdtar`	7.81%	71%	7.81%	100%

Table 3.17: Experimental results with a regular shellcode and with our userland exec implementation.

attacks against `eject` are detected with no false positive at all. The exploit is detected in the very beginning: since a very long argument is passed to the `execve`, this triggers the argument model. The detection accuracy is similar in the case of `bsdtar`, even if in this case there are some false positives. The detection of the shellcode happens with the first `open` of the unexpected special file `/dev/tty`. It must be underlined that most of the true alerts are correctly fired at system call level; this means that malicious *calls* are flagged by our IDS because of their unexpected arguments, for instance.

On the other hand, exploiting the userland exec an attacker launches an otherwise normal executable, but of course such executable has different system calls, in a different order, and with different arguments than the ones expected in the monitored process. This reflects in the fact that we achieved a 100% DR with no increase in false positives, as each executable we have run through SELF has produced a Markov model which significantly differs from the learned one for the exploited host process.

3.5 Concluding Remarks

In this chapter we first described in detail two contributions to host-based anomaly detection and then demonstrated how these techniques can be successfully applied to circumvent anti-forensics tools often used by intruders to avoid to leave traces on the file-system.

First, we described a host-based IDS based on the analysis of system calls arguments and sequences. In particular, we analyzed previous literature on the subject, and found that there exists only a handful of works which take into account the anomalies in such arguments. We improved the models suggested in one of these works, we added a stage of clustering in order to characterize normal invocations of calls and to better fit models to arguments, and finally we complemented it with Markov models in order to capture correlation between system calls.

We showed how the prototype is able to correctly contextualize alarms, giving the user more information to understand what caused any false positive, and to detect variations over the execution flow, as opposed to punctual variations over single instances. We also demonstrated its improved detection capabilities, and a reduction of false positives. The system is auto-tuning and fully unsupervised, even if a range of parameters can be set by the user to improve the quality of detection.

A possible future extension of the technique we described is the analysis of complementary approaches (such as Markov model merging or the computation of distance metrics) to better detect anomalies in the case of long system call sequences, which we identified as a possible source of false positives. In the following section, we describe how a deterministic behavioral model, i.e., an FSA, often shows a better and more accurate characterization of the process flow. However, to achieve better FPR of S^2A^2DE we had to add several stochastic models to avoid many false detections; this, as expected, is paid at the price of higher computational overheads.

Secondly, we demonstrated that a good alternative to using distance metrics between Markov models is the exploiting deterministic models for the control flow. More precisely, we showed how the deterministic dataflow relations between arguments implemented in FSA-DF can be improved by using better statistical models. In addition, we partially addressed the problem of spurious datasets by introducing a Gaussian string model, which has been shown to be more resilient to outliers (i.e. too long or too short strings). We also proposed a new model for counting the frequency of traversal of edges on FSA-DF, to make it able to detect DoS attacks. Both systems needed an improved model for string (path) similarity. We adapted the Symbol SOM algorithm to make it suitable for computing a distance between two paths. We believe that this is the core contribution.

We tested and compared the original prototypes with an hybrid solution where the Symbol SOM and the edge traversal models are applied to the FSA, and a version of S^2A^2DE enhanced with the Symbol SOM and the correction to the execution arguments model. Both the new prototypes have the *same* DRs of the original ones, but significantly *lower* FPRs. This is paid in terms of a non-negligible limit to detection speed, at least in our proof of concept implementation.

Future efforts will focus on re-engineering the prototypes to use an in-kernel system call interceptor, and generically improve their performance. We are studying how to speed up the Symbol SOM node search algorithm, in order to bring the throughput to a rate suitable for online use.

Last, we analyzed the wide class of *definitive* anti-forensics techniques which try to eliminate evidence by avoiding disk usage. In particular, we focused on in-memory injection techniques. Such techniques are widely used by modern attack tools (both commercial and open source).

As memory dump and analysis is inconvenient to perform, often not part of standard operating procedures, and does not help except in case of in-memory resident backdoors and rootkits, we proposed an alternative approach to circumvent such techniques. We illustrated how a prototype which analyzes (using learning algorithms) the sequence and the arguments of system calls to detect intrusions can be used to detect in-memory injections of executable code, and in-memory execution of binaries.

We proposed an experimental setup using vulnerable versions of two widely used programs on FreeBSD, `eject` and `bsdtar`. We described the creation of a training and testing dataset, how we adapted or created exploits for such vulnerabilities, and how we recorded audit data. We also developed an advanced in-memory execution payload, based on SELF, which implements the "userland exec" technique through an injectable shellcode and a self-loading object (a specially-crafted, statically linked ELF file). The payload executes any statically linked binary in the memory space of a target process without calling the kernel and, more importantly, without leaving any trace on the hard disk of the attacked machine.

We performed several experiments, with excellent DRs for the *exploits*, but even more importantly with a 100% DR for the in-memory execution payload itself. We can positively conclude that our technique yields promising results for creating a forensic audit trail of otherwise "invisible" injection techniques. Future developments will include a more extensive testing with different anti-forensics techniques, and the development of a specifically designed forensic output option for our prototype.

Anomaly Detection of Web-based Attacks 4

Anomaly detection techniques have been shown to be very effective at mitigating the widespread of malicious activity against web applications. In fact, nowadays, the underground criminals' preferred way of spreading malware consists in compromising vulnerable web applications, deploying a phishing-, spamming-, malware-kit and infecting an enormous amount of users that simply visit the website using a vulnerable browser (or plug-in). Further exacerbating the situation is the use of botnets to exhaustively compromise vast amounts of web applications. Thus, due to their popularity, web applications play a significant role in the malware spreading work-flow. As a consequence, by blocking attacks against web applications the majority of potential infections can be avoided. Indirectly, all the visitors of the website benefit from the adoption of web-based IDSs.

Unfortunately, even the most advanced anomaly detectors of web-based attacks are not with their drawbacks. In particular, in this chapter we discuss our contributions to overcome two relevant training issues, overfitting and concept-drift, that both manifest themselves as increased FPRs. First, we address overfitting due to training data scarcity, which results in under-trained activity models with poor generalization capabilities. Consequently, a considerable amount of normal events is classified as malicious. Secondly, we propose a simple but extremely effective mechanism to detect changes in the monitored system to adapt the models to what we named web application concept drift.

4.1 Preliminaries

The two contributions described in Section 4.2 and 4.3 are both based on the generic anomaly detection architecture described in the following. In this section, a generic model of an anomaly-based detector of attacks against web applications is given. In addition, the details of the datasets used to evaluate the effectiveness of both the approach are described.

4.1.1 Anomaly Detectors of Web-based Attacks

Unless differently stated, we use the shorthand term *anomaly detector* to refer to anomaly-based detectors that leverage unsupervised machine learning techniques. Let us first overview the basic concepts.

Without loss of generality, a set of web applications A can be organized into a set of *resource paths* or *components* R, and named *parameters* P. For example, $A = \{a_1, a_2\}$ may contain a blog application $a_1 =$ `blog.example.com` and an e-commerce application $a_2 =$ `store.example.com`. They can be decomposed into their resource paths:

$$\begin{array}{cc} R_1 = & R_2 = \\ \left\{ \begin{array}{l} r_{1,1} = /\texttt{article}/, \\ r_{1,2} = /\texttt{comments}/, \\ r_{1,3} = /\texttt{comments}/\texttt{edit}/, \\ r_{1,4} = /\texttt{account}/, \\ r_{1,5} = /\texttt{account}/\texttt{password}/ \end{array} \right\} & \left\{ \begin{array}{l} r_{2,1} = /\texttt{list}/, \\ r_{2,2} = /\texttt{cart}/, \\ r_{2,3} = /\texttt{cart}/\texttt{add}/, \\ r_{2,4} = /\texttt{account}/, \\ r_{2,5} = /\texttt{account}/\texttt{password}/ \end{array} \right\} \end{array}$$

In this example, resource path $r_{1,5}$ might take a set of parameters, as part of the HTTP request. For instance, the following request to $r_{1,5} =$ `/account/password/`

```
GET /account/password/id/12/oldpwd/14m3/newpwd/1337/ HTTP/1.1
```

can be parsed into the following abstract data structure:

$$P_{1,5} = \left\{ \begin{array}{llll} \langle p_{1,5,1} = & \texttt{id} & v_{1,5,1} = & \texttt{12}\rangle, \\ \langle p_{1,5,2} = & \texttt{oldpw} & v_{1,5,2} = & \texttt{14m3}\rangle, \\ \langle p_{1,5,3} = & \texttt{newpw} & v_{1,5,3} = & \texttt{1337}\rangle \end{array} \right\}.$$

A generic web application IDS based on unsupervised learning techniques captures the system activity $\mathbb{I}$ as a sequence of *requests* $Q = \{q_1, q_2, \ldots\}$, similar to those exemplified in Section 2.1, issued by web clients to the set of monitored web applications. Each request $q \in Q$ is represented by a tuple $\langle a_i, r_{i,j}, P_q \rangle$, where P_q is a set of parameter name-value pairs such that $P_q \subseteq P_{i,j}$.

During the initial *training phase*, the anomaly detection system learns the characteristics of the monitored web applications in terms of *models*. As new web application, resource path, and parameter instances are observed, the sets A, R, and P are updated. For each unique parameter $p_{i,j,k}$ observed in association with a particular application a_i and path $r_{i,j}$, a set of models that characterize the various features of the parameter is constructed. An activity profile (Definition 2.1.3) associated with each unique parameter instance is generated $c_{(.)} = \langle m^{(1)}, m^2, \dots, m^{(u)}, \dots, m^{(U)} \rangle$, which we name *(parameter) profile* or *model composition*. Therefore, for each application a_i and resource path $r_{i,j}$, a set $\mathcal{C}_{i,j}$ of model compositions is constructed, one for each parameter $p_{i,j,k} \in P_{i,j}$. The *knowledge base* of an anomaly detection system trained on web application a_i is denoted by $\mathcal{C}_{a_i} = \bigcup_j \mathcal{C}_{i,j}$. A graphical representation of how a knowledge base is modeled for multiple web applications is depicted in Figure 4.1.

Example 4.1.1 In webanomaly, a profile for a given parameter $p_{i,j,k}$ is the following tuple:

$$c_{i,j,k} = \langle m^{(\text{tok})}, m^{(\text{int})}, m^{(\text{len})}, m^{(\text{char})}, m^{(\text{struct})} \rangle.$$

Similarly to LibAnomaly:

$m^{(\text{tok})}$ models parameter values as a set of legal tokens (i.e., the set of of possible values for the parameter `gender`, observed during training).

$m^{(\text{int})}$ and $m^{(\text{len})}$ describe normal intervals for literal integers and string lengths, respectively, using the Chebyshev inequality.

$m^{(\text{char})}$ models the character strings as a ranked frequency histogram, named *Idealized Character Distribution* (ICD), that are compared using the χ^2 or G tests.

$m^{(\text{struct})}$ models sets of character strings by inducing a HMM. The HMM encodes a probabilistic grammar that represents a superset of the strings observed in a training set. Aside from the addition of $m^{(\text{int})}$, which is a straightforward generalization of

$m^{(\text{len})}$ to numbers, the interested reader may refer to [Kruegel et al., 2005] for further details.

In addition to requests, the structure of user sessions can be taken into account to model the normal states of a server-side application. In this case, the anomaly detector does not consider individual requests independently, but models their sequence. This model captures the legitimate order of invocation of the resources, according to the application logic. An example is when a user is required to invoke an authentication resource (e.g., `/user/auth`) before requesting a private page (e.g., `/user/profile`). In [Kruegel et al., 2005], a session S is defined as a sequence of resources in R. For instance, given $R = \{r_1, r_2, \ldots, r_{10}\}$, a sample session is $S = \langle r_3, r_1, r_2, r_{10}, r_2 \rangle$.

Some systems also model HTTP responses that are returned by the server. For example, in [Kruegel et al., 2005], a model $m^{(\text{doc})}$ is presented that takes into account the structure of documents (e.g., *HyperText Markup Language* (HTML), XML, and *JavaScript Object Notation* (JSON)) in terms of partial trees that include security-relevant nodes (e.g., `<script />` nodes, nodes containing DOM event handlers, and nodes that contain sensitive data such as credit card numbers). These trees are iteratively merged as new documents are observed, creating a superset of the allowed document structure and the positions within the tree where client-side code or sensitive data may appear. A similar approach is adopted in [Criscione et al., 2009].

Note 4.1.1 (Web Application Behavior) The concept of *web application behavior*, used in the following, is a particular case of the system behavior as defined by Definition 2.1.4. Depending on the accuracy of each model, the behavior of a web application reflects the characteristics and functionalities that the application offers and, as a consequence, the content of the inputs (i.e., the requests) that it process and the outputs (i.e., the responses) that it produces. Thus, unless differently stated, we use the same term to indicate both the ideas.

After training, the system is switched to *detection mode*, which is performed online. The models trained in the previous phase are queried to determine whether or not the new parameters observed are anomalous. Without going into the details of a particular implementation, each parameter is compared to all the applicable models

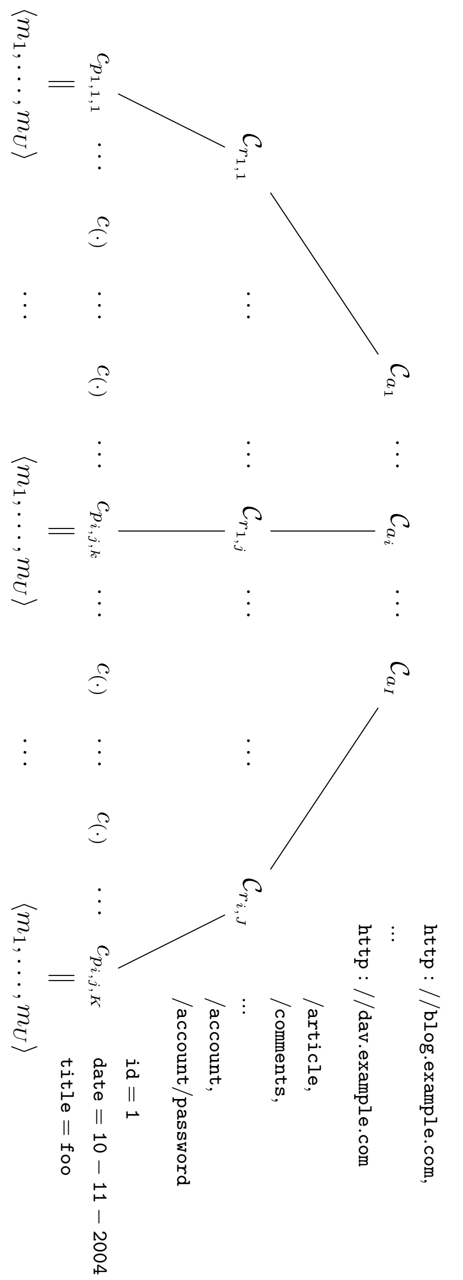


Figure 4.1: Overview of web application model construction.

and an aggregated anomaly score between 0 and 1 is calculated by composing the values returned by the various models. If the anomaly score is above a certain threshold, an alert is generated. For example, in [Kruegel et al., 2003b], the anomaly score represents the probability that a parameter value is anomalous and it is calculated using a Bayesian network that encodes the probability that a parameter value is actually normal or anomalous given the scores returned by the individual models.

4.1.2 A Comprehensive Detection System to Mitigate Web-based Attacks

The approach described in [Kruegel et al., 2005] are further exploited in a recent work, [Criscione et al., 2009], which we partially contributed to. In particular, we defined a web application anomaly detector that is able to detect a real-world threats against the clients (e.g., malicious JavaScript code, trying to exploit browser vulnerabilities), the application (e.g., cross-site scripting, permanent content injection), and the database layer (e.g., SQL injection). A prototype of the system, called Masibty, has been evaluated on a set of real-world attacks against publicly available applications, using both simple and mutated versions of exploits, in order to assess the resilience to evasion. Masibty has the following key characteristics:

- its models are designed with the explicit goal of not requiring an attack-free dataset for training, which is an unrealistic requirement in real-world applications. Even if in [Cretu et al., 2008a] techniques are suggested to filter outliers (i.e., attacks) from the training data, in absence of a ground truth there can be no guarantee that the dataset will be effectively free of attacks. Using such techniques before training Masibty would surely improve its detection capabilities.

- As depicted in Figure 4.2, Masibty intercepts and process both HTTP requests (i.e., *PAnomaly*) and responses (i.e., *XSSAnomaly*) and protects against both server-side and client-side attacks, an extremely important feature in the upcoming "Web 2.0" era of highly interactive websites based mainly on user contributed content. In particular, we devised two novel anomaly detection models — referred to as "engines" — based on the representation of the responses as trees.

- Masibty incorporates an optional data protection component, i.e., *QueryAnomaly* that extracts and parses the SQL queries sent to the database server. This component is part of the analysis of HTTP requests, and thus is not merely a reverse proxy to the database. In fact, it allows to bind the requests to the SQL queries that they generate, directly or indirectly. Hence, queries can be modeled although are not explicitly passed as a parameter of the HTTP requests.

4.1.3 Evaluation Data

The experiments in Section 4.2 and 4.3 were conducted using a dataset drawn from real-world web applications deployed on both academic and industry web servers. Examples of representative applications include payroll processors, client management, and online commerce sites. For each application, the full content of each HTTP connection observed over a period of several months was recorded. The resulting flows were then filtered using the most advanced signature-based detection system, Snort[1], to remove known attacks. In total, the dataset contains 823 distinct web applications, 36,392 unique resource paths, 16,671 unique parameters, and 58,734,624 HTTP requests.

Note 4.1.2 (Dataset privacy) To preserve the privacy, the dataset used has been given to the Computer Security Laboratory of UC Santa Barbara under strict contractual agreements that denies to disclose specific information identifying the web applications themselves.

Two sets of 100,000 and 1,000 attacks was introduced into the dataset used for the approach described in Section 4.2 and 4.3, respectively. These attacks were real-world examples and variations upon XSS (e.g., CVE-2009-0781), SQL injections (e.g., CVE-2009-1224), and command execution exploits (e.g., CVE-2009-0258) that manifest themselves in request parameter values, which remain the most common attacks against web applications. Representative examples of these attacks include:

- malicious JavaScript inclusion

[1]Source and attack signatures available for download at `http://snort.org`.

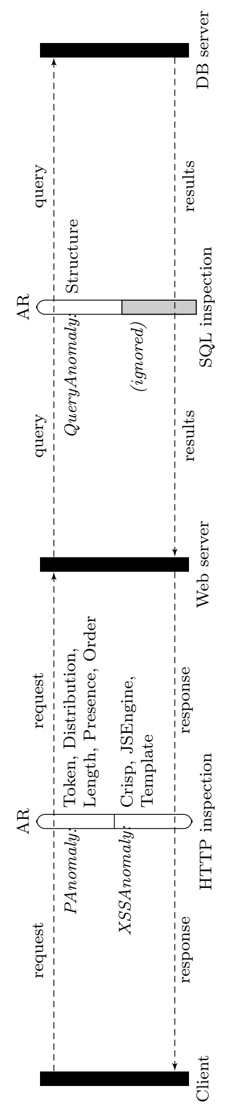


Figure 4.2: The logical structure of Masibty.

```
<script src=”http://example.com/malware.js”></script>
```

- bypassing login authentication

  ```
  ’ OR ’x’=’x’--
  ```

- command injection

  ```
  cat /etc/passwd | mail attacker@gmail.com \#
  ```

More precisely, the XSS attacks are variations on those listed in [Robert Hansen, 2009], the SQL injections were created similarly from [Ferruh Mavituna, 2009], and the command execution exploits were variations of common command injections against the Linux and Windows platforms.

4.2 Training With Scarce Data

In this section, we describe our contributions [Robertson et al., 2009] to cope with the difficulty of obtaining sufficient *amounts* of training data. As we explained in Section 2.2.2 and 2.4, this limitation has heretofore not been well studied. We developed this technique to work with webanomaly and, in general, to any learning-based anomaly detection system against web attacks. In fact, the problem described in the following is significantly evident in the case of web applications. However, it can be easily applied to other anomaly-based system, as long as they rely on behavioral models and learning techniques such as those described in Section 3.2.

The issue that motivates this work is that the number of web application component invocations is non-uniformly distributed. In fact, we noticed that relatively few components are dominant in the traffic and the remainder components are accessed relatively infrequently. Therefore, for those components, it is difficult to gather enough training data to accurately model their normal behavior. In statistics, the infrequently accessed population is known as the "long tail". Note that, however, this does not necessarily imply a power law distribution. Clearly, components that are infrequently accessed lead to undertrained models (i.e., models that do not capture the normal behavior accurately). Consequently, models are subject to overfitting due to lack of data, resulting in increases in the FPR.

In particular, in this section, we describe our joint work with UC Santa Barbara in which we propose to mitigate this problem by exploiting natural similarities among all web applications. In particular, we show that the values of the parameters extracted from HTTP requests can generally be categorized according to their type, such as an integer, date, or string. Indeed, our experiments demonstrate that parameters of similar type induce similar models of normal behavior. This result is then exploited to supplement a local scarcity of training data for a given web application component with similar data from other web applications.

This section is structured as follows. In Section 4.2.1 the problem of the non-uniform distribution to the different components of a web application is discussed. In addition, we provide evidence that it occurs in the real world. In Section 4.2.2 approach to address the problem of model profile undertraining by using the notion of *global profiles* that exploit similarities between web application parameters of similar type. In Section 4.2.3 the application of global profiles is evaluated on a large dataset of real-world traffic from many web applications, and demonstrate that utilizing global profiles allows anomaly detectors to accurately model web application components that would otherwise be associated with undertrained models.

4.2.1 Non-uniformly distributed training data

To describe the undertraining problem, in this section we refer to the aforementioned, generic architecture of a web-based anomaly detector. To address the problem of undertraining, we leverage the set of models described in the Example 4.1.1.

Because anomaly detection systems dynamically learn specifications of normal behavior from training data, it is clear that the quality of the detection results critically relies upon the quality of the training data. For example, as mentioned in Section 2.4, a training dataset should be attack-free and should accurately represent the normal behavior of the modeled features. To our knowledge, the difficulty of obtaining sufficient amounts of training data to accurately model web applications is less well-known. In a sense, this issue is similar to those addressed by statistical analysis methods with missing data [Little and Rubin, 1987]. Although a training procedure would benefit from such mechanisms, they require a complete redesign of the training algorithm specific to each model. Instead, a non-obtrusive approach that can improve an existing system without modifying the undertrained models is more desirable. Typically, anomaly-based detectors cannot assume the presence of a testing environment that can be leveraged to generate realistic training data that exercises the web application in a safe, attack-free environment. Instead, an anomaly detection system is typically deployed in front of live web applications with no *a priori* knowledge of the applications' components and their behavior.

In particular, in the case of low-traffic web applications, problems arise if the rate of client requests is inadequate to allow models to train in a timely manner. Even in the case of high-traffic web applications, however, a large subset of resource paths might fail to receive enough requests to adequately train the associated models. This phenomenon is a direct consequence of the fact that resource path invocations issued by web clients often follow a non-uniform distribution. To illustrate this point, Figure 4.3 plots the normalized

Resource Path	Requests
`/article`	95.0%
`/comments`	3.0%
`/comments/edit`	1.8%
`/account`	0.2%
`/account/password`	0.02%

Table 4.1: Client access distribution on a real-world web application (based on 500,000 requests per day).

cumulative distribution function of web client resource path invocations for a variety of real-world, high-traffic web applications (details on the source of this data are provided in Section 4.2.3). Although several applications have an approximately uniform client access distribution, a clear majority exhibits highly skewed distributions. Indeed, in many cases, a large percentage of resource paths receive a comparatively minuscule number of requests. Thus, returning to the example mentioned in Section 4.1.1, assuming an overall request volume of 500,000 requests per day, the resource path set would result in the client access distribution detailed in Table 4.1.

Profiles for parameters to resource paths such as `/article` will receive ample training data, while profiles associated with account components will be undertrained. The impact of the problem is also magnified by the fact that components of a web application that are infrequently exercised are also likely to contain a disproportionately large portion of security vulnerabilities. This could be a consequence of the reduced amount of testing that developers invariably perform on less prominent components of a web application, resulting in a higher rate of software flaws. In addition, the relatively low request rate from users of the web application results in a reduced exposure rate for these flaws. Finally, when flaws are exposed and reported, correcting the flaws may be given a lower priority than those in higher traffic components of a web application.

4.2.2 Exploiting global knowledge

The approach described in this section is based on a key observation: parameters associated with the invocation of components belonging

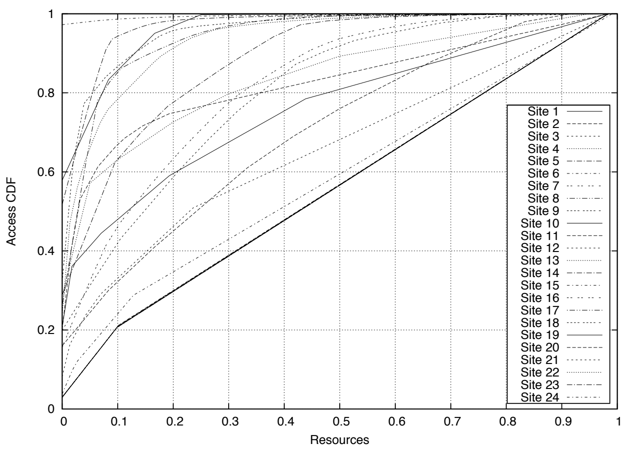


Figure 4.3: Web client resource path invocation distributions from a selection of real-world web applications.

to different web applications often exhibit a marked similarity to each other. For instance, many web applications take an integer value as a unique identifier for a class of objects such as a blog article or comment, as in the case of the `id` parameter. Similarly, applications also accept date ranges similar to the `date` parameter as identifiers or as constraints upon a search request. Similarly, as in the case of the `title` parameter, web applications often expect a short phrase of text as an input, or perhaps a longer block of text in the form of a comment body. In some sense, each of these groupings of similar parameters can be considered as distinct *parameter types*, though this need not necessarily correspond to the concept of types as understood in the programming languages context.

Our approach is based in the following assumption. Parameters of the same type tend to induce model compositions that are similar to each other in many respects. Consequently, if the lack of training data for a subset of the components of a web application prevents an anomaly detection system from constructing accurate profiles for the parameters of those components, we claim that it is possible to substitute, with an acceptably low decrease in the DR, profiles for similar parameters of the same type that were learned when enough training data was available to construct accurate profiles. It must be under-

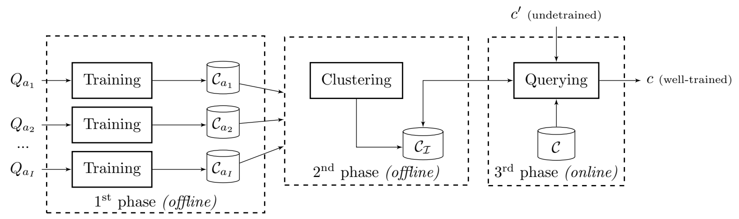


Figure 4.4: Overall procedure. Profiles, both undertrained and well-trained, are collected from a set of web applications. These profiles are processed offline to generate the global knowledge base $\mathcal{C}$ and index $\mathcal{C}^I$. $\mathcal{C}$ can then be queried to find similar global profiles.

lined that the substitution operates at the granularity of *parameters* rather than *requests* (which may contain more than one parameter). This increases the likelihood of finding applicable profile similarities, and allows for the substitution of models taken from radically different components. However, although the experiments on real-world data, described in Section 4.2.3, confirm that the aforementioned insight is realistic, our hypothesis might not hold in some very specific settings. Thus, to minimize the risks brought by migrating global knowledge across different deployments, we interpreted this result only as an insight and developed a robust criterion able to find similar profiles independently from the actual types of the modeled parameters.

As schematized in Figure 4.4 our approach is composed of three phases.

First phase (offline) Enhancement of the training procedure originally implemented in [Kruegel et al., 2005].

Second phase (offline) It is divided into three sub-steps.

1. A *global knowledge base* of profiles $\mathcal{C} = \bigcup_{a_i} \mathcal{C}_{a_i}$ is constructed, where $\mathcal{C}_{a_i}$ are knowledge bases containing only well-trained, stable profiles from anomaly detection systems previously deployed on a set of web applications $\bigcup_i a_i$.
2. A knowledge base $\mathcal{C}^I = \bigcup_{a_i} \mathcal{C}^I_{a_i}$ of undertrained profiles is then constructed as an *index* into $\mathcal{C}$, where $\mathcal{C}^I_{a_i}$ is

a knowledge base of undertrained profiles from the web application a_i.

3. Finally, a *mapping* $f : \left\{\mathcal{C}^I\right\} \times \mathcal{C}_{a_i} \mapsto \mathcal{C}$ is defined.

Third phase (online) For any new web application where insufficient training data is available for a component's parameter, the anomaly detector first extracts the undertrained profile c' from the local knowledge base $\mathcal{C}_{a_i}$. Then, the global knowledge base $\mathcal{C}$ is queried to find a similar, previously constructed profile $f\left(\mathcal{C}_I, c'\right) = c \in \mathcal{C}$. The well-trained profile c is then substituted for the undertrained profile c' in the detection process.

The following sections detail how $\mathcal{C}_I$ and $\mathcal{C}$ are constructed, and how f maps elements in $\mathcal{C}_I$ and $\mathcal{C}_{a_i}$ to elements in $\mathcal{C}$.

4.2.2.1 First Phase: Enhanced Training

A significant refinement of the individual models described in [Kruegel et al., 2005] is the criterion used to determine the length of the training phase. An obvious choice is to fix a constant training length, e.g., a thousand requests. Unfortunately, an appropriate training length is dependent upon the complexity of modeling a given set of features. Therefore, we have developed an automated method that leverages two stop criteria, *model stability* and *model confidence*, to determine when a model has observed enough training samples to accurately approximate the normal behavior of a parameter.

Model Stability As new training samples are observed early in the training phase, the state of a model typically exhibits frequent and significant change as its approximation of the normal behavior of a parameter is updated. Informally, in an information-theoretic sense, the average information gain of new training samples is high. As a model's state converges to a more precise approximation of normal behavior, however, its state exhibits infrequent and incremental changes. In other words, the information gain of new training samples approaches zero, and the model stabilizes. Note that, we refer to the information gain as an informal and intuitive concept to explain the rationale behind the development of a sound model stability criterion. By no means we claim that our procedure relies on the actual information gain to calculate when a model converges to stability.

Each model self-assesses its stability during the training phase by maintaining a history of snapshots of its internal state. At regular intervals, a model checks if the sequence of deltas between each successive historical state is monotonically decreasing and whether the degree of change drops below a certain threshold. If both conditions are satisfied, then the model considers itself stable. Let $\kappa^{(u)}_{\text{stable}}$ denote the number of training samples required for a model to achieve stability. A profile is considered stable when all of its constituent models are stable.

Definition 4.2.1 (Profile stability) Let κ_{stable} be the number of training samples required for a single model to achieve stability. The *aggregated stability* measure is defined as:

$$\kappa_{\text{stable}} = \max_{u \in U} \kappa^{(u)}_{\text{stable}}. \tag{4.1}$$

The notion of model stability is also leveraged in the third phase, as detailed in Section 4.2.2.2.

Note 4.2.1 Instead of describing the internal stop criterion specific to each model, if any, we developed a model-agnostic minimization algorithm detailed in Section 4.2.2.2 (and evaluated in Section 4.2.3.2) that allows one to trade off detection accuracy against the number of training samples available.

Model Confidence A related notion to model stability is that of *model confidence*. Recall that the goal of a model is to learn an abstraction of the normal behavior of a feature from the training data. There exists a well-known tension in the learning process between accurately fitting the model to the data and generalizing to data that has not been observed in the training set. Therefore, in addition to detecting whether a model has observed enough training samples to accurately model a feature, each model incorporates a self-confidence measure $z^{(u)}$ that represents whether the model has generalized so much as to lose all predictive power.

For example, the confidence of a token model should be directly related to the probability that a token set is present.

Definition 4.2.2 (Token model confidence) The *token model confidence* is defined as:

$$z^{(\text{tok})} = \tau = \frac{1}{2} + \frac{n_c - n_d}{4n\,(n-1)}, \tag{4.2}$$

where n_c and n_d are the number of concordant and discordant pairs, respectively, between the unique number of observed samples and the total number of samples observed at each training step, and n is the total number of training samples.

Note that, $z_{m^{(\text{tok})}}$ is an adaptation of the τ coefficient [Kendall, 1938].

For the integer and length models, the generality of a particular model can be related to the statistical dispersion of the training set.

Definition 4.2.3 (Integer model confidence) Given the observed standard deviation σ, minimum observed value u, and maximum observed value v, the *confidence of an integer or length model* is defined as:

$$z^{(\{\text{int,len}\})} = \begin{cases} 1 - \frac{\sigma}{v-u} & \text{if } v - u > 0 \\ 1 & \text{otherwise} \end{cases}. \tag{4.3}$$

The confidence of a character distribution model is determined by the variance of observed ICDs. Therefore, the confidence of this model is given by a similar construction as the previous one, except that instead of operating over observed values, the measure operates over observed variances.

Definition 4.2.4 (Char. distr. model confidence) Given the standard deviation of observed variances σ, the minimum observed variance u, and the maximum observed variance v, the *confidence of a character distribution model* is:

$$z^{(\text{char})} = \begin{cases} 1 - \frac{\sigma}{v-u} & \text{if } v - u > 0 \\ 1 & \text{otherwise} \end{cases}. \tag{4.4}$$

Finally, the HMM induced by the structure model can be directly analyzed for generality. We perform a rough estimate of this by computing the mean probability for any symbol from the learned alphabet to be emitted at any state.

Definition 4.2.5 (Struct. model confidence) Given a structural model, i.e. an HMM, specified by the tuple $\langle \mathbb{S}, \mathbb{O}, M_{\mathbb{S}\times\mathbb{S}}, P(\mathbb{S}, \mathbb{O}), P(\mathbb{S}) \rangle$, where $\mathbb{S}$ is the set of states, $\mathbb{O}$ is the set of emissions, $M_{\mathbb{S}\times\mathbb{S}}$ is the state transition probability matrix, $P(\mathbb{S}, \mathbb{O})$ is the emission probability distribution over the set of states, and $P(\mathbb{S})$ is the initial probability assignment, *the structural model confidence*:

$$z^{(\text{struct})} = 1 - \frac{1}{|\mathbb{S}|\,|\mathbb{O}|} \sum_{i=1}^{|\mathbb{S}|} \sum_{j=1}^{|\mathbb{O}|} P(s_i, o_j). \tag{4.5}$$

At the end of this phase, for each web application a_i, the profiles are stored in the corresponding knowledge base $\mathcal{C}_{a_i}$ and are ready to be processed by the next phase.

4.2.2.2 Second Phase: Building a global knowledge base

This phase is divided into the three sub-steps described in the following.

Well-trained profiles The construction of $\mathcal{C}$ begins by merging a collection of knowledge bases $\{\mathcal{C}_{a_1}, \mathcal{C}_{a_2}, \ldots, \mathcal{C}_{a_n}\}$ that have previously been built by a web application anomaly detector over a set of web applications $\bigcup_i a_i$. The profiles in $\mathcal{C}$ are then clustered in order to group profiles that are semantically similar to each other. Profile clustering is performed in order to time-optimize query execution when indexing into $\mathcal{C}$, as well as to validate the notion of parameter types. In this work, an agglomerative hierarchical clustering algorithm using group average linkage was applied, although the clustering stage is, in principle, agnostic as to the specific algorithm. For an in-depth discussion of clustering algorithms and techniques, we refer the reader to [Xu and Wunsch, 2005].

Central to any clustering algorithm is the distance function, which defines how distances between the objects to be clustered are calculated. A suitable distance function must reflect the semantics of the objects under consideration, and should satisfy two conditions:

- the overall similarity (i.e., inverse of the distance) between elements within the same cluster is maximized, and
- the similarity between elements within different clusters is minimized.

We define the distance between two profiles to be the sum of the distances between the models comprising each profile. More formally.

Definition 4.2.6 (Profile distance) The *distance between two profiles* c_i and c_j is defined as:

$$d(c_i, c_j) = \frac{1}{|c_i \bigcap c_j|} \sum_{m_i^{(u)}, m_j^{(u)} \in c_i \bigcap c_j} \delta_u \left(m_i^{(u)}, m_j^{(u)} \right), \tag{4.6}$$

where $\delta_u : \mathcal{M}_u \times \mathcal{M}_u \mapsto [0,1]$ is the distance function defined between models of type $u \in U = \{\text{tok}, \text{int}, \text{len}, \text{char}, \text{struct}\}$.

The token model $m^{(\text{tok})}$ is represented as a set of unique tokens observed during the training phase. Therefore, two token models $m_i^{(\text{tok})}$ and $m_j^{(\text{tok})}$ are considered similar if they contain similar sets of tokens. More formally.

Definition 4.2.7 (Token models distance) The *distance function for token models* is defined as the Jaccard distance [Cohen et al., 2003]:

$$\delta_{\text{tok}} \left(m_i^{(\text{tok})}, m_j^{(\text{tok})} \right) = 1 - \frac{\left| m_i^{(\text{tok})} \bigcap m_j^{(\text{tok})} \right|}{\left| m_i^{(\text{tok})} \bigcup m_j^{(\text{tok})} \right|}. \tag{4.7}$$

The integer model $m^{(\text{int})}$ is parametrized by the sample mean μ and variance σ of observed integers. Two integer models $m_i^{(\text{int})}$ and $m_j^{(\text{int})}$ are similar if these parameters are also similar.

Definition 4.2.8 (Integer models distance) The *distance function for integer models* is defined as:

$$\delta_{\text{int}} \left(m_i^{(\text{int})}, m_j^{(\text{int})} \right) = \frac{\left\| \frac{\sigma_i^2}{\mu_i^2} - \frac{\sigma_j^2}{\mu_j^2} \right\|}{\frac{\sigma_i^2}{\mu_i^2} + \frac{\sigma_j^2}{\mu_j^2}}. \tag{4.8}$$

Note 4.2.2 (Length models distance) As the string length model is internally identical to the integer model, its distance function δ_{len} is defined similarly.

Recall that the character distribution model $m^{(\text{char})}$ learns the frequencies of individual characters comprising strings observed during the training phase. These frequencies are then ranked and coalesced into a n bins to create an ICD. Two character distribution models $m_i^{(\text{char})}$ and $m_j^{(\text{char})}$ are considered similar if each model's ICDs are similar. More formally.

Definition 4.2.9 (Char. distr. models distance) The *character distribution models distance* is defined as:

$$\delta_{\text{char}}\left(m_i^{(\text{char})}, m_j^{(\text{char})}\right) = \sum_{l=0}^{n} \frac{\|b_i\,(l) - b_j\,(l)\|}{\max_{k=i,j} b_k\,(l)}, \tag{4.9}$$

where $b_i\,(k)$ is the value of bin k for $m_i^{(\text{char})}$.

The structural model $m^{(\text{struct})}$ builds an HMM by observing a sequence of character strings. The resulting HMM encodes a probabilistic grammar that can produce a superset of the strings observed during the training phase. The HMM is specified by the tuple $\langle \mathbb{S}, \mathbb{O}, M_{\mathbb{S}\times\mathbb{S}}, P\,(\mathbb{S}, \mathbb{O})\,, P\,(\mathbb{S})\rangle$. Several distance metrics have been proposed to evaluate the similarity between HMMs [Stolcke and Omohundro, 1993a; Lyngsøet al., 1999; Stolcke and Omohundro, 1994a; Juang and Rabiner, 1985]. Their time complexity, however, is non-negligible. Therefore, we adopt a less precise, but considerably more efficient, distance metric between two structural models $m_i^{(\text{struct})}$ and $m_j^{(\text{struct})}$.

Definition 4.2.10 (Struct. models distance) The *structural models distance* is defined as the Jaccard distance between their respective emission sets:

$$\delta_{\text{struct}}\left(m_i^{(\text{struct})}, m_j^{(\text{struct})}\right) = 1 - \frac{|\mathbb{O}_i \bigcap \mathbb{O}_j|}{|\mathbb{O}_i \bigcup \mathbb{O}_j|}. \tag{4.10}$$

Undertrained profiles Once the global knowledge base $\mathcal{C}$ has been built, the next step is to construct a knowledge base of undertrained models $\mathcal{C}^I$. This knowledge base is composed of profiles that have been deliberately undertrained on $\kappa \ll \kappa_{\text{stable}}$ for each of the web applications a_i. Because each member of $\mathcal{C}^I$ uniquely corresponds to a member in $\mathcal{C}$ and undertrained profiles are built for each well-trained profile in $\mathcal{C}$, a bijective mapping $f' : \mathcal{C}^I \mapsto \mathcal{C}$ exists between

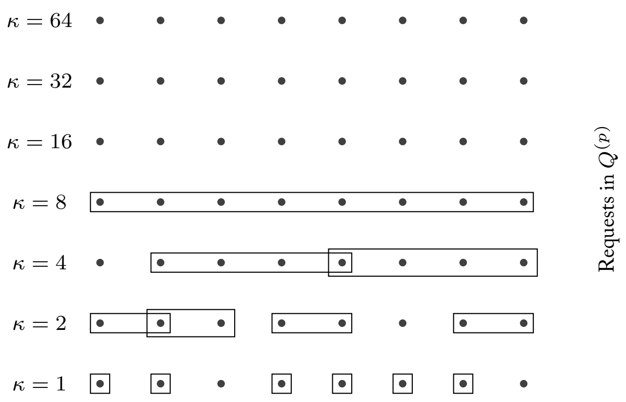


Figure 4.5: Partitioning of the training set Q for various κ.

$\mathcal{C}^I$ and $\mathcal{C}$. Therefore, when a web application parameter is identified as likely to be undertrained, the corresponding undertrained profile c' can be compared to a similar undertrained profile in $\mathcal{C}^I$, which is then used to select a corresponding stable profile from $\mathcal{C}$. This operation requires the knowledge base $\mathcal{C}^I$ to be indexed. The construction of the indices relies on the notion of model stability, described in Section 4.2.2.1.

The undertrained profiles that comprise $\mathcal{C}^I$ are constructed using the following procedure. Let $Q^{(p)} = \{q_1^{(p)}, q_2^{(p)}, \ldots\}$ denote a sequence of client requests containing parameter p for a given web application. Over this sequence of requests, profiles are deliberately undertrained on randomly sampled κ-sequences. Each of the resulting profiles is then added to the set $\mathcal{C}^I$. Figure 4.5 depicts this procedure for various κ. Note that the random sub-sampling is performed with the goal of inducing undertraining to show that clustering is feasible and leads to the desired grouping even – and especially – in the presence of undertraining.

Recall that κ_{stable} is the number of training samples required for a profile to achieve stability. Then, a profile is considered undertrained when the number of training samples it has observed, κ, is significantly less than the number required to achieve stability. Therefore, an undertrained knowledge base $\mathcal{C}^I$ is a set of profiles that have observed κ samples such that $\kappa \ll \kappa_{\text{stable}}$.

The selection of an appropriate value for κ is central to both the efficiency and the accuracy of this process. Clearly, it is desirable to minimize κ in order to be able to index into $\mathcal{C}$ as quickly as possible once a parameter has been identified to be subject to undertraining at runtime. On the other hand, setting κ too low is problematic, as Figure 4.6 indicates. For low values of κ, profiles are distributed with relatively high uniformity within $\mathcal{C}^I$, such that clusters in $\mathcal{C}^I$ are significantly different than clusters of well-trained profiles in $\mathcal{C}$. Therefore, slight differences in the state of the individual models can cause profiles that are close in $\mathcal{C}^I$ to map to radically different profiles in $\mathcal{C}$. As $\kappa \to \kappa_{\text{stable}}$, however, profiles tend to form meaningful clusters, and tend to approximate those found in $\mathcal{C}$. Therefore, as κ increases, profiles that are close in $\mathcal{C}^I$ become close in $\mathcal{C}$ under f – in other words, f becomes *robust* with respect to model semantics.

Note 4.2.3 (Robustness) Our use of the term "robustness" is related, but not necessarily equivalent, to the definition of robustness in statistics (i.e., the property of a model to perform well even in the presence of small changes in the underlying assumptions).

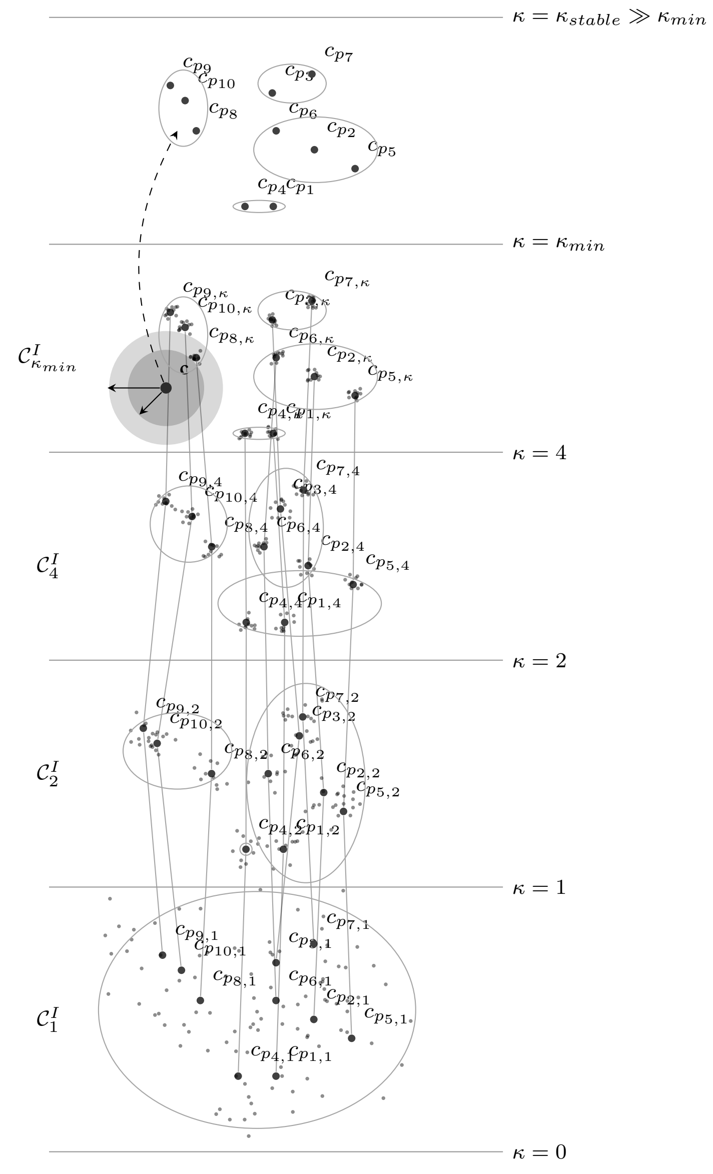
Mapping undertrained profiles to well-trained profiles A principled criterion is required for balancing quick indexing against a robust profile mapping. Accordingly, we first construct a candidate knowledge base $\mathcal{C}^I_\kappa$ for a given κ, as in the case of the global knowledge base construction. Then, we define a *robustness metric* as follows. Let $H^I = \bigcup_i h^I_i$ be the set of clusters in $\mathcal{C}^I$, and $H = \bigcup_i h_i$ be the set of clusters in $\mathcal{C}$. Let $g : H^I \mapsto \mathbb{N}$ be a mapping from an undertrained cluster to the maximum number of elements in that cluster that map to the same cluster in $\mathcal{C}$. The robustness metric $\rho : \{\mathcal{C}^I\} \mapsto [0, 1]$ is then defined as

$$\rho\left(\mathcal{C}^I\right) = \frac{1}{|\mathcal{C}^I|} \sum_i g\left(h^I_i\right). \tag{4.11}$$

With this metric, an appropriate value for κ can now be chosen as

$$\kappa_{\text{min}} = \min_\kappa \left(\rho\left(\mathcal{C}^I_\kappa\right) \geq \rho_{\text{min}}\right), \tag{4.12}$$

where ρ_{min} is a minimum robustness threshold.



$\kappa = \kappa_{stable} \gg \kappa_{min}$
c_{p_7}
c_{p_9}
$c_{p_{10}}$
c_{p_3}
c_{p_8}
c_{p_6}
c_{p_2}
c_{p_5}
c_{p_4} c_{p_1}
$\kappa = \kappa_{min}$
$\mathcal{C}^I_{\kappa_{min}}$
$\mathbf{c}$
$c_{p_{9,\kappa}}$ $c_{p_{10,\kappa}}$ $c_{p_{8,\kappa}}$ $c_{p_{7,\kappa}}$ $c_{p_{3,\kappa}}$ $c_{p_{6,\kappa}}$ $c_{p_{2,\kappa}}$ $c_{p_{5,\kappa}}$ $c_{p_{4,\kappa}}$ $c_{p_{1,\kappa}}$
$\kappa = 4$
$\mathcal{C}^I_4$
$c_{p_{9,4}}$ $c_{p_{10,4}}$ $c_{p_{8,4}}$ $c_{p_{7,4}}$ $c_{p_{3,4}}$ $c_{p_{6,4}}$ $c_{p_{2,4}}$ $c_{p_{5,4}}$ $c_{p_{4,4}}$ $c_{p_{1,4}}$
$\kappa = 2$
$\mathcal{C}^I_2$
$c_{p_{9,2}}$ $c_{p_{10,2}}$ $c_{p_{8,2}}$ $c_{p_{7,2}}$ $c_{p_{3,2}}$ $c_{p_{6,2}}$ $c_{p_{2,2}}$ $c_{p_{5,2}}$ $c_{p_{4,2}}$ $c_{p_{1,2}}$
$\kappa = 1$
$\mathcal{C}^I_1$
$c_{p_{9,1}}$ $c_{p_{10,1}}$ $c_{p_{8,1}}$ $c_{p_{7,1}}$ $c_{p_{3,1}}$ $c_{p_{6,1}}$ $c_{p_{2,1}}$ $c_{p_{5,1}}$ $c_{p_{4,1}}$ $c_{p_{1,1}}$
$\kappa = 0$

Figure 4.6: Procedure for building global knowledge base indices.

With the construction of a global knowledge base $\mathcal{C}$ and an undertrained knowledge base $\mathcal{C}^I$, online querying of $\mathcal{C}$ can then be performed.

4.2.2.3 Third Phase: Querying and Substitution

Given an undertrained profile c' from an anomaly detector deployed over a web application a_i, the mapping $f : \{\mathcal{C}^I\} \times \mathcal{C}_{a_i} \mapsto \mathcal{C}$ is defined as follows. A nearest-neighbor match is performed between c' and the previously constructed clusters H^I from $\mathcal{C}^I$ to discover the most similar cluster of undertrained profiles. This is done to avoid a full scan of the entire knowledge base, which would be prohibitively expensive due to the cardinality of $\mathcal{C}^I$.

Then, using the same distance metric defined in Equation (4.6), a nearest-neighbor match is performed between c' and the members of H^I to discover the most similar undertrained profile c^I. Finally, the global, well-trained profile $f'\left(c^I\right) = c$ is substituted for c' for the web application a_i.

To make explicit how global profiles can be used to address a scarcity of training data, consider the example introduced in Section 4.1.1. Since the resource path `/account/password` has received only 100 requests (see Table 4.1: `/account/password` has received 0.02% of 500,000 total requests), the profiles for each of its parameters {`id`, `oldpw`, `newpw`} are undertrained, as determined by each profile's aggregate stability measure. In the absence of a global knowledge base, the anomaly detector would provide no protection against attacks manifesting themselves in the values passed to any of these parameters.

If, however, a global knowledge base and index are available, the situation is considerably improved. Given $\mathcal{C}$ and $\mathcal{C}^I$, the anomaly detector can simply apply f for each of the undertrained parameters to find a well-trained profile from the global knowledge base that accurately models a parameter with similar characteristics *from another web application*. Then, these profiles can be substituted for the undertrained profiles for each of {`id`, `oldpw`, `newpw`}. As will be demonstrated in the following section, the substitution of global profiles provides an acceptable detection accuracy for what would otherwise be an unprotected component (i.e., without a global profile 0% of the attacks against that component would be detected).

Note 4.2.4 (HTTP parameters "semantics") It is important to underline that the goal of this approach is not that of understanding the types of the fields nor their semantic. However, if the system administrator and the security officer had enough time and skills, the use of manual tools such as live debuggers or symbolic execution engines may help to infer the types of the fields and, once available, this information may be exploited as a lookup criterion. Unfortunately, this would be a human-intensive task, thus far from being automatic.

Instead, our goal is to find groups of similar models, regardless of what this may imply in terms of types. According to our experiments, similar models tend to be associated to similar types. Thus, in principle, our method cannot be used to infer the parameters' types but can certainly give insights about parameters with similar features.

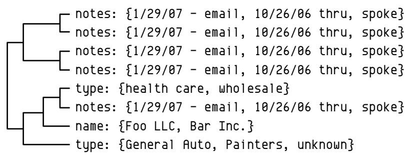
4.2.3 Experimental Results

The goal of this evaluation is three-fold. First, we investigate the effects of profile clustering, and support the hypothesis of parameter types by examining global knowledge base clusters. Then, we discuss how the quality of the mapping between undertrained profiles and well-trained profiles improves as the training slice length κ is increased. Finally, we present results regarding the accuracy of a web application anomaly detection system incorporating the application of a global knowledge base to address training data scarcity.

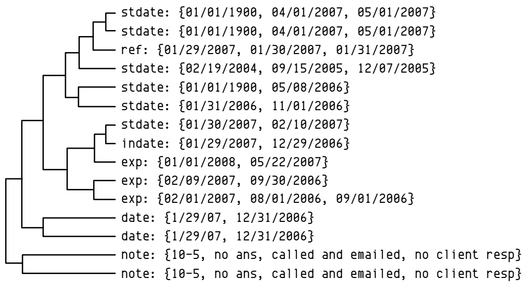
4.2.3.1 Profile clustering quality

To evaluate the accuracy of the clustering phase, we first built a global knowledge base $\mathcal{C}$ from a collection of well-trained profiles. The profiles were trained on a subset of the data mentioned in Section 4.1.3; this subset was composed of 603 web applications, 27,990 unique resource paths, 9,023 unique parameters, and 3,444,092 HTTP requests. The clustering algorithm described in Section 4.2.2.2 was then applied to group profiles according to their parameter type. Sample results from this clustering are shown in Figure 4.7b. Each leaf node corresponds to a profile and displays the parameter name and a few representative sample values corresponding to the parameter.

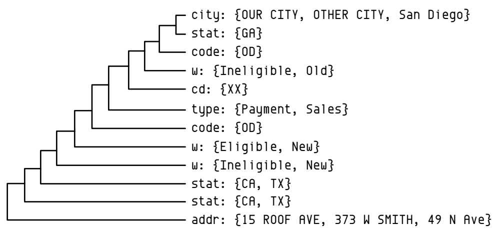
As the partial dendrogram indicates, the resulting clusters in $\mathcal{C}$ are accurately clustered by parameter type. For instance, date parameters are grouped into a single hierarchy, while unstructured text strings are grouped into a separate hierarchy.


notes: {1/29/07 - email, 10/26/06 thru, spoke}
notes: {1/29/07 - email, 10/26/06 thru, spoke}
notes: {1/29/07 - email, 10/26/06 thru, spoke}
notes: {1/29/07 - email, 10/26/06 thru, spoke}
type: {health care, wholesale}
notes: {1/29/07 - email, 10/26/06 thru, spoke}
name: {Foo LLC, Bar Inc.}
type: {General Auto, Painters, unknown}

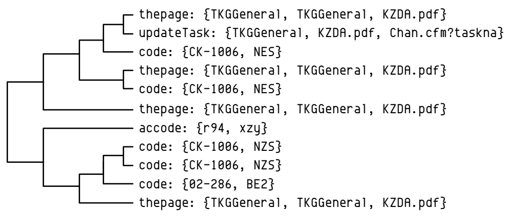
(a) $\kappa = 64$


stdate: {01/01/1900, 04/01/2007, 05/01/2007}
stdate: {01/01/1900, 04/01/2007, 05/01/2007}
ref: {01/29/2007, 01/30/2007, 01/31/2007}
stdate: {02/19/2004, 09/15/2005, 12/07/2005}
stdate: {01/01/1900, 05/08/2006}
stdate: {01/31/2006, 11/01/2006}
stdate: {01/30/2007, 02/10/2007}
indate: {01/29/2007, 12/29/2006}
exp: {01/01/2008, 05/22/2007}
exp: {02/09/2007, 09/30/2006}
exp: {02/01/2007, 08/01/2006, 09/01/2006}
date: {1/29/07, 12/31/2006}
date: {1/29/07, 12/31/2006}
note: {10-5, no ans, called and emailed, no client resp}
note: {10-5, no ans, called and emailed, no client resp}

(b) $\kappa_{stable} \simeq 10^3$



(c) $\kappa = 8$


thepage: {TKGGeneral, TKGGeneral, KZDA.pdf}
updateTask: {TKGGeneral, KZDA.pdf, Chan.cfm?taskna}
code: {CK-1006, NES}
thepage: {TKGGeneral, TKGGeneral, KZDA.pdf}
code: {CK-1006, NES}
thepage: {TKGGeneral, TKGGeneral, KZDA.pdf}
accode: {r94, xzy}
code: {CK-1006, NZS}
code: {CK-1006, NZS}
code: {02-286, BE2}
thepage: {TKGGeneral, TKGGeneral, KZDA.pdf}

(d) $\kappa = 32$

FIGURE 4.7: Clustering of $\mathcal{C}$, (a-b), and $\mathcal{C}^I$, (c-d). Each leaf (a profile) is labeled with the parameter name and samples values observed during training. As κ increases, profiles are clustered more accurately.

The following experiment investigates how κ affects the quality of the final clustering.

4.2.3.2 Profile mapping robustness

Recall that in order to balance the robustness of the mapping f between undertrained profiles and global profiles against the speed with which undertraining can be addressed, it is necessary to select an appropriate value for κ. To this end, we generated undertrained knowledge bases for increasing values of $\kappa = 1, 2, 4, 8, 16, 32, 64$ from the same dataset used to generate $\mathcal{C}$, following the procedure outlined in Section 4.2.2.2. Partial dendrograms for various κ are presented in Figure 4.7c, 4.7d, 4.7a.

At low values of κ (e.g., Figure 4.7c), the clustering process exhibits non-negligible systemic errors. For instance, the parameter `stat` clearly should be clustered as a token set of states, but instead is grouped with unstructured strings such as cities and addresses. A more accurate clustering would have dissociated the token and string profiles into well-separated sub-hierarchies.

As shown in Figure 4.7d, larger values of κ lead to more meaningful groupings. Some inaccuracies are still noticeable, but the clustering process of the sub-hierarchy is significantly better that the one obtained at $\kappa = 8$. A further improvement in the clusters is shown in Figure 4.7a. At $\kappa = 64$, the separation between dates and unstructured strings is sharper; except for one outlier, the two types are recognized as similar and grouped together in the early stages of the clustering process.

Figure 4.8 plots the profile mapping robustness $\rho(\cdot)$ against κ for different cuts of the dendrogram, indicated by D_{max}. D_{max} is a threshold representing the maximum distance between two clusters. Basically, for low D_{max}, the "cut" will generate many clusters with a few elements; on the other hand, for high values of D_{max} the algorithm will tend to form less clusters, each having a larger number of elements. Note that this parameter is known to have two different possible interpretations: it could indicate either a threshold on the real distance between clusters, or a "cut level" in the dendrogram constructed by the clustering algorithm. Although they are both valid, we prefer to utilize the former.

Figure 4.8 shows two important properties of our technique. First, it demonstrates that the robustness is fairly insensitive to D_{max}. Sec-

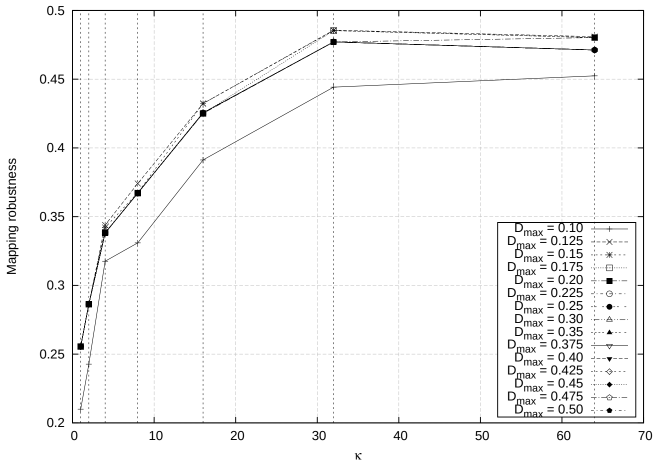


FIGURE 4.8: Plot of profile mapping robustness for varying κ.

ond, the robustness of the mapping increases with κ until saturation at $32 \leq \kappa \leq 64$. This not only confirms the soundness of the mapping function, but it also provides insights on the appropriate choice of κ_{min} to minimize the delay to global profile lookup while maximizing the robustness of the mapping.

4.2.3.3 Detection accuracy

Having studied the effects of profile clustering and varying values for κ upon the robustness of the profile mapping f, a separate experiment was conducted in order to evaluate the detection accuracy of a web application anomaly detector incorporating $\mathcal{C}$, the global knowledge base constructed in the previous experiments. In particular, the goal of this experiment is to demonstrate that an anomaly detector equipped with a global knowledge base exhibits an improved detection accuracy in the presence of training data scarcity.

The data used in this experiment was a subset of the full dataset described above, and was completely disjoint from the one used to construct the global knowledge base and its indices. It consisted of 220 unique real-world web applications, 8,402 unique resource paths, 7,648 distinct parameters, and 55,290,532 HTTP requests.

The intended threat model is that of an attacker attempting to compromise the confidentiality or integrity of data exposed by a web application by injecting malicious code in request parameters.

Note 4.2.5 (Threat model) Although the anomaly detector used in this study is capable of detecting more complex session-level anomalies, we restrict the threat model to request parameter manipulation because we do not address session profile clustering.

To establish a worst-case bound on the detection accuracy of the system, profiles for each observed request parameter were deliberately undertrained to artificially induce a scarcity of training data for *all* parameters. That is, for each value of $\kappa = 1, 2, 4, 8, 16, 32, 64$, the anomaly detector prematurely terminated profile training after κ samples, and then used the undertrained profiles to query $\mathcal{C}$. The resulting global profiles were then substituted for the undertrained profiles and evaluated against the rest of the dataset. The sensitivity of the system was varied over the interval $[0, 1]$, and the resulting ROC curves for each κ are plotted in Figure 4.9.

As one can clearly see, low values of κ result in the selection of global profiles that do not accurately model the behavior of the undertrained parameters. As κ increases, however, the quality of the global profiles returned by the querying process increases as well. In particular, this increase in quality closely follows the mapping robustness plot presented in Figure 4.8. As predicted, setting $\kappa = 32, 64$ leads to fairly accurate global profile selection, with the resulting ROC curves approaching that of fully-trained profiles. This means that even if the component of a web application has received only a few requests (i.e., 64), by leveraging a global knowledge base it is possible to achieve effective attack detection. As a consequence, our approach can improve the effectiveness of real-world web application firewalls and web application anomaly detection systems.

Clearly, the detection accuracy will improve as more training samples (e.g., 128, 256) become available. However, the goal of this experiment was to evaluate such an improvement with a very limited training set, rather than showing the detection maximum accuracy achievable. From these results, we conclude that for appropriate values of κ, the use of a global knowledge base can provide reasonably accurate detection performance even in the presence of training data scarcity.

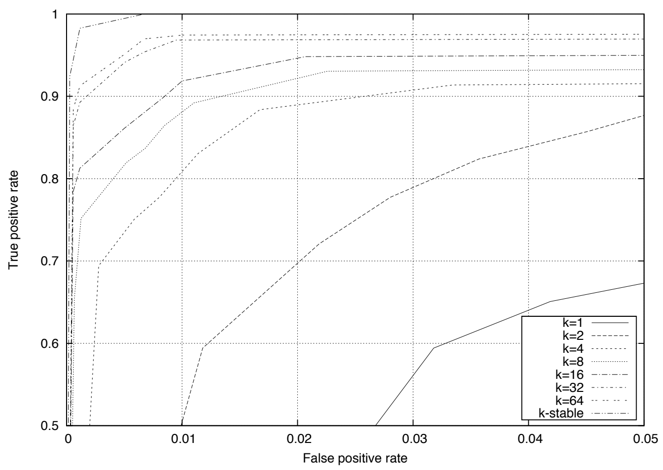


FIGURE 4.9: Global profile ROC curves for varying κ. In the presence of severe undertraining ($\kappa \ll \kappa_{\text{stable}}$), the system is not able to recognize most attacks and also reports several false positives. However, as κ increases, detection accuracy improves, and approaches that of the well-trained case ($\kappa = \kappa_{\text{stable}}$).

One concern regarding the substitution of global profiles for local request parameters is that a global profile that was trained on another web application may not detect valid attacks against the undertrained parameter. Without this technique, however, recall that a learning-based web application anomaly detector would otherwise have no effective model whatsoever, and therefore the undertrained parameter would be unprotected by the detection system (i.e., zero true positive rate). Furthermore, the ROC curves demonstrate that while global profiles are in general not as precise as locally-trained models, they do provide a significant level of detection accuracy.

Note 4.2.6 If global profiles were found to be as accurate as local profiles, this would constitute an argument against anomaly detection itself.

More precisely, with $\kappa = 1$, undertraining condition and system off, only 67.5% of the attacks are detected, overall, with around 5% of false positives. On the other hand, with $\kappa = 64$ (undertraining and system on), more than 91% of the attacks are detected with less than 0.2% of false positives (*vs.*, 0.1% of false positives in the case of no undertraining and system off). Therefore, we conclude that, assuming no mistrust among the parties that share the global knowledge base, our approach is a useful technique to apply in the presence of undertrained models and, in general, in the case of training data scarcity. Note that the last assumption is often satisfied because one physical deployment (e.g., one server protected by our system) typically hosts several web applications under the control of the same system administrator or institution.

Therefore, we conclude that our approach is a useful technique to apply in the presence of undertrained models and, in general, in case of training data scarcity.

Note 4.2.7 (Configuration parameters) Our methods provide explicit guidance, based on the training data of the particular deployment, to choose the value of the only parameter required to trade off accuracy *vs.* length of the training phase. In addition, the "goodness" of the approach with respect to different choices of such a parameter is evaluated on real-world data in Section 4.

In particular, the ROC curve shows how the sampling size (κ) affects the detection accuracy, thus offer some guidance to the user. In addition, Figure 8 shows that the system stabilizes for $\kappa > 32$, thus some more guidance about "what is a good value" is given.

4.3 Addressing Changes in Web Applications

In this section, we describe our proposal [Maggi et al., 2009c] to cope with FP due to changes in the modeled web applications. Recall that detection is performed under the assumption that attacks cause significant changes (i.e., anomalies) in the application behavior. Thus, any activity that does not fit the expected, learned models is flagged as malicious. This is true regardless of the type of system activity, thus it holds for other types of IDSs than web-based ones.

In particular, one issue that has not been well-studied is the difficulty of adapting to changes in the behavior of the protected applications. This is an important problem because today's web applications are user-centric. That is, the demand for new services causes continuous updates to an application's logic and its interfaces.

Our analysis, described in Section 4.3.1, reveals that significant changes in the behavior of web applications are frequent. We refer to this phenomenon as *web application concept drift*. In the context of anomaly-based detection, this means that legitimate behavior might be misclassified as an attack after an update of the application, causing the generation of false positives. Normally, whenever a new version of an application is deployed in a production environment, a coordinated effort involving application maintainers, deployment administrators, and security experts is required. That is, developers have to inform administrators about the changes that are rolled out, and the administrators have to update or re-train the anomaly models accordingly. Otherwise, the amount of FPs will increase significantly. In [Maggi et al., 2009c] we describe a solution that makes these tedious tasks unnecessary. Our technique examines the responses (HTML pages) sent by a web application. More precisely, we check the forms and links in these pages to determine when new elements are added or old ones removed. This information is leveraged to recognizes when anomalous inputs (i.e., HTTP requests) are due to previous, legitimate updates —changes— in a web application. In such cases, FPs are suppressed by automatically and selectively re-training models. Moreover, when possible, model parameters can be automatically updated without requiring any re-training.

Note 4.3.1 (Re-training) Often, a complete re-training of all the models is expensive in terms of time; typically, it requires $O(P)$ where P represents the number of HTTP messages required to train a model.

More importantly, such re-training is not always feasible since new, attack-free training data is unlikely to be available immediately after the application has changed. In fact, to collect a sufficient amount of data the new version of the application must be executed and real, legitimate clients have to interact with it in a controlled environment. Clearly, this task requires time and efforts. More importantly, those parts that have changed in the application must be known in advance.

Our technique focuses on the fundamental problem of *detecting* those parts of the application that have changed and that will cause FPs if no re-training is performed. Therefore, the technique is agnostic with respect to the specific training procedure, which is IDS-specific and can be different from the one we propose.

The core of this contribution is the exploiting of HTTP responses, which we show to contain important insights that can be effectively leveraged to update previously learned models to take changes into account. The results of applying our technique on real-world data demonstrate that learning-based anomaly detectors can automatically *adapt to changes*, and by doing this, are able to reduce their FPR without decreasing their DR significantly. Note that, in general, relying on HTTP responses may lead to the issue of poisoning of the models used to characterize them: this limitation is discussed in details in Section 4.3.2.3 in comparison with the advantages provided by our technique.

In Section 4.3.1 the problem of concept drift in the context of web applications is detailed. In addition, we provide evidence that it occurs in practice, motivating why it is a significant problem for deploying learning-based anomaly detectors in the real world. The core of our contribution is described in Section 4.3.2 where we detail the technique based on HTTP response models that can be used to distinguish between legitimate changes in web applications and web-based attacks. A version of webanomaly incorporating these techniques has been evaluated over an extensive real-world data set, demonstrating its ability to deal with web application concept drift and reliably detect attacks with a low false positive rate. The results of this evaluation are discussed in Section 4.3.3.

4.3.1 Web Application Concept drift

In this section the notion of web application concept drift is defined. We rely upon the generalized model of learning-based anomaly detectors of web attacks described in Section 4.1.1. Secondly, evidence that concept drift is a problem that exists in the real world is provided to motivate why it should be addressed.

4.3.1.1 Changes in Web Applications' Behavior

In machine learning, changes in the modeled behavior are known as *concept drift* [Schlimmer and Granger, 1986]. Intuitively, the *concept* is the modeled phenomenon; in the context of anomaly detection it may be the structure of requests to a web server, the recurring patterns in the payload of network packets, etc. Thus, variations in the main features of the phenomena under consideration result in changes, or *drifts*, in the *concept*. In some sense, the *concept* corresponds to the normal web application behavior (see also Definition 2.1.4).

Although the generalization and abstraction capabilities of modern learning-based anomaly detectors are resilient to noise (i.e., small, legitimate variations in the modeled behavior), concept drift is difficult to detect and to cope with [Kolter and Maloof, 2007]. The reason is that the parameters of the models may stabilize to different values. For example, the string length model described in Section 3.3.1.1 —and also used in webanomaly— learns the sample mean and variance of the string lengths that are observed during training. In webanomaly, during detection, the Chebyshev inequality is used to detect strings with lengths that significantly deviate from the mean, taking into account the observed variance. As shown in Section 3.3.2, the variance allows to account for small differences in the lengths of strings that will be considered normal.

On the other hand, the mean and variance of the string lengths can completely change because of legitimate and permanent modifications in the web application. In this case, the normal mean and variance will stabilize, or drift, to different values. If appropriate retraining or manual updates are not performed, the model will classify benign, new strings as anomalous. These examples allow us to better define the web application concept drift.

Definition 4.3.1 (Web Application Concept Drift) The *web application concept drift* is a permanent modification of the *normal web ap-*

plication behavior.

Since the behavior of a web application is derived from the system activity $\mathbb{I}$ during normal operation, a permanent *change* in $\mathbb{I}$ causes the concept drift. Changes in web applications can manifest themselves in several ways. In the context of learning-based detection of web attacks, those changes can be categorized into three groups: *request* changes, *session* changes, and *response* changes.

Request changes Changes in requests occur when an application is upgraded to handle different HTTP requests. These changes can be further divided into two groups: *parameter value* changes and *request structure* changes. The former involve modifications of the actual value of the parameters, while the latter occur when parameters are *added* or *removed*. Parameter *renaming* is the result of removal plus addition.

Example 4.3.1 (Request change) A new version of a web forum introduces internationalization (I18N) and localization (L10N). Besides handling different languages, I18N and L10N allow several types of strings to be parsed as valid dates and times. For instance, valid strings for the `datetime` parameter are '`3 May 2009 3:00`', '`3/12/2009`', '`3/12/2009 3:00 PM GMT-08`', '`now`'. In the previous version, valid date-time strings had to conform to the regular expression '`[0-9]{1,2}/[0-9]{2}/[0-9]{4}`'. A model with good generalization properties would learn that the field `datetime` is composed of numbers and slashes, with no spaces. Thus, other strings such as '`now`' or '`3/12/2009 3:00 PM GMT-08`' would be flagged as anomalous. Also, in our example, `tz` and `lang` parameters have been added to take into account time zones and languages. To summarize, the new version introduces two classes of changes. Clearly, the parameter domain of `datetime` is modified. Secondly, new parameters are added.

Changes in HTTP requests directly affect the request models:

1. parameter value changes affect any models that rely on the parameters' *values* to extract features. For instance, consider two of the models used in the system described in [Kruegel et al., 2005]: $m^{(\text{char})}$ and $m^{(\text{struct})}$. The former models the strings' character distribution by storing the frequency of all

the symbols found in the strings during training, while the latter models the strings' structure as a stochastic grammar, using a HMM. In the aforementioned example, the I18N and L10N introduce new, legitimate values in the parameters; thus, the frequency of numbers in $m^{(\text{char})}$ changes and new symbols (e.g., '`-`', '`[a-zA-Z]`' have to be taken into account. It is straightforward to note that $m^{(\text{struct})}$ is affected in terms of new transitions introduced in the HMM by the new strings.

2. Request structure changes may affect any type of request model, regardless of the specific characteristics. For instance, if a model for a new parameter is missing, requests that contain that parameter might be flagged as anomalous.

Session changes Changes in sessions occur whenever resource path sequences are *reordered*, *inserted*, or *removed*. Adding or removing application modules introduces changes in the session models. Also, modifications in the application logic are reflected in the session models as reordering of the resources invoked.

Example 4.3.2 (Session change) A new version of a web-based community software grants read-only access to *anonymous* users (i.e., without authentication), allowing them to display contents previously available to subscribed users only. In the old version, legitimate sequences were ⟨`/site, /auth, /blog`⟩ or ⟨`/site, /auth, /files`⟩, where `/site` indicates the server-side resource that handles the public site, `/auth` is the authentication resource, and `/blog` and `/files` were formerly private resources. Initially, the probability of observing /`auth` before `/blog` or `/files` is close to one (since users need to authenticate before accessing private material). This is no longer true in the new version, however, where `/files|/blog|/auth` are all possible after `/site`.

Changes in sessions impact all models that rely on the sequence of resources that are invoked during the normal operation of an application. For instance, consider the model $m^{(\text{sess})}$ described in [Kruegel et al., 2005], which builds a probabilistic finite state automaton that captures sequences of *resource paths*. New arcs must be added to take into account the changes mentioned in the above example. These types of models are sensitive to strong changes in the session structure and should be updated accordingly when they occur.

Response changes Changes in responses occur whenever an application is upgraded to produce different responses. Interface redesigns and feature addition or removal are example causes of changes in the responses. Response changes are common and frequent, since page updates or redesigns often occur in modern websites.

Example 4.3.3 (Response change) A new version of a video sharing application introduces Web 2.0 features into the user interface, allowing for the modification of user interface elements without refreshing the entire page. In the old version, relatively few nodes of documents generated by the application contained client-side code. In the new version, however, many nodes of the document contain event handlers to trigger asynchronous requests to the application in response to user events. Thus, if a response model is not updated to reflect the new structure of such documents, a large of number of false positives will be generated due to *legitimate* changes in the characteristics of the web application responses.

4.3.1.2 Concept Drift in the Real World

To understand whether concept drift is a relevant issue for real-world websites, we performed three experiments. For the first experiment, we monitored 2,264 public websites, including the Alexa Top 500's and other sites collected by querying Google with popular terms extracted from the Alexa Top 500's. The goal was to identify and quantify the changes in the forms and input fields of popular websites at large. This provides an indication of the frequency with which real-world applications are updated or altered.

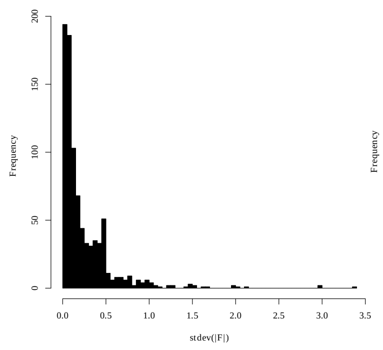
First Experiment: Frequency of Changes Once every hour, we visited one page for each of the 2,264 websites. In total, we collected 3,303,816 pages, comprising more than 1,390 snapshots for each website, between January 29 and April 13, 2009. One tenth of the pages were manually selected to have a significant number of forms, input fields, and hyperlinks with `GET` parameters. By doing this, we gathered a considerable amount of information regarding the HTTP messages generated by some applications. Examples of these pages are registration pages, data submission pages, or contact form pages. For the remaining websites, we simply used their home pages. Note that, the pages we selected may not be fully representative of the whole website. To overcome this limitation and to further confirm our intuition, we performed a more detailed experiment — described in Section 4.3.1.2 — on the source code of large, real-world web applications.

For each website w, each page sample crawled at time t is associated with a tuple $|F|_t^{(w)}, |I|_t^{(w)}$, the cardinality of the sets of forms and input fields, respectively. By doing this, we collected samples of the variables $|F|^w = |F|_{t_1}^w, \ldots, |F|_{t_n}^w$, $|I|^w = |I|_{t_1}^w, \ldots, |I|_{t_n}^w$, with $0 < n < 1,390$. Figure 4.10 shows the relative frequency of the variables

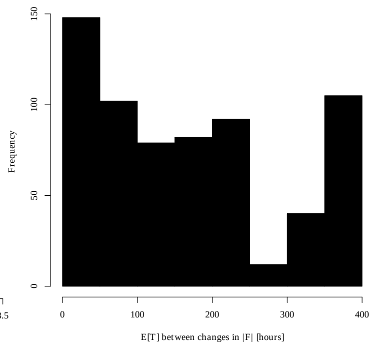
$$\begin{aligned} X_I &= \text{stdev}(|I|^{(w_1)}), \ldots, \text{stdev}(|I|^{(w_k)}) \\ X_F &= \text{stdev}(|F|^{(w_1)}), \ldots, \text{stdev}(|F|^{(w_k)}). \end{aligned}$$

This demonstrates that a significant amount of websites exhibit variability in the response models, in terms of elements modified in the pages, as well as request models, in terms of new forms and parameters. In addition, we estimated the expected time between changes of forms and inputs fields, $E[T_F]$ and $E[T_I]$, respectively. In terms of forms, 40.72% of the websites drifted during the observation period. More precisely, 922 out of 2,264 websites have a finite $E[T_F]$. Similarly, 29.15% of the websites exhibited drifts in the number of input fields, i.e., $E[T_I] < +\infty$ for 660 websites. Figure 4.10 shows the relative frequency of (b) $E[T_F]$, and (d) $E[T_I]$. $E[T_F]$. This confirms that a non-negligible portion of the websites exhibit significantly frequent changes in the responses.

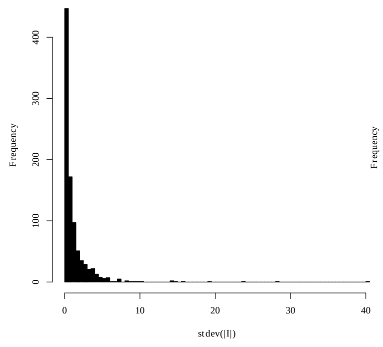
Second Experiment: Type of Changes We monitored in depth three large, data-centric web applications over several months: Yahoo! Mail, YouTube, and MySpace. We dumped HTTP responses captured by emulating user interaction using a custom, scriptable web browser implemented with Html-Unit. Examples of these interactions are as follows: visit the home page, login, browse the inbox, send messages, return to the home page, click links, log out. Manual inspection revealed some major changes in Yahoo! Mail. For instance, the most evident change consisted of a set of new features added to the search engine (e.g., local search, refined address field in maps search), which manifested themselves as new parameters found in the web search page (e.g. to take into account the country or the ZIP code). User



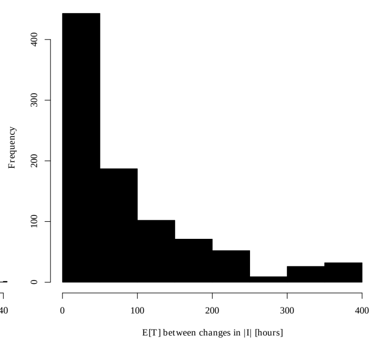
(a) Changes of forms.



(b) Avg. time between changes in $|F|$.



(c) Changes of inputs.



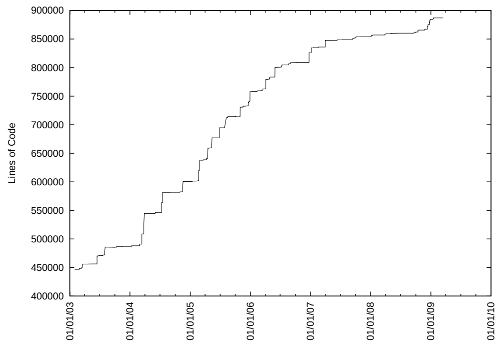
(d) Avg. time between changes in $|I|$

Figure 4.10: Relative frequency of the standard deviation of the number of forms (a) and input fields (c). Also, the distribution of the expected time between changes of forms (b) and input fields (d) are plotted. A non-negligible portion of the websites exhibits changes in the responses. No differences have been noticed between home pages and manually-selected pages.

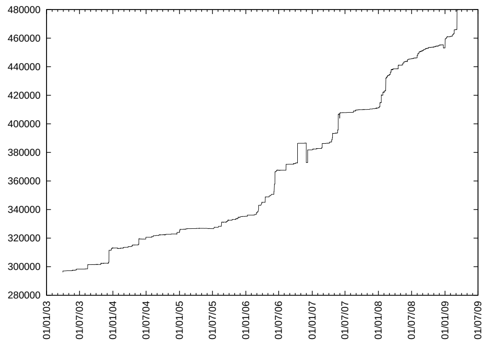
pages of YouTube were significantly updated with new functionalities between 2008 and 2009. For instance, the new version allows users to rearrange widgets in their personal pages. To account for the position of each element, new parameters are added to the profile pages and submitted asynchronously whenever the user drags widgets within the layout. The analysis on MySpace did not reveal any significant change. The results of these two experiments show that changes in server-side applications are common. More importantly, these modifications often involve the way user data is represented, handled, and manipulated.

Third Experiment: Abundance of Code Change We analyzed changes in the requests and sessions by inspecting the code repositories of three of the largest, most popular open-source web applications: WordPress, Movable Type, and PhpBB. The goal was to understand whether upgrading a web application to a newer release results in significant changes in the features that are used to determine its behavior. In this analysis, we examined changes in the source code that affect the manipulation of HTTP responses, requests, and session data. We used StatSVN, an open-source tool for tracking and visualizing the activity of *SubVersioN* (SVN) repositories (e.g., the number of lines changed or the most active developers). We modified StatSVN to incorporate a set of heuristics to compute approximate counts of the lines of code that, directly or indirectly, manipulate HTTP session, request or response data. In the case of *PHP Hypertext Preprocessor* (PHP), examples representative of such lines include, but are not limited to, `_REQUEST|_SESSION|_POST|_GET|session_-[a-z]+|http_|strip_tags|addslashes`. In order to take into account data manipulation performed through library functions (e.g., WordPress' custom `Http` class), we also generated application-specific code patterns by manually inspecting and filtering the core libraries. Figure 4.11 shows, over time, the lines of code in the repositories of PhpBB, WordPress, and Movable Type that manipulate HTTP responses, requests and, sessions. These results show the presence of significant modifications in the web application in terms of relevant lines of code added or removed. More importantly, such modifications affect the way HTTP data is manipulated and, thus, impact request, response or session models.

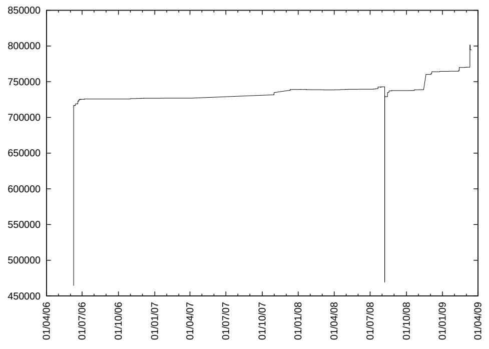
The aforementioned experiments confirm that the class of changes we described in Section 4.3.1.1 is common in real-world web applications. Therefore, anomaly detectors for web applications must incorporate procedures to prevent false alerts due to concept drift. In particular, a mechanism is needed to discriminate between legitimate and malicious changes, and respond accordingly. Note that, this somehow coincides with the original definition of ID (i.e., distinguishing among normal *vs.* malicious activity); however, our focus



(a) PhpBB



(b) WordPress



(c) Movable Type

Figure 4.11: Lines of codes in the repositories of PhpBB, WordPress, and Movable Type, over time. Counts include only the code that manipulates HTTP responses, requests and sessions.

is to recognizes changes in the application *activity* that could lead to concept drift as opposed to spotting out changes in the application *behavior* (i.e., ID).

4.3.2 Addressing concept drift

In this section, a technique is presented to distinguish between legitimate changes in web application behavior and evidence of malicious behavior.

4.3.2.1 Exploiting HTTP responses

An HTTP response is composed of a header and a body. Under the hypothesis of content-type `text/html`, the body of HTTP responses contains a set of links L_i and forms F_i that refer to a set of target resources. Each form also includes a set of input fields I_i. In addition, each link $l_{i,j} \in L_i$ and form $f_{i,j} \in F_i$ has an associated set of parameters.

A request q_i to a resource r_i returns a response $resp_i$. From $resp_i$ the client follows a link $l_{i,j}$ or submits a form $f_{i,j}$. Either of these actions generates a new HTTP request to the web application with a set of parameter key-value pairs, resulting in the return of a new HTTP response to the client, r_{i+1}, the body of which contains a set of links L_{i+1} and forms F_{i+1}. According to the specifications of the HTTP, this process continues until the session has ended (i.e., either the user has explicitly logged out, or a timeout has occurred). We then define:

Definition 4.3.2 (Candidate Resource Set) Given a resource *r* within a web application, the *candidate resource set* of *r* is defined as:

$$\begin{aligned} \text{candidates(r)} &:= L_r \cup F_r = \\ &= \{l_1, l_2, \ldots, l_N\} \cup \\ &\quad \{f_1, f_2, \ldots, f_M\}. \end{aligned}$$

where:

- $l_{(\cdot)} := resp.\texttt{a}_{(\cdot)}.\texttt{href}$,
- $f_{(\cdot)} := resp.\texttt{form}_{(\cdot)}.\texttt{action}$,

- $resp.$ is the `plain/text` body of the response $resp$,
- $resp.$`<element>`$_{(\cdot)}$.`<attribute>` is the content of the attribute,
- `<attribute>`, of the HTML `<element>`$_{(\cdot)}$.

Our key observation is that, at each step of a web application session, a subset of the *potential* target resources is given exactly by the content of the current resource. That is, given r_i, the associated sets of links L_i and forms F_i directly encode a significant sub-set of the possible r_{i+1}. Furthermore, each link $l_{i,j}$ and form $f_{i,j}$ indicates a precise set of expected parameters and, in some cases, the set of legitimate values for those parameters that can be provided by a client.

Consider a hypothetical banking web application, where the current resource $r_i = /$`account` presented to a client is an account overview. The response $resp_i$ may contain:

```
<body class="account">
  /* ... */

  <form action="/search" method="get">
    <input name="term" type="text" value="Insert term" />
    <input type="submit" value="Search!" />
  </form>

  <h1><a href="/account/transfer">Transfer funds</a></h1>
  <a href="/account/transfer/submit">Submit transfer</a>

  /* ... */

  <tr><td><a href="/account/history?aid=328849660322">See details</a
      ></td></tr>
  <tr><td><a href="/account/history?aid=446825759916">See details</a
      ></td></tr>

  /* ... */

  <a href="/logout">Logout</a>

  /* ... */

  <h2>Feedback on this page?</h2>
  <form action="/feedback" method="post" class="feedback-form">
```

```
    <select name="subject">
      <option>General</option>
      <option>User interface</option>
      <option>Functionality</option>
    </select>
    <textarea name="message" />
    <input type="submit" />
  </form>

  /* - */
</body>
```

containing a set of links L_i, represented as their URL, for instance:

$$L_i = \left\{ \begin{array}{l} \texttt{/account/history?aid=328849660322,} \\ \texttt{/account/history?aid=446825759916,} \\ \texttt{/account/transfer/submit,} \\ \texttt{/account/transfer,} \\ \texttt{/logout} \end{array} \right\}.$$

Forms are represented as their target action. For instance: $F_i = \{$`/feedback`, `/search`$\}$.

From L_i and F_i, we can deduce the set of legal candidate resources for the next request r_{i+1}. Any other resource would, by definition, be a deviation from a legal session flow through the web application as specified by the application itself. For instance, it would not be expected behavior for a client to directly access `/account/transfer/submit` (i.e., a resource intended to submit an account funds transfer) from r_i. Furthermore, for the resource `/account/history`, it is clear that the web application expects to receive a single parameter `aid` with an account number as an identifier.

In the case of the form with target `/feedback`, let the associated input elements be:

```
    <select name="subject">
      <option>General</option>
      <option>User interface</option>
      <option>Functionality</option>
    </select>
    <textarea name="message" />
```

It immediately follows that any invocation of the `/feedback` resource from r_i should include the parameters `subject` and `message`.

In addition, the legal set of values for the parameter `subject` is given by enumerating the enclosed `<option />` tags. Valid values for the new `tz` and `datetime` parameters mentioned in the example of Section 4.3.1.1 can be inferred using the same algorithm. Any deviation from these specifications could be considered evidence of malicious behavior.

In this section we described why responses generated by a web application constitute a specification of the intended behavior of clients and the expected inputs to an application's resources. As a consequence, when a change occurs in the interface presented by a web application, this will be reflected in the content of its responses. Therefore, as detailed in the following section, an anomaly detection system can take advantage of response modeling to detect and adapt to changes in monitored web applications.

4.3.2.2 Adaptive response modeling

In order to detect changes in web application interfaces, the response modeling of webanomaly has been augmented with the ability to build L_i and F_i from the HTML documents returned to a client. The approach is divided into two phases.

Extraction and parsing The anomaly detector parses each HTML document contained in a response issued by the web application to a client. For each `<a />` tag encountered, the contents of the `href` attribute is extracted and analyzed. The link is decomposed into tokens representing the protocol (e.g., `http`, `https`, `javascript`, `mailto`), target host, port, path, parameter sequence, and anchor. Paths are subject to additional processing; for instance, relative paths are normalized to obtain a canonical representation. This information is stored as part of an abstract document model for later processing.

A similar process occurs for forms. When a `<form />` tag is encountered, the `action` attribute is extracted and analyzed as in the case of the link `href` attribute. Furthermore, any `<input />`, `<textarea />`, or `<select />` and `<option />` tags enclosed by a particular `<form />` tag are parsed as parameters to the corresponding form invocation. For `<input />` tags, the `type`, `name`, and `value` attributes are extracted. For `<textarea />` tags, the `name` attribute is extracted. Finally, for `<select />` tags, the `name` attribute is extracted, as well as the content of any enclosed `<option />` tags. The target of the form and its pa-

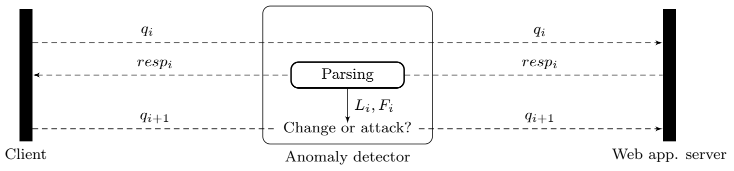


FIGURE 4.12: A representation of the interaction between the client and the web application server, monitored by a learning-based anomaly detector. After request q_i is processed, the corresponding response $resp_i$ is intercepted and link L_i and forms F_i are parsed to update the request models. This knowledge is exploited as a change detection criterion for the subsequent request q_{i+1}.

rameters are recorded in the abstract document model as in the case for links.

Analysis and modeling The set of links and forms contained in a response is processed by the anomaly engine. For each link and form, the corresponding target resource is compared to the existing known set of resources. If the resource has not been observed before, a new model is created for that resource. The session model is also updated to account for a potential transition from the resource associated with the parsed document and the target resource by training on the observed session request sequence.

For each of the parameters parsed from links or forms contained in a response, a comparison with the existing set of known parameters is performed. If a parameter has not already been observed (e.g., the new `tz` parameter), a profile is created and associated with the target resource model.

Any values contained in the response for a given parameter are processed as training samples for the associated models. In cases where the total set of legal parameter values is specified (e.g., `<select />` and `<option />` tags), the parameter profile is updated to reflect this. Otherwise, the profile is trained on subsequent requests to the associated resource.

As a result of this analysis, the anomaly detector is able to adapt to changes in session structure resulting from the introduction of new resources. In addition, the anomaly detector is able to adapt to changes in request structure resulting from the introduction of new parameters and, in a limited sense, to changes in parameter values.

4.3.2.3 Advantages and limitations

Due to the response modeling algorithm described in the previous section, an anomaly detector is able to automatically adapt to many common changes observed in web applications as modifications are made to the interface presented to clients. Both changes in session and request structure such as those described in Section 4.3.1.1 can be accounted for in an automated fashion.

Example 4.3.4 (I18N and L10N) The aforementioned modification is correctly handled as it consists in an addition of the `tz` parameter and a modification of the `datetime` parameter.

Furthermore, we claim that web application anomaly detectors that do not perform response modeling cannot reliably distinguish between anomalies caused by legitimate changes in web applications and those caused by malicious behavior. Therefore, as will be shown in Section 4.3.3, any such detector that solely monitors requests is more prone to false positives in the real world.

Clearly, the technique relies upon the assumption that the web application has not been compromised. Since the web application, and in particular the documents it generates, is treated as an oracle for whether a change has occurred, if an attacker were to compromise the application in order to introduce a malicious change, the malicious behavior would be learned as normal by the detector. Of course, in this case, the attacker would already have access to the web application. However, modern anomaly detectors like webanomaly observes all requests and responses to and from untrusted clients, therefore, any attack that would compromise response modeling would be detected and blocked.

Example 4.3.5 An attacker could attempt to evade the anomaly detector by introducing a malicious change in the HTTP responses and then exploits the change detection technique that would interpret the new malicious request as a legit change.

For instance, the attacker could incorporate a link that contain a parameter used to inject the attack vector. To this end, the attacker would have to gain control of the server by leveraging an existing vulnerability of the web application (e.g., a buffer overflow, a SQL injection). However, the HTTP requests used by the attacker to exploit the vulnerability will trigger several models (e.g., the string length model, in the case of a buffer overflow) and, thus, will be flagged as anomalous[2].

In fact, our technique does not alter the ability of the anomaly detector to detect attacks. On the other hand, it avoids many false positives, as demonstrated in Section 4.3.3.2.

Besides the aforementioned assumption, three limitations are important to note.

- The set of target resources may not always be statically derivable from a given resource. For instance, this can occur when client-side scripts are used to dynamically generate page content, including links and forms. Accounting for dynamic behavior would require the inclusion of script interpretation. This, however, has a high overhead, is complex to perform accurately, and introduces the potential for denial of service attacks against the anomaly detection system. For these reasons, such a component is not implemented in the current version of webanomaly, although further research is planned to deal with dynamic behavior.

- The technique does not fully address changes in the behavior of individual request parameters in its current form. In cases where legitimate parameter values are statically encoded as part of an HTML document, response modeling can directly account for changes in the legal set of parameter values. Unfortunately, in the absence of any other discernible changes in the response, changes in parameter values provided by clients cannot be detected. However, heuristics such as detecting when all clients switch to a new observable behavior in parameter values (i.e., all clients generate anomalies against a set of models in a similar way) could serve as an indication that a change in legitimate parameter behavior has occurred.

- The technique cannot handle the case where a resource is the result of a parametrized query and the previous response has not been observed by the anomaly detector. In our experience,

[2]The threat model assumes that the attacker can interact with the web application only by sending HTTP requests.

however, this does not occur frequently in practice, especially for sensitive resources.

4.3.3 Experimental Results

In this section, we show that our techniques reliably distinguish between legitimate changes and evidence of malicious behavior, and present the resulting improvement in terms of detection accuracy.

The goal of this evaluation is twofold. We first show that concept drift in modeled behavior caused by changes in web applications results in lower detection accuracy. Second, we demonstrate that our technique based on HTTP responses effectively mitigates the effects of concept drift. To this end, we adopted the training dataset described in Section 4.1.3.

4.3.3.1 Effects of concept drift

In the first experiment, we demonstrate that concept drift as observed in real-world web applications results in a significant negative impact on false positive rates.

1. webanomaly was trained on an unmodified, filtered data set. Then, the detector analyzed a test data set Q to obtain a baseline ROC curve.
2. After the baseline curve had been obtained, the test data set was processed to introduce new behaviors corresponding to the effects of web application changes, such as upgrades or source code refactoring, obtaining Q_{drift}. In this manner, the set of changes in web application behavior was explicitly known. In particular, as detailed in Table 4.2:

   sessions 6,749 new session flows were created by introducing requests for new resources and creating request sequences for both new and known resources that had not previously been observed;

   parameters new parameter sets were created by introducing 6,750 new parameters to existing requests;

   values the behavior of modeled features of parameter values was changed by introducing 5,785 mutations of observed values in client requests.

For example, each sequence of resources

$$\langle \texttt{/login, /index, /article} \rangle$$

might be transformed to

$$\langle \texttt{/login, /article} \rangle.$$

Similarly, each request like `/categories` found in the traffic might be replaced with `/foobar`. For new parameters, a set of link or form parameters might be updated by changing a parameter name and updating requests accordingly.

It must be noted that in all cases, responses generated by the web application were modified to reflect changes in client behavior. To this end, references to new resources were inserted in documents generated by the web application, and both links and forms contained in documents were updated to reflect new parameters.

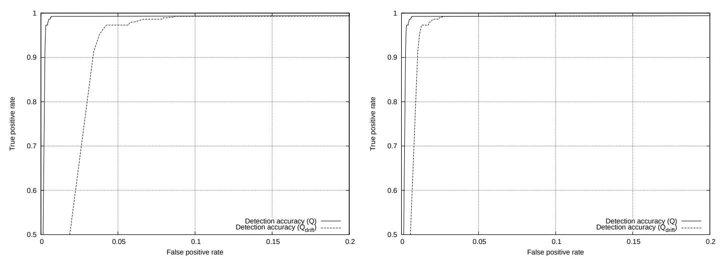
3. webanomaly– without the HTTP response modeling technique enabled – was then run over Q_{drift} to determine the effects of concept drift upon detector accuracy.

The resulting ROC curves are shown in Figure 4.13a. The consequences of web application change are clearly reflected in the increase in false positive rate for Q_{drift} versus that for Q. Each new session flow and parameter manifests as an alert, since the detector is unable to distinguish between anomalies due to malicious behavior and those due to legitimate change in the web application.

4.3.3.2 Change detection

The second experiment quantifies the improvement in the detection accuracy of webanomaly in the presence of web application change. As before, the detector was trained over an unmodified filtered data set, and the resulting profiles were evaluated over both Q and Q_{drift}. In this experiment, however, the HTTP response modeling technique was enabled.

Figure 4.13b presents the results of analyzing HTTP responses on detection accuracy. Since many changes in the behavior of the



(a) Response modeling disabled. (b) Response modeling enabled.

Figure 4.13: Detection and false positive rates measured on Q and Q_{drift}, with HTTP response modeling enabled in (b).

web application and its clients can be discovered using our response modeling technique, the false positive rate for Q_{drift} is greatly reduced over that shown in Figure 4.13a, and approaches that of Q, where no changes have been introduced. The small observed increase in false positive rate can be attributed to the effects of changes in parameter values. This occurs because a change has been introduced into a parameter value submitted by a client to the web application, and no indication of this change was detected on the preceding document returned to the client (e.g., because no `<select />` were found).

Table 4.2 displays the individual contributions to the reduction of the false positive rate due to the response modeling technique. Specifically, the total number of anomalies caused by each type of change, the number of anomalies erroneously reported as alerts, and the corresponding reduction in the false positive rate is shown. The results displayed were generated from a run using the optimal operating point (0.00144, 0.97263) indicated by the knee of the ROC curve in Figure 4.13b. For changes in session flows and parameters sets, the detector was able to identify an anomaly as being caused by a change in web application behavior in all cases. This resulted in a large net decrease in the false positive rate of the detector with response modeling enabled. The modification of parameters is more problematic, though; as discussed in Section 4.3.2.3, it is not always apparent that a change has occurred when that change is limited to the type of behavior a parameter's value exhibits.

From the overall improvement in false positive rates, we conclude

Change type	Anomalies	FP	Reduction
New session flows	6,749	0	100.0%
New parameters	6,750	0	100.0%
Modified parameters	5,785	4,821	16.6%
Total	19,284	4,821	75.0%

Table 4.2: Reduction in the false positive rate due to HTTP response modeling for various types of changes.

that HTTP response modeling is an effective technique for distinguishing between anomalies due to legitimate changes in web applications and those caused by malicious behavior. Furthermore, any anomaly detector that does not do so is prone to generating a large number of false positives when changes do occur in the modeled application. Finally, as it has been shown in Section 4.3.1, web applications exhibit significant long-term change in practice, and, therefore, concept drift is a critical aspect of web application anomaly detection that must be addressed.

4.4 Concluding Remarks

In this chapter we described in detail two approaches to mitigate training issues. In particular, we presented a technique to reduce FPs due to scarce training data and a mechanism to automatically recognize changes in the monitored applications, such as code updates, and re-train the models without any human intervention.

First, we have described our efforts to cope with an issue that must be addressed by web application anomaly detection systems in order to provide a reasonable level of security. The impact of this issue is particularly relevant for commercial web-based anomaly detection systems and web application firewall, which have to operate in real-world environment where sufficient training data might be unavailable within a reasonable time frame.

We have proposed the use of global knowledge bases of well-trained, stable profiles to remediate a local scarcity of training data by exploiting global similarities in web application parameters. We found that although using global profiles does result in a small reduction in detection accuracy, the resulting system, when given appropriate parameters, does provide reasonably precise modeling of otherwise unprotected web application parameters.

A possible future extension of the described approach is to investigate the use of other types of models, particularly HTTP response and session models. An additional line of future work is the application of different clustering methods in order to improve the efficiency of the querying procedure for high-cardinality knowledge bases.

Secondly, we have identified the natural dynamicity of web applications as an issue that must be addressed by modern anomaly-based web application anomaly detectors in order to prevent increases in the false positive rate whenever the monitored web application is changed. We named this frequent phenomenon the *web application concept drift*.

In [Maggi et al., 2009c] we proposed the use of novel HTTP response modeling techniques to discriminate between legitimate changes and anomalous behaviors in web applications. More precisely, responses are analyzed to find new and previously unmodeled parameters. This information is extracted from anchors and forms elements, and then leveraged to update request and session models. We have evaluated the effectiveness of our approach over an extensive real-world data set of web application traffic. The results show that the resulting system can detect anomalies and avoid false alerts in the presence of concept drift.

We plan to investigate the potential benefits of modeling the behavior of JavaScript code, which is becoming increasingly prevalent in modern web applications. Also, additional, richer, and media-dependent response models must be studied to account for interactive client-side components, such as Adobe Flash and Microsoft Silverlight applications.

Network and Host Alert Correlation 5

Either as part of an IDS or as a post-processing step, the alert correlation task is meant to recognize relations among alerts. For instance, a simple correlation task is to suppress all the alerts regarding unsuccessful attacks. A more complex example envisions a large enterprise network monitored by multiple IDSs, both host- and network-based. Whenever a misbehaving application is detected on a certain machine, the HIDS interacts with the NIDS that is monitoring the network segment of that machine to, say, validate the actual detection result. A more formal definition of alert correlation and alert correlation systems can be found in Section 2.1.2.

In this section we focus on practical aspects of alert correlation by first presenting the model we used in our prototype tools. Based on this model we concentrate on two challenging phases, namely *aggregation*, which has to do with grouping similar alerts together, and *correlation*, which is the actual correlation phase. In particular, in Section 5.1 we investigate the use of fuzzy metrics to time-based aggregation, which has been proposed recently as a simple but effective aggregation criterion. Unfortunately, given a threshold of, say, $10s$, this approach will fail if two alerts are at, say, $10.09s$ to each other. In Section 5.3 we focus on the actual correlation task. We exploit non-parametric statistical tests to create a simple but robust correlation system that is capable of recognizing series of related events without relying on previous knowledge. Last, in Section 5.2 we propose a methodology to compare alert correlation systems which are as difficult as IDSs to evaluate.

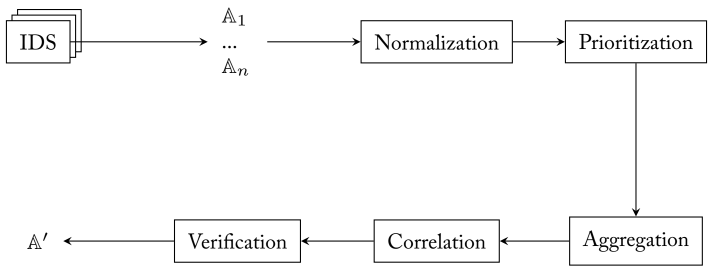


FIGURE 5.1: Simplified version of the correlation approach proposed in [Valeur et al., 2004].

5.1 Fuzzy Models and Measures for Alert Fusion

In this section we focus on the aggregation of IDS alerts, an important component of the alert correlation process as shown on Figure 5.1. We describe our approach [Maggi et al., 2009b], which exploit fuzzy measures and fuzzy sets to design simple and robust alert aggregation algorithms. Exploiting fuzzy sets, we are able to robustly state whether or not two alerts are "close in time", dealing with noisy and delayed detections. A performance metric for the evaluation of fusion systems is also proposed. Finally, we evaluate the fusion method with alert streams from anomaly-based IDS.

Alert aggregation becomes more complex when taking into account *anomaly detection* systems, because no information on the type or classification of the observed attack is available to any of the fusion algorithms. Most of the algorithms proposed in the current literature on correlation make use of the information regarding the matching attack provided by misuse detectors; therefore, such methods are inapplicable to purely anomaly based intrusion detection systems. Although some approaches, e.g., [Portokalidis and Bos, 2007], incorporate techniques to correlate automatically-generated signatures, such techniques are aimed at creating more general signatures in order to augment the DR. Instead, our focus is that of reducing the FPR by fusing more information sources. However, as described in Section 2.1, since anomaly and misuse detection are symmetric, it is reasonable and noteworthy to try to integrate different approaches through an alert fusion process.

Toward such goal, we explore the use of fuzzy measures [Wang and Klir, 1993] and fuzzy sets [Klir and Folger, 1987] to design simple, but robust aggregation algorithms. In particular, we contribute to one of the key issues, that is how to state whether or not two alerts are "close in time". In addition, uncertainties on both timestamp measurements and threshold setting make this process even more difficult; the use of fuzzy sets allows us to precisely define a time-distance criterion which "embeds" unavoidable errors (e.g., delayed detections).

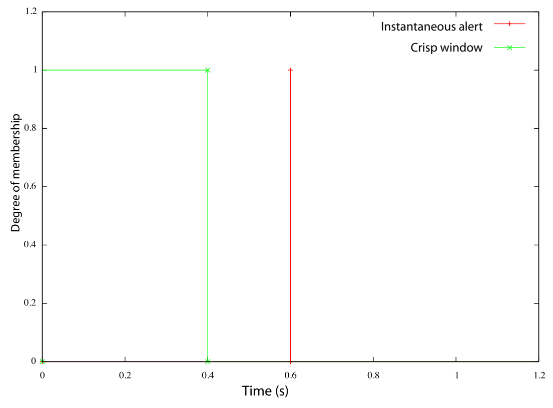
5.1.1 Time-based alert correlation

As proposed in most of the reviewed literature, a first, naive approach consists in exploiting the *time distance* between alerts for aggregation; the idea is to aggregate "near" alerts. In this section we focus on this point, starting by the definition of "near".

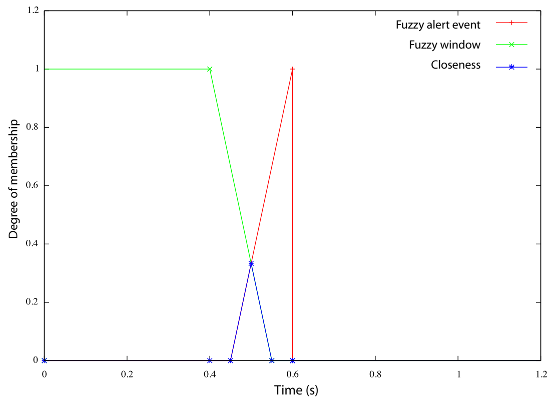
Time-distance aggregation might be able to catch simple scenarios like remote attacks against remote applications vulnerabilities (e.g., web servers). For instance, consider the scenario where, at time t_0, an attacker violates a host by exploiting a vulnerability of a server application. An IDS monitoring the system recognizes anomalous activity at time $t_n = t_0 + \tau_n$. Meanwhile, the attacker might escalate privileges and break through another application; the IDS would detects another anomaly at $t_h = t_0 + \tau_h$. In the most general case t_n is "close" to t_h (with $t_n < t_h$), so if $t_h - t_n \leq T_{near}$ the two alerts belong to the same attack.

The idea is to *fuse* alerts if they are both close in time, raised from the same IDS, and refer to the same source and destination. This intuition obviously needs to be detailed. First, the concept of "near" is not precisely defined; secondly, errors in timestamping are not taken into account; and, a crisp time distance measure is not robust. For instance, if $|t_h - t_n| = 2.451$ and $T_{near} = 2.450$ the two alerts are obviously near, but not aggregated, because the above condition is not satisfied. To overcome such limitations, we propose a *fuzzy* approach to time-based aggregation.

In the following, we will use the well-known dot notation, as in object-oriented programming languages, to access a specific alert attribute: e.g., $a.start\_ts$ indicates the value of the attribute $start\_ts$ of the alert a. Moreover, we will use $T_{(\cdot)}$ to indicate a threshold variable: for instance, T_{near} is the threshold variable called (or regarding to) "near".



(a)



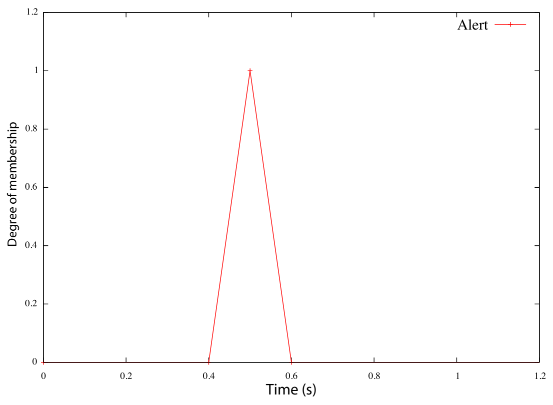
(b)

FIGURE 5.2: Comparison of crisp (a) and fuzzy (b) time-windows. In both graphs, one alert is fired at $t = 0$ and another alert occurs at $t = 0.6$. Using a crisp time-window and instantaneous alerts (a), the distance measurement is not robust to neither delays nor erroneous settings of the time-window size. Using fuzzy-shaped functions (b) provides more robustness and allows to capture the concept of "closeness", as implemented with the T-norm depicted in (b). Distances in time are normalized in [0,1] (w.r.t. the origin).

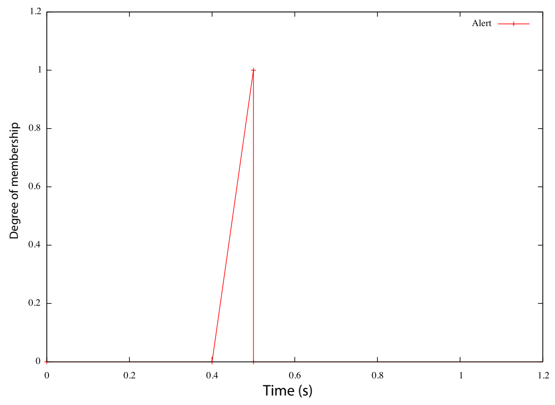
We rely on fuzzy sets for modeling both the uncertainty on the timestamps of alerts and the time distance in a robust manner. Regarding the uncertainty on measurements, we focus on delayed detections by using triangle-shaped fuzzy sets to model the occurrence of an alert. Since the measured timestamp may be affected by errors or delays, we extend the singleton shown in Figure 5.2 (a) with a triangle, as depicted in Figure 5.2 (b). We also take into account uncertainty on the dimension of the aggregation window: instead of using a crisp window (as in Figure 5.2 (a)), we extend it to a trapezoidal fuzzy set, resulting in a more robust distance measurement. In both graphs, one alert is fired at $t = 0$ and another alert occurs at $t = 0.6$. Using a crisp time-window and instantaneous alerts (Figure 5.2 (a)), the distance measurement is not robust to neither delays nor erroneous settings of the time-window size. Using fuzzy-shaped functions (Figure 5.2 (b)) provides more robustness and allows to capture the concept of "closeness", as implemented with the `T`-norm depicted in Figure 5.2 (b). In Figure 5.2 distances in time are normalized in [0,1] (w.r.t. the origin).

Note that, Figure 5.3 compares two possible manners to model uncertainty on alert timestamps: in Figure 5.3 (a) the alert is recorded at 0.5 seconds but the measurement may have both positive (in the future) and negative (in the past) errors. Figure 5.3 (b) is more realistic because positive errors are not likely to happen (i.e., we cannot "anticipate" detections), while events are often delayed, especially in network environments. In comparison to our proposal of using "fuzzy timestamps", the IDMEF describes event occurrences using *three* timestamps (`Create-`, `Detect-`, and `Analyzer-Time`): this is obviously more generic and allows the reconstruction of the entire event path from the attack to the analyzer that reports the alert. However, all timestamps are not always known: for example, the IDS might not have such a feature, thus the IDMEF `Alerts` cannot be completely filled.

As stated above, the main goal is to measure the time distance between two alerts, a_1 and a_2 (note that, in the following, a_2 occurs after a_1). We first exemplify the case of instantaneous alerts, that is $a_i.start\_ts = a_i.end\_ts = a_i.ts$ for $i \in \{1, 2\}$. To state whether or not a_1 is close to a_2 we use a T_{near} sized time window: in other words, an interval spreading from $a_1.ts$ to $a_1.ts + T_{near}$ (Figure 5.2 (a) shows the described situation for $a_1.ts = 0$, $T_{near} = 0.4$, and $a_2.ts = 0.6$: values have been normalized to place a_1 alert in the



(a)



(b)

Figure 5.3: Comparing two possible models of uncertainty on timestamps of single alerts.

origin).

Extending the example shown in Figure 5.2 (a) to uncertainty in measurements is straightforward; let us suppose to have an average uncertainty of 0.15 seconds on the measured (normalized w.r.t. the origin) value of $a_2.ts$: we model is as a triangle-shaped fuzzy set as the one drawn in Figure 5.2 (b).

In the second place, our method takes into account uncertainty regarding the thresholding of the distance between two events, modeled in Figure 5.2 (b) by the trapezoidal fuzzy window: the smoothing factor, 0.15 seconds, represents potential errors (i.e., the time values for which the membership function is neither 1 nor 0). Given these premises, the fuzzy variable "near" is defined by a T-norm [Klir and Folger, 1987] as shown in Figure 5.2 (b): the resulting triangle represents the alerts overlapping in time. In the example we used $\min(\cdot)$ as T-norm.

In the above examples, simple triangles and trapezoids have been used but more accurate/complex membership functions could be used as well. However, we remark here that trapezoid-like sets are conceptually different from triangles as the former have a membership value of 100%; this means certainty on the observed/measured phenomenon (i.e., closeness), while lower values mean uncertainty. Trapezoids-like functions should be used whenever a 100%-certainty interval is known: for instance, in Figure 5.2 (b) if two alerts occur within 0.4 seconds they are near at 100%; between 0.4 and 0.55 seconds, such certainty decreases accordingly.

Note 5.1.1 (Extended crisp window) It is important to underline that, from a purely practical point of view, one may achieve the same effect on alert fusion by building a more "flexible" crisp time window (e.g., by increasing its size by, say, 15%). However, from a theoretical point of view, it must be noted that our approach explicitly models the errors that occur during the alert generation process. On the other hand, by increasing a time window of a certain percentage would only give more flexibility, without giving any semantic definition of "near alerts". There is a profound difference between saying "15% larger" and assigning a precise membership function to a certain variable and defining T-norms for aggregation.

The approach can be easily generalized to take into account non-instantaneous events, i.e. $a_i.start\_ts < a_i.end\_ts$. In this case,

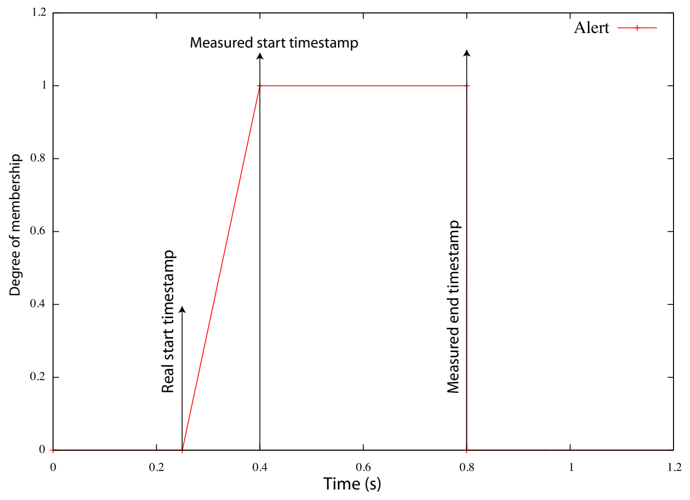


FIGURE 5.4: Non-zero long alert: uncertainty on measured timestamps are modeled.

alerts delays have to be represented by a trapezoidal fuzzy set. Note that, smoothing parameters are configurable values as well as fuzzy set shapes, in order to be as generic as possible w.r.t. the particular network environment; in the most general case, such parameters should be estimated before the system deployment. In Figure 5.4 the measured value of $a_i.start\_ts$ is 0.4, while $a_i.end\_ts = 0.8$; moreover, a smoothing factor of 0.15 seconds is added as a negative error, allowing the real $a_i.start\_ts$ to be 0.25.

5.1.1.1 Alert pruning

As suggested by the IDMEF `Confidence` attribute both the IDS described in Chapter 3 and [Zanero and Savaresi, 2004; Zanero, 2005b] — that we also use in the following experiments — feature an `attack_belief` value. For a generic alert a the `attack_belief` represents the deviation of the anomaly score, $a.score$, from the threshold, T. The concept of anomaly score is typical of anomaly detectors and, even if there is no agreement about its definition, it can intuitively interpreted as an internal, absolute indicator of abnormality. To complete our approach we consider the `attack_belief` attribute for alert pruning, after the aggregation phase.

Many IDSs, including the ones that we proposed, rely on probabilistic scores and thresholds in order to isolate anomalies from normal activity; thus, we implemented a first naive belief measure:

$$\text{Bel}(a) = |a.score - T_{anomaly}| \in [0, 1] \tag{5.1}$$

We remark that $a.score \in [0, 1] \wedge T_{anomaly} \in [0, 1] \Rightarrow \text{Bel}(a) \in [0, 1]$. Also note that in the belief theory [Wang and Klir, 1993; Klir and Folger, 1987] $\text{Bel}(B)$ indicates the belief associated to the hypothesis (or proposition) B; with an abbreviate notation, we indicate $\text{Bel}(a)$ meaning the belief of the proposition "the alert a is associated to a real anomaly". In this case, the domain of discourse is the set of all the alerts, which contains both the alerts that are real anomalies and the alerts that are not.

The belief theory has been used to implement complex decision support systems, such as [Shortliffe, 1976], in which a more comprehensive belief model has been formalized taking into account both belief and misbelief. The event (mis)belief basically represents the amount of evidence is available to support the (non-)occurrence of that event. In a similar vein, we exploit both the belief and the *a-priori* misbelief, the FPR. The FPR tells how much the anomaly detector is likely to report false alerts (i.e., the *a-priori* probability of reporting false alerts, or the so called *type I error*), thus it is a good candidate for modeling the misbelief of the alert. The more the FPR increases, the more the belief associated to an alert should decrease. In the first place, such intuition can be formalized using a linear-scaling function:

$$\text{Bel}_{lin}(a) = (1 - \underbrace{FPR}_{\text{misbelief}})\text{Bel}(a) \tag{5.2}$$

However, such a scaling function (see, dashed-line plot in Figure 5.5) makes the belief to decrease too fast w.r.t. the misbelief. As Figure 5.5 shows, a smoother rescaling function is the following:

$$\text{Bel}_{exp}(a) = (1 - FPR)\text{Bel}(a)e^{FPR} \tag{5.3}$$

The above defined `attack_belief` is exploited to further reduce the resulting set of alerts. Alerts are reported only if

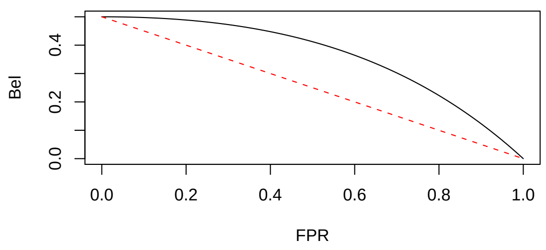


FIGURE 5.5: A sample plot of $\text{Bel}_{exp}(a)$ (with $|a.score - T_{anomaly}|, FPR \in [0, 1]$) compared to a linear scaling function, i.e. $\text{Bel}_{lin}(a)$ (dashed line)

$$a_i.attack\_belief > T_{bel}$$

where T_{bel} must be set to reasonable values according to the particular environment (in our experiments we used 0.66). The attack belief variable may be modeled using fuzzy variables with, for example, three membership functions labeled as "Low", "Medium", and "High" (IDMEF uses the same three-values assignment for each alert `Confidence` attribute and we used crisp numbers for it), but this has not been implemented yet in our working prototype.

Let us now to summarize the incremental approaches; the first "*naive time-distance*" uses a crisp window (as in Figure 5.2 (a)) to groups alerts w.r.t. their timestamps. The second approach, called "*fuzzy time-distance*", extends the first method: it relies on fuzzy sets (as shown in Figure 5.2 (b)) to take into account uncertainty on timestamping and window sizing. The last version, "*fuzzy time-distance with belief-based alert pruning*", includes the last explained criterion. In the following we will show the results of the proposed approaches on a realistic test bench we have developed using alerts generated by our IDS on the complete IDEVAL testing dataset (both host and network data).

5.2 Mitigating Evaluation Issues

This section investigates the problem of testing an alert fusion engine. In order to better understand the data we used, the experimental setup is here detailed and, before reporting the results obtained with the fuzzy approach introduced, we also briefly sketch the theoretical foundations, the accuracy, and the peculiarities of the overall implementation of the IDSs used for gathering the alerts. At the end of the section, we compare the results of the aggregation phase in comparison with an analysis of the accuracy of an alternative fusion approach [Qin and Lee, 2003].

5.2.1 A common alert representation

The output format of our prototypes has been designed with IDMEF in mind, to make the normalization process — the first step Figure 5.1 — as straightforward as possible. Being more precise, alerts are reported by our tools as a modified version of the original Snort [Roesch, 2009] alert format, in order to take into account both the peculiarities of the anomaly detection approach, namely the lack of attacks name, and suggestions from the IDMEF data model. In this way, our testing tools can easily implement conversion procedures from and to IDMEF.

Basically, we represent an alert as a tuple with the following 9 attributes:

```
(ids_id, src, dst, ids_type, start_ts, end_ts, attack_class, attack_bel,
                              target_resource)
```

The `ids_id` identifies the IDS, of type `ids_type` (host or network), that has reported the alert; `src` and `dst` are the IP source and destination addresses, respectively; `start_ts` and `end_ts` are the detection *Network Time Protocol* (NTP) [Mills, 1992] timestamps: they are both mandatory, even if the alert duration is unknown (i.e., `start_ts` equals `end_ts`). The discrete attribute `attack_class` (or name) indicates the name of the attack (if known); anomaly detectors cannot provide such an information because recognized attacks names are not known, by definition. Instead, anomaly detectors are able to quantify "how anomalous an activity is" so we exploited this characteristic and defined the `attack_belief`. Finally, the `target_resource` attribute stores the TCP port (or service name), if known, or the program begin attacked. An example instance is the following one:

```
(127.0.0.1-z7012gf, 127.0.0.1, 127.0.0.1, H, 0xbc723b45.0xef449129,
        0xbc723b45.0xff449130, -, 0.141044, fdformat(2351))
```

The example reports an attack detected from an host IDS (`ids_type = H`), identified by `127.0.0.1-z7012gf`: the attack against `fdformat` is started at NTP-second 0xbc723b45 (picosecond 0xef449129) and finished at NTP-second 0xbc723b45 (picosecond 0xff449130); the analyzer detected the activity as an attack with a belief of 14.1044%. An equivalent example for the network IDS (`ids_type = N`) anomaly detector would be something like:

```
(172.16.87.101/24-a032j11, 172.16.87.103/24, 172.16.87.100/24, N,
 0xbc723b45.0xef449129, 0xbc723b45.0xff449130, -, 0.290937, smtp)
```

Here the attack is against the `smtp` resource (i.e., protocol) and the analyzer believes there is an attack at 29.0937%.

5.2.2 Proposed Evaluation Metrics

Alert fusion is a relatively new problem, thus evaluation techniques are limited to a few approaches [Haines et al., 2003]. The development of solid testing methodologies is needed from both the theoretical and the practical points of view (a problem shared with IDS testing).

The main goal of fusion systems is to reduce the amount of alerts the security analyst has to check. In doing this, the *global* DR should ideally not decrease while the *global* FPR should be reduced as much as possible. This suggests that the first sub-goal of fusion systems is the *reduction* of the *global* FPR (for instance, by reducing the total number of alerts fired by source IDS through a rejection mechanism). Moreover, the second sub-goal is to limit the decreasing of the *global* DR. Let us now formalize the concepts we just introduced.

We indicate with $\mathbb{A}_i$ the *alert set* fired by the i-th IDS. An *alert stream* is actually a structure $\langle \mathbb{A}_i, \prec \rangle$ over $\mathbb{A}_i$, where the binary relation $\prec$ means "occurs before". More formally

$$\forall a, a' \in \mathbb{A}_i : a \prec a' \Rightarrow a.start\_ts < a'.start\_ts.$$

with $a.start\_ts, a'.start\_ts \in \mathbb{R}^+$. We assume that common operations between sets such as the union $\cup$ preserves the order or, otherwise, that the order can be recomputed; hence, given the union between two alert sets $\mathbb{A}_k = \mathbb{A}_i \cup \mathbb{A}_j$, the alert stream (i.e., ordered set) can be always reconstructed.

In the second place, we define the two functions $d : \mathbb{A} \times \mathbb{O} \mapsto [0,1]$, $f : \mathbb{A} \times \mathbb{O} \mapsto [0,1]$ representing the computation of the DR and the FPR, respectively, that is: $DR_i = \mathrm{d}(\mathbb{A}_i, \hat{\mathbb{O}})$, $FPR_i = \mathrm{f}(\mathbb{A}_i, \hat{\mathbb{O}})$. The set $\mathbb{O}$ contains each and every true alert; $\hat{\mathbb{O}}$ is a particular instance of $\mathbb{O}$ containing alerts occurring in the system under consideration (i.e., those listed in the truth file). We can now define the *global* DR, $DR_{\mathbb{A}}$, and the *global* FPR, $FPR_{\mathbb{A}}$, as:

$$\mathbb{A} = \bigcup_{i \in I} \mathbb{A}_i \tag{5.4}$$

$$DR_{\mathbb{A}} = \mathrm{d}\left(\mathbb{A}, \hat{\mathbb{O}}\right) \tag{5.5}$$

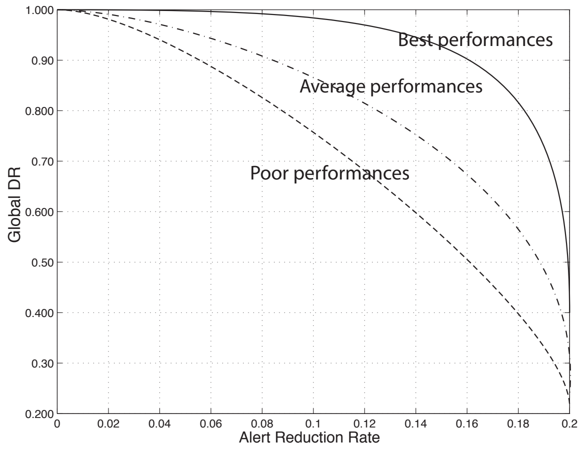
$$FPR_{\mathbb{A}} = \mathrm{f}\left(\mathbb{A}, \hat{\mathbb{O}}\right) \tag{5.6}$$

The output of a generic alert fusion system, is another alert stream $\mathbb{A}'$ with $|\mathbb{A}'| \leq |\mathbb{A}|$; of course, $|\mathbb{A}'| = |\mathbb{A}|$ is a degenerative case. To measure the "impact" of an alert fusion system we define the *Alert Reduction Rate* (ARR) which quantifies:

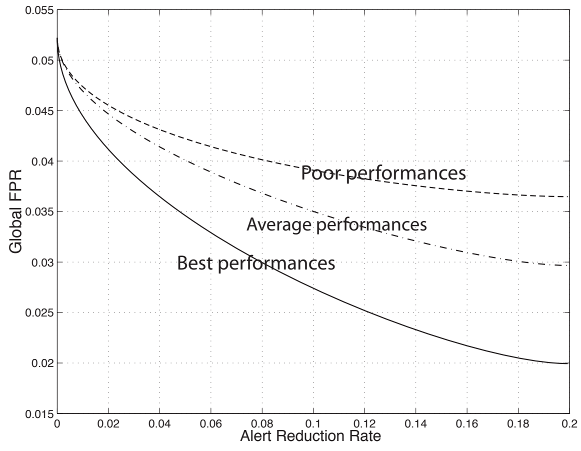
$$ARR = \frac{|\mathbb{A}'|}{|\mathbb{A}|}. \tag{5.7}$$

The aforementioned sub-goals can now be formalized into two performance metrics. The ideal correlation system would maximize both ARR and $\frac{DR_{\mathbb{A}'}}{DR_{\mathbb{A}}}$; meanwhile, it would minimize $\frac{FPR_{\mathbb{A}'}}{FPR_{\mathbb{A}}}$. Of course, $ARR = 1$ and $ARR = 0$ are degenerative cases. Low (i.e., close to 0.0) $\frac{FPR_{\mathbb{A}'}}{FPR_{\mathbb{A}}}$ means that the fusion system has significantly decreased the original FPR; in a similar vein, high (i.e., close to 1.0) $\frac{DR_{\mathbb{A}'}}{DR_{\mathbb{A}}}$ means that, while reducing false positive alerts, the loss of detection capabilities is minimal.

Therefore, a fusion system is better than another if it has *both* a greater $\frac{DR_{\mathbb{A}'}}{DR_{\mathbb{A}}}$ and a lower $\frac{FPR_{\mathbb{A}'}}{FPR_{\mathbb{A}}}$ rate than the latter. This criterion by no means has to be considered as complete or exhaustive. In addition,



(a)



(b)

Figure 5.6: Hypothetical plots of (a) the global DR, $DR_{\mathbb{A}'}$, vs. ARR and (b) the global FPR, $FPR_{\mathbb{A}'}$, vs. ARR.

it is useful to compare $DR_{\mathbb{A}'}$ and $FPR_{\mathbb{A}'}$ plots, vs. ARR, of different correlation systems obtaining diagrams like the one exemplified in Figure 5.6; this gives a graphical idea of which correlation algorithm is better. For instance, Figure 5.6 show that the algorithm labeled as Best performances is better than the others, because it shows higher $FPR_{\mathbb{A}'}$ reduction while $DR_{\mathbb{A}'}$ does not significantly decrease.

Note 5.2.1 (Configuration parameters) One limitation of our approach is that it relies on the attack belief threshold. Currently, there is no rigorous procedure for deriving it in practice. In our experiments, we run the aggregation algorithm several times on the same data-set under the same conditions until acceptable values of DR and FPR.

Similarly, the current version of our algorithm relies on alpha cut values for deciding whether or not two alerts can be fused or not. However, note that this is different from setting thresholds on time, e.g., fusing alerts that are delayed, say, 60 seconds. In fact, such a threshold would have *direct* impact on the fusion procedure; instead, different values of alpha cuts and shapes of trapezoid fuzzy sets have a much smoother effects on the results. In addition, a user of our system may not be aware of what does it means for two alerts to be, say, 60-seconds-close in time; it might be easier for the user to just configure the system to fuse alerts that are 50%-close.

5.2.3 Experimental Results

In these experiments we used two prototypes for network- and host-based anomaly detection. In particular, we use the system described in Chapter 3 (host-based) and [Zanero and Savaresi, 2004; Zanero, 2005b] (network-based). These tools were used to generate alert streams for testing the three variants of the fuzzy aggregation algorithm we proposed. The same tools were also used to generate alerts data for testing our correlation algorithm detailed in Section 5.3.

Because of the well known issues of IDEVAL and to avoid biased interpretation of the results, in our testing we used the IDEVAL dataset with the following simplification: we just tried to fuse the stream of alerts coming from a HIDS sensor and a NIDS sensor, which is monitoring the whole network. To this end, we ran the two prototypes on the whole 1999 IDEVAL testing dataset, using the network dumps and the host-based logs from `pascal`. We ran the NIDS prototype on `tcpdump` data and collected 128 alerts for attacks against the host `pascal.eyrie.af.mil`. The NIDS also generated 1009 alerts related to other hosts. Using the HIDS prototype we generated 1070 alerts from the dumps of the host `pascal.eyrie.af.-mil`. The NIDS was capable of detecting almost 66% of the attacks with less than 0.03% of false positives; the HIDS performs even better with a DR of 98% and 1.7% of false positives. However, in this experiment we focus on the aggregated results rather than the detection capabilities of the single system.

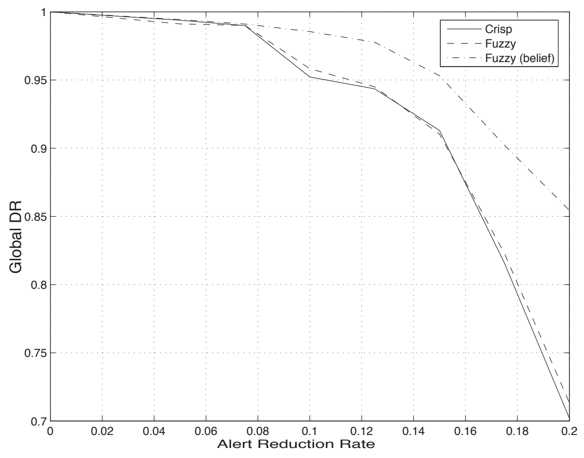
In the following, we use this shorthand notation: Net is the substream of all the alerts generated by the NIDS. $HostP$ is the substream of all the alerts generated by the HIDS installed on `pascal.-eyrie.af.mil`, while $NetP$ regards all the alerts (with `pascal` as a target) generated by the NIDS; finally, $NetO = Net \backslash NetP$ indicates all the alerts (with all but `pascal` as a target) generated by the NIDS.

Using the data generated as described above, and the metrics proposed in Section 5.2.2, we compared three different versions of the alert aggregation algorithms described in Section 5.1. In particular we compare the use of crisp time-distance aggregation, the use use of a simple fuzzy time-distance aggregation; and finally, the use of `attack_belief` for alert pruning.

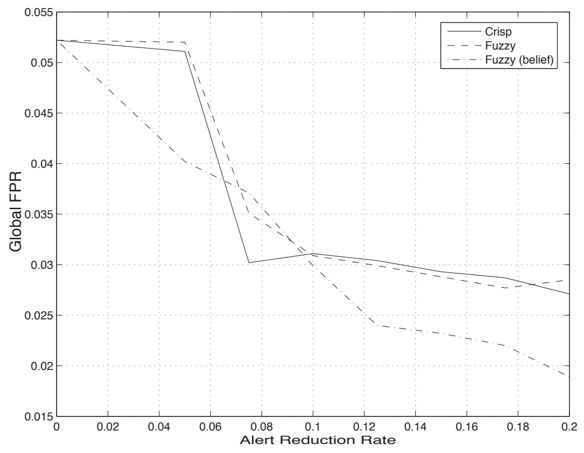
Numerical results are plotted in in Figure 5.7 for different values of ARR. As we discussed in Section 5.2, Figure 5.7 (a) refers to the reduction of DR while Figure 5.7 (b) focuses on FPR. $DR_{\mathbb{A}'}$ and $FPR_{\mathbb{A}'}$ were calculated using the complete alert stream, network and host, at different values of ARR. The values of ARR are obtained by changing the parameters values: in particular, we set $T_{bel} = 0.66$, the alpha cut of T_{near} to 0.5, the window size to 1.0 seconds, and varied the smoothing of the trapezoid between 1.0 and 1.5 seconds, and the alert delay between 0 and 1.5 seconds. It is not useful to plot the increase in false negatives, as it can be easily deduced from the decrease in DR.

The last aggregation algorithm, denoted as "Fuzzy (belief)", shows better performances since the $DR_{\mathbb{A}'}$ is always higher w.r.t. the other two aggregation strategies; this algorithm also causes a significant reduction of the $FPR_{\mathbb{A}'}$. Note that, taking into account the `attack_-belief` attribute makes the difference because it avoids true positives to be discarded; on the other hand, *real* false positive are not reported in the output alert stream because of their low belief.

It is difficult to properly compare our approach with other other fusion approaches proposed in the reviewed literature, because the



(a)



(b)

Figure 5.7: Plot of the $DR_{\mathbb{A}'}$ (a) and $FPR_{\mathbb{A}'}$ (b) vs. ARR. "Crisp" refers to the use of the crisp time-distance aggregation; "Fuzzy" and "Fuzzy (belief)" indicates the simple fuzzy time-distance aggregation and the use of the `attack_belief` for alert discarding, respectively.

latter were not specifically tested from the aggregation point of view, separately from the correlation one. Since the prototypes used to produce results are not released, we are only able to compare our approach with the *overall* fusion systems presented by others.

5.3 Using Non-parametric Statistical Tests

In this section we analyze the use of different types of statistical tests for the correlation of anomaly detection alerts. We show that the technique proposed in [Qin and Lee, 2003] and summarized in Section 2.3, one of the few proposals that can be extended to the anomaly detection domain, strongly depends on good choices of a parameter which proves to be both sensitive and difficult to estimate. Rather than using the *Granger Causality Test* (GCT) we propose alternative approach [Maggi and Zanero, 2007] based on a set of simpler statistical tests. We proved that our criteria work well on a simplified correlation task, without requiring complex configuration parameters. More precisely, in [Maggi and Zanero, 2007] we explored the use of *statistical causality tests*, which have been proposed for the correlation of IDS alerts, and which could be applied to anomaly based IDS as well. We redefine the problem in terms of a simpler statistical test, and experimentally validate it.

5.3.1 The Granger Causality Test

We tested the sensitivity of the GCT to the choice of two parameters: the sampling time of the time series that represent the alerts, w, and the time lag l, i.e., the order of the AR model being fitted. In [Qin and Lee, 2003] the sampling time was arbitrarily set to $w = 60s$, while the choice of l is not documented. However, our experiments show that the choice of these parameters can strongly influence the results of the test. As explained in [Maggi et al., 2009b], other values of l lead to awkward results.

We used the same dataset described in Section 5.2.3, with the following approach: we tested if $NetP$ has any causal relationship[1] with $HostP$; we also tested $HostP \not\leadsto NetP$. Since only a reduced number of alerts was available and since it was impossible to aggregate alerts along the "attack name" attribute (unavailable on anomaly detectors), we preferred to construct only two, large time series: one from $NetP$ (denoted as $NetP(k)$) and one from $HostP$ (denoted as $HostP(k)$). In the experiment reported in [Qin and Lee, 2003] the sampling time, w, has been fixed at 60 seconds, although we

[1]In the following we will use the symbol "$\leadsto$" to denote "Granger-causes" so, for instance, "A $\leadsto$ B" has to be read as "A causes B"; the symbol "$\not\leadsto$" indicates the negated causal relationship, i.e., "does not Granger-cause".

tested different values from 60 to 3600 seconds. In our simple experiment, the expected result is that $NetP \rightsquigarrow HostP$, and that $HostP \not\rightsquigarrow NetP$ (the $\rightsquigarrow$ indicates "causality" while $\not\rightsquigarrow$ is its negation).

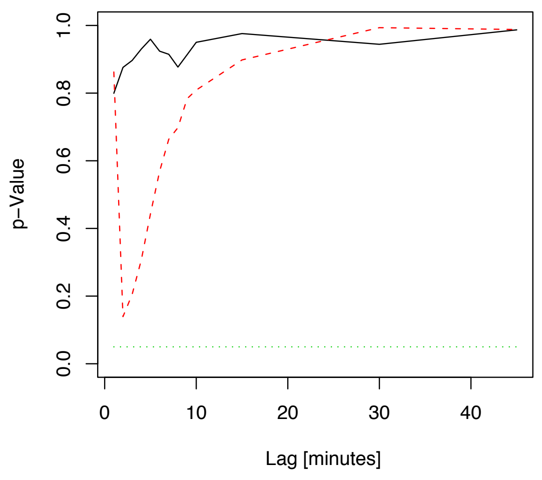
Since the first experiment ($w = 60$) led us to unexpected results (i.e., using a lag, l, of 3 minutes, both $NetP(k) \not\rightsquigarrow HostP(k)$, and vice versa) we decided to plot the test results (i.e., p-value and GCI) vs. both l and w. In Figure 5.8 (a) l is reported in minutes while the experiment has been performed with $w = 60s$; the dashed line is the p-value of the test "$NetP(k) \rightsquigarrow HostP(k)$", the solid line is opposite one. As one may notice, neither the first nor the second test passed, thus nothing can be concluded: fixing the test significance at $\alpha = 0.20$ is the only way for refusing the null hypothesis (around $l \simeq 2.5$ minutes) to conclude both that $NetP \rightsquigarrow HostP$" and that $HostP \not\rightsquigarrow NetP$; the GCI plotted in Figure 5.8 (b) confirms the previous result. However, for other values of l the situation changes leading to opposite results.

Figure 5.9 shows the results of the test for $w = 1800$ seconds and $l \in [50, 300]$ minutes. If we suppose a test significance of $\alpha = 0.05$ (dotted line), for $l \simeq 120$ the result is that $HostP \rightsquigarrow NetP$ while $NetP \not\rightsquigarrow HostP$, the opposite of the previous one. Moreover, for other values of l the p-value leads to different results.

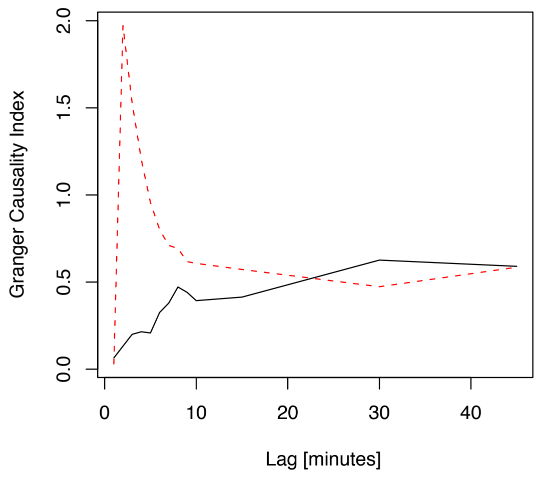
The last experiment results are shown in Figure 5.10: for $l > 230$ minutes, one may conclude that $NetP \rightsquigarrow HostP$ and $HostP \not\rightsquigarrow NetP$; i.e., the expected result, even if the p-value for the second test is close to $\alpha = 0.05$.

The previous experiments show only that the GCT failed on the dataset we generated, not that it does not work well in general. It may be the case that it is not suitable for "blind" alerts (i.e., without any information about the attack) reported by anomaly detectors: in fact, [Qin and Lee, 2003] used time series built from hyper-alerts resulting from an aggregation along *all* attributes: instead, in the anomaly-based case, the lack of attack names does not allow such hyper-alerts to be correctly constructed. In any case, the test result significantly depends on l (i.e., the test is asymptotic); this means that an appropriate choice of l will be required, depending on specific environmental conditions. The optimal l is also likely to change over time in a given environment.

A possible explanation is that the GCT is significant only if both the linear regression models are optimal, in order to calculate the

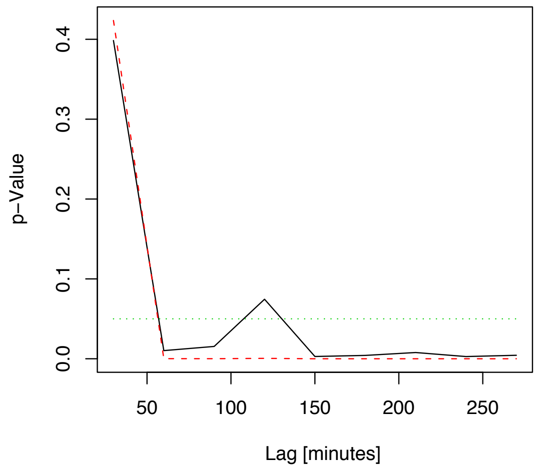


(a)

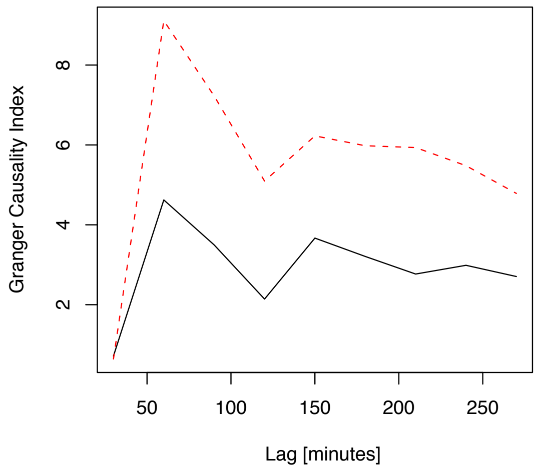


(b)

Figure 5.8: p-value (a) and GCI (b) vs. l (in minutes) for the first GCT experiment ($w = 60.0$ seconds): "$NetP(k) \rightsquigarrow HostP(k)$" (dashed line), "$HostP(k) \rightsquigarrow NetP(k)$" (solid line).



(a)



(b)

FIGURE 5.9: p-value (a) and GCI (b) vs. l (in minutes) for the first Granger causality test experiment ($w = 1800.0$ seconds): "$NetP(k) \leadsto HostP(k)$" (dashed line), "$HostP(k) \leadsto NetP(k)$" (solid line).

correct residuals. If we use the *Akaike Information Criterion* (AIC) [Akaike, 1974] to estimate the optimal time lag $\hat{l}$ over different windows of data, we find out that $\hat{p}$ wildly varies over time, as it is shown in Figure 5.11. The fact that there is no stable optimal choice of l, combined with the fact that the test result significantly depends on it, makes us doubt that the Granger causality test is a viable option for general alert correlation. The choice of w seems equally important and even more difficult to perform, except by guessing.

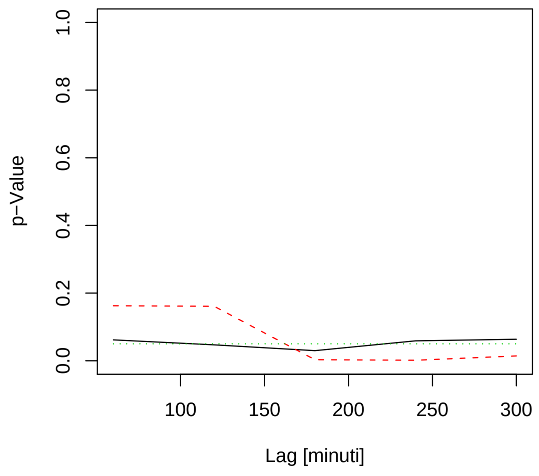
Of course, our testing is not conclusive: the IDEVAL alert sets may simply not be adequate for showing causal relationships. Another, albeit more unlikely, explanation, is that the GCT test may not be suitable for anomaly detection alerts: in fact, in [Qin and Lee, 2003] it has been tested on misuse alerts. But in fact there are also theoretical reasons to doubt that the application of the Granger test can lead to stable, good results. First, the test is asymptotic w.r.t. l meaning that the results reliability decreases as l increases because of the loss of degrees of freedom. Second, it is based on the strong assumption of *linearity* in the auto-regressive model fitting step, which strongly depends on the observed phenomenon. In the same way, the stationarity assumption of the model does not always hold.

5.3.2 Modeling alerts as stochastic processes

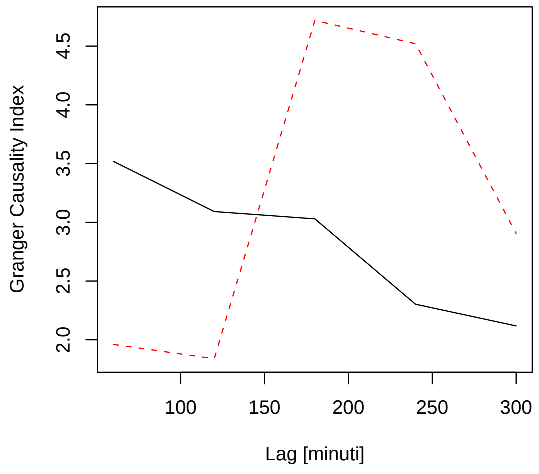
Instead of interpreting alert streams as time series (as proposed by the GCT-based approach), we propose to change point of view by using a stochastic model in which alerts are modeled as (random) events in time. This proposal can be seen as a formalized extension of the approach introduced in [Valeur et al., 2004] (see Figure 5.1), which correlates alerts if they are fired by different IDS within a "negligible" time frame, where "negligible" is defined with a crisp, fixed threshold.

For simplicity, once again we describe our technique in the simple case of a single HIDS and a single NIDS which monitors the whole network. The concepts, however, can be easily generalized to take into account more than two alert streams, by evaluating them couple by couple. For each alert, we have three essential information: a timestamp, a "target" host (fixed, in the case of the HIDS, to the host itself), and the generating sensor (in our case, a binary value).

With a self-explaining notation, define the following random variables: T_{NetP} are the arrival times of network alerts in $NetP$ (T_{NetO}, T_{HostP} are similarly defined); ε_{NetP} (ε_{NetO}) are the de-



(a)



(b)

Figure 5.10: p-value (a) and GCI (b) vs. l (in minutes) for the first Granger causality test experiment ($w = 3600.0$ seconds): "$NetP(k) \rightsquigarrow HostP(k)$" (dashed line), "$HostP(k) \rightsquigarrow NetP(k)$" (solid line).

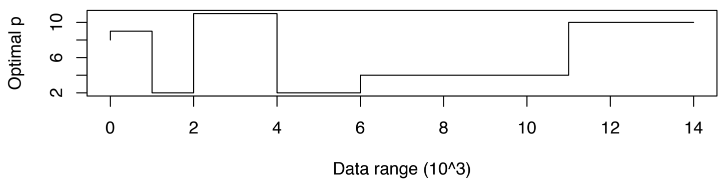


Figure 5.11: The optimal time lag $p = \hat{l}$ given by the AIC criterion strongly varies over time.

lays (caused by transmission, processing and different granularity in detection) between a specific network-based alert regarding `pascal` (not `pascal`) and the corresponding host-based one. The actual values of each $T_{(\cdot)}$ is nothing but the set of timestamps extracted from the corresponding alert stream. We reasonably assume that ε_{NetP} and T_{NetP} are stochastically independent (the same is assumed for ε_{NetO} and T_{NetO}). Figure 5.12 shows how delays between network and host alerts are calculated.

In an *ideal* correlation framework with two equally perfect IDS with a 100% DR and 0% FPR, if two alert streams are correlated (i.e., they represent independent detections of the same attack occurrences by different IDSs [Valeur et al., 2004]), they also are "close" in time. $NetP$ and $HostP$ should evidently be an example of such a couple of streams. Obviously, in the real world, some alerts will be missing (because of false negatives, or simply because some of the attacks are detectable only by a specific type of detector), and the distances between related alerts will therefore have some higher variability. In order to account for this, we can "cut off" alerts that are too far away from a corresponding alert in the other time series, presuming them to be singletons. In our case, knowing that single attacks did not last more than $400s$ in the original dataset, we tentatively set a cutoff threshold at this point.

Under the given working assumptions, the proposed stochastic model is used to formalize the correlation problem as a set of two statistical hypothesis tests:

$$\begin{array}{ccc} H_0 & & H_1 \\ T_{HostP} \neq T_{NetP} + \varepsilon_{NetP} & vs. & T_{HostP} = T_{NetP} + \varepsilon_{NetP} \end{array} \tag{5.8}$$

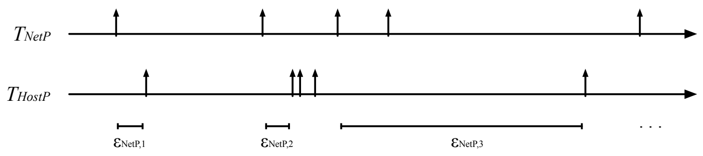


FIGURE 5.12: How delays between network and host alerts are calculated.

$$T_{HostP} \neq T_{NetO} + \varepsilon_{NetO} \quad vs. \quad T_{HostP} = T_{NetO} + \varepsilon_{NetO} \tag{5.9}$$

Let $\{t_{i,k}\}$ be the observed timestamps of $T_i, \forall i \in \{HostP, NetP, NetO\}$, the meaning of the first test is straightforward: within a random amount of time, ε_{NetP}, the occurring of a host alert, $t_{HostP,k}$, is preceded by a network alert, $t_{NetP,k}$. If this does not happen for a statistically significant amount of events, the test result is that alert stream T_{NetP} is *uncorrelated* to T_{HostP}; in this case, we have *enough statistical evidence* for refusing H_1 and accepting the null one. Symmetrically, refusing the null hypothesis of the second test means that the $NetO$ alert stream (regarding to all hosts but `pascal`) is correlated to the alert stream regarding `pascal`.

Note that, the above two tests are strongly related to each other: in an ideal correlation framework, it cannot happen that both "$NetP$ is correlated to $HostP$" and "$NetO$ is correlated to $HostP$": this would imply that the network activity regarding to all hosts but `pascal` (which raises $NetO$) has to do with the host activity of `pascal` (which raises $HostP$) with the same order of magnitude of $NetP$, that is an intuitively contradictory conclusion. Therefore, the second test acts as a sort of "robustness" criterion.

From our alerts, we can compute a sample of ε_{NetP} by simply picking, for each value in $NetP$, the value in $HostP$ which is closest, but greater (applying a threshold as defined above). We can do the same for ε_{NetO}, using the alerts in $NetO$ and $HostP$.

The next step involves the *choice of the distributions* of the random variables we defined above. Typical distributions used for modeling random occurrences of timed events fall into the family of exponential *Probability Density Functions* (PDFs) [Pestman, 1998]. In particular, we decided to fit them with Gamma PDFs, because our

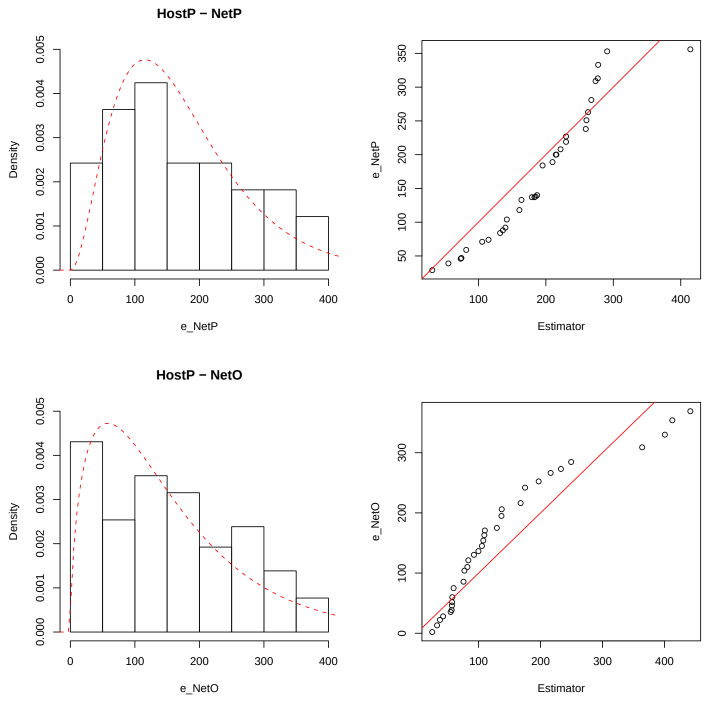


Figure 5.13: Histograms vs. estimated density (red dashes) and Q-Q plots, for both $\hat{f_O}$ and $\hat{f_P}$.

experiments show that such a distribution is a good choice for both the ε_{NetP} and ε_{NetO}.

The estimation of the PDF of ε_{NetP}, $f_P := f_{\varepsilon_{NetP}}$, and ε_{NetO}, $f_O := f_{\varepsilon_{NetO}}$, is performed using the well known *Maximum Likelihood* (ML) technique [Venables and Ripley, 2002] as implemented in the GNU S package: the results are summarized in Figure 5.13. f_P and f_O are approximated by

$$\text{Gamma}(3.0606, 0.0178)$$

$$\text{Gamma}(1.6301, 0.0105)$$

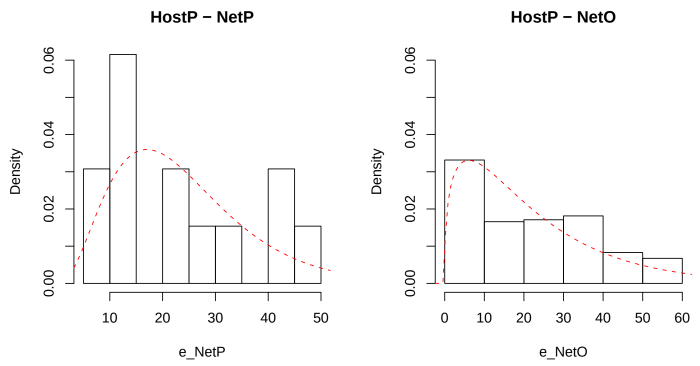


FIGURE 5.14: Histograms vs. estimated density (red dashes) for both $\hat{f}_O$ and $\hat{f}_P$ (IDEVAL 1998)

respectively (standard errors on parameters are 0.7080, 0.0045 for f_P and 0.1288, 0.009 for f_O). From now on, the estimator of a given density f will be indicated as $\hat{f}$.

Figure 5.13 shows histograms vs. estimated density (red, dashed line) and quantile-quantile plots (Q-Q plots), for both $\hat{f}_O$ and $\hat{f}_P$. We recall that Q-Q plots are an intuitive graphical "tool" for comparing data distributions by plotting the quantile of the first distribution against the quantile of the other one.

Considering that the samples sizes of $\varepsilon_{(\cdot)}$ are around 40, Q-Q plots empirically confirms our intuition: in fact, $\hat{f}_O$ and $\hat{f}_P$ are both able to explain real data well, within inevitable but negligible estimation errors. Even if $\hat{f}_P$ and $\hat{f}_O$ are both Gamma-shaped, it must be noticed that they significantly differ in their parametrization; this is a very important result since it allows to set up a proper criterion to decide whether or not ε_{NetP} and ε_{NetO} are generated by the same phenomenon.

Given the above estimators, a more precise and robust hypotheses test can be now designed. The Test 5.8 and 5.9 can be mapped into two-sided *Kolmogorov-Smirnoff* (KS) tests [Pestman, 1998], achieving the same result in terms of decisions:

$$\begin{array}{ccc} H_0 & & H_1 \\ \varepsilon_{NetP} \sim f_P & vs. & \varepsilon_{NetP} \nsim f_P \end{array} \tag{5.10}$$

$$\varepsilon_{NetO} \sim f_O \quad vs. \quad \varepsilon_{NetO} \nsim f_O \tag{5.11}$$

where the symbol $\sim$ means "has the same distribution of". Since we do not know the real PDFs, estimators are used in their stead. We recall that the KS-test is a *non-parametric* test to compare a sample (or a PDF) against a PDF (or a sample) to check how much they differs from each other (or how much they fit). Such tests can be performed, for instance, with `ks.test()` (a `GNU R` native procedure): resulting p-values on IDEVAL 1999 are 0.83 and 0.03, respectively.

Noticeably, there is a significant statistical evidence to accept the null hypothesis of Test 5.10. It seems that the ML estimation is capable of correctly fitting a Gamma PDF for f_P (given ε_{NetP} samples), which double-checks our intuition about the distribution. The same does not hold for f_O: in fact, it cannot be correctly estimated, with a Gamma PDF, from ε_{NetO}. The low p-value for Test 5.11 confirms that the distribution of ε_{NetO} delays is completely different than the one of ε_{NetP}. Therefore, our criterion doest not only recognize noisy delay-based relationships among alerts stream *if they exists*; it is also capable of detecting if such a correlation does not hold.

We also tested our technique on alerts generated by our NIDS and HIDS running on IDEVAL 1998 (limiting our analysis to the first four days of the first week), in order to cross-validate the above results. We prepared and processed the data with the same procedures we described above for the 1999 dataset. Starting from almost the same proportion of host/net alerts against either `pascal` or other hosts, the ML-estimation has computed the two Gamma densities shown in Figure 5.14: f_P and f_O are approximated by Gamma$(3.5127, 0.1478)$ and Gamma$(1.3747, 0.0618)$, respectively (standard errors on estimated parameters are 1.3173, 0.0596 for f_P and 0.1265, 0.0068 for f_O). These parameter are very similar to the ones we estimated for the IDEVAL 1999 dataset. Furthermore, with p-values of 0.51 and 0.09, the two KS tests confirm the same statistical discrepancies we observed on the 1999 dataset.

The above numerical results show that, by interpreting alert streams as random processes, there are several (stochastic) dissimilarities between net-to-host delays belonging to the same net-host attack session, and net-to-host delays belonging to different sessions. Ex-

	IDEVAL 1998		IDEVAL 1999	
	α	β	α	β
$\hat{f_P}$	3.512 (1.317)	0.147 (0.059)	3.060 (0.708)	0.017 (0.004)
$\hat{f_O}$	1.374 (0.126)	0.061 (0.006)	1.630 (0.128)	0.010 (0.009)

Table 5.1: Estimated parameters (and their estimation standard error) for the Gamma PDFs on IDEVAL 1998 and 1999.

ploiting these dissimilarities, we may find out the correlation among streams in an unsupervised manner, without the need to predefine any parameter.

5.4 Concluding Remarks

In this chapter we first described and evaluated a technique which uses fuzzy sets and measures to fuse alerts reported by the anomaly detectors. After a brief framework description and precise problem statement, we analyzed previous literature about alert fusion (i.e., aggregation and correlation), and found that effective techniques have been proposed, but they are not really suitable for anomaly detection, because they require a-priori knowledge (e.g., attack names or division into classes) to perform well.

Our proposal defines simple, but robust criteria for computing the time distance between alerts in order to take into account uncertainty on both measurements and the threshold-distance sizing. In addition, we considered the implementation of a post-aggregation phase to remove non-aggregated alerts according to their belief, a value indicating how much the IDS believes the detected attack to be real. Moreover, we defined and used some simple metrics for the evaluation of alert fusion systems. In particular, we propose to plot both the DR and the FPR vs. the degree of output alert reduction vs. the size of the input alert stream.

We performed experiments for validating our proposal. To this end, we used two prototypes we previously developed: a host anomaly detector, that exploits the analysis of system calls arguments and behavior, and a network anomaly detector, based on unsupervised payload clustering and classification techniques that enables an effective outlier detection algorithm to flag anomalies. During our experiments, we were able to outline many shortcomings of the IDEVAL dataset (the only available IDS benchmark) when used for evaluating alert fusion systems. In addition to known anomalies in network and host data, IDEVAL is outdated both in terms of background traffic and attack scenarios.

Our experiments showed that the proposed fuzzy aggregation approach is able to decrease the FPR at the price of a small reduction of the DR (a necessary consequence). The approach defines the notion of "closeness" in time as the natural extension of the naive, crisp way; to this end, we rely both on fuzzy set theory and fuzzy measures to semantically ground the concept of "closeness". By definition, our method is robust because it takes into account major uncertainties on timestamps; this means the choice of window size is less sensitive to fluctuations in the network delays because of the smoothing allowed by the fuzziness of the window itself. Of course, if the delays are varying too much, a dynamic resizing is still necessary. The biggest challenge with our approach would be its extension to the correlation of distributed alerts: in the current state, our modeling is not complete, but can potentially be extended in such a way; being the lack of alert features the main difficult.

Secondly, we analyzed the use of of different types of statistical tests for the correlation of anomaly detection alerts, a problem which has little or no solutions available today. One of the few correlation proposals that can be applied to anomaly detection is the use of a GCT. After discussing a possible testing methodology, we observed that the IDEVAL datasets traditionally used for evaluation have various shortcomings, that we partially addressed by using the data for a simpler scenario of correlation, investigating only the link between a stream of host-based alerts for a specific host, and the corresponding stream of alerts from a network based detector.

We examined the usage of a GCT as proposed in earlier works, showing that it relies on the choice of non-obvious configuration parameters which significantly affect the final result. We also showed that one of these parameters (the order of the models) is absolutely critical, but cannot be uniquely estimated for a given system. Instead of the GCT, we proposed a simpler statistical model of alert generation, describing alert streams and timestamps as stochastic variables, and showed that statistical tests can be used to create a reasonable criterion for distinguishing correlated and non correlated streams. We proved that our criteria work well on the simplified correlation task we used for testing, without requiring complex configuration parameters.

These were exploratory works, and further investigations on longer sequences of data, as well as further refinements of the tests and the criteria we proposed, are surely needed. Another possible extension of this work is the investigation of how these criteria can be used to correlate anomaly and misuse-based alerts together, in order to bridge the gap between the existing paradigms of intrusion detection. In particular, another possible correlation criteria could be build upon the observation that a compromised host behaves differently (e.g., its response time increases).

Conclusions 6

The main question that we attempted to answer with our research work is, basically, *to what extent classic ID approaches can be adapted and integrated to mitigate todays' Internet threats*. Examples of such threats include attacks against web applications like SQL injections, client-side malware that force the browser to download viruses or connect to botnets, and so forth. Typically, these attempts are labeled as *malicious activities*. The short answer to the question is the following.

As long as the technologies (e.g., applications, protocols, devices) will prevent us to reach a sophistication level such that malicious activity is seamlessly camouflaged as normal activity, then ID techniques will constitute an effective countermeasure. This is because IDSs — in particular those that leverage anomaly-based techniques — are specifically designed to detect unexpected events in a computer infrastructure. The crucial point of anomaly-based techniques is they are conceived to be generic. In principle, they indeed make no difference between an alteration of a process' control flow caused by an attempt to interpret a crafted JavaScript code and one due to a buffer overflow being exploited. Thus, as long as the benign activity of a system is relatively simple to model, then ID techniques are the building block of choice to design effective protections.

Like many other researches, this work is meant to be a longer and more articulated answer to the aforementioned question. The concluding remarks of Chapter 3, 4, and 5, summarize the results of our contributions to mitigate the issues we identified in host and web IDSs, and alert correlation systems. A common line to the works presented in this thesis is the problem of false detections, which is certainly one of the most significant barriers to the wide adoption of anomaly-based systems. In fact, the tools that are available to the public already offer superb detection capabilities that can recognize all the known [1] threats and, in theory, are effective also against unknown malicious activities. However, a large share of the alerts fired by these tools are negligible; either because they regard threats that do not apply to the actual scenario (e.g., unsuccessful attacks or vulnerable software version mismatches) or because the recognized anomaly does not reflect an attack at all (e.g., a software is upgraded and a change is confused with an threat). False detections mean time spent by the security officers to investigate the possible causes of the attack, thus, false detections mean costs.

We demonstrated that most of the false detections, especially false positives, can be prevented by carefully designing the models of normal activity. For example, we have been able to suppress many false positives caused by too strict model constraints. In particular, we substituted the "crisp" checks performed by deterministic relations learned over system calls' arguments with "smoother" models (e.g., Gaussian distributions instead of simple ranges) that, as we shown in Section 3.3.1, preserve the detection capabilities and decrease the rate of false alerts.

We also have shown that another good portion of false positives can be avoided by solving training issues. In particular, our contributions on detection of attacks against web applications have identified that, in certain cases, training is responsible for more than 70% of the false positives. We proposed to solve this issue by dynamically updating models of benign activity while the system is running. Even though our solution may pose new, limited risks in some situations, it is capable of suppressing all the false detections due to incomplete training; and, given the low base-rate of attacks we pointed out in Section 2.4.2, the resulting system offers a good balance between protection and costs (due to false positives).

[1] Recall that a large portion of the systems proposed in the literature have been tested using Snort as a baseline, as we argue in Section 2.4

Another important, desirable feature of an IDS is its capability of recognizing logically-related alerts. Note that, however, this is nothing but a slightly more sophisticated alert reduction mechanism, which once again means decreasing the effort of the security officer. In fact, as we have shown in the last chapter of this work, alert aggregation techniques can be leveraged to reduce the amount of false detections, not just for compressing them into more compact reports. However, alert correlation is a very difficult task and many efforts have been proposed to address it. Our point is that the research on this topic is still very limited and no common directions can be identified as we highlighted in Section 2.5.

The main future directions of our research have been already delineated in each chapter's conclusions. In addition, there are a few ideas that are not mature enough and thus have not been developed yet, even though they were planned to be included in this work. Regarding client-side protection mechanisms for browsers, we are designing a Firefox extension to learn the structure of benign web pages in terms of objects of different types typically contained. Here, the term "structure" refers to both the *types* of objects (i.e., images, videos) and the *order* these objects are fetched with and from which domains. We believe that this is a minimal set of features that can be used to design very simple classification features to distinguish between legit and malicious pages. One of our goals is to detect pages that contain an unexpected "redirection pattern" due to `<iframe />`s leveraged by drive-by downloads.

Regarding server-side protection techniques we are currently investigating the possibility of mapping some features of an HTTP request, such as the order of the parameters or the characteristics of their content, with the resulting SQL query, if any. If such a mapping is found, many of the existing anomaly detection techniques used to protect the DBs can be coupled with web-based IDS to offer a double layer protection mechanism. This idea can be leveraged to distinguishing between high-risk requests, which permanently modify a resource, and low-risk ones, which only temporarily affect a web page.

Bibliography

H. Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.

Jesse Alpert and Nissan Hajaj. We knew the web was big... Available online at `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`, Jul 2008.

J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, J. P. Anderson Co., Ft. Washington, Pennsylvania, April 1980.

Thomas Augustine, Us Naval Academy, Ronald C. Dodge, and Us Military Academy. Cyber defense exercise: Meeting learning objectives thru competition, 2006.

S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Chalmers University of Technology, Dept. of Computer Engineering, G
"oteborg, Sweden, March 2000a.

Stefan Axelsson. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security*, 3(3):186–205, August 2000b.

Hal Berghel. Hiding data, forensics, and anti-forensics. *Communications ACM*, 50(4):15–20, 2007. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1232743.1232761.

S. Bhatkar, A. Chaturvedi, and R. Sekar. Dataflow anomaly detection. *Security and Privacy, 2006 IEEE Symposium on*, page 15, May 2006. ISSN 1081-6011. doi: 10.1109/SP.2006.12.

Damiano Bolzoni, Sandro Etalle, Pieter H. Hartel, and Emmanuele Zambon. Poseidon: a 2-tier anomaly-based network intrusion detection system. In *IWIA*, pages 144–156. IEEE Computer Society, 2006. ISBN 0-7695-2564-4.

Damiano Bolzoni, Bruno Crispo, and Sandro Etalle. Atlantides: an architecture for alert verification in network intrusion detection systems. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 1–12, Berkeley, CA, USA, 2007. USENIX Association. ISBN 978-1-59327-152-7.

Mariusz Burdach. In-memory forensics tools. available online at `http://forensic.seccure.net/`, 2009.

J. B. D. Cabrera, L. Lewis, and R.K. Mehara. Detection and classification of intrusion and faults using sequences of system calls. *ACM SIGMOD Record*, 30(4), 2001.

Sanghyun Cho and Sungdeok Cha. Sad: web session anomaly detection based on parameter estimation. In *Computers & Security*, volume 23, pages 312–319, 2004. doi: DOI:10.1016/j.cose.2004.01.006.

Privacy Rights Clearinghouse. A chronology of data breaches. Technical report, Privacy Rights Clearinghouse, July 2009.

William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, Jul 1995. Morgan Kaufmann. ISBN 1-55860-377-8.

W.W. Cohen, P. Ravikumar, and S.E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, 2003.

Core Security Technologies. CORE Impact. `http://www.coresecurity.com/?module=ContentMod&action=item&id=32`, 2009.

Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. Vigilante: end-to-end containment of internet worms. *SIGOPS Oper. Syst. Rev.*, 39(5):133–147, 2005. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1095809.1095824.

Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, and Angelos D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. *Security and Privacy, IEEE Symposium on*, 0:81–95, 2008a. doi: http://doi.ieeecomputersociety.org/10.1109/SP.2008.11.

Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, and Angelos D. Keromytis. Casting out Demons: Sanitizing Training Data for Anomaly Sensors. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy (S&P 2008)*, pages 81–95, Oakland, CA, USA, May 2008b. IEEE Computer Society.

Claudio Criscione, Federico Maggi, Guido Salvaneschi, and Stefano Zanero. Integrated detection of attacks against browsers, web applications and databases. In *European Conference on Computer Network Defence - EC2ND 2009*, 2009.

Frédéric Cuppens and Alexandre Miage. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 202, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1543-6.

O. Dain and R. Cunningham. Fusing heterogeneous alert streams into scenarios. In *Proceedings of the ACM Workshop on Data Mining for Security Applications*, pages 1–13, Nov 2001.

H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Comput. Networks*, 31(8):805–822, 1999.

H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion-detection systems. *Annals of Telecommunications*, 55(7):361–378, 2000.

H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format. Technical report, France Telecom and Guardian and TNT, Mar. 2006.

Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pages 85–103, London, UK, 2001. Springer-Verlag. ISBN 3-540-42702-3.

Dorothy E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/TSE.1987.232894.

S. Eckmann, G. Vigna, and R. Kemmerer. STATL: An attack language for state-based intrusion detection. In *Proceedings of the ACM Workshop on Intrusion Detection*, Athens, Nov. 2000.

AK Elmagarmid, PG Ipeirotis, and VS Verykios. Duplicate Record Detection: A Survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.

Facebook. Statistics. Available online at `http://www.facebook.com/press/info.php?statistics`, 2009.

Ferruh Mavituna. SQL Injection Cheat Sheet. `http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/`, June 2009.

Christof Fetzer and Martin Suesskraut. Switchblade: enforcing dynamic personalized system call models. In *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 273–286, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-013-5. doi: http://doi.acm.org/10.1145/1352592.1352621.

Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symp. on Security and Privacy*, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7417-2.

J.C. Foster and V. Liu. Catch me if you can... In *Blackhat Briefings 2005*, Las Vegas, NV, August 2005. URL `http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-foster-liu-update.pdf`.

Vanessa Frias-Martinez, Salvatore J. Stolfo, and Angelos D. Keromytis. Behavior-Profile Clustering for False Alert Reduction in Anomaly Detection Sensors. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2008)*, Anaheim, CA, USA, December 2008.

Alessandro Frossi, Federico Maggi, Gian Luigi Rizzo, and Stefano Zanero. Selecting and Improving System Call Models for Anomaly Detection. In Ulrich Flegel and Michael Meier, editors, *DIMVA*, Lecture Notes in Computer Science. Springer, 2009.

Simson Garfinkel. Anti-Forensics: Techniques, Detection and Countermeasures. In *Proceedings of the 2nd International Conference on i-Warfare and Security (ICIW)*, pages 8–9, 2007.

SL Garfinkel and A. Shelat. Remembrance of data passed: a study of disk sanitization practices. *Security & Privacy Magazine, IEEE*, 1(1):17–27, 2003.

M. Geiger. Evaluating Commercial Counter-Forensic Tools. In *Proceedings of the 5th Annual Digital Forensic Research Workshop*, 2005.

Jonathon T. Giffin, David Dagon, Somesh Jha, Wenke Lee, and Barton P. Miller. Environment-sensitive intrusion detection. In *Proceedings of the Recent Advances in Intrusion Detection (RAID)*, pages 185–206, 2005.

Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

Sarah Granger. Social engineering fundamentals, part i: Hacker tactics. Available online at `http://www.securityfocus.com/infocus/1527`, Dec 2001.

Grugq. The art of defiling: defeating forensic analysis. In *Blackhat briefings 2005*, Las Vegas, NV, August 2005. URL `http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-grugq.pdf`.

P. Haccou and E. Meelis. *Statistical analysis of behavioural data. An approach based on timestructured models*. Oxford university press, 1992.

Joshua Haines, Dorene Kewley Ryder, Laura Tinnel, and Stephen Taylor. Validation of sensor alert correlators. *IEEE Security and Privacy*, 01(1):46–56, 2003. ISSN 1540-7993.

J. Han and M. Kamber. *Data Mining: concepts and techniques*. Morgan-Kauffman, 2000.

Ryan Harris. Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. In *Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06)*, volume 3 of *Digital Investigation*, pages 44–49, September 2006. URL `http://www.sciencedirect.com/science/article/B7CW4-4KCPVBY-4/2/185856560fb7d3b59798e99c909f7754`.

Harry. Exploit for CVE-2007-1719. available online at `http://www.milw0rm.com/exploits/3578`, 2007.

S. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *J. of Computer Security*, 6:151–180, 1998.

Thorsten Holz. A short visit to the bot zoo. *IEEE Security & Privacy*, 3(3):76–79, 2005.

Kenneth L. Ingham, Anil Somayaji, John Burge, and Stephanie Forrest. Learning dfa representations of http for protecting web applications. *Computer Networks*, 51(5):1239–1255, April 2007. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2006.09.016.

S. Jha, K. Tan, and R. A. Maxion. Markov chains, classifiers, and intrusion detection. In *CSFW '01: Proc. of the 14th IEEE Workshop on Computer Security Foundations*, page 206, Washington, DC, USA, 2001. IEEE Computer Society.

DN Joanes and CA Gill. Comparing Measures of Sample Skewness and Kurtosis. *The Statistician*, 47(1):183–189, 1998.

N.F. Johnson and S. Jajodia. Exploring steganography: Seeing the unseen. *COMPUTER*, 31(2):26–34, 1998.

Wen-Hua Ju and Y. Vardi. A hybrid high-order Markov chain model for computer intrusion detection. *Journal of Computational and Graphical Statistics*, 10:277–295, 2001.

B.H. Juang and LR Rabiner. A probabilistic distance measure for hidden Markov models. *AT&T Bell Laboratories Technical Journal*, 64(2):391–408, 1985.

Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *KDD '02: Proceedings of the eighth ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 366–375, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-567-X.

M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1–2):81–93, June 1938.

George J. Klir and Tina A. Folger. *Fuzzy sets, uncertainty, and information*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987. ISBN 0-13-345984-5.

Calvin Ko, George Fink, and Karl Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proc. of the 10th Ann. Computer Security Applications Conf.*, volume XIII, pages 134–144. IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.

T. Kohonen. *Self Organizing Maps*. Springer-Verlag, 2000.

Teuvo Kohonen and Panu Somervuo. Self-organizing maps of symbol strings. *Neurocomputing*, 21(1-3):19–30, 1998. URL `http://www.sciencedirect.com/science/article/B6V10-3V7S70G-3/2/1439043d6f41db0afa22e46c6f2b809a`.

J.Z. Kolter and M.A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.

C. Kruegel, D. Mutz, F. Valeur, and G. Vigna. On the detection of anomalous system call arguments. In *Proceedings of the 2003 European Symp. on Research in Computer Security*, Gjøvik, Norway, Oct. 2003a.

Christopher Kruegel and William Robertson. Alert Verification: Determining the Success of Intrusion Attempts. In *Proceedings of the Workshop on the Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2004)*, Dortmund, North Rhine-Westphalia, GR, July 2004.

Christopher Kruegel, Thomas Toth, and Engin Kirda. Service-Specific Anomaly Detection for Network Intrusion Detection. In

Proceedings of the Symposium on Applied Computing (SAC 2002), Spain, March 2002.

Christopher Kruegel, Darren Mutz, William Robertson, and Fredrik Valeur. Bayesian Event Classification for Intrusion Detection. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, Las Vegas, NV, USA, December 2003b.

Christopher Kruegel, William Robertson, and Giovanni Vigna. A Multi-model Approach to the Detection of Web-based Attacks. *Journal of Computer Networks*, 48(5):717–738, July 2005.

Christopher Kruegel, Darren Mutz, William Robertson, Fredrick Valeur, and Giovanni Vigna. LibAnomaly (website). available online at `http://www.cs.ucsb.edu/~seclab/projects/libanomaly/index.html`, 2009.

K. Labib and R. Vemuri. NSOM: A real-time network-based intrusion detection system using self-organizing maps. Technical report, Department of Applied Science, University of California, Davis, 2002.

Sin Yeung Lee, Wai Lup Low, and Pei Yuen Wong. Learning fingerprints for a database intrusion detection system. In *ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security*, pages 264–280, London, UK, 2002. Springer-Verlag. ISBN 3-540-44345-2.

Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symp.*, San Antonio, TX, 1998.

Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4):227–261, 2000. ISSN 1094-9224. doi: http://doi.acm.org/10.1145/382912.382914.

Wenke Lee and Dong Xiang. Information-Theoretic Measures for Anomaly Detection. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P 2001)*, pages 130–143, Oakland, CA, USA, May 2001. IEEE Computer Society.

Richard Lippmann, Robert K. Cunningham, David J. Fried, Isaac Graf, Kris R. Kendall, Seth E. Webster, and Marc A. Zissman. Results of the DARPA 1998 offline intrusion detection evaluation. In *Recent Advances in Intrusion Detection*, 1999.

Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Comput. Networks*, 34(4):579–595, 2000. ISSN 1389-1286.

R.J.A. Little and D.B. Rubin. *Statistical analysis with missing data*. Wiley New York, 1987.

Rune B. Lyngsø, Christian N. S. Pedersen, and Henrik Nielsen. Metrics and similarity measures for hidden markov models. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 178–186. AAAI Press, 1999. ISBN 1-57735-083-9.

F. Maggi, S. Zanero, and V. Iozzo. Seeing the invisible - forensic uses of anomaly detection and machine learning. *ACM Operating Systems Review*, April 2008.

Federico Maggi and Stefano Zanero. On the use of different statistical tests for alert correlation. In Christopher Kruegel, Richard Lippmann, and Andrew Clark, editors, *RAID*, volume 4637 of *Lecture Notes in Computer Science*, pages 167–177. Springer, 2007. ISBN 978-3-540-74319-4.

Federico Maggi, Matteo Matteucci, and Stefano Zanero. Detecting intrusions through system call sequence and argument analysis (preprint). *IEEE Transactions on Dependable and Secure Computing*, 99(1), 2009a. ISSN 1545-5971. doi: http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.69.

Federico Maggi, Matteo Matteucci, and Stefano Zanero. Reducing False Positives In Anomaly Detectors Through Fuzzy Alert Aggregation. *Information Fusion*, 2009b.

Federico Maggi, William Robertson, Christopher Kruegel, and Giovanni Vigna. Protecting a moving target: Addressing web application concept drift. In Engin Kirda and Davide Balzarotti, editors, *RAID*, Lecture Notes in Computer Science. Springer, 2009c.

M. V. Mahoney. Network traffic anomaly detection based on packet bytes. In *Proceedings of the 19th Annual ACM Symposium on Applied Computing*, 2003.

M. V. Mahoney and P. K. Chan. An analysis of the 1999 DARPA / Lincoln laboratory evaluation data for network anomaly detection. In *Proceedings of the 6th International Symp. on Recent Advances in Intrusion Detection (RAID 2003)*, pages 220–237, Pittsburgh, PA, USA, Sept. 2003a.

Matthew V. Mahoney and Philip K. Chan. Learning rules for anomaly detection of hostile network traffic. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, page 601, 2003b. ISBN 0-7695-1978-4.

M.V. Mahoney and P.K. Chan. Detecting novel attacks by identifying anomalous network packet headers. Technical Report CS-2001-2, Florida Institute of Technology, 2001.

John McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed By Lincoln Laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, November 2000. ISSN 1094-9224. doi: http://doi.acm.org/10.1145/382912.382923.

N. Merhav, M. Gutman, and J. Ziv. On the estimation of the order of a Markov chain and universal data compression. *IEEE Trans. Inform. Theory*, 35:1014–1019, Sep 1989.

Metasploit. Mafia: Metasploit anti-forensics investigation arsenal. available online at `http://www.metasploit.org/research/projects/antiforensics/`, 2009.

C. C. Michael and Anup Ghosh. Simple, state-based approaches to program-based anomaly detection. *ACM Transactions on Information and System Security*, 5(3):203–237, 2002. ISSN 1094-9224.

David L. Mills. Network time protocol (version 3). Technical report, University of Delaware, 1992.

Miniwatts Marketing Grp. World Internet Usage Statistics. `http://www.internetworldstats.com/stats.htm`, January 2009.

Kevin Mitnick. *The art of deception*. Wiley, 2002.

George M. Mohay, Alison Anderson, Byron Collie, Rodney D. McKemmish, and Olivier de Vel. *Computer and Intrusion Forensics*. Artech House, Inc., Norwood, MA, USA, 2003. ISBN 1580533698.

Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. M2d2: A formal data model for IDS alert correlation. In Wespi et al. [2002], pages 115–127. ISBN 978-3-540-00020-4.

D. Mutz, F. Valeur, C. Kruegel, and G. Vigna. Anomalous System Call Detection. *ACM Transactions on Information and System Security*, 9(1):61–93, February 2006.

Darren Mutz, William Robertson, Giovanni Vigna, and Richard A. Kemmerer. Exploiting execution context for the detection of anomalous system calls. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2007)*, Gold Coast, Queensland, AU, September 2007. Springer.

National Vulnerability Database. Exploit for CVE-2007-1719. available online at `http://nvd.nist.gov/nvd.cfm?cvename=CVE-2007-1719`, 2007a.

National Vulnerability Database. Exploit for CVE-2007-3641. available online at `http://nvd.nist.gov/nvd.cfm?cvename=CVE-2007-3641`, 2007b.

J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Network and Distributed System Security Symposium (NDSS)*. Citeseer, 2005.

Jr. Nick L. Petroni, AAron Walters, Timothy Fraser, and William A. Arbaugh. Fatkit: A framework for the extraction and analysis of digital forensic data from volatile system memory. *Digital Investigation*, 3(4):197–210, december 2006. URL `http://www.sciencedirect.com/science/article/B7CW4-4MD9G8V-1/2/7de54623f0dc5e1ec1306bfc96686085`.

Peng Ning, Yun Cui, Douglas S. Reeves, and Dingbang Xu. Techniques and tools for analyzing intrusion alerts. *ACM Trans. Inf.*

Syst. Secur., 7(2):274–318, 2004. ISSN 1094-9224. doi: http://doi.acm.org/10.1145/996943.996947.

Ofer Shezaf and Jeremiah Grossman and Robert Auger. Web Hacking Incidents Database. `http://www.xiom.com/whid-about`, January 2009.

Dirk Ourston, Sara Matzner, William Stump, and Bryan Hopkins. Applications of Hidden Markov Models to detecting multi-stage network attacks. In *Proceedings of the 36th Annual Hawaiian International Conference on System Sciences*, page 334, 2003.

Wiebe R. Pestman. *Mathematical Statistics: An Introduction*. Walter de Gruyter, 1998.

S. Piper, M. Davis, G. Manes, and S. Shenoi. *Detecting Hidden Data in Ext2/Ext3 File Systems*, volume 194 of *IFIP International Federation for Information Processing*, chapter 20, pages 245–256. Springer, Boston, 2006.

Pluf and Ripe. Advanced antiforensics self. available online at `http://www.phrack.org/issues.html?issue=63&id=11&mode=txt`, 2005.

Mark Pollitt. Computer forensics: an approach to evidence in cyberspace. In *Proceedings of the National Information Systems Security Conference*, volume II, pages 487–491, Baltimore, Maryland, 1995.

Phillip A. Porras, Martin W. Fong, and Alfonso Valdes. A mission-impact-based approach to infosec alarm correlation. In Wespi et al. [2002], page 35. ISBN 978-3-540-00020-4.

Georgios Portokalidis and Herbert Bos. Sweetbait: Zero-hour worm detection and containment using low- and high-interaction honeypots. *Comput. Netw.*, 51(5):1256–1274, 2007. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2006.09.005.

Georgios Portokalidis, Asia Slowinska, and Herbert Bos. Argos: an emulator for fingerprinting zero-day attacks. In *Proc. ACM SIGOPS EUROSYS'2006*, Leuven, Belgium, April 2006.

Bruce Potter. The Shmoo Group Capture the CTF project. `http://cctf.shmoo.com`, 2006.

Nicholas Puketza, Mandy Chung, Ronald A. Olsson, and Biswanath Mukherjee. A software platform for testing intrusion detection systems. *IEEE Software*, 14(5):43–51, 1997. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/52.605930.

Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, and Ronald A. Olsson. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering*, 22(10):719–729, 1996. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/32.544350.

Xinzhou Qin and Wenke Lee. Statistical causality analysis of infosec alert data. In *RAID*, pages 73–93, 2003.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.

M. Ramadas. Detecting anomalous network traffic with self-organizing maps. In *Recent Advances in Intrusion Detection 6th International Symposium, RAID 2003, Pittsburgh, PA, USA, September 8-10, 2003, Proceedings*, Mar 2003.

Marcus J. Ranum. The six dumbest ideas in computer security. `http://www.ranum.com/security/computer_security/editorials/dumb/`, Sept. 2005.

S. Ring and E. Cole. Volatile Memory Computer Forensics to Detect Kernel Level Compromise. In *Proceedings of the 6th International Conference on Information And Communications Security (ICICS 2004)*, Malaga, Spain, October 2004. Springer.

Ivan Ristic. mod_security: Open Source Web Application Firewall. http://www.modsecurity.org/, June 2008.

Robert Hansen. XSS (Cross Site Scripting) Cheat Sheet. `http://ha.ckers.org/xss.html`, June 2009.

William Robertson. webanomaly - Web Application Anomaly Detection. `http://webanomaly.googlecode.com`, 2009.

William Robertson, Federico Maggi, Christopher Kruegel, and Giovanni Vigna. Effective anomaly detection with scarce training data. In *Annual Network & Distributed System Security Symposium*, 2009.

Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of USENIX LISA 99*, 1999.

Martin Roesch. Snort - Lightweight Intrusion Detection for Networks (website). `http://www.snort.org`, 2009.

Hettich S. and Bay S. D. KDD Cup '99 Dataset. `http://kdd.ics.uci.edu/`, 1999.

Benjamin Sangster. `http://www.itoc.usma.edu/research/dataset/index.html`, April 2009.

Benjamin Sangster, T. J. O'Connor, Thomas Cook, Robert Fanelli, Erik Dean, William J. Adams, Chris Morrell, and Gregory Conti. Toward instrumenting network warfare competitions to generate labeled datasets. In *USENIX Security's Workshop*. USENIX, August 2009.

Bradley Schatz. Bodysnatcher: Towards reliable volatile memory acquisition by software. In *Proceedings of the 7th Annual Digital Forensic Research Workshop (DFRWS '07)*, volume 4 of *Digital Investigation*, pages 126–134, September 2007. URL `http://www.sciencedirect.com/science/article/B7CW4-4P06CJD-3/2/081292b9131eca32c02ec2f5599bb807`.

J.C. Schlimmer and R.H. Granger. Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, volume 1, pages 502–507, 1986.

Secunia. Secunia's 2008 annual report. Available online at `http://secunia.com/gfx/Secunia2008Report.pdf`, 2008.

R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P 2001)*, pages 144–155, Oakland, CA, USA, May 2001. IEEE Computer Society. ISBN 0-7695-1046-9.

R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *CCS '02: Proceedings* *of the 9th ACM Conference on Computer and communications security*, pages 265–274, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-612-9.

Monirul I. Sharif, Kapil Singh, Jonathon T. Giffin, and Wenke Lee. Understanding precision in host based intrusion detection. In *RAID*, pages 21–41, 2007.

Edward Hance Shortliffe. *Computer-based medical consultations: MYCIN*. Elsevier, 1976.

Adam Singer. Social media, web 2.0 and internet stats. Available online at `http://thefuturebuzz.com/2009/01/12/social-media-web-20-internet-numbers-stats/`, Jan 2009.

Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 4–4, Berkeley, CA, USA, 2004. USENIX Association.

Anil Somayaji and Stephanie Forrest. Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symp.*, Denver, CO, Aug. 2000.

Panu J. Somervuo. Online algorithm for the self-organizing map of symbol strings. *Neural Netw.*, 17(8-9):1231–1239, 2004. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2004.08.004.

Y. Song, SJ Stolfo, and AD Keromytis. Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic. In *Proc of the 16th Annual Network and Distributed System Security Symposium (NDSS)*, 2009.

Sourcefire. Snort Rules. `http://www.snort.org/snort-rules`, 2009.

A. Stolcke and S. Omohundro. Hidden Markov Model Induction by Bayesian Model Merging. *Advances in Neural Information Processing Systems*, pages 11–11, 1993a.

A. Stolcke and S. Omohundro. Inducing Probabilistic Grammars by Bayesian Model Merging. *Lecture Notes in Computer Science*, pages 106–106, 1994a.

Andreas Stolcke and Stephen Omohundro. Hidden Markov Model induction by bayesian model merging. In *Advances in Neural Information Processing Systems*, volume 5, pages 11–18. Morgan Kaufmann, 1993b.

Andreas Stolcke and Stephen M. Omohundro. Inducing probabilistic grammars by bayesian model merging. In *Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications*, pages 106–118, London, UK, 1994b. Springer-Verlag. ISBN 3-540-58473-0.

Andreas Stolcke and Stephen M. Omohundro. Best-first Model Merging for Hidden Markov Model Induction. Technical Report TR-94-003, ICSI, Berkeley, CA, USA, 1994c.

Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilber t, Mar tin Szydlowski andRichard Kemmerer, and Christopher Kruegel andGiovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *CCS 2009*, Chicago, November 2009. ACM.

Alon Swartz. The exploittree repository. `http://www.securityforest.com/wiki/index.php/Category:ExploitTree`, 2009.

G. Tandon and P. Chan. Learning rules from system call arguments and sequences for anomaly detection. In *ICDM Workshop on Data Mining for Computer Security (DMSEC)*, pages 20–29, 2003.

Steven J. Templeton and Karl Levitt. A requires/provides model for computer attacks. In *NSPW '00: Proceedings of the 2000 workshop on New security paradigms*, pages 31–38, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-260-3. doi: http://doi.acm.org/10.1145/366173.366187.

The SANS Institute. The twenty most critical internet security vulnerabilities. `http://www.sans.org/top20/`, Nov. 2005.

Walter N. Thurman and Mark E. Fisher. Chickens, eggs, and causality, or which came first? *Am. J. of Agricultural Economics*, 1998.

Dean Turner, Marc Fossi, Eric Johnson, Trevor Mark, Joseph Blackbird, Stephen Entwise, Mo King Low, David McKinney, and Candid Wueest. Symantec Global Internet Security Threat Report

– Trends for 2008. Technical Report XIV, Symantec Corporation, April 2009.

Alfonso Valdes and Keith Skinner. Probabilistic Alert Correlation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, pages 54–68, London, UK, 2001. Springer-Verlag. ISBN 3-540-42702-3.

F. Valeur, G. Vigna, C. Kruegel, and R.A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, 2004. ISSN 1545-5971. doi: http://dx.doi.org/10.1109/TDSC.2004.21.

WN Venables and B.D. Ripley. *Modern Applied Statistics with S*. Springer, 2002.

Giovanni Vigna and Richard A. Kemmerer. NetSTAT: A Network-based Intrusion Detection System. *Journal of Computer Security*, 7 (1):37–71, 1999. ISSN 0926-227X.

Giovanni Vigna, William Robertson, and Davide Balzarotti. Testing Network-based Intrusion Detection Systems Using Mutant Exploits. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS 2005)*, Washington DC, USA, October 2004. ACM.

Jouni Viinikka, Hervé Debar, Ludovic Mé, and Renaud Séguier. Time series modeling for ids alert management. In *ASIACCS '06: Proceedings of the 2006 ACM Symp. on Information, computer and communications security*, pages 102–113, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-272-0.

David Wagner and Drew Dean. Intrusion detection via static analysis. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P 2001)*, pages 156–168, Oakland, CA, USA, 2001. IEEE Computer Society. doi: 10.1109/SECPRI.2001.924296.

Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*. Springer-Verlag, September 2004.

Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, Seattle, WA, USA, September 2005. Springer-Verlag.

Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2006)*, Hamburg, GR, September 2006. Springer-Verlag.

Zhenyuan Wang and George J. Klir. *Fuzzy Measure Theory*. Kluwer Academic Publishers, Norwell, MA, USA, 1993. ISBN 0306442604.

Robert Watson. Openbsm. `http://www.openbsm.org`, 2006.

Robert N. M. Watson and Whayne Salamon. The FreeBSD audit system. In *UKUUG LISA Conference*, Durham, UK, Mar. 2006.

Andreas Wespi, Giovanni Vigna, and Luca Deri, editors. *Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002 Zurich, Switzerland, October 16–18, 2002 Proceedings*, volume 2516 of *Lecture Notes in Computer Science*, 2002. Springer. ISBN 978-3-540-00020-4.

R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

K. Yamanishi, J.-I. Takeuchi, G. J. Williams, and P. Milne. Online unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Knowledge Discovery and Data Mining*, 8 (3):275–300, 2004.

Dit-Yan Yeung and Yuxin Ding. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition*, 36: 229–243, Jan. 2003.

Robert H'obbes' Zakon. Hobbes' internet timeline v8.2. Available online at `http://www.zakon.org/robert/internet/timeline/`, Nov 2006.

S. Zanero. Improving self organizing map performance for network intrusion detection. In *SDM 2005 Workshop on "Clustering High Dimensional Data and its Applications"*, 2005a.

Stefano Zanero. Behavioral intrusion detection. In Cevdet Aykanat, Tugrul Dayar, and Ibrahim Korpeoglu, editors, *19th Int'l Symp. on Computer and Information Sciences - ISCIS 2004*, volume 3280 of *Lecture Notes in Computer Science*, pages 657–666, Kemer-Antalya, Turkey, Oct. 2004. Springer. ISBN 3-540-23526-4.

Stefano Zanero. Analyzing tcp traffic patterns using self organizing maps. In Fabio Roli and Sergio Vitulano, editors, *Proceedings 13th International Conference on Image Analysis and Processing - ICIAP 2005*, volume 3617 of *Lecture Notes in Computer Science*, pages 83–90, Cagliari, Italy, Sept. 2005b. Springer. ISBN 3-540-28869-4.

Stefano Zanero and Sergio M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 412–419. ACM Press, 2004. ISBN 1-58113-812-1.

Index

List of Acronyms

AIC Akaike Information Criterion 191

ARMAX Auto Regressive Moving Average eXogenous 46

AR Auto Regressive 46

ARR Alert Reduction Rate 184

ANSI American National Standard Institute 63

ASCII American Standard for Information Interxchange 27

BIC Bayesian Information Criterion 81

BMU Best Matching Unit 94

BSM Basic Security Module 25

CDF Cumulative Density Function 84

CDX Cyber Defense eXercise 54

CPU Central Processing Unit 25

CTF Capture The Flag 49

DARPA Defense Advanced Research Projects Agency 90

DB DataBase 14

DNS Domain Name System 29

DOM Document Object Model 43

DoS	Denial of Service	5
DR	Detection Rate	19
ELF	Executable Linux Format	109
FN	False Negative	19
FNR	False Negative Rate	96
FPR	False Positive Rate	19
FP	False Positive	20
FSA	Finite State Automaton	37
FTP	File Transfer Protocol	14
GCI	Granger Causality Index	46
GCT	Granger Causality Test	46
HIDS	Host-based Intrusion Detection System	25
HMM	Hidden Markov Model	37
HTML	HyperText Markup Language	119
HTTP	HyperText Transfer Protocol	6
ICD	Idealized Character Distribution	118
IDEVAL	Intrusion Detection eVALuation	29
IDMEF	Intrusion Detection Message Exchange Format	18
IDS	Intrusion Detection System	7
ID	Intrusion Detection	8
IETF	Internet Engineering Task Force	18
ISP	Internet Service Provider	7
IP	Internet Protocol	28
IRC	Internet Relay Chat	61

JSON	JavaScript Object Notation . . . 119
KS	Kolmogorov-Smirnoff . . . 199
LERAD	Learning Rules for Anomaly Detection . . . 31
MDL	Minimum Description Length . . . 81
ML	Maximum Likelihood . . . 198
NIDS	Network-based Intrusion Detection System . . . 25
NTP	Network Time Protocol . . . 182
PC	Program Counter . . . 39
PDF	Probability Density Function . . . 197
PHP	PHP Hypertext Preprocessor . . . 155
PID	Process IDentifier . . . 39
ROC	Receiving Operating Characteristic
SOM	Self Organizing Map . . . 29
SQL	Structured Query Language . . . 6
S^{2}A^{2}DE	Syscall Sequence Arguments Anomaly Detection Engine 63
SSH	Secure SHell . . . 14
SVN	SubVersioN . . . 155
TCP	Trasmission Control Protocol . . . 26
TN	True Negative . . . 19
TP	True Positive . . . 18
TTL	Time To Live . . . 29
ULISSE	Unsupervised Learning IDS with 2-Stages Engine . . . 31
URL	Uniform Resource Locator . . . 1

VPN	Virtual Private Network	14
XML	eXtensible Markup Language	87
XSS	Cross-Site Scripting	13

List of Figures

List of Tables

Colophon

This document was typeset using the XeTeX typesetting system created by the Non-Roman Script Initiative and the memoir class created by Peter Wilson. The body text is set 10pt with Adobe Caslon Pro. Other fonts include `Envy Code R`, Optima Regular and. Most of the drawings are typeset using the TikZ/PGF packages by Till Tantau.