Revisiting Anomaly-based Network Intrusion Detection Systems

Damiano Bolzoni

Revisiting Anomaly-based Network Intrusion Detection Systems

Damiano Bolzoni

Composition of the Graduation Committee:

Prof. dr. S. Etalle	Universiteit Twente (promotor)
Prof. dr. P.H. Hartel	Universiteit Twente (promotor)
Prof. dr. ir. B.R.M.H. Haverkort	Universiteit Twente
Prof. dr. S.J. Mullender	Universiteit Twente
Prof. dr. L.V. Mancini	Università La Sapienza di Roma
Dr. ir. H.J. Bos	Vrije Universiteit Amsterdam
Dr. F. Piessens	Katholieke Universiteit Leuven
Dr. ir. R.N.J. Veldhuis	Universiteit Twente

Distributed and Embedded Security Group
P.O. Box 217, 7500 AE, Enschede, The Netherlands.

This research is supported by the research program Sentinels of STW (http://www.sentinels.nl), under the project number 06679.

CTIT PhD Thesis Series Number 09-147
Centre for Telematics and Information Technology (CTIT)
P.O. Box 217-7500 AE Enschede-The Netherlands.

IPA: 2009-14
The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithms).

ISBN: 978-90-365-2853-5
ISSN: 1381-3617
DOI: 10.3990/1.9789036528535

Cover design: Damiano Bolzoni and Nienke Timmer.
Printed by Wöhrmann Print Service.

Revisiting Anomaly-based Network Intrusion Detection Systems

DISSERTATION

to obtain
the doctor's degree at the University of Twente
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday, 25th of June 2009 at 13.15

by

Damiano Bolzoni

born on 19th of January 1981,
in Torino, Italy

The dissertation is approved by:

Prof. dr. S. Etalle	Universiteit Twente (promotor)
Prof. dr. P.H. Hartel	Universiteit Twente (promotor)

Ambition:
aspire to climb as high as you can dream

Abstract

Intrusion detection systems (IDSs) are well-known and widely-deployed security tools to detect cyber-attacks and malicious activities in computer systems and networks.

A signature-based IDS works similar to anti-virus software. It employs a signature database of known attacks, and a successful match with current input raises an alert. A signature-based IDS cannot detect unknown attacks, either because the database is out of date or because no signature is available yet.

To overcome this limitation, researchers have been developing anomaly-based IDSs. An anomaly-based IDS works by building a model of *normal* data/usage patterns during a training phase, then it compares new inputs to the model (using a similarity metric). A significant deviation is marked as an anomaly. An anomaly-based IDS is able to detect previously unknown, or modifications of well-known, attacks as soon as they take place (i.e., so called zero-day attacks) and targeted attacks.

Cyber-attacks and breaches of information security appear to be increasing in frequency and impact. Signature-based IDSs are likely to miss an increasingly number of attack attempts, as cyber-attacks diversify. Thus, one would expect a large number of anomaly-based IDSs to have been deployed to detect the newest disruptive attacks. However, most IDSs in use today are still signature-based, and few anomaly-based IDSs have been deployed in production environments.

Up to now a signature-based IDS has been easier to implement and simpler to configure and maintain than an anomaly-based IDS, i.e., it is easier and less expensive to use. We see in these limitations the main reason why anomaly-based systems have not been widely deployed, despite research that has been conducted for more than a decade.

To address these limitations we have developed SilentDefense, a comprehensive anomaly-based intrusion detection architecture that outperforms competitors not only in terms of attack detection and false alert rates, but it reduces the user

effort as well. Several integrated components constitute the architecture of SilentDefense: each component can work independently, but they can be plugged into several configurations to offer diverse (automated) facilities to users, thus reducing user effort. In particular, SilentDefense:

- improves the well-known detection algorithm PAYL (for the HTTP protocol, from 90% up to 100% detection rate and from 0,17% down to 0,0016% false alert rate) by adding a neural network that pre-processes network traffic;
- reduces the number of false positive alerts (between 50% and 100% fewer alerts) by correlating alerts generated by an intrusion detection system (be it signature- or anomaly-based) monitoring the incoming traffic with a content-based analysis of the outgoing traffic;
- automatically generates regular expressions to validate incoming HTTP requests, that users can edit to tune the anomaly-based detection engine;
- automates the classification of anomaly-based alerts by extracting the payload of previous alerts which can be classified using both standard and user-defined taxonomies.

SilentDefense is the first systematic attempt to develop an anomaly-based intrusion detection system with a high degree of usability. However, SilentDefense (and in general anomaly detection) is not an alternative to signature-based systems. In fact, we believe that the best chance to detect an attack is provided by the combination of the two approaches. A signature-based system works better for well-known applications and systems (e.g., the Windows operating systems), while SilentDefense can detect zero-day and targeted attacks. The latter attack type usually targets custom-developed software, such as web applications developed by corporations for their internal users or proprietary systems used in critical infrastructures.

Acknowledgements

In the past four years I have met many people, and had a lot of fun. It is difficult to summarize everything in a couple of pages, but I will try my best!

The most important thing I have learnt in 15 years of running is that even the fastest runner of all time needs a skilled coach to become a "champion". During my PhD, I was lucky enough to have two very skilled coaches. As my science training is far from being completed, I hope to learn more from them.

Sandro has been my daily supervisor, however he has been much more than that. He is a genuine friend, and a superlative teacher. When I first met him in Milan in February 2005 I understood he was not a common researcher, and that very first feeling has been proven right in the following years. What still puzzles me about Sandro is his ability to grasp new ideas and concepts so quickly from scratch (ranging from intrusion detection to risk management). He did so well that Emmanuele and I thought he could be a perfect business partner... ;)

I have been writing with Pieter my very first scientific paper and some crucial parts of this thesis. It took me quite a while to understand how he wants things to be written down and organized. However, I think we have been managing this task nicely. I truly thank Pieter for having improved my well known "writing laziness", and forced me to provide stronger motivations to explain crude experimental numbers (and I am pretty sure that is the main reason for our latest success).

I have known Emmanuele for 13 years. We met in high school, attended the same university, worked together first in KPMG and now here at the University of Twente. Emmanuele is brilliant, and quite often when one of us comes out with a new idea, the other one is already thinking how to improve it and refine it (even until 6am :) Our passion brought us to many different places around the world (literally). In Rome, when we attended the "Young IT experts" meeting (our first IT event) whilst still in high school, London, San Francisco, and Las Vegas. Despite the fact we are always busy with something, we manage to have a

lot of fun and I hope we will keep going on in the same way! (mai ipse dixit fu più azzeccato..."a giugno professore!" :)

Emmanuele and I have been dreaming about starting up our own business since we were university students. It took us several years to have all of the ingredients, and we could not have done this without help and support from Sandro. SecurityMatters is the next big challenge for the three of us, and no matter what will happen next, we already proved to be a great team.

Some very important people I had the pleasure to meet: Andreas, Anna, Daniel, Dimitrios, the DIES staff (including Marlous, Nicole and Thelma), i ragazzi del DSI, Dulce, Emma, Enrico (who kindly accepted to be a paranimf), il mio osteopata Fabio, Filippo e Francesca (e la piccola Violante), tutti i ragazzi (e non) del campo di San Giuliano, tutti i ragazzi (e non) dell'ufficio KPMG di Treviso (in particolare Rudy e Marco), Laura, Lianne, Luca, the Macandra residents, diversi Marco, Mirko e Sonia, Nadine, Nienke (thanks for the Dutch abstract), many Olgas, il Prof (mio Maestro nella corsa), Sasha, Simone, Stefano, i miei amici di Schio e dintorni, my Russian friends in A'dam and R'dam (Dina, Ira, Lena, Sonya, Tanya), the AC Tion guys, Uros, Wouter, and last but not least Zlatko.

And now, the most important acknowledgement.

Cara Mamma,

mi hai scritto diverse lettere struggenti nel corso di questi anni (Carlotta le definisce, come ben sai, "da libro Cuore"). Non ho mai risposto a quelle lettere, perchè non sapevo trovare le parole per consolare il tuo dolore. Se ho deciso di venire in Olanda non è stato per sfuggire a qualcosa, o qualcuno. Ciò a cui io aspiro non è possibile in Italia, questo lo sai bene anche tu, e se ne avessi l'opportunità rifarei ogni cosa dal principio.

C'è comunque qualcosa che rimpiango: avrei voluto poterti aiutare di più durante la malattia di papà. Hai sacrificato te stessa per permettere a me di continuare a sognare: tra tutti i regali che si possono ricevere dai propri genitori, questo è senz'altro il più grande.

Non potrò mai saldare il mio debito con te: il dottorato è quanto ho da offrirti in cambio di tutti i sacrifici che hai fatto e dei dispiaceri che hai sopportato. Per quanto i kilometri possano separarci, non possono cambiare i sentimenti. Un affettuoso abbraccio va anche a mia nonna Anna e a mia sorella Carlotta, ed un pensiero a mio Papà che oggi non può essere qui con noi.

Enschede, June 2009.

Contents

1 Introduction **1**

1.1 Motivation 3

1.1.1 Running a signature-based IDS 4

1.1.2 Running an anomaly-based IDS 4

1.2 Research Question 6

1.3 Thesis Overview 9

1.4 Conclusion and Outlook 11

2 Taxonomy of Intrusion Detection Systems **13**

2.1 Host- or Network-based Systems 15

2.1.1 Host-based Systems 15

2.1.2 Network-based Systems 15

2.1.3 Honeypots 16

2.2 Signature- or Anomaly-based Systems 17

2.2.1 Signature-based Systems 17

2.2.2 Anomaly-based Systems 18

2.3 State-of-the-art of Anomaly-based Intrusion Detection Systems . . 19

2.3.1 Classification of Anomaly-based Network Intrusion Detection Systems 20

2.3.2 Payload- vs Header-based Approaches 21

2.3.3 Building the Model 25

2.3.4 Similarity Metric 26

2.4 Conclusion 26

3 POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System 29
3.1 Architecture . . . 30
3.1.1 SOM Classification Model . . . 30
3.1.2 PAYL Classification Model . . . 32
3.1.3 POSEIDON . . . 34
3.2 Tuning and Benchmarks . . . 35
3.2.1 Benchmarks . . . 36
3.3 Related Work . . . 40
3.4 Conclusion . . . 42

4 ATLANTIDES: an Architecture for Alert Verification in Network Intrusion Detection Systems 43
4.1 Preliminaries . . . 45
4.1.1 The Base-rate Fallacy . . . 45
4.1.2 False Positives in Signature-based Systems . . . 46
4.1.3 False Positives in Anomaly-based Systems . . . 47
4.2 Architecture . . . 47
4.2.1 The Output Anomaly Detector . . . 49
4.3 Benchmarks . . . 50
4.3.1 Tests with a Private Data Set . . . 50
4.3.2 Tests with the DARPA 1999 Data Set . . . 51
4.4 Related Work . . . 53
4.5 Conclusion . . . 56

5 Boosting Web Intrusion Detection Systems by Inferring Regular Languages 57
5.1 Detecting Data-flow Attacks to Web Applications . . . 59
5.1.1 Exploiting Regularities . . . 59
5.1.2 Regular and Irregular Parameters . . . 60
5.2 Sphinx Detection Engine . . . 60
5.2.1 Building the Model . . . 61
5.2.2 The Regular-text Methodology . . . 62
5.2.3 The Raw-data Methodology . . . 66
5.2.4 Using the Model . . . 66
5.3 Benchmarks . . . 67
5.3.1 Comparative Benchmarks . . . 69

5.3.2 Testing the Regular-expression Engine 69
5.3.3 Testing Sphinx on the Complete Input 71
5.4 Signature Generation for Signature-based Systems 72
5.5 Related Work 74
5.6 Conclusion 75

6 Panacea: Automating the Classification of Attacks for Anomaly-based Network Intrusion Detection Systems 77
6.1 Architecture 79
6.1.1 Alert Information Extractor 79
6.1.2 Attack Classification Engine 82
6.1.3 Implementation 86
6.2 Benchmarks 86
6.2.1 Tests with DS_A 89
6.2.2 Tests with DS_B 90
6.2.3 Tests with DS_C 91
6.2.4 Summary of Benchmark Results 92
6.2.5 Usability in Panacea 93
6.3 Related Work 94
6.4 Conclusion 95

7 Concluding Remarks 97

Chapter 1

Introduction

At the dawn of the computer age, computer systems were quite simple. A basic input output system allowed a single user to perform a sequence of tasks. No real computer security mechanism was in place, as a single computer was almost as big as a room: security was provided by keys and locks on building doors. The technological advances in hardware and software computer technology first, and the growth of the Internet second, revolutionized the computer (and the human) world. Nowadays, computer systems have evolved to multi-user systems that allow many tasks to run concurrently, and they fit a palm of a hand. The Internet connects computer systems all around the world and users can access diverse online services, ranging from e-banking to on-line communities. Computers have penetrated our life in depth, and in general they have positively affected our life style. However, there are negative sides as well.

Computers store and carry all sorts of personal and confidential data, e.g., credit card numbers, medical records, etc. The same applies to corporations, who handle vital information for their business through computer networks, e.g., confidential industrial process data. Thus, digital information is an asset of increasing value and the safeguarding of computer systems and of the data they contain has long become a critical issue for modern society. The information processed by our computers and carried by our networks is also of great interest to modern criminals, who are still interested in bank robberies, extorting money, fraud, and stealing secrets for profit. The criminals are diverting their attention to the digital world.

As computers got small enough to fit on a desk, and operating systems became more complex, computer architects began to include the computer equivalent of locks and keys in their systems. The first security mechanisms put in place to allow only authorized users to operate were user accounts and passwords. Be-

cause malicious users can bypass such a simple line of defence, a new concept made its way. Similar to what happens in the real world, there was a need for digital watchers, to monitor system operations and alert on suspicious events. An Intrusion Detection System (IDS) is the computer system equivalent of a burglar alarm: this concept was introduced by Anderson [14] in 1980 and later formalized by Denning [45] in her seminal work. At the very beginning, an IDS was just used to monitor system logs (e.g., a user logging in at midnight, when nobody is supposed to be working, is suspicious).

The need to allow different users to operate and access diverse resources on the same system required security mechanisms to evolve. Today, three heterogeneous technologies currently work in combination: authentication, authorization and auditing (the "gold standard", Lampson [75]). Authentication is the process of veryfing the identity of a party who requires access to a certain resource, by means of some credentials she can show (e.g., a password). Authorization determines (through access control mechanisms and policies) how parties can interact, which actions parties are allowed to perform and which data parties are allowed to access or modify. Usually, a party is first requested to authenticate to interact with another. During the auditing process, activity evidences are collected and analysed to discover security violations and diagnose their causes. Sandhu and Samarati [108] make the link between auditing and intrusion detection explicit, as they call the latter a case of on-line auditing. Meanwhile the IDS has evolved as well, to monitor system internals more in depth (e.g., syscalls) and to cope with network monitoring.

The universal use of the Internet has made it more difficult to achieve a high security level. Nowadays, systems work in distributed environments and they are typically built upon multiple heterogeneous technologies. The gold standard fails to protect modern systems. Attackers target web applications instead of Telnet ports, and they write automatic scanners to discover exploitable network services. For instance, think of web applications released by different sources, ranging from professional to hobbyist programmers: an error in the user authentication mechanism can expose the whole application to attacks. Lampson argues that this is due to the unwillingness of people to pay for the direct and indirect costs of achieving a high level of security [75] . Secure programming and configuration set up, user inconvenience and obsolete features all lead to increased costs. As a result, it is difficult to ensure software quality and resilience.

Cyber-attacks and breaches of information security appear to be increasing in frequency and impact (see the Internet Storm Center [134] for weekly and monthly single attack rates). Few observers are willing to ignore the possibility that future attacks could have much more severe consequences than what has been observed

to date. Hence, a more powerful second line of defence is needed more than ever, to patrol modern computer networks.

Despite the fact that intrusion detection has been an active topic of research for the last decade to enhance attack detection, current IDSs have hardly increased their effectiveness over the years, and most advances in intrusion detection have remained within the academic domain.

1.1 Motivation

The general idea behind an IDS is that if someone voluntarily attempts to tamper with a system, she will alter some activities/parameters of the system itself. The outcomes of her activities are supposed to deviate from the normal behaviour of the system. Ideally, an IDS is devised to detect and report such anomalies. An IDS can be classified according to several features, e.g., the kind of data its engine analyses (host/network data) and the way it detects anomalies (signature or anomaly-based).

A signature-based IDS works similar to anti-virus software. It employs a signature database of well-known attacks, and a successful match with current input raises an alert. Signatures generally target widely used applications or systems for which security vulnerabilities are widely advertised. Similarly to anti-virus software, which fails to identify unknown viruses (either because the database is out of date or because no signature is available yet), a signature-based IDS fails to detect unknown attacks.

To overcome this limitation, researchers have been developing anomaly-based IDSs. An anomaly-based IDS works by building a model of *normal* data/usage patterns, then it compares (using a similarity metric) the current input with the model. A significant difference is marked as an anomaly. The main feature of an anomaly-based IDS is its the ability to detect previously unknown (or modifications of well-known) attacks as soon as they take place. An anomaly-based IDS can also adapt to custom-developed systems/applications, which are widely deployed, and for which it is not cost effective to maintain a set of signatures.

As cyber-attacks diversify, signature-based IDSs are likely to miss an increasingly large part of attack attempts. Thus, one would expect a large number of anomaly-based IDSs to have been deployed to detect the newest disruptive attacks. However, most IDSs in use today are signature-based, and few anomaly-based IDSs have been deployed in production environments. The reason for this is that – also according to Kruegel and Toth [70] – a signature-based IDS is easier to implement and simpler to configure and maintain than an anomaly-based IDS, i.e., it is easier and less expensive to use.

Usability is a broad concept, and the first idea that comes to mind is usually a user-friendly graphical interface. Although usability has a lot to do with interface design – to quote Nielsen [93] – it is not a single, one dimensional property. The key notion for us is that software shows a higher degree of usability than its competitors if it accomplishes a certain task 1) better and/or 2) faster. We will now show how a signature-based IDS can be easier to run than an anomaly-based IDS by considering their basic tasks: data analysis, alert verification and attack response (Kahn et al. [64]).

1.1.1 Running a signature-based IDS

At deployment time, a signature-based IDS requires little work to set up. It can be deployed almost with an out-of-the-box configuration. Users can perform a thorough selection of needed signatures, a task known as *tuning*, to avoid future false alerts: e.g., a signature for the IIS web server is useless when only Apache-based installations are in use. Should an alert turn out to be false or unrelated with the monitored environment, the IT security team can deactivate the signature to avoid further time-wasting analysis. Tuning is performed once, and then updated from time to time. Thanks to tuning, a signature-based IDS generally generates a low rate of false alerts. Users can write custom signatures as well, to detect certain events and patterns, or refine existing rules to improve detection, thereby altering the behaviour of the detection engine. Users can manipulate the IDS engine as they wish.

A signature-based IDS raises classified alerts (e.g., buffer overflow or SQL Injection), and this classification is assigned "off-line" during the development of the signature. The importance of classification is threefold. First, security teams can prioritize alerts without having to inspect them. Second, security teams deploy automatic defensive countermeasures to react to certain disruptive attacks as soon as they take place (e.g., dropping network traffic when a buffer overflow is detected, or changing some rules in firewall systems). Third, alerts can be correlated with each other, and with other system logs, e.g., firewall logs, (1) to detect multi-step attacks [97], i.e., attacks that require the attacker to execute several actions to achieve her goal, and (2) to reduce false alerts by identifying root causes [63]. Thus, determining the attack class helps these tasks.

1.1.2 Running an anomaly-based IDS

The deployment of an anomaly-based IDS typically requires expert personnel. Several parameters need to be configured, such as the duration of the training phase or the similarity metric. Every environment being different, guidelines are

hard to give. Each anomaly-based IDS is also different from the others (while all signature-based IDSs work in a similar manner). Users gain little knowledge from subsequent installations; hence deployment tasks are likely to be trial-and-error processes. Users mainly criticize three aspects of the management of current anomaly-based IDSs [106], each of which increases the user effort needed to run the IDS. Namely:

1. An anomaly-based IDS generally raises a high number of false alerts.
2. An anomaly-based IDS usually works as a black-box.
3. An anomaly-based IDS raises alerts without an exact classification.

False alerts Because of its statistical nature, an anomaly-based IDS is bound to raise an higher number of false alerts than a signature-based IDS. As a matter of fact, the classification of a certain input for a signature-based IDS is predetermined (for malicious inputs), while the classification of inputs for an anomaly-based IDS depends on the training data. Thus, different anomaly-based model instances could classify the same input differently.

False alerting is a well-known problem of IDSs in general [101], and anomaly-based IDSs in particular. Security teams need to verify each raised alert, thus an ineffective system (i.e., a system prone to raise a high number of false alerts) will require more personnel for its management. Two distinct statistical phenomena affect the rate of false alerts.

First, most anomaly-based detection engines employ statistical models, a distance function, and a threshold value to detect anomalies. There is an intrinsic direct link between attacks detected and false alerts raised [114]. When adjusting the threshold value to detect a larger number of attacks, the number of false alerts increases as well. It is impossible to achieve 100% attack detection and 0% false alert rates at the same time, and users have to balance these opposite phenomena by setting an appropriate threshold value.

Secondly, Axelsson [17] demonstrates that since intrusions are rare events, and because a detection engine cannot achieve both a 100% detection rate and a 0% false alert rate, a greater rate of false alerts will be generated than the expected rate. This problem is commonly known as the *base-rate fallacy*, and it stems directly from Bayes' theorem. We provide a more detailed explanation of the base-rate fallacy problem in Section 4.1.1.

Black-box approach An anomaly-based IDS carries out detection as a black-box. Users have little control: quite often, they can adjust only the similarity met-

ric used to discern licit traffic from malicious activities. Because most anomaly-based IDSs employ complex mathematical models (think of neural networks), users can neither precisely understand how the IDS engine discerns normal input nor refine the IDS model to avoid certain false alerts or to improve attack detection. Users need to spend a good deal of time to understand the inner working of the system, and they must be experts.

Lack of attack classification An anomaly-based IDS raises an alert every time its model differs from the current input. The cause of the anomaly itself is unknown to the IDS. It holds little information to determine the attack class (other than the targeted IP address/TCP port and the IP source of attack). Because the model employed by the detection engine is locally built during a certain time frame, each model is likely to be different. Hence, it is difficult to develop an off-line classifier suitable for *any* anomaly-based IDS instance. Because an anomaly-based IDS is supposed to detect unknown attacks, or slight modifications of well-known attacks, manual classification or the application of some heuristics are currently the only possible choices. However, manual classification is not feasible and the heuristics deliver results in a restricted context only (because the "traits" of each attack must be known before). Because alerts come unclassified, no automatic countermeasure can be activated to react to a certain threat.

Because of all the difficulties listed above for an anomaly-based IDS, a signature-based IDS is the obvious choice when users have to monitor complex systems, such as modern corporate networks, despite its inability to deal with zero day attacks. Users can accomplish tuning more easily to avoid false alerts (thereby saving overall time), they can write their own set of signatures to detect attacks and a signature-based IDS automatically performs alert classification. An anomaly-based IDS lacks these features because researchers have mainly focused on enhancing attack detection (Werlinger et al. [122]), and they have not considered usability. As a matter of fact, users could improve the usability of current anomaly-based IDSs by setting the similarity metric in such a way that the IDS generates fewer alerts (i.e., recall the base-rate fallacy). However, the detection rate would be (negatively) affected as well, thereby reducing the only advantage an anomaly-based IDS has over a signature-based IDS.

1.2 Research Question

Based on the analysis above of the problems of an anomaly-based IDS, this work focuses on answering the following practical question:

"How can we extend current anomaly- network-based IDSs to improve their usability, yet delivering an effective IDS?"

We focus on network-based IDSs because they are widely deployed and they can monitor several different platforms at the same time (see Section 2.1 for a detailed overview). Although anomaly- host-based IDSs present similar issues, they depend more on the internals of an operating system (think of how Linux and Windows handle system calls differently). Thus, a certain enhancement could only be applicable to few host-based IDSs.

Having described three main short comings in anomaly-based intrusion detection in Section 1.1, we have identified the following sub-questions to address:

1. Can we improve the false alert rate of an anomaly-based IDS in an automatic way?

2. Can we allow users to control and adjust the behaviour of an anomaly-based IDS?

3. Can we automate the classification of attacks detected by an anomaly-based IDS?

To address these questions we have developed SilentDefense, a comprehensive anomaly-based intrusion detection architecture that outperforms competitors not only in terms of attack detection and false alert rates, but it reduces the user effort as well. Several integrated components constitute the architecture of SilentDefense: each component can work independently, but they can be plugged into several configurations to offer diverse (automated) facilities to users, thus reducing user effort. Here we provide a brief functional description of each component and the usability issue it addresses (see Section 1.3 for the main technical contributions). Figure 1.1 shows a possible configuration with all SilentDefense components present.

POSEIDON is the core component of SilentDefense, as it is employed also by ATLANTIDES and Sphinx. It has been originally designed to detect anomalies in computer networks and outperform leading competitors in terms of detection and false alert rates, thereby reducing the workload for users.

ATLANTIDES is the first system to date that can reduce the number of false alerts of any IDS (both signature- and anomaly-based) in an automatic way, thereby reducing the time spent to verify alerts. Although extensively researched, the problem of reducing the false alert rate (also called alert verification) still persists for anomaly-based IDSs. Previous systems are tailored towards signature-based IDSs only, as they require information about triggered signatures to work.

Sphinx is a web-based IDS that, though being anomaly-based, allows users to have tight control over the detection engine behaviour and the way it detects anomalies, in an intuitive way. Hence, users can (1) reduce the time for verification by enhancing the false alert rate (eliminating some normal traces that have been considered malicious) and (2) improve effectiveness by adding custom detection rules. The topic of helping users to change the way an anomaly-based IDS detection engine works has been briefly addressed by Robertson et al. [106]. The main problem is how to present to users the detection models in an easy and simple way for them to understand how to refine the model(s). Sphinx allows the users to have a tight control over the detection engine by presenting the detection models in the form of regular expressions.

Panacea is the first system to date that automates the classification of attacks detected by an anomaly-based IDS, thereby reducing the time users have to spend to classify raised alerts. A simple approach to solve this problem is to generate some heuristics to match an attack to some previous knowledge. Although effective, this approach is not scalable because it requires a good deal of manual labour, also to update the heuristics w.r.t. new attacks, and it strictly relies on the expertise of the analyst(s). Panacea provides an automatic way to build attack classifiers, but it also allows the user to define her own attack classification.

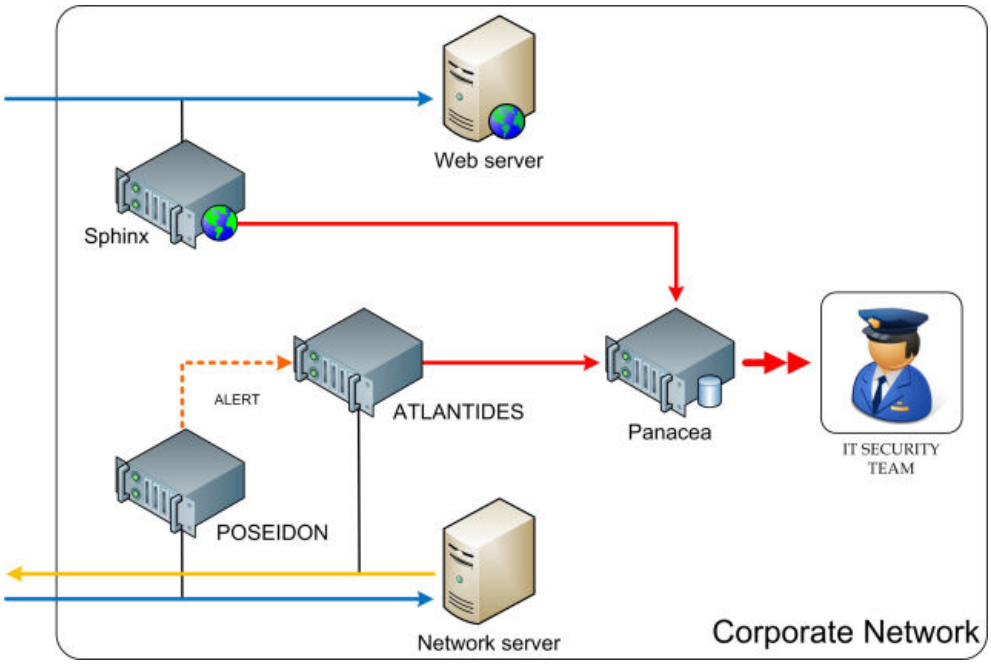


Figure 1.1: A possible configuration of SilentDefense components. See Section 1.3 for a technical description of each component.

1.3 Thesis Overview

We begin by describing the state of the art and a taxonomy of intrusion detection systems in Chapter 2. In Chapter 3 we present POSEIDON, our anomaly-based network intrusion detection system, and compare it to previous similar systems. POSEIDON is the core of several other systems we have developed. In Chapter 4 we introduce ATLANTIDES, a system to reduce false positives raised by any network intrusion detection system. In Chapter 5 we describe an intrusion detection system specifically tailored to detect attacks to web applications. We call this system Sphinx. In Chapter 6 we present a system (Panacea) to automatically classify alerts raised by an anomaly-based IDS.

We now elaborate further the contribution of each chapter in more detail and present some technical internals of SilentDefense. Figure 1.2 depicts the cross-component relationships of SilentDefense components.

Taxonomy of Intrusion Detection Systems (Chapter 2) We provide a taxonomy of intrusion detection systems and some definitions that we use through this work. We then report on the state of the art of network anomaly-based intrusion detection, which is the main topic of this work. We compare different anomaly-based detection engines, based on the data they analyse (e.g., network packet headers or connection meta-data) and the algorithm they use to detect anomalies (e.g., neural networks). We show how the different approaches can be more effective than others in detecting a certain attack. We conclude by showing how an anomaly-based IDS builds its detection model and how the similarity metric is used to detect anomalies. This work appears in a book chapter [7], which is joint work with S. Etalle.

POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System (Chapter 3) We present the architecture of POSEIDON, an anomaly- network-based IDS. POSEIDON combines a Self-organizing Map (a neural network) and a modified version of the Wang and Stolfo's PAYL algorithm [121] (based on n-gram analysis). This combination proves to enhance detection and false alert rates, and to be more effective than the original algorithm, which was, at that time, the leading anomaly-based IDS. We motivate the design choices for this architecture and detail its advantages over other solutions. This work appears in a refereed conference paper [4], which is joint work with E. Zambon, S. Etalle and P.H. Hartel.

ATLANTIDES: an Architecture for Alert Verification in Network Intrusion Detection Systems (Chapter 4) ATLANTIDES is a system designed to lower

the false alert rate of network intrusion detection systems, and it works in combination with both signature- and anomaly-based IDSs. ATLANTIDES processes the alerts raised by an IDS monitoring incoming data and analyses the outgoing data generated by the network service in response to the suspicious input. The analysis of the output is performed by an instance of POSEIDON, modified to monitor outgoing traffic rather than incoming. Should the output turn out to be anomalous, ATLANTIDES forwards the alert raised to the security team. The alert is otherwise dropped. ATLANTIDES works in a completely automatic manner, after a quick set up, and it does not require expert personnel for its operation, thereby improving the usability. Benchmarks prove that our system is effective in reducing false alerts without reducing true alerts. In Section 4.2.1 we also provide a quick method to automatically set the threshold value for POSEIDON. This work appears in a refereed conference paper [1], which is joint work with B. Crispo and S. Etalle.

Boosting Web Intrusion Detection Systems by Inferring Regular Languages (Chapter 5) Sphinx is an anomaly-based IDS tailored to monitor web application and web services in general. With Sphinx we introduce the concept of a "positive signature". Signatures employed by an IDS usually match attack traces. On the other hand, a positive signature describes the normal (expected) input of the web application parameters. Sphinx analyses the parameters and automatically infers a regular expression (i.e., the positive signature) to validate the content of each of them. Because a positive signature is in the form of a regular expression, it is easy for users to adjust *any* positive signature and make it more accurate, thereby changing the behaviour of the detection engine. For some input parameters, it is not possible to generate precise positive signatures (think of email messages or binary data). The content of those parameters is then processed by a modified instance of POSEIDON. Comparative benchmarks against other web-anomaly-based IDSs prove that our system is more effective in detecting attacks, with a lower false alert rate. This work appears in a refereed conference paper [2], which is joint work with S. Etalle.

Panacea: Automating the Classification of Attacks for Anomaly-based Network Intrusion Detection Systems(Chapter 6) We present a system to automate the classification of alerts raised by an anomaly-based IDS, Panacea. It works by analysing the payload of raised alerts from which it extracts some meta-information. The meta-information is then passed to a supervised machine learning algorithm that builds classification profiles to match attack classes. The machine learning algorithm requires to be trained, i.e., to observe a number of samples whose classification is known. The training can be fully automated or sup-

ported by an analyst. In the former case, Panacea receives alerts incoming from a signature-based IDS and extracts all required information. In the latter case, alert sources can be diverse, and the analyst provides the corresponding attack classification for a certain alert. Once the training is complete, the system switches to classification mode and processes incoming anomaly-based alerts to output an attack class. Panacea is heuristic-independent and supports user-defined attack classifications as well. This work appears in a refereed conference paper [3], which is joint work with S. Etalle and P.H. Hartel.

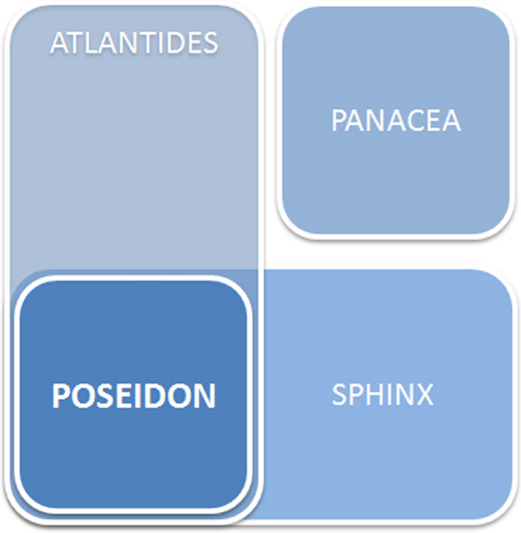


Figure 1.2: Cross-component relationships of SilentDefense components.

1.4 Conclusion and Outlook

After a decade of theoretical and practical research, anomaly detection techniques can be considered mature. Yet, anomaly-based IDSs failed to evolve from the research laboratories to real network environments. Although effective, they do not meet usability requirements and users find signature-based IDS more practical to use.

In this thesis we answer the main research question positively, by approaching the research problem from a different point of view than previous research. Our contributions do not only relate to well-known problems (i.e., reducing false alerts), but address important issues that have not been extensively explored before; in particular, we address sub-questions related to main usability issues by developing several tools to extend current anomaly-based IDSs.

Some of our tools (POSEIDON and Sphinx) address usability indirectly, by raising fewer false alerts than competitors, thus reducing the time users need to spend to manage the IDS. Other tools (ATLANTIDES and Panacea) have been specifically designed to automate various tasks of an anomaly-based IDS, that are usually run manually. The ultimate goal of our work is to close the gap with signature-based IDSs and eventually to allow users to combine these two different approaches to protect computer networks. Panacea and Sphinx are the most concrete effort to bridge anomaly- and signature-based IDSs.

Despite the number of issues we address, our extensions are of course not complete; there is considerable room for further improvement. We do not address the problem of training an anomaly-based IDS engine. The quality of data used to build the detection model is a crucial factor, which can influence both effectiveness and usability. A model with a poor quality is likely to flag a higher number of legal events as malicious than a good quality model, thus increasing the amount of time required to process alerts.

The topic of correlation of host- and network-based IDSs alerts has been already addressed in the past. However, research should address also how the simultaneous combination of the two approaches can improve the overall detection effectiveness and accuracy. Host-based IDSs usually impact on performance, thus they would need to run only when a possible threat is detected. A network-based IDS could activate the host-based IDS "on-demand".

Public data sets for IDS testing date back to 1999. Those data sets do not reflect neither the data transferred nowadays on the Internet nor the latest attack techniques. Researchers are forced to collect their own private data set to assess the effectiveness of their systems, but, because of privacy issues, such data sets are not usually disclosed publicly. As a result, it is difficult to asses the effectiveness of an IDS and to compare different systems under the same conditions. The development of a public, modern and faithful data set for IDS testing should be a primary goal for the IDS research community.

Chapter 2

Taxonomy of Intrusion Detection Systems*

In the multifaceted world of intrusion detection, systems have only a few underlying principles. In this chapter, first we introduce the definitions and a taxonomy for IDSs, specifically in terms of the data source (i.e., which data is analysed) and the detection model (i.e., the underlying algorithm) employed. Then we focus on anomaly-based network intrusion detection, which is the main topic of this work, and provide a detailed overview.

Let us first set the stage: we assume the presence of an application A (or system) that exchanges information over a network. An input is any finite string of characters and we say that S is the set of all possible inputs. Typically, A is not designed to deal with *any* input $i \in S$ but only with a subset of S, which we call I_N (the normal inputs, e.g., in the case of a web service I_N will consist of all valid user requests made to the web application). Similarly, we call $I_D \subset S$ the set containing all possible *dangerous* inputs (attacks, e.g. allowing an attacker to deviate the normal behaviour). We assume that $I_D \bigcap I_N = 0$ over the time (i.e., an input that was considered normal during, e.g., the working hours will not be considered dangerous during the weekend).

Definition 1 *An intrusion detection system monitors computer systems and networks to determine if a malicious event (i.e., an intrusion) has occurred. Each time a malicious event is detected, the IDS raises an alert.*

*This chapter is a major revision of the book chapter Approaches in Anomaly-based Network Intrusion Detection Systems, Intrusion Detection Systems, volume 38 of Advances in Information Security, pages 1 - 15, Springer, 2008.

Definition 2 *A true positive (TP) is a real alert, raised in response to an intrusion attempt.*

Definition 3 *A false positive (FP) is a false alert, raised in response to a non-malicious behaviour.*

Definition 4 *A true negative (TN) is the event when no alert is raised and no intrusion attempt takes place.*

Definition 5 *A false negative (FN) is the event when no alert is raised but a real intrusion attempt takes place.*

False positives, rather than false negatives, influence the overall user experience of an IDS (see Chapter 4 for a detailed discussion). Users are aware that some attack attempts will go unnoticed, but on the other hand a system that often raises false alerts is likely to be ignored after a while.

Definition 6 *The effectiveness of an IDS is determined by its* completeness *and its* accuracy *[42, 43].*

- completeness $= \#TP/(\#TP + \#FN)$
- accuracy $= \#TP/(\#TP + \#FP)$

Here, $\#TP$ is the number of true positives, $\#FN$ is the number of false negatives and $\#FP$ is the number of false positives raised during a given time frame. In other words, the completeness is the detection rate and the accuracy the false positive rate. In this work we will refer to the latter definitions rather than completeness and accuracy.

Definition 7 *A zero-day attack exploits a software vulnerability that has not been made public yet.*

Being unknown to security experts, zero-day attacks are unlikely to be detected by a present IDS (predominantly signature-based, see Section 2.3). On the other hand, cyber criminals can easily get access to zero-day attacks in the well-developed underground market. Often, even straightforward mutations of known attacks cannot be recognized either by a present IDS.

Definition 8 *A targeted attack is crafted to attack specific (often custom) systems/applications within a network.*

2.1 Host- or Network-based Systems

The first distinction we can draw to characterize an IDS is the main source of data used to feed its detection model. An IDS can be either host- or network-based. In this section, for both approaches, we present the definition, typical advantages and disadvantages of the approach and some examples of real systems. We focus mainly on network-based systems because they are the most popular nowadays.

2.1.1 Host-based Systems

Definition 9 *A host-based IDS (HIDS) monitors a single machine (or a single application) and audits data traced by the hosting operating system (or application).*

Typical examples of audited data are system calls (their parameters and their order), resource usage, and/or system logs.

HIDSs have been deployed first to attempt the detection of malicious or unauthorized events against computer systems: the first implementations were mainly designed to analyse system logs [14]. Nowadays, the host-based approach is less attractive than a decade ago. First, modern operating systems have grown in complexity, "driven" by the explosive growth of the Internet, thus it is more difficult to achieve an extensive monitoring. Secondly, system administrators are usually concerned about the impact of an HIDS on host performance. A notable HIDS (it is usually called a "web application firewall") is ModSecurity [128]. It is a module (i.e., a pluggable software component) for the Apache web server. ModSecurity intercepts incoming requests, runs the analysis and, in case a request is considered suspicious, can drop it, thereby preventing the request from being processed by the Apache instance.

2.1.2 Network-based Systems

Definition 10 *A network-based IDS (NIDS) monitors a network segment and analyse the traffic which flows through the segment.*

The NIDS detection engine can analyse different data: network streams (i.e., connection properties such as endpoints, bytes exchanged by peers, connection time) or network payloads.

A network-based intrusion detection system (NIDS) is considered an effective second line of defence against network-based attacks directed at computer systems and networks [18, 43], and – due to the increasing severity and likelihood of

such attacks from the Internet – NIDSs are employed in almost all large-scale IT infrastructures [11]. A typical example of NIDS is Snort [107, 143].

The main advantage of the NIDS approach is the possibility to monitor data and events without affecting host performance. On the other hand, the fact it is not host-based turns out to be one of the main disadvantages (especially for systems analysing the payload of network packets). For instance, a NIDS cannot function properly in combination with applications or application protocols which apply data encryption (e.g. SSH and SSL), unless the encryption key is provided. A possible solution to this makes use of a host-based component to access data after decryption, but this causes an overhead on the monitored host. This problem is going to grow in importance when IPv6 will gradually replace IPv4: in fact, one of the main design goals of IPv6 is the authentication and confidentiality of data (through cryptography).

Another common problem for a NIDS is the reconstruction of network traffic. Data streams are split into TCP segments and IP datagrams. In order to analyse the content, the system needs to reassemble the traffic into the original form. Modern networks operate at high speed (up to 10Gbs): while the traffic reconstruction would be theoretically possible for an arbitrarily powerful system, a NIDS faces performance and implementation constraints. First, the NIDS must save a significant amount of data for a "long" time, depending on system time-outs and data throughput (this is resource consuming). Secondly, operating systems implement heterogeneous network stacks and handle data reconstruction differently. Therefore the NIDS engine should implement some context-awareness functionalities. All of these limitations resulted in the so-called evasion and insertion attacks, formalised by Ptacek and Newsham [104]. Attackers craft communications to fool the NIDS, e.g., by *overwriting* inside NIDS memory some data previously sent or by forcing the NIDS to drop data (that has not been analysed yet) after sometime.

The EMERALD system [90] attempted to merge the advantages offered by both the HIDS and the NIDS approaches into a (virtually) single IDS. The problems of data normalization from different sources, event fusion and correlation and suitable metric definition are still open issues. These problems stopped the development of improvements after the first proof of concept of EMERALD.

2.1.3 Honeypots

A honeypot is a closely monitored computer system that is deliberately deployed to be attacked and compromised. Honeypots are not used by normal users, therefore they are hit only by malicious activities. Although such systems cannot be categorized as IDSs (since they passively wait for attackers to hit), honeypots

gather information that is not usually available to IDSs, e.g., by recording and tracking any activity performed by the attacker.

Honeypots can emulate either a fully working computer system (high-interaction) or only part(s) of it, such as the network stack (low-interaction). High-interaction honeypots (e.g., Sebek [146] and Argos [102]) are useful to capture and study exploits for vulnerabilities that are still unknown, as the attacker can gain full system control. Low-interaction honeypots (e.g., *honeyd* [103]) are more limited, but they are useful to learn about network probes or worm activity.

Honeypots are usually deployed in research networks to capture, e.g., malware instances and consequently update and upgrade anti-virus software and IDSs to cope with new threats.

2.2 Signature- or Anomaly-based Systems

The second classification one can use to categorize an IDS is based on the model it uses to detect attacks. As this feature is often considered the most important, we provide a detailed overview of both approaches, signature- and anomaly-based. In this section we provide a definition for each approach and show its main strengths and weaknesses.

2.2.1 Signature-based Systems

A signature-based IDS (SBS), e.g., Snort [107, 143], is based on pattern-matching techniques: the IDS contains a database of known-attack *signatures* (much like an anti-virus software) and tries to match these signatures with the analysed data. When a match is found, an alert is raised. Formally:

Definition 11 *A* signature-based *IDS is based on a model* $M_S \subseteq I_D$ *of* dangerous *inputs: if an input element* $i \in M_S$ *then the IDS raises an alert.*

This approach usually yields good results in terms of few false positives, but has drawbacks: first, in most systems *all* new (i.e., zero-day) or polymorphic attacks, where a change in the attack payload does not affect the attack effectiveness, will go unnoticed until the signature database is updated. The IDS is not likely to detect even slight modifications of a known attack. Thus, attackers have a window of opportunity to gain control of the system or application under attack. Although this limitation is considered acceptable for detecting attacks to, e.g., the OS, it makes an SBS less suitable for protecting a web-based service, because of the *ad hoc* and dynamic nature of web traffic. Secondly, an SBS needs to be updated regularly, increasing the IT personnel workload and required skills.

2.2.1.1 Developing Signatures

Developing a signature is a thorny task [100]. Once an attack is made public, experts first need to carefully analyse it. The attack could exploit either a well-known vulnerability or a new one. The signature should aim to detect the way the attack exploits a given vulnerability rather than the attack payload only: e.g., a buffer overflow can be exploited by different attack vectors, but those vectors will show a common (minimum) length. The reason for this is that, since it is easy for an attacker to modify the attack payload while not altering the attack effectiveness, there are more chances to detect attack variations with just one (or few) signature. Abstracting the attack is not always possible, and it usually requires a good deal of work: it is difficult to find the right balance between an overly specific signature (which is not able to detect a simple attack variation) and an overly general one (which will classify legitimate traffic as an attack attempt).

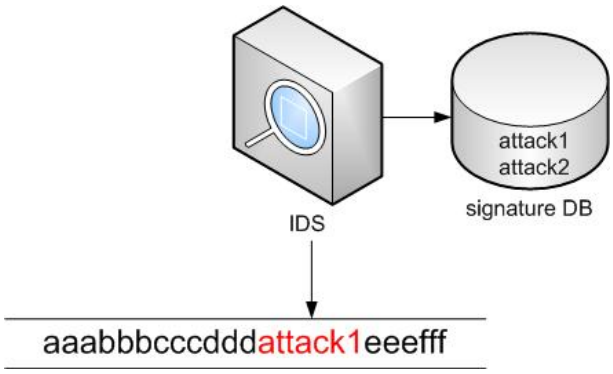


Figure 2.1: A signature-based IDS.

2.2.2 Anomaly-based Systems

An anomaly-based IDS (ABS) builds a statistical model which describes the normal behaviour of the monitored system/network. Intuitively, an ABS works by training itself to recognize acceptable behaviour and then raising an alert for any behaviour outside the boundaries of its training. Formally:

Definition 12 *An* anomaly-based *IDS makes use of a model* $M_A \subseteq I_N$ *of* normal *inputs: if* $i \notin M_A$ *then the IDS raises an alert.*

Typically, M_A is defined implicitly by using an abstract model M_{abs} and a similarity function $\phi(M_{abs}, i) \rightarrow \{yes, no\}$, to discern normal inputs from anomalous ones. An example of a similarity function is the distance d such that $\phi(M_{abs}, i) =$

$d(M_{abs}, i) < t$, where t is a threshold value . The model M_{abs} is built during a *training phase*: starting from a training set $T \subseteq I_N$ (a consistent dump of the application/system input) one builds M_{abs} (and – implicitly – M_A) in such a way that every input in T is included in M_A. For the model to reflect faithfully the "normal" activity, it is important that M_A does not contain malicious inputs (see Section 2.3.3 for a detailed discussion).

The main advantage of an ABS is that it can detect *zero-day* and polymorphic attacks: novel attacks can be detected as soon as they take place. Since they do not require any a-priori knowledge of the application/system, an ABS can protect better than an SBS *ad hoc* systems such as custom-developed applications (e.g., web applications, as also argued by Vigna [106]). Thus, an ABS is suitable to detect *targeted* attacks too. On the negative side, because of the statistical nature of its model, an ABS is bound to raise a number of false positives, and the value of the threshold actually determines a compromise between the number of false positives and the number of false negatives the IT security personnel is willing to accept [114]. An ABS is generally difficult to configure and use: as the detection model is usually a black-box, users have little control over the way the system detects attacks and on the false positive/negative rates.

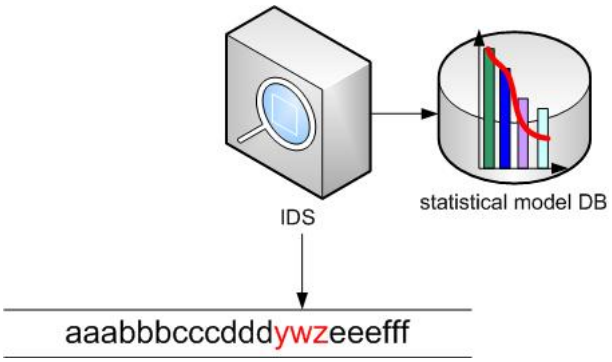


Figure 2.2: An anomaly-based IDS.

2.3 State-of-the-art of Anomaly-based Intrusion Detection Systems

Most IDSs in use today are network- and signature-based. System administrators prefer a signature-based NIDS (SNIDS) over an anomaly-based systems (ANIDS) because it is – according to Kruegel and Toth [70] – easier to implement and simpler to configure and maintain, despite the fact that a SNIDS could raise

a significant amount of false negatives. However, as new attacks are devised with increasing frequency every day (see the Internet Storm Center's web site [134] for weekly and monthly single attack rates), the ANIDS approach becomes increasingly attractive.

2.3.1 Classification of Anomaly-based Network Intrusion Detection Systems

An ANIDS can extract information to detect attacks from different sources: packet headers, packet payload or both. *Header information* is mainly useful to recognize attacks aiming at vulnerabilities of the network stack implementation or probing the operating system to identify active network services. On the other hand, *payload information* is most useful to identify attacks against vulnerable applications (since the connection that carries the attack is established in a normal way) [121]. Without pretending to be globally better than other types of ABSs, payload-based systems have importance of their own, as they are particularly suitable for detecting popular attacks such as those against the HTTP protocol, and worms (see Wang and Stolfo [119] and Costa et al. [35] for a discussion). An ANIDS can be classified according to:

(a) the underlying algorithm it uses;

(b) whether it analyses the features of each packet separately or of the whole connection, and how data are correlated;

(c) the kind of data it analyses. In particular, whether the ANIDS engine analyses either the packet headers or the packet payload.

Regarding the underlying algorithm, Debar et al. [42, 43] define four different possible approaches, but only two of them have been successfully employed in the last decade: algorithms based on statistical models and those based on neural networks. The former is the most widely used: according to Debar et al. [42] more than 50% of existing ANIDSs employ statistical models. An ANIDS based on neural networks works in a similar way, but instead of building a statistical model, it trains a neural network which is then in charge of recognizing regular traffic from anomalous one.

Concerning feature (b) the distinction one has to make is between *packet-oriented* and *connection-oriented* systems. A packet-oriented system uses a single packet as minimal information source, while a connection-oriented system considers features of the whole communication before establishing whether it is anomalous or not. Theoretically, a connection-oriented system could use as input

the content (payload) of a whole communication (allowing – at least in principle – a more precise analysis), but this would require a longer computational time, which could limit the throughput of the system by introducing extra latency time. In practice, a connection-oriented system typically takes into account the number of sent/received bytes, the duration of the connection and layer-4 protocol used. According to the Wang and Stolfo benchmarks [121], a payload-based system does not show a significant increase in performance when it also reconstructs the connection, instead of just considering the packets in isolation. In practice, most ANIDSs are packet-oriented (see also Table 2.1).

The last, more practically relevant distinction we can make is between *header-based* and *payload-based* systems. A header-based system considers only packet headers (layer-3 and, if present, layer 4 headers) to detect malicious activities; a payload-based system analyses the payload data carried by the layer-4 protocol; there are also hybrid systems which mix information gathered observing packet headers and (if present) layer-4 payload data. We will elaborate on this distinction in the rest of this section. Before we do so, we present a table reporting the most important systems; which have been benchmarked with public data sets (either DARPA 1998 [81] or DARPA 1999 [83] data sets, which contain a full dump of the packets, or the KDD 99 [23] data set, which contains only connection meta data extracted from the DARPA 1999 data set).

iSOM [92] uses a one-tier architecture, consisting of a Self-organizing Map [67], to detect two attacks in the 1999 DARPA data set: the first attack against the SMTP service and the other attack against the FTP service. IntelligentIDS [46] extracts information from the connection meta data once it has been reassembled. PHAD [84] combines 34 different values extracted from the packet headers. MADAM ID [79] extracts information from audit traffic and builds classification models (specifically designed for certain types of intrusion) using data mining techniques. SSAD [71] (Service Specific Anomaly Detection) combines different information such as type, length and payload distribution (computing character frequencies and aggregating then them into six groups) of the request. PAYL [121] and POSEIDON (see Chapter 3) detect anomalies only looking at the full payload. Table 2.1 summarizes the properties of the systems mentioned.

2.3.2 Payload- vs Header-based Approaches

We now elaborate on the differences in effectiveness between payload-based and header-based systems. We begin by showing some examples of attacks that can be detected by the systems of one kind, but not by all systems.

System	Detection Engine	Semantic Level	Analysed Data
iSOM	NN	PO + CO	Meta data
IntelligentIDS	NN	CO	Meta data
PHAD	S	PO	H
MADAM ID	S	CO	Meta data
SSAD	S	PO	H + P
PAYL	S	PO	P
POSEIDON	NN + S	PO	P

Table 2.1: The most important ANIDSs: NN stands for Neural Networks, S for Statistical model, PO is Packet-Oriented while CO is Connection-Oriented, H and P stand for Header and Payload respectively.

2.3.2.1 Attacks Detectable by Header-based Systems

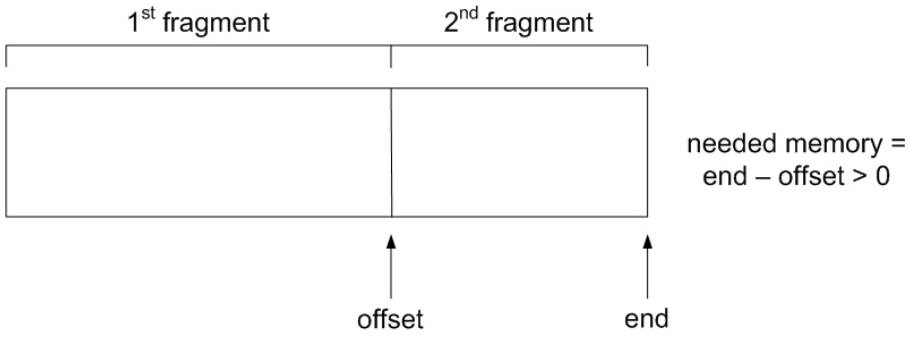
The attacks presented in this section are extracted from the DARPA 1999 data set. Although nowadays these attacks are not effective, they were considered serious threats at the time the data set was put together. iSOM, IntelligentIDS, PHAD and SSAD are likely to detect these attacks (in particular, the latter two systems).

The teardrop exploit [130] is a remote Denial of Service attack that exploits a flaw in the implementation of older TCP/IP stacks: such implementations do not handle properly IP fragments which overlap. Figure 2.3 shows how the attack takes place: the attacker sends fragmented packets forged so that they overlap each other when the receiving host tries to reassemble them. If the host does not check the boundaries properly, it will try to allocate a memory block with a negative size, causing a kernel panic and crashing the OS.

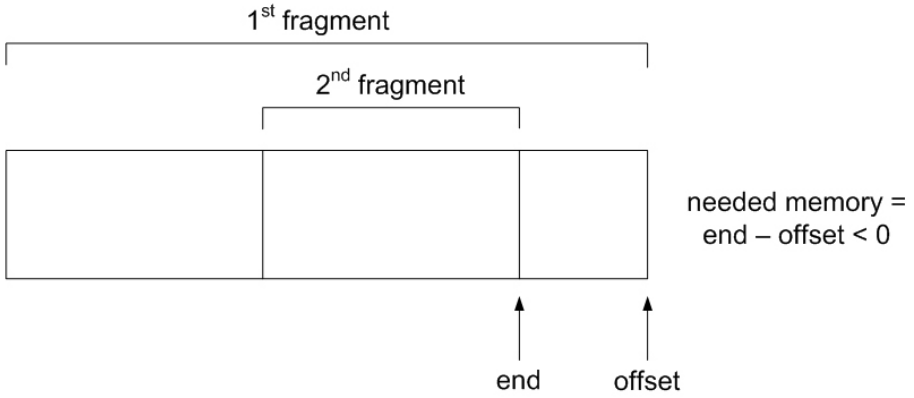
An IDS can detect this attack only by looking for two specially fragmented IP datagrams, analysing the headers. This attack exploits a vulnerability at the network layer (layer 3 of the ISO/OSI network stack [127]).

The Land attack [130] is a remote Denial of Service attack that is effective against some older TCP/IP implementations: the attack involves sending a spoofed TCP SYN packet (connection initiation) with the same source and destination IP address (the target address) and the same (open) TCP port as source and destination.

Some implementations cannot handle this theoretically impossible condition, causing the operating system to go into a loop as it tries to resolve a repeated connection to itself. An IDS detects this attack by looking at packet headers, since TCP SYN segments do not carry any payload. This attack exploits a vulnerability at the transport layer (layer 4 of the ISO/OSI network stack).



(a) Correct case



(b) Incorrect case

Figure 2.3: A correct and incorrect case of fragment management by the network layer.

2.3.2.2 Attacks Detectable by Payload-based Systems

The following attacks we present can be detected only by looking at the payload of the network connection. The attacks work by injecting some malicious content into the vulnerable application. As the attack is carried at application level, headers do not show any significant information for the IDS to detect the attack. SSAD, PAYL and POSEIDON are likely to detect these attacks.

SQL Injection is a technique that exploits vulnerabilities of (web-based) applications which are interfaced to an SQL database: if the application does not sanitize potentially harmful characters first [149], an intruder can *inject* an SQL query in the database to force it to output sensitive data from database tables (e.g., user passwords and personal details), or to execute arbitrary commands with high user-privileges. SQL Injection attacks are considered a serious threat and are constantly listed in the "Top Ten Most Critical Web Application Security Vulnerabilities" [148] by "The Open Web Application Security Project".

For instance, the following HTTP request is actually a well-known attack [142] against the Content Management System (CMS) PostNuke [138]. The attack uses a request parameter (*start* in the following example) whose content is not properly sanitized.

```
http://[target]/[postnuke_dir]/modules.php?op=modload&
name=Messages&file=readpmsg&start=0%20UNION%20SELECT%20
pn_uname,null,pn_uname,pn_pass,pn_p
```

When a SQL Injection attack is carried out successfully, the output is still an HTML page but within the usual HTML tags it is possible to find information stored in the database table(s), whose access was originally regulated by the web application. In the previous example, the attacker can get hold of the user passwords stored in the web application user table.

The PHF attack [129] exploits a badly written CGI script, that was shipped with the Apache web server, to execute commands with the privilege level of the HTTP server user. This script relies on the vulnerable function *escape_shell_cmd()*. The purpose of the function is to prevent passing shell meta-characters to critical library calls (such as *system*, which execute *any* command in the argument list). The function contains a bug in the argument handling, and it is possible to execute arbitrary commands within the HTTP server context by crafting a special HTTP request. In the following example, the attacker inserts the command to execute in the parameter *Qalias*.

```
http://[target]/cgi-bin/phf?Qalias=x\%0A/bin/cat\%20/etc/passwd
```

A successful attack forces the HTTP server to execute the command and to send back to the attacker the output generated during the execution (the list of system users in the previous example). To detect a PHF attack, an IDS can monitor HTTP requests for invocations of the *phf* command with arguments that specify commands to be run.

The above examples reflect the unsurprising fact that header-based systems are more suitable to detect attacks directed at vulnerabilities of the network and transport layers than the application layer; we can also include in this category all of the probing techniques used before a real attack takes place (port/host scanning). On the other hand, payload-based systems are more suitable to identify attacks trying to exploit vulnerabilities at the application level, where sensitive data is stored and most of the systems can be subverted.

Nowadays, this second kind of attack is the most common: this is due both to the large success of web-based services, and to the fact that network stack implementations are becoming more resilient against attacks. Because of this, we believe that payload-based systems will be increasingly useful in the future. We believe that this trend not only favours payload-based over header-based ANIDSs, but also ABSs w.r.t. SBSs.

On the other hand, a payload-based ABS is likely to miss the detection of attacks such as Denial of Service or password brute forcing. These attacks work by repeating a licit action (e.g., connecting to a web server or attempting to authenticate) hundred times in a short period of time. The anomaly cannot be found by analysing the content, but rather by analysing the number of connections (or actions) a host performs. Connection-oriented systems like iSom, IntelligentIDS and MADAM ID employ models which could detect such attacks.

We can draw the conclusion that current ABSs are not able to detect all of the possible attack classes, but focus on subsets.

2.3.3 Building the Model

Good training is of crucial importance for the effectiveness of the system. The model M_A used by an ABS should reflect the behaviour of the system *in absence of attacks*, otherwise the ABS may fail to recognize an attack as such. Because of this, the ABS should be trained with a *clean* (i.e., attack-free) data set. However, obtaining such a data set is difficult in practice: a casual dump of network traffic is likely to be *noisy* (i.e., to contain attack attempts), and the longer the traffic is dumped, the higher is the chance to include noise.

The standard way to deal with this is by sanitizing the data set manually. This relies completely on the expertise of the IT personnel which must analyse a large amount of data. The process is labour intensive, also because the detection model needs to be updated regularly to adapt to environment changes. Manual inspection can be aided by an automatic inspection using an SBS, which can pre-process the training data and discover well-known attacks (e.g. web-scanners, old exploits, etc.) An SBS however will not detect all attacks in the data, leaving the training set with a certain amount of noise.

The duration of the training is influenced by opposing constraints. On one hand the training phase should be long enough to allow the system to build a faithful model: a too short training phase could lead to a coarse data classification, which – in the detection phase – translates into flagging legitimate traffic too often as anomalous (false positives). Nevertheless, during a longer training phase, a good deal of noise is likely to be incorporated in the model. On the other hand, applications change on a regular base (this is particularly true in the context of web applications, which are highly dynamic), and each time a software change determines a noticeable change in the input of the application, one needs to re-train the model. The larger the training set required, the higher is the required workload to maintain the IDS.

Desiderata To summarize, the quality of the model M_A is crucial to achieve a low rate of both false positives and false negatives. For M_A we can define the following set of desiderata:

- To avoid false positives, M_A should contain all foreseeable non-malicious inputs.

- To avoid false negatives, M_A should be disjoint from the set of possible attacks,
- M_A should be simple to build, i.e., the shorter the training phase required to build a faithful M_A, the better it is.

2.3.4 Similarity Metric

As we have seen, to determine whether a certain input is anomalous or not, an ABS compares the input to its model, by using a similarity function. As most of the ABSs nowadays are based on statistical models, the similarity function is usually a distance function, whose output is compared to a threshold value. The choice of this function depends on the model the ABS employs, and how many features it takes into the account. For instance, for a single feature (e.g., the length of a parameter content inside a HTTP request) the distance can be defined by using the Chebyshev inequality (see [73]). The Mahalanobis distance considers multiple correlated mean and standard deviation values at once (see [121]).

The value of the threshold has an important impact on the completeness and accuracy: a low threshold yields a high number of alarms, and therefore a low false negative rate, but a high false positive rate. On the other hand, a high threshold yields a low number of alarms in general (therefore a high number of false negatives, but a low number of false positives). Therefore, setting the threshold requires skill: its "optimal" value depends on the environment being monitored and on the quality of the training data.

To represent graphically how the threshold influences completeness and accuracy, we can use a parametric curve, the Receiver Operating Characteristic (ROC), typical of the signal analysis field. The ROC curve plots the completeness value (also called *detection rate*) versus the FP value, as the threshold value is varied. Figure 2.4 shows an example of an ROC curve.

2.4 Conclusion

This chapter introduces some definitions and a taxonomy for intrusion detection systems. We have shown how an IDS can be classified on the basis of the data source it analyses and the detection model it employs. We present a detailed overview of anomaly-based intrusion detection, which is the main topic of this work (in particular, anomaly-network-based detection). Table 2.2 summarizes the main advantages and disadvantages of signature-based IDSs w.r.t. anomaly-based IDSs.

ABSs have been extensively researched but there are still major open issues that limit the application of an ABS in real environments, despite its advantage over an SBS in detecting new attacks. In this work, we address the problems of reducing false positives, making an ABS more user-friendly by allowing a user to interact with the detection engine and raising classified alerts generated by an ABS. In the following chapters, we will first

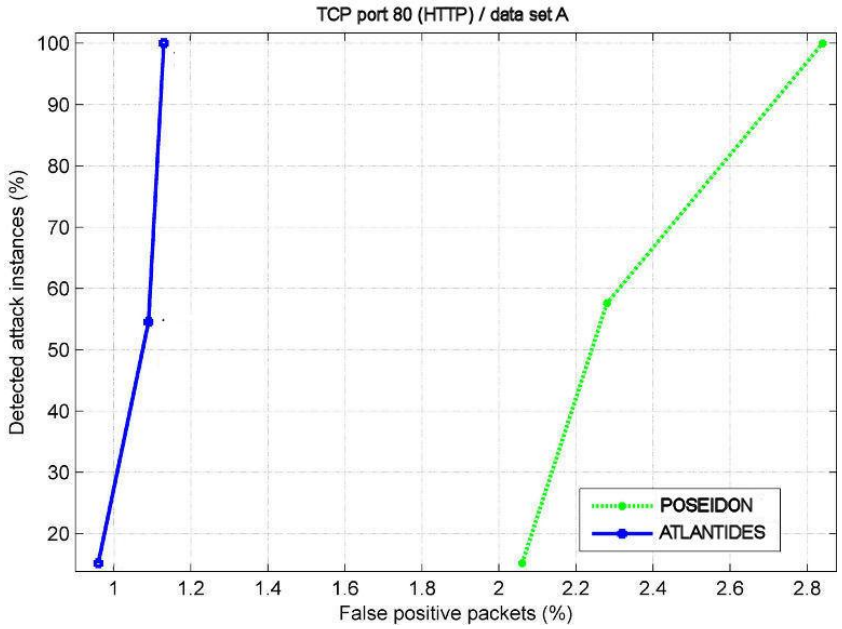


Figure 2.4: Example of ROC curve. Here, left is better than right and above is better than below. A point left-top indicates an IDS that correctly detects (almost) every attack, with very few false positives left. On the other hand, a point on the low-right side indicates an IDS that fails to detect some real attacks, and flags a good deal of licit traffic as anomalous.

discuss the core component (Chapter 3) of the SilentDefense architecture, before getting back into the usability issues and introducing each component individually.

Detection model	**Advantages**	**Disadvantages**
Signature-based	Low false positive rate Does not require training Classified alerts	Cannot detect new attacks Requires continuous updates Tuning could be a thorny task
Anomaly-based	Can detect new attacks Self-learning	Prone to raise false positives Black-box approach Unclassified alerts Requires initial training

Table 2.2: Advantages and disadvantages of SBSs versus ABSs.

Chapter 3

POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System*.

In this chapter we present POSEIDON (Payl Over Som for Intrusion DetectiON): a two-tier ANIDS. The first tier consists of a Self-organizing Map (SOM), and it is used exclusively to classify payload data; the second tier consists of a slight modification of the well-known PAYL algorithm [121] for the detection of anomalies.

POSEIDON is payload-based: it uses only the destination address and service port number to build a profile for each endpoint monitored, and it does *not* consider other header features. Mahoney and Chan [85] call the payload the *legitimate* data of the 1999 DARPA data set, implying that we can legitimately expect that our system performs in real environments similarly to what it does on the DARPA benchmarks (see Section 3.2.1).

Let us now explain the reasons that brought us to develop this architecture. We believe that the Achilles' heel of the PAYL algorithm lies in the classification it adopts: the algorithm uses packet payload length information to classify packets and thus to define detection models. This, together with the fact that - for efficiency reasons - models have to be merged, yields in our opinion a too low *intra-model* similarity: two packet payloads can be dissimilar in content but similar (or equal) in length. Because they are classified according to the latter parameter they will be classified in the same cluster, but their different byte distributions will negatively affect the way the detection engine detects anomalies. We think that a better way to tackle the data classification is to consider the packet payload data in the classification phase too.

There are several classification algorithms. We think that a SOM - in general - yields a high quality classification, i.e. models with a high intra-model similarity and high inter-model dissimilarity. First, a SOM can deal better with high-dimensional data, than al-

*This chapter is a minor revision of the paper with the same title published in the Proceedings of the 4th IEEE International Workshop on Information Assurance (IWIA '06), pages 144 - 156, IEEE Computer Society, 2006. The paper includes the pseudo code of POSEIDON, which is not reported in this work.

gorithms such as K-means and K-medoids [125]. Secondly, a SOM works in an unsupervised manner. Although SOMs have been used before for intrusion detection, we believe that a SOM is not as effective when it comes to the detection phase, i.e. finding whether a given packet is anomalous w.r.t. the profile it has been classified in. In a SOM, detection means comparing the current packet quantization error with matching profile quantization error: this method can be heavily influenced by payload byte order and thus perform poorly.

For the detection, we believe that the n-gram algorithm used by PAYL is more suitable. By combining a SOM with the n-gram algorithm we obtain an architecture that combines the advantages of the SOM (the realization of profiles with high intra-cluster similarity) with those of PAYL (the ability to detect when a packet is anomalous w.r.t. a given profile), without having to take into account the length of the packet. This combination should generate good detection profiles and the results we have obtained on the DARPA benchmark substantiate our beliefs.

3.1 Architecture

Our starting point is the PAYL architecture. Our algorithm receives as input a packet and *classifies* the packet, without prejudice for any of its properties, such as length, destination port or application data semantics. The idea is that the classifier keeps as much information as possible about packets (e.g. high-dimensional data) for the anomaly detection phase: we also want the classifier to operate in an unsupervised manner. This is a typical clustering problem which can be properly tackled using neural networks in general and Self-Organizing Maps (SOMs) [67] in particular. SOMs have been widely used in the past both to classify network data and to find anomalies. Our architecture combines a SOM, used for pre-processing, with a modified PAYL algorithm.

We now give a high-level description of the algorithms underlying POSEIDON. We first describe the SOM. Later in the section, we introduce PAYL, focusing on the main differences between our approach and the PAYL approach towards classification of network data.

3.1.1 SOM Classification Model

A Self-organizing Map is a kind of artificial neural network [67]. It works by emulating the cognitive classification process typical of the human brain and it is based on competitive learning. Like any other neural network, the basic component of the network structure is a neuron, and a SOM usually employs a single bi-dimensional neuron grid, either rectangular or hexagonal. Each neuron n has a weight vector w_n associated, and the dimension of this vector is proportional to the input dimension. A SOM has two main properties: (1) it works in un-supervised mode (i.e., the user does not need to provide any information regarding the classification) and (2) it maps high-dimensional data into a two-dimension space (the grid).

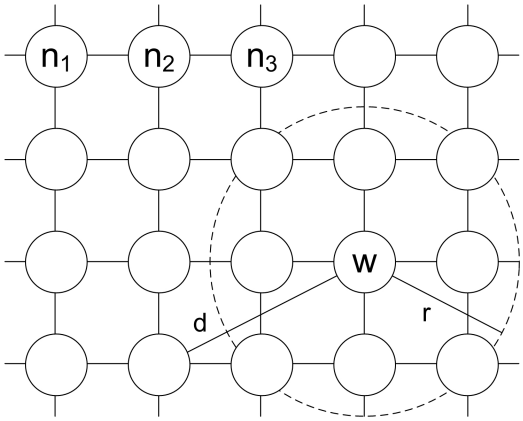
The goal of learning in a SOM is accomplished by modifying the weight vector of neurons in a certain manner. To select which weight vectors must be updated and how, two parameters are used: the learning rate, that controls the "influence" of the input on the neuron weight array components, and the update radius, that controls which neurons to modify.

To accomplish the classification, a SOM goes through three phases: initialization, training, and classification.

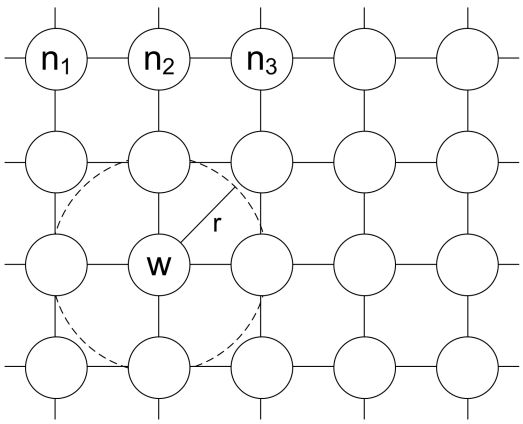
```
∀n∈N do
      dist = Manhattan_dist(x,n_i)
      if(dist < best)
            best = dist
            winner = n_i
      end if
end do

∀n∈N do
      if(trig_dist(winner,n_i) < r)
            update_neuron(n_i,x,α)
      end if
end do

decrease(α)
decrease(r)
```



At iteration t = 0



After some iterations

Figure 3.1: Training of a Self-organizing Map.

Initialization First of all, some parameters (number of neurons, learning rate, radius, and number of training samples) have to be fixed. Each value influences the SOM classification. For instance, a small network with few neurons will classify different data inputs in the same neuron while a large network will produce a too sparse classification.

Although these parameters are crucial, it is difficult to provide general guidelines, and optimal values come from experiments.

To initialize the array of neuron weights, it is possible to either sub-sample some training inputs or to use random values (in the same range of input values). The latter approach is faster and, in case there is a large number of training samples at disposal, it will not affect the classification.

Training The training phase consists of a number of iterations (also called *epochs*): each training sample is used for one iteration only, thus the number of samples determines the number of iterations. At each iteration, the current input sample x is compared using a mathematical distance (e.g., Euclidean or Manhattan distance) to each neuron weight vector w_n in the network (this approach is known as "competitive learning"). The neuron with the smallest distance is selected as winner (or *best matching unit*, BMU). After the winner has been found, the algorithm continues by updating "neighbouring" neurons of the winner neuron. A neuron is a neighbour if the trigonometric distance with the winner is smaller than the current radius value. Each neighbour's weight array n_w is then updated using the following formula $n'_{w_i} = n_{w_i} + \alpha * (x_i - n_{w_i})$. Eventually, the radius and the learning rate values are decreased as follows: $r = r_{init} * \frac{\#i-i}{\#i}$ and $\alpha = 1.0 + (\alpha_{init} - 1.0) * \frac{\#i-i}{\#i}$, where r_{init} is the initial radius value, α_{init} is the initial learning rate value, $\#i$ is the total number of iterations and i is the i-th iteration.

Classification To classify an input x, the SOM proceeds similarly to the training phase. The input is compared to each neuron weight vector w_n: once the winner has been found, that indicates the classification class of the input.

3.1.2 PAYL Classification Model

PAYL detects anomalies by combining an n-gram [40] analysis algorithm with a classification method based on clustering of packet payload data length. N-gram analysis allows to capture features of data payload in an efficient way, and it has been used before in the context of computer security (Forrester and Hofmeyr [51]).

PAYL employs a set of detection profiles (one for each endpoint it monitors): a profile is identified by the unique pair destination address i and destination port j. Each profile contains a set of *models*: a model M_{ijl} stores incrementally the resulting values of the n-gram analysis for packet payloads of length l, thus each payload length has a different model, in a given profile.

The n-gram analysis (PAYL uses 1-gram analysis) computes relative byte frequency values for a payload of length l. The byte frequency for byte value b_h is $\frac{\#b_h}{l}$. Once this value has been computed, PAYL updates the corresponding M_{ijl} model. A model stores two data: mean byte frequency (i.e., relative byte frequencies span across several payloads of length l) and byte frequency standard deviation for each byte value (i.e., how relative byte frequencies change across payloads). There are 256 possible different byte values in

each payload, thus a model needs two vectors of size 256 for storing. Figure 3.2 shows the internals of PAYL.

During the detection phase, the same values are computed for incoming packets and then compared to model values: a significant difference from the model parameters produces an alert. To compare a payload to a model, PAYL uses a simplified version of the Mahalanobis distance, which has the advantage of taking into account the average and variance values of the variables measured:

$$dist_{Mahalanobis}(M, x) = \sqrt{\sum_{h=1}^{256} \frac{\mu_{M_h} - x_h}{\delta^2_{M_h}}}$$

Here, M and x are the selected PAYL M_{ijl} model and the current input payload of length l respectively. μ_{M_h} and $\delta^2_{M_h}$ are the mean byte frequency and the byte frequency standard deviation for byte value b_h respectively (and stored inside the model M_{ijl}). x_h is the byte frequency for byte value b_h ($\frac{\#b_h}{l}$) for the current input payload.

The maximum number of detection models used by PAYL is: $\#i * \#j * l$, where $\#i$ is the number of destination addresses, $\#j$ is the number of destination ports and l is the length of the longest payload. Thus, the total amount of memory required by PAYL is at most $\#i * \#j * l * 2 * 256 * k$, where k is a constant representing the space required to store a floating point number in the physical memory on a given architecture.

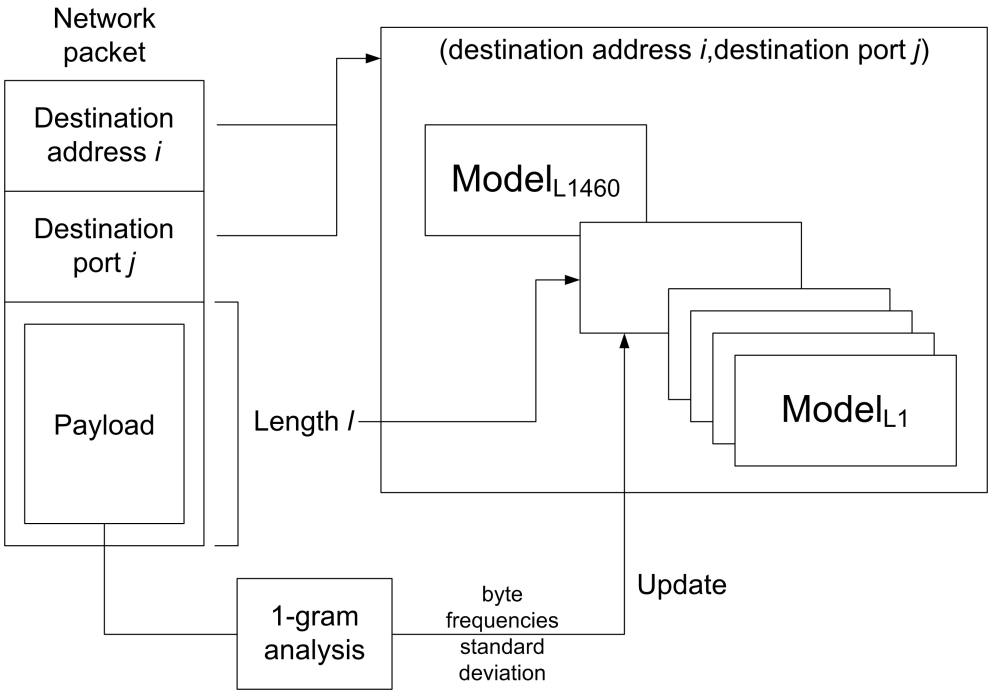


Figure 3.2: Internals of PAYL.

To reduce the otherwise large number of models to be computed, PAYL organizes models in clusters. After comparing two neighbouring models using the Manhattan distance, if the distance is smaller than a given threshold t, models are merged: the means and variances are updated to produce a new combined distribution. This process is repeated until no more models can be merged. Experiments with PAYL show [121] that a reduction in the number of models of up to a factor of 16 can be achieved.

3.1.3 POSEIDON

POSEIDON combines the SOM with the models used by PAYL as follows. First, we pre-process each packet by feeding the SOM with the payload data. The full packet payload is analysed by the SOM, by comparing it with each neuron weight array. Because in a Ethernet Local Area Network data in a TCP segment can contain up to 1460 bytes, each neuron has a 1460 long weight array. In case the current data payload is shorter than 1460 bytes, additional (null) bytes are appended to reach a length of 1460 bytes. The SOM returns the value of the most similar neuron (*winning neuron*). Afterwards, PAYL selects the model using the SOM neuron value instead of the payload length. Technically, instead of using model M_{ijl}, PAYL uses the model M_{ijn} where i and j are the usual destination address and port and n is the classification derived from the neural network. Then, relative byte frequency and standard deviation values are computed as usual. Figure 3.3 shows the internals of POSEIDON.

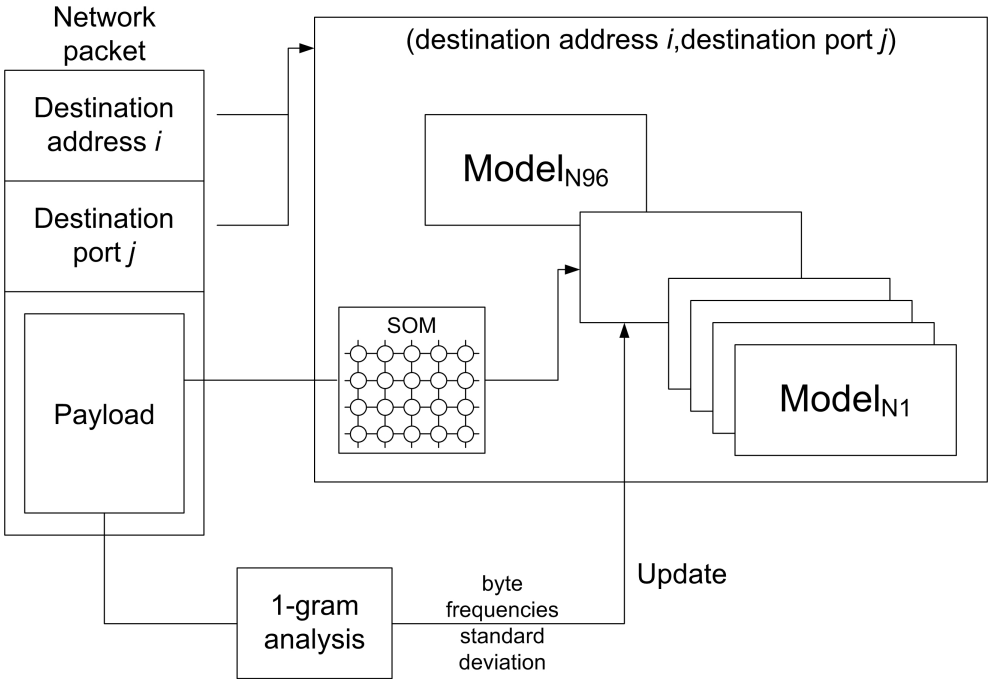


Figure 3.3: Internals of POSEIDON.

Having added a SOM to the system, we must allow for both the SOM and PAYL to be trained separately. In case the training is accomplished off-line, i.e., the training data is dumped in advance, it is possible to either train both the SOM and then PAYL with the same data or to train the two components with different data. We tested both modes during our benchmarks and did not observe significant differences in the behaviour of the system. It is possible to train the system on-line as well, by capturing live network traffic: should that be the case, the SOM and PAYL are trained with different data (similarly to the second off-line mode we introduced). Regarding memory consumption, we have to revise the amount of detection models PAYL uses to: $\#i * \#j * \#n$, where the new parameter $\#n$ indicates the number of SOM network neurons.

To calculate the total amount of memory required by POSEIDON we have to take into consideration the memory required to store a SOM as well: $\#n * l * k$, where l is the usual length of the longest payload and k the constant representing the space required in physical memory to store a floating point number. Thus, the total amount of memory required by POSEIDON is: $\#n * l * k \ + \ \#i * \#j * \#n * 2 * 256 * k$.

Resilience to mimicry attacks Mimicry attacks [117, 118] can evade payload-based ABSs that analyse byte frequencies. By carefully crafting an attack payload the attacker is able to fool the IDS. The modified payload contains additional bytes, which are useless to carry on the attack, but match the statistics of normal profiles. Examples of mimicry attacks against PAYL have been shown by Fogla et al. [50]. The original PAYL algorithm is vulnerable to mimicry attacks since it models only 1-gram byte distributions.

POSEIDON is resilient to mimicry attacks thanks to the combination of the SOM with PAYL. The SOM analyses the input by taking into consideration the byte value at i-th position within the whole payload. Thus, extra bytes inserted by the attacker would be taken into consideration as well, resulting in a different classification than normal traffic.

3.2 Tuning and Benchmarks

In this section, we show the results of our benchmarks and compare the performance of POSEIDON with PAYL and PHAD. PAYL and PHAD are the two reference ADS based on payload. They are the only two payload-based ABSs which have published their detection rate on the DARPA 1999 data set.

SOM parameters tuning The SOM algorithm needs several parameters on start-up: the total number of network neurons, the function used to compute the distance between vectors and the values of the *learning rate* and *update radius*. For the sake of transparency, we report here the values used in our experiments.

Concerning the number of neurons, a small network would yield a too course classification, while a large network will produce a sparse classification. In addition, it is worth

bearing in mind that the computational load increases quadratically with the number neurons.

Experimenting with different initialization parameters and using the *quantization error* method [67] to evaluate the classification given by the network, we found the best SOM with the following parameters:

- Number of neurons: 96 (rectangular network of 12 by 8)
- Learning rate: 0.1
- Update radius: 4
- Distance function: Manhattan

Hinneburg et al. [55] state that the Manhattan distance performs better than the Euclidean distance in presence of high-dimensional data: our experiments confirm this statement also in the case of network data analysis.

3.2.1 Benchmarks

We have benchmarked POSEIDON against PAYL (also by replicating the experiments on PAYL) and PHAD, using the same data used by PAYL and PHAD: the DARPA 1999 data set [83]. This standard data set is used as reference by a number of researchers (e.g. [84, 92, 121]), and offers the possibility of comparing the performance of various systems.

The DARPA 1999 data set is a synthetic set of network dumps. The data generated for the evaluation span over 5 weeks and can be divided in training and testing data. Training data (week 1 and 3 of traffic) is intended to be completely free of attacks, while testing data (week 4 and 5) is intended to consist entirely of attack scenarios. An additional week of traffic (week 2) is provided with labelled attacks, i.e., attacks are clearly marked with temporal timestamps and classified. Testing data did not originally contain any timestamps or classification of attacks, as the evaluation was run at researcher's site and results were sent back to the DARPA team. The process used to generate training data or attacks is not deeply presented. The data is claimed to be similar to that observed during several months of sampling data from a number of Air Force bases (see Lippman et al. [83]), but the data set lacks of statistics to evaluate and establish similarities. The data set embodies nearly 300 attack instances, which exploit vulnerabilities both at network and application levels, divided into four main categories ("User to Root", "Remote to Local User", "Denial of Service", and "Probe/Surveillance"). The simulated network contains several machines equipped with various operating systems (ranging from Windows 95 to Solaris), but testing data mainly contains attacks to four of them. No evidence is given that the network configuration applies to an Air Force base.

This data set has been criticized because of the environment in which data were collected and because attacks were adaptations of scripts or malware collected from a variety

of sources [87]; as explained by Mahoney and Chan [85], it is possible to tune an IDS in such a way that it scores particularly well on this particular data set: some attributes – specifically: remote client address, TTL, TCP options and TCP window size – have a small contribution to the DARPA simulation, but have a large contribution to real traffic. An IDS that takes into account the above-mentioned attributes is likely to score better on the DARPA set than in real life. Since our system does not consider these attributes, we can legitimately expect that the system in real life performs as well as it does on the DARPA benchmark.

3.2.1.1 PAYL and POSEIDON

To compare our model with PAYL, we apply the same restrictions and conditions used by Wang and Stolfo [121]: we focus only on inbound TCP packets, with data payload, directed to hosts 172.016.0.0/16 and ports 1-1024.

We train the SOM clustering algorithm using internal network traffic of week 1 and week 3 (12 days, 2.444.591 packets, attack free): for each different protocol we use a different SOM. Then, we use the same data to build PAYL models taking advantage of the classification given by the neural network. After this double training phase, it is possible to use the testing weeks (4 and 5) to benchmark the network intrusion-detection algorithm. This data contains several attack instances (97 payload-based attacks are detectable applying the same traffic filter mentioned above), as well as legal traffic, directed against different hosts of the internal network: the attack source can be situated both inside and outside the network.

To compute detection and false positive rates we apply the same approach used by the PAYL authors. We consider an attack to be successfully detected when at least one packet carrying the attack payload is correctly flagged as malicious; all of the other non-detected packets carrying the attack payload are not considered to be false negatives. On the other hand, each packet incorrectly flagged as malicious is considered to be a false positive. Thus, the detection rate is attacked-based, while the false positive rate is packet-based.

Figure 3.4 shows a detailed comparison of PAYL and POSEIDON in terms of detected attack instances (reported on the y axis) w.r.t. the false positive percentage (x axis). Table 3.1 reports a summary of these results: the first column reports PAYL's statistics as we have inferred them from the graphs reported by Wang and Stolfo [121]. The second column reports the figures we obtained by repeating Wang and Stolfo's benchmarks. In the repeated PAYL experiments we used an **un-clustered** architecture, which yields on one hand a higher number of profiles, and on the other hand a different classification. The third column reports POSEIDON's result. POSEIDON outperforms PAYL on every benchmarked protocol. However, there is a remark about FTP protocol (see the next paragraph).

Remark During the FTP protocol benchmarks we found a high rate of false positives (more than 3000 packets) both with PAYL and with POSEIDON: all of these packets are

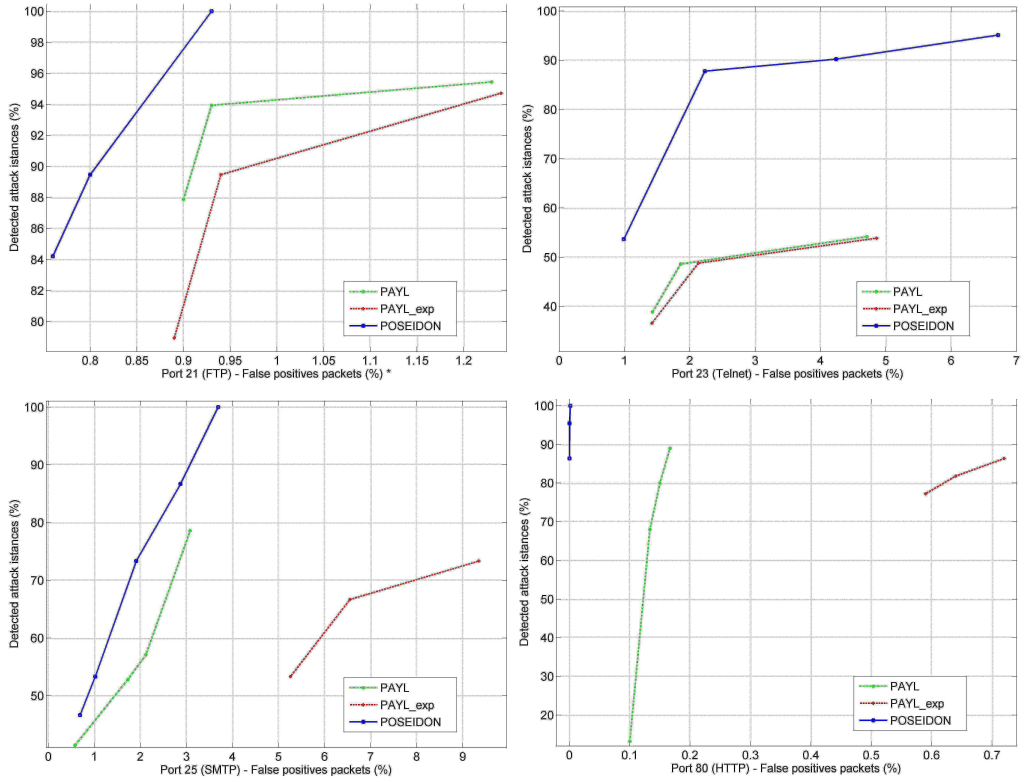


Figure 3.4: Detection rates for ports 21 (FTP), 23 (Telnet), 25 (SMTP) and 80 (HTTP): the x-axis and y-axis present false positive rate (packet percentage) and detection rate (attack instance percentage) respectively. POSEIDON yields a higher detection rate compared with PAYL at the same false positive rate. For the graph relative to port 21 see Remark in Section 3.2.1.1.

sent by the same source host, which is sending FTP commands in a way that is typical of the Telnet protocol (one character per packet, with the TCP flag *PUSH* set). These packets are marked as an attack because the training model does not contain this kind of traffic over the FTP control channel port, although it is normal traffic. During the experiments with our implementation of PAYL, we found the same behaviour, although Wang and Stolfo do not report it: for this reason we decided to present benchmarks results of PAYL and POSEIDON also without taking into account these packets (the figures marked with an asterisk * in Table 3.1 and the graph in Figure 3.4).

3.2.1.2 PHAD and POSEIDON

Table 3.2 compares our results with PHAD: it is not possible to make a full comparison between the two systems, because of the restrictions used by the PHAD authors (they restrict to a maximum total amount of 100 false positives during 10 days of testing).

		PAYL	PAYL_exp	POSEIDON
Number of profiles used		4065	(11312 - unclustered)	1622
HTTP	DR	89,00%	90,00%	100,00%
	FP	0,17%	0,73%	0,0016%
FTP	DR	95,50%	94,74%	100,00%
	FP	1,23%	11,41% (1,21%*)	11,31% (0,93%*)
Telnet	DR	54,17%	53,65%	95,12%
	FP	4,71%	4,94%	6,72%
SMTP	DR	78,57%	73,34%	100,00%
	FP	3,08%	8,35%	3,69%
Overall DR with FP $< 1\%$		58,8% (57/97)		73,2% (71/97)*

Table 3.1: Comparison between PAYL, our implementation of PAYL (PAYL_exp) and POSEIDON; DR stands for detection rate (attack instance), while FP is the false positive rate (packets). The last row reports the detection rate when the FP stays below 1% (number of detected attack instances over total number of detectable attacks, 97). Regarding the FP rate in FTP protocol (marked with a *), we have to calculate this value in a different way than for the other protocols (see Remark in Section 3.2.1.1 for a detailed description).

Nonetheless, we could legitimately compare the two systems on the HTTP protocol, on which POSEIDON meets the restrictions above.

Summary of benchmark results Unfortunately, there is no other public available data set suitable to compare our approach with previous work on anomaly intrusion detection: many authors use the KDD 99 data set [23] in which regrettably payload data is

Type	Attack	PHAD	POSEIDON
Probe	ntinfoscan	66,67% (2/3)	100% (3/3)
Denial of Service	apache2	100% (3/3)	100% (3/3)
	back	0% (0/4)	100% (4/4)
	crashiis	71,43% (5/7)	100% (7/7)
Remote to Local	phf	66,67% (2/3)	100% (3/3)
	ppmacro	33,34% (1/3)	100% (3/3)
Overall detection rate		65% (13/20)	100% (20/20)

Table 3.2: Comparison between PHAD and POSEIDON detection rates.

discarded. Because we use payload information, we can not use this data set to benchmark POSEIDON and models that use this data set are not directly comparable with ours.

Concluding, benchmarks show that our architecture is more effective than PAYL in terms of detection and false positive rates. Our modification to the original PAYL algorithm consists of replacing the packet classification based on payload length with a classification based on the actual payload data. The new classification performed by the SOM leads to better results because PAYL builds more homogeneous detection models, as similar payloads are incorporated in the same model although they could show a significant difference in length.

3.3 Related Work

In this section we report on related work. First we describe other neural network-based systems then we address statistics-based systems.

Neural Network-based Systems We start by presenting other neural network-based ANIDSs. We cannot benchmark these systems with POSEIDON because their authors use either private data sets (Cannady [27], Labib and Vemuri [74] and Ramadas et al. [105]), or data sets that do not contain payload information (Depren et al. [46]) or do not provide precise statistics (Nguyen [92]).

Cannady [27] proposes a SOM-based NIDS in which network packets are first classified according to nine features and then presented to the neural network. Attack traffic is generated using a security audit tool. The author extends this work in [28, 29].

Nguyen [92] uses a one-tier architecture, consisting of a SOM, to detect two attacks in the 1999 DARPA data set: the first one (*mailbomb*) against the SMTP service, and the other one (*guessftp*) against FTP.

Labib and Vemuri [74] use a SOM to identify Denial of Service attacks. They discard information about payload and use only packet header information; their data is collected from a private network (described in a general way) and is not publicly available.

Ramadas et al. [105] use a SOM to detect attacks against DNS and HTTP services (using a private data set): they use a pre-processor to summarize some connection parameters (source and destination host and port) and then add several values to track connections behaviour: the information is then merged in a data structure used to fire events related to the connection and to feed the neural network.

Depren et al. [46] present a hybrid IDS based on SOMs and benchmark it on the KDD 99 data set [23]. They feed the neural networks (one for each protocol type) with six features extracted from each connection (duration, protocol type, service type, status, total bytes sent and received) and then use the quantization error method to detect anomalies. The system is connection-oriented, therefore attacks can be detected only when the connection is completely re-assembled. Regarding their architecture, the authors state that the

SOM used to model TCP connections uses 1515 neurons; which in our opinion is quite large, if compared with the ones used by our system.

Statistics-based Systems We now report on statistics-based ANIDSs. Again, we cannot benchmark them against POSEIDON because they either use only header information (Staniford et al. [111], Javitz and Valdes [59]) or employ benchmarking data that is not publicly available (Kruegel et al. [71]).

Barbará et al. [21, 20] use data mining techniques to detect attacks on network infrastructures: their system ADAM first applies association rules techniques to identify abnormal events in traffic data; then a classification algorithm is used to classify the abnormal events into normal instances and abnormal instances. The original work has been expanded in Barbará et al. [22]. Lee et al. [78, 80] propose a comprehensive framework based on data mining. For a complete overview of data mining techniques applied to intrusion-detection see Julisch [62].

The NIDES [59], PHAD [84] and SPADE [111] systems rely on statistical models computed on normal network traffic: they work by extracting features from the packet header fields and trigger an alarm when they recognize a significant deviation from the normal model; most of the features extracted are related to IP addresses (source and destination), destination service port and TCP connection state (PHAD uses up to 34 attributes coming from Ethernet, IP and application layer protocols packets.) Our approach differs from the ones mentioned here in the following aspects. First, it is payload-based, we use only destination address and service port numbers to build a profile for each port monitored, without taking care of other header features (of the above systems only PHAD considers payload information, we have compared it with our system in the previous section.) Second, we have a two-tier architecture in which the SOM is used only to pre-process information.

Shifting to payload-based systems, Kruegel et al. [71] show that it is possible to find the description of a system that computes a payload byte distribution and combines this information with extracted packet header features. They first sort the resultant ASCII characters by frequency and then aggregate them into six groups. As argued by Wang and Stolfo [121], this leads to a course classification of the payload.

PAYL works in a way similar to Kruegel et al. [71] but models the full byte distribution based on payload data length and operates a clustering phase to cover possible missing lengths. The PAYL architecture is made up of a single tier, while our architecture has two different layers: the first one, made up by a SOM, is delegated to classify packets only using payload data information, without using payload length value. The second layer is a modified version of PAYL that computes byte distribution models using the classification information coming from the first layer and extracting destination IP address and service port from packets header.

Zanero [124] presents a two-tier payload-based system that combines a self-organizing map with a modified version of SmartSifter [123]. While this architecture is similar to POSEIDON, a full comparison is not possible because the benchmarks of Zanero [124]

concern only the FTP service and no details are given about experiments execution. A two-tier architecture for intrusion-detection is also outlined in Zanero and Savaresi [125].

3.4 Conclusion

We present an approach to network intrusion detection that combines two different techniques: a self-organizing map and the PAYL algorithm. We modify the original PAYL to take advantage of the unsupervised classification given by the SOM, which then functions as pre-processing stage.

Our experiments on the DARPA data set show that our approach reduces the number of profiles used by PAYL (payload length can vary between 0 and 1460 in a Ethernet Local Area Network, while the SOM neural network used in our experiments has less than one hundred neurons.) Our experiments show that PAYL without SOM requires 3 times as many profiles as with the SOM pre-processing (see Table 3.1).

We have extensively benchmarked our system w.r.t. PAYL [121] (also by replicating the PAYL experiments) and PHAD [84] using the 1999 DARPA benchmark [83]. PAYL and PHAD are the reference ABSs based on payload analysis. On this data set, our experiments show a *higher* detection rate and *lower* number of false positives than PAYL and PHAD, and a reduction of the number of profiles used w.r.t. PAYL. This is the main contribution of POSEIDON to the usability issues: by raising fewer false alerts, the system reduces the user burden as well.

We benchmark POSEIDON extensively against the PAYL algorithm and data sets showing a higher detection rate and a lower false positive rate for POSEIDON.

POSEIDON is not only used to detect network attacks, but it is also the core component of SilentDefense. In the following chapters, we will show how, by using an adapted POSEIDON core, ATLANTIDES and Sphinx perform their tasks, and how they contribute to usability issues.

Chapter 4

ATLANTIDES: an Architecture for Alert Verification in Network Intrusion Detection Systems*

The Achilles' heel of a NIDS lies in the large number of *false positives* that occur [89]: practitioners [86, 99] as well as researchers [16, 33, 61] observe that it is common for a NIDS to raise thousands of alerts per day, most of which are false alerts. Julisch [63] states that up to 99% of total alerts may not be related to real security issues. Notably, false positives affect both SBS and ABS [17]. A high rate of false alerts is – according to Axelsson [16] – the limiting factor for the performance of an IDS. False alerts cause an overload for IT personnel [86], who must verify every single alert, a task that is not only labor intensive but also error prone [39]. For instance, when considering a NIDS, with a hundred thousands input packets per hour (which is a reasonable figure for a web server), a false positive rate of 1% still generates a thousand false positives per hour, which is more than a typical company can afford to handle. Indeed, a high false positive rate can even be *exploited* by attackers to overload IT personnel, thereby lowering the defences of the IT infrastructure. Finally, when a NIDS raises too many false positives, system managers tend to ignore alerts raised.

The main reason why a NIDS raises false positives is that – quoting Kruegel and Robertson [69] – it is often run without any (or very limited) information about the network resources it protects (i.e., the context). Chaboya et al. [30] state that the context knowledge (e.g., network and system configurations) can improve significantly alert verification. On the other hand, building and updating a database of the configurations or running vulnerability assessment tools (e.g., Nessus [141]) to provide context knowledge

*This chapter is a minor revision of the paper with the same title published in the Proceedings of the 21st Large Installation System Administration Conference (LISA '07), pages 141 - 152, USENIX Association, 2007. The paper includes the pseudo code of ATLANTIDES, which is not reported in this work.

is expensive and often not feasible when dealing with complex systems (indeed these activities require additional labor of IT personnel, since they cannot be completely automated.) Most current techniques to improve alert verification are tailored for specific attacks [53, 119] (e.g., worm-like) or support only signature-based NIDSs [101, 109].

Our hypothesis is that, in many relevant situations, the context information can be obtained by a systematic (and automatic) *anomaly-based* analysis of the output traffic of the monitored network services; we believe this is possible when the output traffic presents some regularities, which in practice is often the case.

Contribution To substantiate our claims, we have developed *ATLANTIDES* (Architecture for Alert verification in Network Intrusion Detection Systems) an innovative architecture for easing the management of *any* NIDS (be it signature- or anomaly-based) by reducing, in an automatic way, the number of false alarms a NIDS raises. The main idea behind ATLANTIDES is simple: a successful attack often causes an *anomaly* in the *output* of the service [126], thus modifying the normal output outcome. Detecting this anomaly can help in reducing false alerts. For instance, a successful SQL Injection attack [149] against a web application often causes the output of SQL table content (e.g., user/admin credentials) rather than the expected web content.

ATLANTIDES is completely network-based, i.e., it relies only on information gathered over the network, without involving any host-based component. It works by analysing, using n-gram analysis (see Section 3.1.2), and modeling the normal output payload of the monitored network services that is expected to be sent in response to a client request. This normal output is specific to the site; therefore the derived models reflect – in a way – the network/system context. By correlating the anomalies detected on the output with the alerts raised by the NIDS monitoring the input traffic, we can discard a number of the latter as being false alerts. This way we obtain a system that raises considerably fewer false positives than the original NIDS, without this correlation system.

In the past, simple correlations between input and output traffic have already been used to identify possible worm attacks [53, 119]. To the best of our knowledge, ATLANTIDES is the first proposed solution for alert verification that:

- Works in combination with both signature- and anomaly-based NIDSs.
- Operates in a completely automatic way after a quick setup, without any further human involvement (i.e., reducing the IT personnel overload), thus improving usability and easing NIDS management.

We benchmarked ATLANTIDES in combination with Snort, as well as in combination with POSEIDON. We carried out benchmarks both on a private data set as well as on the common DARPA 1999 data set (for the sake of completeness and to allow duplication of our results). In seven out of eight cases, our benchmarks show a reduction of false positives between 50% and 100%.

Since nowadays attacks against connection-less protocols are less common (see the Common Vulnerabilities and Exposures [147] database for detailed statistics), we have

designed ATLANTIDES with the explicit goal of reducing false positives when monitoring network services based on the TCP protocol (e.g., HTTP, SMTP and FTP) where a response is typically sent by the server to the client

Limitations of the approach Because ATLANTIDES is based on output payload analysis, our architecture is designed for TCP-based client/server network services (such as HTTP). Like all (external) payload-based analysis, ATLANTIDES cannot work properly with encrypted data unless the cryptographic keys are provided.

Although we do not aim to handle all kinds of possible attacks (e.g., worms or DDoS attacks work by generating a huge number of legal connections), we believe our solution can improve the accuracy of a NIDS without any additional component installed directly on the monitored hosts. An additional component could affect under certain circumstances host performance, i.e., a high number of connections.

Since ATLANTIDES relies on the output generated by the monitored service, an attacker, by carefully crafting an attack payload, could forge a legitimate output after the intrusion to fool the alert verification process. Todd et al. [112] present such a technique to alter the alert verification process. However, this technique has some limitations and works under some circumstances only. The attacker has to inject some shell code (i.e., encoded machine instructions) to be executed by the victim to forge the reply, or to "jump" to some code already in memory. Thus, only vulnerabilities that allow some code injection can be exploited to generate a normal request. Secondly, the memory area the attacker fills in with injected code must be large enough to contain both the attack and the response-forging code. For instance, most buffer overflow attacks target buffers which are normally small in size [30], thus it could be difficult to include the necessary payload . Todd et al. note that a payload-based IDS could examine the additional code used to forge the fake response and detect the attack.

4.1 Preliminaries

In this section, we introduce the concepts used in the rest of the chapter and explain how false positives arise in SBSs and ABSs.

4.1.1 The Base-rate Fallacy

Because intrusions are rare events, even a low false alert rate does not result in an high detection rate (this is known in the literature as the *base-rate fallacy*, Axelsson [17]).

Assuming an IDS analyses 1.000.000 packets per day, and 100 attacks target the environment the IDS monitors, the probability of an intrusion is $P(I) = 10^{-4}$. Let $P(A|I)$ be the probability of detection and $P(A|\neg I)$ the probability of raising a false alert. Then, it is possible to calculate the probability that a raised alert is actually true ($P(I|A)$) by using the Bayes theorem:

$$P(I|A) = \frac{P(I)*P(A|I)}{P(I)*P(A|I)+P(\neg I)*P(A|\neg I)}$$

With an (unrealistic) detection rate $P(A|I) = 1.0$ and a low false alert rate $P(A|\neg I) = 10^{-3}$ we obtain a final value $P(I|A) = 0.09$. This means that nine tenth of alerts are not related to real attack attempts. With more realistic values $P(A|I) = 0.7$ and $P(I|A) = 0.01$, we obtain that 99% of all raised alerts will be false.

4.1.2 False Positives in Signature-based Systems

An SBS raises an alert every time the current input matches a signature loaded into its database. Consider for example the *path traversal attack*, which allows access to files, directories, and commands residing outside the (given) web document root directory. This attack occurs in a web application that does not properly sanitize inputs which are used to retrieve some file content from the file system, and it allows the attacker to read files outside the safe web application context (e.g., the user password file). The most elementary path traversal attack uses the "`../`" character sequence to alter the resource location requested in the URL. Variations include valid and invalid Unicode-encoding ("`..%u2216`" or "`..%c0%af`"), URL encoded characters ("`%2e%2e%2f`"), and double URL encoding ("`..%255c`") of the backslash character (excerpted from the *WASC Threat Classification* [149]).

To detect this attack class, an SBS (using an out-of-the-box configuration) raises an alert each time it sees the pattern "`../`" in the incoming traffic. Unfortunately, this pattern could be present in legal traffic too; some content management systems insert relative paths in request parameters to load files, e.g., related to the user language, which causes an SBS to raise a high number of false alerts. These false alerts can be avoided by deactivating the specific rule. On the other hand, this could prevent the NIDS from detecting this sort of attacks.

Tuning Signature-Based Systems The main reasons why alerts raised by an SBS turn out to be either false or irrelevant are the following:

- A signature can be too general (see Section 2.2.1.1)
- The monitored environment is not susceptible to a certain vulnerability
- Mis-configured network devices or services producing atypical output (usually, in this case, it is possible to observe recurrent and periodic phenomena).

A good deal of false positives can be suppressed by a *tuning* activity: this activity, based on deactivation of unneeded signatures, requires a thorough analysis of the environment by qualified IT personnel. Finally, to remain effective, an SBS requires continuous configuration updates to reflect changes in the environment: new vulnerabilities are discovered daily, new signatures are released regularly, and systems may be patched, thereby (possibly adding or) removing vulnerabilities.

4.1.3 False Positives in Anomaly-based Systems

The high false positive rate of an ABS is generally cited as one of the main disadvantages. The most commonly used *tuning* procedure for an ABS is finding an optimal threshold value, i.e., the best compromise between a high number of detected attacks and a low (or acceptable) number of false positives. This is typically carried out manually by trained IT personnel: different improving steps can be necessary to obtain a good balance between detection and false positive rates.

4.2 Architecture

The ATLANTIDES architecture (see Figure 4.1) consists of one external and two internal components. The external component is the NIDS monitoring the incoming traffic. We do not make any assumption about it except that it is capable of raising an alert: ATLANTIDES can work together with any kind of NIDS (signature- or anomaly-based).

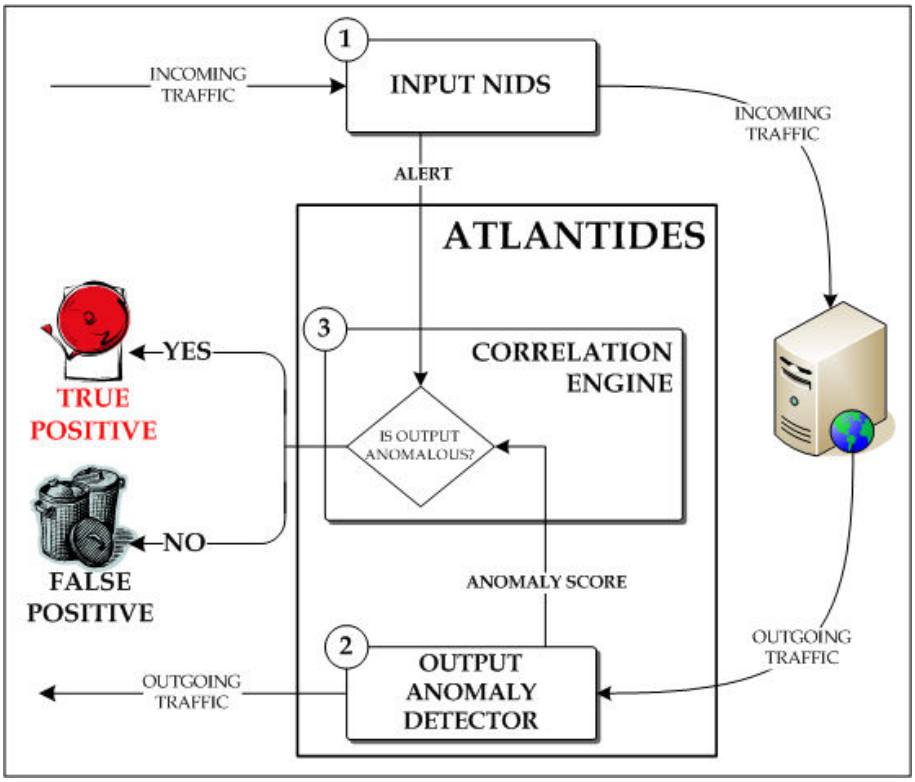


Figure 4.1: The ATLANTIDES architecture.

The first internal component is the output anomaly detector (OAD), which is actually an anomaly-based NIDS monitoring the outgoing traffic. The OAD employs a statistical

model describing the normal output of the monitored network service(s), and flags any behaviour that significantly deviates from the norm as the result of a possible attack.

The second internal component is the correlation engine (CE), which tracks network connections (using stateful-inspection [131]) and correlates alerts related to incoming traffic and raised by the input NIDS with the output produced by the OAD.

ATLANTIDES works as follows (see Figure 4.1). The input NIDS monitors the incoming traffic while, simultaneously, the OAD (after a training phase) analyses the output of network services. When the input NIDS raises an alert, this is forwarded to the CE, together with the information regarding the communication endpoints (i.e., source and destination IP addresses, source and destination TCP ports as well as sequence numbers and communication status) of the packet that raised the alert. The CE uses a hash-table to store this information, using less than 20 bytes per entry: thus, the CE does not require much memory to store the information, and ATLANTIDES can handle even a rate of 1000 alerts per second with a total memory space of 1 MB (in case the connections are kept in memory, e.g., for a maximum time of 60 seconds before being dropped.) At this time, the alert is not considered an incident yet (it is a *pre-alert*) and is not forwarded immediately to IT specialists. Next, the CE marks the communication related to the given endpoints as suspicious and waits for the output of the OAD: if the OAD detects an anomaly in the outgoing traffic related to the tracked communication, then the system considers the alert as an *incident* (i.e., a positive) and the alert is forwarded to the IT specialists for further handling and countermeasure reactions, otherwise it is considered a *false positive* and discarded. The IT personnel can manually set (or adjust) the time value t that the CE waits before dropping an entry from its hash-table, because no output has been produced: during our experiments we fixed this value to 60 seconds. This time could be critical if an attack results in a large data transfer (but in this case the OAD should detect the anomaly in the transferred data) or in the case where the attacker is able to delay the server response (although this seems quite difficult to realize and the literature does not provide any example of such an attack). Although a delay is introduced to allow the OAD to process the data sent back to the client, this does not affect the detection itself: in fact, the delay, in the worst case of no output sent at all, is equal to the time value t.

It should be clear from the architecture that ATLANTIDES will never raise more false positives than its input NIDS. In fact, the output of the OAD is evaluated *only* when an alert has already been raised by the input NIDS: the OAD could classify the alert-related outgoing traffic by mistake as anomalous and then forward the alert as a true positive, but this would have happened in any case, if considering the output of the input NIDS only. Thus, the worst case is that a false positive is not suppressed, but no new false alert can be generated.

On the other hand, we have to discuss the possibility that ATLANTIDES will introduce additional false negatives (w.r.t. the input NIDS). This happens every time the OAD classifies an alert corresponding to a true attack as a false alert. False negatives are a common problem for alert verification systems. Because our solution bases its verification on an anomaly-based engine, the threshold used to discern outgoing traffic can be adjusted

manually by IT specialists to avoid false negatives (previous proposed solutions cannot be tuned in the same way, e.g. [69]).

Missing output response What we just described is the most common behaviour; nevertheless we have to take into account that there are attacks which, e.g., aim to disrupt completely the service or that, exploiting a buffer overflow, radically modify the normal execution. For instance, Chaboya et al. [30] experimentally verified that most of the buffer overflow attacks against an HTTP server do not produce any output after a successful attempt.

In this case, if the OAD does not detect *any* output related to the pre-alert raised by the NIDS, during the time window t, then the pre-alert is considered an incident and is forwarded to an IT security specialist. Although this could be considered rough, because the missing response could occur for reasons other than a successful attack (e.g., an internal error), the ATLANTIDES strategy does not introduce any additional false negatives/positives, since with a single NIDS (monitoring the incoming traffic) the alert would be forwarded anyway.

4.2.1 The Output Anomaly Detector

The OAD is basically a payload-based ANIDS, monitoring the output of a network service rather than the input of it. In our embodiment we choose to use POSEIDON as the OAD, because we are familiar with it and it gives better results than its leading competitor (see Section 3.2.1).

Using an SBS as OAD (e.g., Snort) would present serious limitations. First, an SBS uses signatures to detect anomalies, thus a new set of signatures has to be created. Creating and maintaining a set of signatures for outgoing traffic is a thorny and labor-intensive task, as these signatures heavily depend on local applications, and must be updated each time that modifications to the application output occur. Secondly, it is difficult to characterize the normal output of an application in terms of a signature: two instances of the same application could generate dissimilar output, as user behaviour influences it.

The fact that the OAD is anomaly-based (rather than signature-based) has various advantages. An anomaly-based OAD can adapt to the specific network environment/service and it does not require the definition of new signatures to detect anomalous output, working in an unsupervised way (after initial set up). The disadvantage of being anomaly-based is that our OAD needs an extensive (though unsupervised) *training* phase. However, our OAD (POSEIDON) can incorporate modifications in its model, without starting training over, thus easing the update process when changes occur.

Setting the OAD Threshold As we mentioned in Section 2.3.4, the threshold setting is an important task, usually accomplished manually. While running experiments with ATLANTIDES, we have found an approach to automatically set this parameter (IT personnel can later adjust it as necessary). In our experiments we address this issue:

Figures 4.2 and 4.3 show how different threshold values influence the effectiveness of ATLANTIDES.

To derive a quick way to set up this value, we consider the maximum distance value (t_{max}) between the analysed data and the model observed during the training phase, and calculated using the Mahalanobis distance function used by POSEIDON inside the OAD. Our experiments show that setting the threshold at $\frac{3t_{max}}{4}$, usually yields reasonably good results in terms of accuracy and completeness.

4.3 Benchmarks

To validate our architecture, we benchmark ATLANTIDES in combination with Snort as well as ATLANTIDES in combination with POSEIDON. To carry out the experiments, we employ two different data sets. First, we benchmark the system using a private data set. Secondly, we use the DARPA 1999 data set, as it has the advantage that it allows one to compare experiments. No other data set, containing sufficient data to perform verifiable benchmarks, is publicly available. The detection and false positive rates are calculated using the same approach introduced in Section 3.2.1.

4.3.1 Tests with a Private Data Set

To carry out our validation we consider a private data set we collected at the University of Twente: this is *data set A*. Data were collected on a public network for 5 consecutive working days (24 hours per day), logging only TCP traffic directed to (and originating from) a heavy-loaded web server (about 10 Gigabytes of total traffic per day). This web server hosts the department official web sites as well as student and research staff personal web pages: thus, the traffic contains different types of data such as static and dynamically generated HTML pages and, especially in the outgoing traffic, common format documents (e.g., PDF) as well as raw binary data (e.g., software executables). We did not inject any artificial attack. To see how the system behaves in the sub-optimal situation in which the IT security specialist does not have the time to clean up the training data set (a situation that is likely to occur often in practice), the data set was not made attack-free.

We focus on HTTP traffic because nowadays Internet attacks are mainly directed to web servers and web-based applications [66]: Kruegel et al. [73] state that web-based attacks account for 20%-30% from 1999 to 2004 in the Common Vulnerabilities and Exposures database [147]; Symantec Corporation [144] reports that, in the first-half of year 2008, 65% of total discovered vulnerabilities were related to web services and, during the same period, more than 60% of *easily exploitable vulnerabilities* (whenever the exploitation code is not needed or well-known) affected web applications. Symantec states that typical examples of easily exploitable vulnerabilities are SQL Injection and Cross-Site Scripting (XSS) attacks.

To train the anomaly-detection engines of both POSEIDON and the OAD on data

Protocol		**POSEIDON**	**POSEIDON + ATLANTIDES**
HTTP	DR	100%	100%
	FP	1683 (2,83%)	774 (1,30%)

Table 4.1: Comparison between POSEIDON stand-alone and POSEIDON in combination with ATLANTIDES using data set A; DR stands for detection rate (attack instance percentage), while FP is the false positive rate (packets and corresponding percentage); ATLANTIDES reduces false positives by more than 50% without affecting the detection rate (i.e., without introducing false negatives).

set A, we used a snapshot of the data collected during *working hours* (approximately 3 hours, or 1.8 Gigabytes of data, randomly chosen). The chosen training data set has not been pre-processed and made attack-free: thus it is possible that the model includes some malicious activity (that could negatively affect accuracy). For the same reason, we randomly chose another snapshot (approximately 1.8 Gigabytes of data) to benchmark POSEIDON stand-alone against POSEIDON in combination with ATLANTIDES.

An ABS can achieve a 100% detection rate using a very low threshold value, but this negatively affects the false positive rate too (as we mentioned in Section 4.1.3): we set the threshold of POSEIDON experimentally to achieve the best detection rate (at least one packet detected per attack instance) at the lowest false positive rate possible.

The alerts have been classified as follows: we found evidences of XSS and SQL Injection attacks [149] (and this is not surprising, accordingly to Symantec's report), plus some probes checking for well-known paths (33 attacks in total). Table 4.1 summarizes the results we obtained. We cannot compare ATLANTIDES in combination with Snort on data set A for the reason that Snort does not find any true attack to the system (Snort raises only false alerts): this is not surprising, since Snort has only few signatures devoted to SQL Injections and XSS attacks. By setting a high threshold value in ATLANTIDES we could have removed *all* the false positives, but this would give no indication of the completeness and accuracy of ATLANTIDES. Figure 4.2 shows detailed results of ATLANTIDES on data set A.

4.3.2 Tests with the DARPA 1999 Data Set

The testing environment of the DARPA 1999 data set contains several internal hosts that are attacked by both external and internal attackers: in our tests, we consider only inbound and outbound TCP packets that belong to attack connections against hosts inside the network 172.16.0.0/16, to replicate the experiments with POSEIDON (see Section 3.2.1). We focus on FTP, Telnet, SMTP and HTTP protocols. This is due to the fact that only these protocols, among the ones contained in this data set, provide us with a

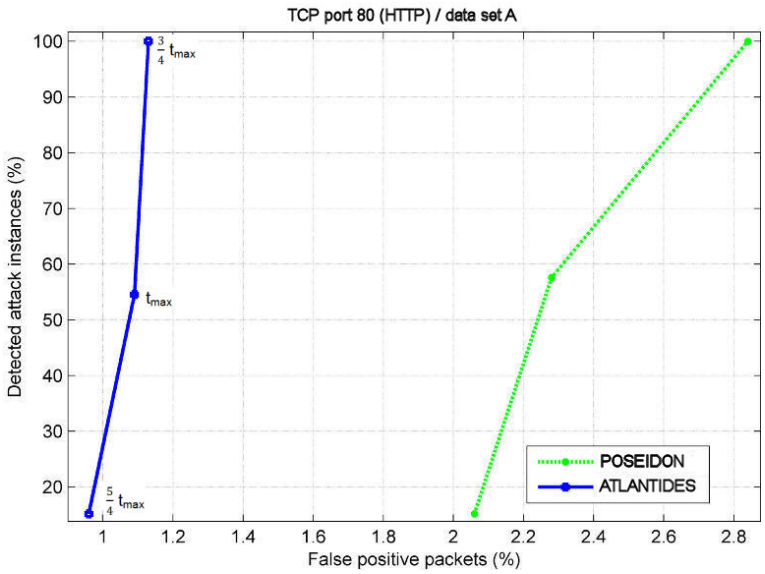


Figure 4.2: Detection rates for POSEIDON in combination with ATLANTIDES using data set A (HTTP protocol): the x-axis and y-axis present false positive rate (packets) and detection rate (attacks instances) respectively. ATLANTIDES presents always a lower false positive rate than POSEIDON, considering the same detection rate. It is possible to notice how different ATLANTIDES threshold settings (t_{max}) affect detection and false positive rates.

sufficient number of samples to train the OAD and, at the same time, allow us to compare our architecture with POSEIDON stand-alone.

We train the OAD of ATLANTIDES with the data of weeks 1 and 3 (attack-free): for each different protocol we use a different OAD instance. Afterwards, we test ATLANTIDES together with both POSEIDON and Snort using week 4 and week 5 traffic. In order to distinguish between true and false positives, we refer to the attack instance table provided by the DARPA data set authors.

Table 4.2 reports a comparison of the detection and false positive rates of Snort stand-alone (first column), Snort in combination with ATLANTIDES (second column), POSEIDON stand-alone (third column) and POSEIDON in combination with ATLANTIDES (fourth column).

In both cases, ATLANTIDES achieves a substantial improvement on the stand-alone system, neither affecting the detection rate nor introducing false negatives; ATLANTIDES reduces the false positive amount by at least 50% on every protocol benchmarked, except for the Telnet protocol when POSEIDON is analysing the input. Here we elaborate more about this discrepancy. Snort and POSEIDON employ two different detection engines, and they raise alerts when different conditions are met. Thus, an alert raised by POSEIDON is likely to be determined by the fact that during the training phase the system did not observe that input traffic. The same applies to the output, and even a slightly different

Protocol		Snort	Snort + ATLANTIDES	POSEIDON	POSEIDON + ATLANTIDES
HTTP	DR	59,9%	59,9%	100%	100%
	FP	599 (0,069%)	5 (0,00057%)	18 (0,0016%)	0 (0,0%)
FTP	DR	31,75%	31,75%	100%	100%
	FP	875 (3,17%)	317 (1,14%)	3303 (11,31%)	373 (1,35%)
Telnet	DR	26,83%	26,83%	95,12%	95,12%
	FP	391 (0,041%)	6 (0,00063%)	63776 (6,72%)	56885 (5,99%)
SMTP	DR	13,3%	13,3%	100%	100%
	FP	0 (0,0%)	0 (0,0%)	6476 (3,69%)	2797 (1,59%)

Table 4.2: Comparison between Snort stand-alone, Snort in combination with ATLANTIDES, POSEIDON stand-alone and POSEIDON in combination with ATLANTIDES using the DARPA 1999 data set: DR stands for detection rate (attack instance percentage), while FP is the false positive rate (packet percentage); ATLANTIDES reduces false positives by more than 50% most of the times, being close to zero in 3 tests, without affecting the detection rate (i.e., without introducing false negatives). ATLANTIDES is not applied to SMTP traffic in combination with Snort because in this case Snort raises no false positives.

input could determine a significant different output (think of the command "ps" or "ps -a", the latter usually brings up a longer list of running processes). The Telnet protocol has a great output variability, and normal traffic in the testing traffic can be mistaken; on the other hand, protocols like HTTP, FTP and SMTP present well-defined protocol schemas to exchange information between client and server.

4.4 Related Work

The problem of alert verification has been addressed using two different kinds of approaches: we have techniques for *identifying true positives*, and techniques for *identifying false positives*. The main difference between our work and the papers described below is that we take into account the outgoing traffic of the system.

Identifying true positives Kruegel and Robertson [69] introduce a plug-in for Snort to verify alerts: the plug-in integrates the Nessus vulnerability scanner into the Snort core. When an alert is raised, this is not immediately forwarded but it is firstly passed to the verification engine. Since every Snort signature comes with a unique identifier (assigned by CVE [147]), this index is used to check the presence of a corresponding Nessus attack script. If found, the script is executed against the target machine/network: the output is extracted and used to flag the alert as either true or false; an output cache is used to avoid further verification for the same alert/target. Although this approach is effective, there are

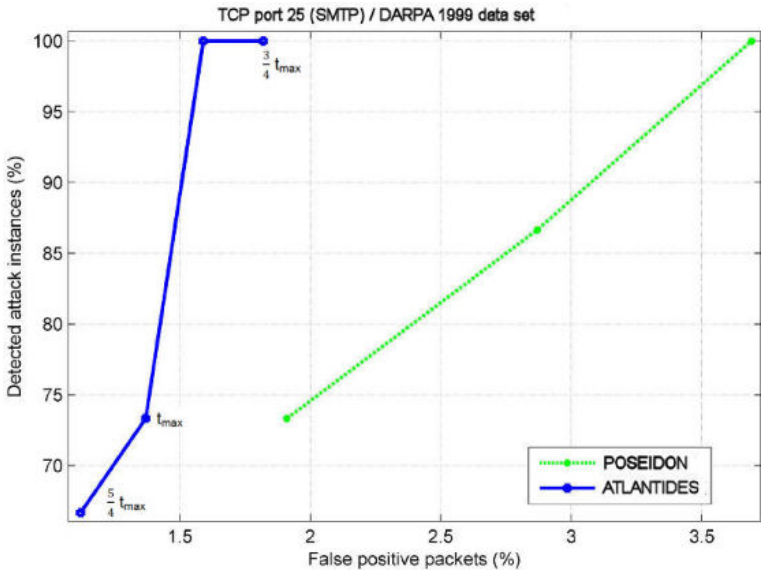


Figure 4.3: Detection rates for POSEIDON in combination with ATLANTIDES using DARPA 1999 data set (SMTP protocol): the x-axis and y-axis present false positive rate (packets) and detection rate (attacks instances) respectively. ATLANTIDES presents always a lower false positive rate than POSEIDON, considering the same detection rate. It is possible to notice how different ATLANTIDES threshold settings affect detection and false positive rates.

several drawbacks: one has to keep the Nessus attack script database up to date, and this approach works only for a SNIDS, while ATLANTIDES can work with both types and in a complete automatic way (i.e., no manual updates are needed).

Ning et al. develop a model [98] and an intrusion-alert correlator [94] to help human analysts during the alert verification phase. This work is based on the observation that most attacks consist of several related stages, with the early stages preparing for the later ones. Hyper-alert correlation graphs are used to represent correlated alerts in an intuitive way. However, this correlation technique is ineffective when attackers use a different (yet not spoofed) IP source address at each attack step. Ning and Cui [94] demonstrate the effectiveness of this approach when applied on a small data set (due to the exponential complexity of hyper-alert graphs): in [95, 97] the same authors present other utilities they developed to facilitate the analysis of large sets of correlated alerts, and report some benchmarks employing network traffic used during the DEFCON 8 Capture the Flag (CTF) event [133]. ATLANTIDES does not present the same limitations on data set size.

Lee and Stolfo [79] develop a hybrid network and host-based framework based on data mining techniques, such as sequential pattern mining and episode rules, to address the problem of improving attack detection while maintaining a low false positive rate. The system detects attacks by combining different models and comparing them with actual

traffic features. Benchmarks have been conducted using the DARPA 1998 data set [82]: the detection rate for different attack typologies has a minimum value of 65% with a false positive rate always below 0.05%. Since Lee and Stolfo use a different data set, we cannot compare directly the two approaches: however, we note that our approach does not use information collected from the operating system hosting the monitored network service(s), thus ATLANTIDES can work on-line without affecting the host performance.

Identifying false positives Pietraszek [101] tackles the problem of reducing false positives by introducing an alert classifier system (**ALAC**, **A**daptive **L**earner for **A**lert **C**lassification) based on machine learning techniques. During the training phase, the system classifies alerts into true and false positives, by attaching a label from a fixed set of user-defined labels to the current alert. Then, the system computes an extra parameter (called *classification confidence*) and presents this classification to a human analyst. The analyst's feedback is used to generate training examples, used by the learning algorithm to build and update its classifiers. After the training phase, the classifiers are used to classify new alerts. To ensure the stability of the system over time, a sub-sampling technique is applied: regularly, the system randomly selects n alerts to be forwarded to the analyst instead of processing them autonomously. This approach relies on the analyst's ability to classify alerts properly and on her availability to operate in real-time (otherwise the system will not be updated in time); we believe that these (demanding) requirements can be considered acceptable for an SNIDS (where the analyst can easily inspect both the signature and network packet(s) that triggered the alert), but it could be difficult to perform the same analysis with an ANIDS. Benchmarks conducted over the 1999 DARPA data set, using Snort to generate alerts, show an overall false positive reduction of over 30% (details on single attack protocols are not given).

The main differences between ALAC and ATLANTIDES are: (a) ALAC does not consider the outgoing traffic, and (b) ALAC relies heavily on the expertise and the presence of an analyst (in ATLANTIDES, all the IT specialist has to do is to set the thresholds).

Julisch [61] presents a semi-automatic approach, based on techniques which discover frequently occurring episodes in a given sequence, for identifying false positives based on the idea of *root cause*: an alert root cause is defined as "the reason for which it occurs". The author observes that in most environments, it is possible to identify a small number of highly predominant (and persistent) root causes: thereby removing such root causes drastically reduces the future alert rate. Benchmarks conducted on a log trace from a commercial SNIDS deployed in a real network show a reduction of 87% of false positives. No further details are given about the testing condition, network topology or traffic typology. We cannot compare directly this approach with ATLANTIDES because the data used by the author is private, nevertheless we can notice that this approach is applicable only to signature-based systems, while ATLANTIDES is effective with anomaly-based systems too.

Analysing output traffic The idea of analysing (and correlating) the output of a (possible) compromised system has been used before in the context of worm detection.

Gu et al. [53] scan the output traffic for specific port numbers. When an anomaly has been detected in the incoming traffic directed to a certain destination service port, their system starts monitoring the output traffic to check whether the host tries to contact other systems using the same destination service port: if this is the case then the system is probably infected by a worm. Wang et al. [119] proceed in a similar way, comparing outgoing to incoming traffic, looking for similarities: when an anomaly has been detected in the incoming traffic, the anomalous traffic is cached and compared to subsequent outgoing traffic (to detect polymorphic worms). A successful match indicates that the host has been infected and that the worm is trying to replicate itself, infecting other hosts. Any other kind of attack will not be handled by the system. In contrast, our solution presents a general architecture to carry out a complete anomaly detection on the output to reduce false positives of any NIDS placed on the input channel. Indeed we have shown that our architecture works well in combination with both a signature and an anomaly-based input NIDS.

4.5 Conclusion

We present ATLANTIDES, a system for automatic alert verification exploiting in a structured way the detection of anomalies in the output traffic of a system. ATLANTIDES can be used to reduce false positives both in signature- and anomaly-based NIDSs.

The core of ATLANTIDES consists of an output anomaly detector (OAD), a modified version of the POSEIDON core, which compares output traffic with a model it has created during the training phase. To reduce *false positives* on the input NIDS (be it signature- or anomaly-based) monitoring the incoming traffic, ATLANTIDES checks if the communication raising an alert in the input NIDS actually produces an anomaly in the outgoing traffic too. In this case (or in the case no output is generated), the alert is forwarded to the IT specialist, otherwise it is discarded. The fact that the OAD is anomaly-based (rather than signature-based) allows it to adapt to the specific network environment/service, and to work in an unsupervised way (at least, after the setup). Anomaly-based systems typically use a distance function and a threshold to discern anomalous from licit traffic. We introduce a simple heuristic to set the ATLANTIDES threshold in an automatic, though effective, way, to further ease the management for IT security specialists (which can in case adjust the threshold value). Benchmarks on a private data set and on the DARPA 1999 data set show that ATLANTIDES reduces false positives between 50% and 100% in most of the cases, without introducing any extra false negative, thereby improving the usability by reducing the work burden for users, and easing the NIDS management.

In the following chapter, we describe the internals of Sphinx, a web- anomaly-based intrusion detection system. Sphinx, unlikely POSEIDON (and ATLANTIDES), analyses specific application layer data, i.e., the HTTP protocol. By doing so, Sphinx improves the detection and false alert rates achieved by POSEIDON.

Chapter 5

Boosting Web Intrusion Detection Systems by Inferring Regular Languages*

In the last decade, the Internet has quickly changed from a static repository of information into a practically unlimited on-demand content generator and service provider. This evolution is mainly due to the increasing success of so-called web applications (later re-branded web services, to include a wider range of services). Web applications make it possible for users to access diverse services from a single web browser, thereby eliminating reliance on tailored client software.

Although ubiquitous, web applications often lack the protection level one expects to find in applications that deal with valuable data: as a result, an attacker intent on acquiring information such as credit card or bank details will often target a web application. Web applications are affected by a number of security issues, primarily due to a lack of expertise in the programming of secure applications. To make things worse, a web application is typically built upon multiple technologies from different sources (such as the open-source community), making it difficult to assess the resulting code quality. Other factors affecting the (in)security of a web application are its size, complexity and extensibility. Even with high quality components, the security of a web application can be compromised if the interactions between those components are not properly designed and implemented, or an additional component is added at a later stage without due consideration (e.g., a vulnerable web application could grant an attacker the control of another system which communicates with it.)

An analysis of the Common Vulnerabilities and Exposures repository [147] conducted

*This chapter is a minor revision of the paper with the same title published in the Proceedings of On the Move to Meaningful Internet Systems Confederated International Conferences (OTM '08), volume 5332 of LNCS, pages 938 - 955, Springer, 2008.

by Robertson et al. [106] shows that web-related security flaws account for more than 25% of the total number of reported vulnerabilities from year 1999 to 2005 (this analysis does not take into account vulnerabilities discovered in web applications developed internally by companies.) The Symantec 2008 Internet Security Threat Report [144] states that most of the *easily exploitable vulnerabilities* (those requiring little knowledge and effort on the attacker side) are related to web applications, in particular SQL Injection and Cross-site Scripting attacks. These statistics show that web applications have become the Achilles' heel in system and network security.

Contribution We present a new approach for anomaly detection devised to detect data-flow attacks to web applications (attacks against the work flow [19, 36] are not taken into consideration) and we introduce Sphinx, an ABS based on it. We exploit the fact that, usually, most of the parameters in HTTP requests present some sort of regularities: by considering those regularities, we divide parameters into "regular" and "irregular" (whose content is highly variable) ones; we argue that, for "regular" parameters, it is possible to exploit their regularities to devise more accurate detection models. We substantiate this with a number of contributions:

- We introduce the concept of "positive signatures", to carry out anomaly-detection on the "regular" parameters. We first infer human-readable regular expressions by analysing the parameter content, then we generate *positive signatures* matching *normal* inputs.
- We build a system, Sphinx, that implements our algorithm to infer regular expressions automatically and to generate positive signatures; positive signatures are later used by Sphinx to build automaton-based detection models to detect anomalies in the corresponding "regular" parameters. For the parameters we call "irregular", Sphinx analyses their content using an adapted version of our NIDS POSEIDON (as it would not be "convenient" to generate a positive signature).
- We extensively benchmark our system against state-of-the-art IDSs such as WebAnomaly [73], Anagram [120] and POSEIDON.

We denote a generated signature as a "positive signature", following the idea that it is as flexible as a signature used by an SBS but matches positive inputs (in contrast with an usual signature used to match malicious inputs). Differently from mathematical and statistical models, a positive signature does not rely on any data frequency/presence observation or threshold. As shown by our benchmarks, positive signatures successfully detect attacks with a low false positive rate for "regular" parameters. Positive signatures aim to improve the usability of the detection engine as well, as users can later modify and improve any positive signatures (e.g., to eliminate some noisy traffic, or to include a licit input that was not observed during the inference).

Our new approach merges the ability of detecting new attacks without prior knowledge (a common feature of an ABS) with the possibility of easily modifying/customizing the behaviour of part of the detection engine (a common feature of an SBS).

Sphinx works with *any* web application, protecting a custom-developed (or close-source) web application as well. By working in an automatic way, Sphinx requires little security knowledge from system administrators, however expert users can easily review regular expressions and make modifications.

We perform thorough benchmarks using three different data sets: the benchmarks show that Sphinx performs better than state-of-the-art ABSs both in terms of detection and false positive rates as well as presenting a better learning curve than competing systems.

5.1 Detecting Data-flow Attacks to Web Applications

Let us describe how a web application handles user inputs. A web application generates an output in response to a *user request*, which is a string containing a number of *parameter names* and their respective *parameter values* (for the sake of simplicity we can disregard parameter-less HTTP requests, as attackers cannot inject attack payloads.) RFC 2616 [49] defines the structure and the syntax of a request with parameters (see Figure 5.1).

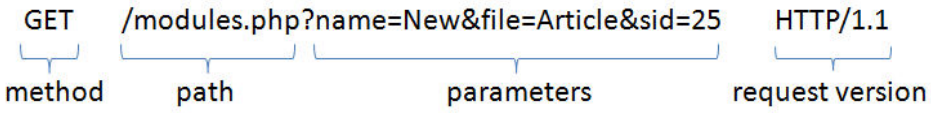


Figure 5.1: A typical HTTP (GET) request with parameters.

We can discard the request version and – for the sake of exposition – the method, since it is just a matter of implementation and our approach works with both GET and POST requests. Of interest to us is the presence of a path, a number of parameter names and of their respective values (in Figure 5.1 the parameter names are “name”, “file” and “sid” and their respective values are “New”, “Article” and “25”). The set of parameters is finite. A value can be any string, though not all of the strings will be accepted. Since no type is defined, the semantics of each parameter is implicitly defined within the context of the web application and such parameters are usually used in a consistent manner (i.e., their syntax is fixed). In the sequel, we refer to the natural projection function: given an input $i = path?p_1 = v_1 \& p_2 = v_2 \& \ldots \& p_n = v_n$, we define $p_n(i) = v_n$ as the function extracting the value of parameter p_n from input i.

5.1.1 Exploiting Regularities

Our claim is that, in the context of web applications, it is possible to exploit the regularities which are not present in other settings to define and build M_A based on the inference of regular automata, which leads to the definition of an IDS that is more effective (yet simpler) than state-of-the-art systems.

Definition 13 *A Sphinx automaton is a mapping from strings on a given alphabet to the set* $\{yes, no\}$ *such as* $\alpha : Strings \rightarrow \{yes, no\}$*; the language it accepts corresponds to* $\{s \in Strings \mid \alpha(s) = yes\}$*. Given a finite set of strings* I *it is easy to construct* α_I*, the automaton which recognizes exactly* I*.*

Commonly, an ABS builds (and uses) a single model M to analyse data. Our proposal takes advantage of the fact that requests to any web application present a fixed syntax, consisting of a sequence of *parameter* = *value* pairs, and instead of building a single model to analyse the input, it builds an *ad hoc* model M_n for each parameter p_n (in practice, we create a separate model for many – not all – parameters.) As already observed by Kruegel and Vigna in [72], this allows to create a more faithful model of the application input. The idea is that of defining M_A implicitly by electing that $i \in M_A$ iff for each parameter n we have that $p_n(i) \in M_n$ (or that $p_n(i)$ is empty).

5.1.2 Regular and Irregular Parameters

So we first divide the parameters in two groups: the *regular parameters* and the *irregular parameters*. The core of our idea is that for the *regular* parameters it is better to define M_n as a *regular language* rather than using state-of-the-art anomaly-based systems. By "better" we mean that this method yields (a) lower false positive rates, (b) same (or higher) detection rates, and (c) a shorter learning phase. We support our thesis by presenting an algorithm realizing this.

For each regular parameter, we build a model using a combination of abstraction and regular expression inference functions that we are going to explain in the following section: we call this the *regular-text* methodology, following the intuition that we can build a model for a parameter that is usually filled by data with a well-defined format (e.g., integer numbers, dates, user session cookies, etc). For the *irregular* parameters we use classical anomaly-based techniques, i.e., n-gram analysis (see Section 3.1.2): we call this the *raw-data* methodology, since it is meant to be more suitable for building the model of parameters containing irregular data (e.g., pieces of blogs or emails, images, binary data etc).

5.2 Sphinx Detection Engine

To substantiate our claims, that it is possible to exploit the regularities in HTTP requests to define a more effective IDS, we present Sphinx: an anomaly-based IDS specifically tailored to detect attacks to a web application data flows. Let us see how it works. Sphinx needs to builds its detection models, for both regular and irregular parameters: we discuss each in turn.

5.2.1 Building the Model

We first outline how we build the model M_A of the application given a *training set* DS; DS is a set of inputs (i.e., HTTP requests), which we assume does not contain fragments of attacks. Typically, DS is obtained by making a dump of the application input traffic during a given time interval. Then, an off-line cleaning step follows (i.e., the malicious traffic is removed) using a combination of signature-based intrusion-detection techniques and manual inspections (see Section 2.3.3).

During the *first* part of the training, we discover the set of parameters used by the web application: DS is scanned a first time and the parameters $\{p_1, \ldots, p_n\}$ are extracted and stored. We call $DS_n = \{p_n(i) \mid i \in DS\}$ the training set for the parameter p_n (i.e., the projection of D_S on the parameter p_n).

In the second step we divide the parameters into two classes: the *regular* parameters (for which we use the new anomaly detection algorithms based on regular expressions) and the irregular parameters. In practice, to decide whether a parameter is "regular", in the sequel we use a simple a-priori syntactic check: if at least the 10% of the samples in DS_n contains occurrences of more than 5 distinct non-alphanumeric characters, we say that p_n is an irregular parameter, otherwise it is a regular one. This criterion for separating (or, better, defining) the regular parameters from the irregular ones appears arbitrary. However, our analysis shows that it gives good results (see Table 5.7 in Section 5.3). We impose a minimum amount of samples (10%) to present more than 5 distinct non-alphanumeric characters to prevent the Sphinx engine from classifying a parameter as "irregular" because of a few anomalous samples. An attacker could in fact exploit this to force the system to classify any parameter as "irregular", by just carefully crafting a single HTTP request.

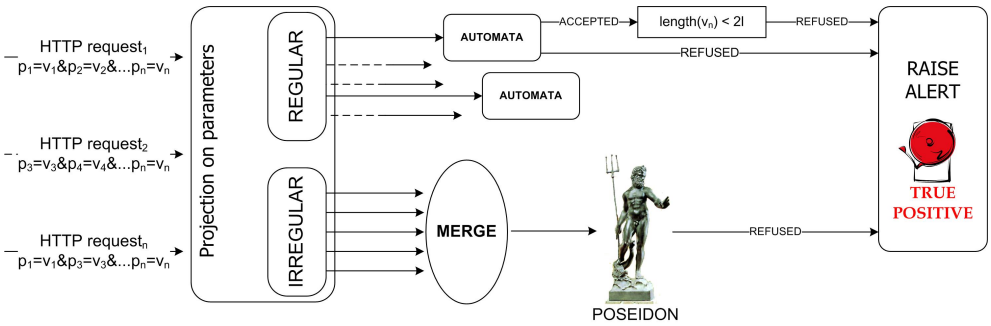


Figure 5.2: Sphinx internals.

In the last step of the training phase we build a model M_n for each of the regular parameters p_n, using the training set DS_n. The *irregular* parameters, in turn, are again grouped together and for them we build a unique model: we could also build a single model per irregular parameter, but this would slow down the learning phase, which is already one of the weak spots of classical anomaly detection techniques.

5.2.2 The Regular-text Methodology

This represents the most innovative aspect of our contribution. The *regular-text* methodology is designed to build a simple model of the "normal" input of the regular parameters. This model is represented by a regular expression and anomalies are detected by the derived finite automaton. We illustrate this methodology by presenting two algorithms realizing it: the first algorithm is called the *simple regular expression generator* (SREG) and it is meant to illustrate the fundamental principles behind the construction of such regular language; the second algorithm is called *complex regular expression generator* (CREG), and can be regarded as a further development of the first one. Here we should mention that standard algorithms to infer regular expressions (see [48] for a detailed overview) cannot be used for intrusion detection because they infer an expression matching exactly the strings in the training data set only, while we need to match an abstraction of it.

5.2.2.1 Simple Regular Expression Generator

Here we introduce our first algorithm. We have a training set DS_n (the training set for parameter n) and we want to build a model M_n of the parameter itself, and a decision procedure to determine for a given input i whether $p_n(i)$ is in M_n or not.

Our first algorithm to generate M_n is based on applying two abstractions to DS_n. The first abstraction function, abs_1, is devised to abstract all letters and all digits (other symbols are left untouched), and works as follows:

$$abs_1(c_1 \dots c_n) = abs_1(c_1), \dots, abs_1(c_n)$$

$$abs_1(c_i) = \begin{cases} \text{“}a\text{”}, & c_i \in \{\text{“}a\text{”}, \dots, \text{“}Z\text{”}\} \\ \text{“1”}, & c_i \in \{\text{“0”}, \dots, \text{“9”}\} \\ c_i & otherwise \end{cases}$$

Thus abs_1 abstracts alphanumerical characters while leaving non-alphanumerical symbols untouched (for the reasons we clarified in Section 5.1.1). The reason for this choice is that, in the context of web applications, the presence of "unusual" symbols, or a concatenation of them, could indicate the presence of attack payloads (Robertson et al. [106] employ a similar approach).

The second abstraction we use is actually a contraction:

$$abs_2(c_1 \dots c_n) = \begin{cases} abs_2(c_2 \dots c_n)\ if\ c_1 = c_2 = c_3 = \text{“}a\text{”}\ or\ c_1 = c_2 = c_3 = \text{“1”} \\ c_1 c_1\ abs_2(c_3 \dots c_n)\ if\ c_1 = c_2 \neq c_3\ and\ c_1 = \text{“}a\text{”}\ or\ c_1 = \text{“1”} \\ c_1\ abs_2(c_2 \dots c_n)\ if\ c_1 \neq c_2\ or\ c_1 = c_2\ and\ c_1 \neq \text{“}a\text{”}\ and\ c_1 \neq \text{“1”} \end{cases}$$

Intuitively, abs_2 collapses all strings of letters (resp. digits) of length not less than two onto strings of letters (resp. digits) of length two. Again, symbols are left untouched,

Input	$abs_1(i)$	$abs_2(i')$
11/12/2007	11/11/1111	11/11/11
addUser	aaaaaaa	aa
C794311F-FC92-47DE-9958	a111111a-aa11-11aa-1111	a11a-aa11-11aa-11

Table 5.1: Some examples of application of abstractions abs_1 and abs_2 on different inputs.

as they may indicate the presence of an attack. Table 5.1 provides some examples of application of abs_1 and abs_2 on different input strings.

These two abstraction algorithms are enough to define our first model, only one detail is still missing. If the samples in DS_n have maximum length say l, then we want our model M_n to contain strings of maximum length $2l$: an input which is much longer than the samples observed in the training set is considered anomalous. In fact, the attacker could encode the attack payload in order to avoid any non-alphanumerical symbol and evade detection (e.g., using URL, hexadecimal or Unicode encoding). However, the resulting payload is likely to be much longer than the original payload. During our experiments we have observed that by setting the maximum length value to $2l$ we were able to detect encoded attack payloads.

Definition 14 *Let DS_n be a training set. Let $l = max\{|x| \mid x \in DS_n\}$. We define the* simple regular-text model *of DS_n to be*

$$M_n^{simple} = \{x \mid |x| \leq 2l \wedge \exists y \in DS_n \; abs_2(abs_1(x)) = abs_2(abs_1(y))\}$$

During the *detection* phase, if $p_n(i) \notin M_n^{simple}$ then an alert is raised. The decision procedure for checking whether $i \in M_n^{simple}$ is given by the finite automaton $\alpha_{M_n^{simple}}$ that recognizes M_n^{simple}. $\alpha_{M_n^{simple}}$ is implemented using a (unbalanced) tree data-structure, therefore adding a new node (i.e., a previously unseen character) costs $O(l)$, where l is the length of the longest observed input. The complexity of building the tree for n inputs is therefore $O(n \cdot l)$. The decision procedure to check, given an input i, if $p_n(i) \in M_n$ has complexity $O(l)$. To simplify things, we can represent this automaton as a regular expression.

Training sets	$abs_2(abs_1(i))$	**SREG**
01/01/1970 30/4/85 9/7/1946	11/11/11 11/1/11 1/1/11	1(1/1(1/11|/11)|/1/11)
41E44909-C86E-45EE-8DA1 0F786C5B-940B-4593-B96D 656E0AB4-B221-422F-92AC	11a11-a11a-11aa-1aa1 1a11a1a-11a-11-a11a 111a1aa1-a11-11a-11aa	1(1(a11-a11a-11aa-1aa1| 1a1aa1-a11-11a-11aa)| a11a1a-11a-11-a11a)

Table 5.2: Examples of how SREG works on different input sets.

5.2.2.2 Complex Regular Expression Generator

The simple SREG algorithm is effective for illustrating how regular expressions can be useful in the context of anomaly detection and how they can be used to detect anomalies in regular parameters. Nevertheless we can improve on SREG in terms of FPs and FNs by using an (albeit more complex) algorithm, which generates a different, more complex model.

The algorithm goes through two different phases. In the *first phase* each DS_n is partitioned in groups with common (shared) prefixes or suffixes (we require at least 3 shared characters, to avoid the generation of useless regular expressions.)

In the *second phase*, the algorithm generates a regular expression for each group as follows. First, it applies abstractions abs_1 and abs_2 on each stored body (we call the *body* the part of the input obtained by removing the common prefix or suffix from it: prefixes and suffixes are handled later.) Secondly, it starts to search for common symbol patterns inside bodies. Given a string s, we define the symbol pattern of s as the string obtained by removing all alphanumerical characters from s.

As bodies could contain different symbol patterns, non-trivial patterns (i.e., patterns of length greater than one) are collected and the pattern matching the highest number of bodies is selected. If some bodies do not match the selected pattern, then the same procedure is repeated on the remaining bodies set till no more non-trivial patterns can be found.

For each non-trivial symbol pattern discovered during the previous step, the algorithm splits each body into sub-strings according to the symbol pattern (e.g., $s = s' - s'' - s'''$ is split in $\{s', s'', s'''\}$ w.r.t. the pattern). Corresponding sub-strings are grouped together (e.g., having strings s_1,s_2 and s_3 the algorithm creates $g1 = \{s'_1, s'_2, s'_3\}$, $g2 = \{s''_1, s''_2, s''_3\}$ and $g3 = \{s'''_1, s'''_2, s'''_3\}$) and a regular expression is generated for each group. This regular expression is then modified according to some heuristics to match also similar strings. Regular expressions are then merged with the corresponding symbol pattern, and any previously found prefix or suffix is eventually added (e.g., $re = (prefix)re_{g1} - re_{g2} - re_{g3}$). Table 5.3 depicts an example of the different steps of CREG.

Training set	**Symbol Pattern**	**Shared Pattern**	**Intermediate Regular Expression**	**Resulting Regular Expression**
al.ias@atwork.com	[. @ .]	[@ .]	$(a^+ [.])^+@(a^+).(a^+)$	$((a\|1)^+ [. \| \_])^+@(a^+ [-])^+.(a^+)$
3l_1t3@hack.it	[_ @ .]		$((a\|1)^+ [\_])^+@(a^+).(a^+)$	
info@dom-ain.org	[@ - .]		$(a^+)@(a^+ [-])^+.(a^+)$	

Table 5.3: Examples of pattern selection, generated regular expressions for samples and the resulting one.

Finally, for bodies which do not share any non-trivial symbol pattern with the other members of the group, a dedicated general regular expression is generated. Table 5.4 shows some examples of generated regular expressions for different sets of strings.

Training sets	**Pattern**	**Regular Expression**
01/01/1970 30/4/85 9/7/1946	[/ / /]	$(1^+/1^+/1^+)$
addUser deleteUser viewUser	N/A	(a^+)User
41E44909_C86E_45EE_8DA1 0F786C5B-940B-4593-B96D 656E0AB4-B221_422F-92AC	Only trivial patterns $\Rightarrow$ general regular expression	$((a\|1)^+[-\|\_])^+$

Table 5.4: Examples of how CREG works on different input sets.

Given the resulting regular expression (i.e., the positive signature), we build a finite automaton accepting the language it represents. The automaton is built in such a way that it accepts (as in the case of SREG) only strings of length less than $2l$. In the detection phase, if an input is not accepted by the automaton, an alert is raised.

5.2.2.3 Effectiveness of Positive Signatures

Because of the novelty of our approach, let us see some concrete examples regarding the potential of positive signatures.

Think of a signature such as "id=1^+", accepting numeric values: the parameter id is virtually protected from any data-flow attack, since only digits are accepted. When we consider more complex signatures, such as "email=$((a|1)^+$ [. | _])$^+$@$(a^+$ [-])$^+$.(a^+)" (extracted from Table 5.3), common attack payloads would be detected by the automaton derived from the regular expression, as they require the injection of different symbol sets and in different orders.

One could argue that it could be sufficient (and simpler) to detect the presence of typical attack symbols (having them classified and categorized) or, better, the presence of a symbol set specific to an attack (e.g., " ' ",", " and " - " for a SQL Injection). However, a certain symbol set is not said to be harmful per se, but it must be somehow related to the context: in fact, the symbol " , " used in SQL Injection attacks, can also be found in some representations of real numbers. A positive signature, in contrast with usual state-of-the-art anomaly detection approaches, provides a sort of context for symbols, thus enhancing the detection of anomalies.

For instance, by April 2009, the CVE database contains more than 3000 SQL Injection and more than 4000 Cross-site Scripting attacks but only less than 250 path traversal and less than 400 buffer overflow attacks (out of a total of more than 30000 entries). Most of the SQL Injection attacks happen to exploit "regular" parameters (integer-based), where

the input is used inside a "SELECT" statement and the attacker can easily add a crafted "UNION SELECT" statement to extract additional information such as user names and their passwords. The same reasoning applies to Cross-site Scripting attacks. Positive signatures can significantly enhance the detection of these attacks.

5.2.3 The Raw-data Methodology

The raw-data methodology is used to handle the "irregular" parameters. For them, using regular automata to detect anomalies is not a good idea: the input is so heterogeneous that any automaton devised to recognize a "reasonable" super set of the training set would probably accept any input. Indeed, for this kind of heterogeneous parameters we can better use a classical anomaly detection engine, based on statistical content analysis (e.g., n-gram analysis). In the present embodiment of Sphinx we use our own POSEIDON (which performs well in our benchmarks, see Section 3.2.1), but we could have used any other payload-based ANIDS.

5.2.4 Using the Model

When the training phase is completed, Sphinx switches to detection mode. As a new HTTP request comes, parameters are extracted applying the projections $p_1, \ldots, p_n$. Sphinx stores, for each parameter analyzed during the training phase, information regarding the model to use to test its input. For a parameter that was labelled as "regular", the corresponding automaton is selected. If it does not accept the input, then an alert is raised. If the parameter was labelled "irregular", the content is analyzed using the adapted version of POSEIDON (which uses a single model for all of the irregular parameters). If the content is considered to deviate from the "normal" model, an alert is raised. Sphinx raises an alert also in the case a parameter has never been analysed before and suddenly materializes in a HTTP request, since we consider this eventuality as an attempt to exploit a vulnerability by an attacker.

5.2.4.1 Editing and Customizing Positive Signatures

One of the most criticized disadvantages of an ABS lies in its "black-box" approach (see Section 2.2.2).

A signature, as a general rule, provides more freedom to customize the detection engine behaviour. The use of positive signatures opens the new possibility of performing a thorough tuning for an ABS, thereby modifying the detection models of regular parameters. Classical anomaly-based detection systems do not offer this possibility as their models aggregate collected information, making it difficult (if not impossible) to remove/add arbitrary portions of data.

Positive signatures allow IT specialists to easily (and quickly) modify or customize the detection models when there is a need to do so (see Table 5.5), like when (1) some

Positive Signature	Problem	Action
id=1^+ | (a^+ ['|,|-|;])$^+$	The payload of a SQL Injection attack was included in the training set	The IT specialist manually changes the signature ⇒ id=1^+
date=1^+/1^+/1^+	A new input ("19-01-1981") is observed after the training phase, thereby increasing the false positive rate	The IT specialist re-trains the detection model with the new input and the positive signature ⇒ date=1^+/1^+/1^+ | 1^+-1^+-1^+ is automatically generated

Table 5.5: Some examples of signature customization.

malicious traffic, that was incorporated in the model during the training phase, has to be purged (to decrease the false negative rate) and (2) a new (previously unseen) input has to be added to the model (to decrease the false positive rate).

5.3 Benchmarks

The quality of the data used in benchmarks (and the way it was collected) greatly influences the number of successfully detected attacks and false alerts: test data should be representative of the web server(s) to monitor, and the attack test bed should reflect modern attack vectors. Presently the only (large) public data set for testing intrusion-detection systems is the DARPA data set, dating back to 1999. Although this data set is still widely used (since public data sets are scarce), it presents significant shortcomings that make it unsuitable to test our system: e.g., only four attacks related to web (and most of them target web server vulnerabilities) are available and traffic typology is outdated (see [85, 87] for detailed explanations about its limitations). So, to carry out our experiments we collected three different data sets from three different sources: real production web sites that strongly rely on user parameters to perform their normal activity.

Data set	Web Application	# of samples (HTTP requests)
DS_A	PostNuke	∼460000 (1 month)
DS_B	(Private) User forum	∼290000 (2 weeks)
DS_C	CS department's web site (CGI & PHP scripts)	∼85000 (2 weeks)

Table 5.6: Collected data sets: code name for tests, source and number of samples.

The first data set is a deployment of the widely-known content management system PostNuke. The second comes from a (closed-source) user forum web application, and

it contains user messages sent to the forum, which present a variable and heterogeneous content. The third data set has been collected from the web server of our department, where PHP and CGI scripts are mainly used. Each data set contains both GET and POST requests: the Sphinx engine can interpret the body of POST requests as well, since only the request syntax changes from a GET request, but the content, in case of regular parameters, looks similar. In case of encoded content (for instance, a white space is usually encoded as the hexadecimal value "%20"), the content is first decoded by Sphinx engine and then processed.

We collected a number of samples sufficient to perform extensive training and testing (never less than two weeks of traffic and in one case a month, see also Table 5.6). Data used for training has been made attack-free by using Snort to remove well-known attacks and by manually inspecting to purge remaining noise.

Determining a criterion for regular and irregular parameters In Section 5.2.1 we introduce the criterion, based on the number of unique symbols in the content of each parameter, used by the detection engine to determine when a parameter is considered either regular (validated with a positive signature) or irregular (validated with an adapted version of the detection engine of POSEIDON).

Intuitively, we want to have as many regular parameters as possible, because a positive signature offers a higher degree of usability for users than the POSEIDON detection engine. Hence we should select a high value of unique symbols that are allowed in the parameter content. However, the higher the number of allowed unique symbols, the less precise the generated positive signature is (and the higher the chance to consider an attack as licit traffic, generating a false negative). Parameters which usually carry large portions of data are likely to have a high number of unique symbols (think of a blog message with punctuation marks).

So, we chose the value of unique symbols allowed as follows. For each data set, we first counted how many unique symbols each parameter contains. Because attacks usually contain a good deal of symbols, we then manually inspected the content of parameters with at least 3 unique symbols, to verify that the symbol set for those parameters does not overlap with any attack symbol set. The attack symbol sets are generated using attack test beds which contain SQL Injection, Cross-site Scripting and Path Traversal attacks,

	# unique symbols						
	1	2	3	4	5	6	7 or more
DS_A	39 (N)	45 (N)	47 (N)	50 (N)	55 (N)	61 (N)	62 (Y)
DS_B	51 (N)	58 (N)	60 (N)	75 (N)	78 (N)	80 (Y)	84 (Y)
DS_C	321 (N)	325 (N)	330 (N)	332 (N)	334 (N)	339 (Y)	341 (Y)

Table 5.7: Number of regular parameters when allowing a certain number of unique symbols. In brackets we report whether an overlap between any parameter symbol set and any attack symbol set occurs (Y) or not (N).

and contain at least 3 unique symbols). Our analysis indicates that by allowing up to 5 unique symbols in the parameter symbol sets there are no overlaps with attack symbol sets. Table 5.7 reports our findings.

Users can adjust the value for the unique symbols in regular parameters to generate more precise positive signatures, and therefore reducing the chance of missing a real attack.

5.3.1 Comparative Benchmarks

To test the effectiveness of Sphinx, we compare it to three state-of-the-art systems, which have been either developed specifically to detect web attacks or have been extensively tested with web traffic.

First, WebAnomaly (Kruegel et al. [72]) combines five different detection models, namely attribute length, character distribution, structural inference, attribute presence and order of appearance, to analyze HTTP request parameters. Second, Anagram (Wang et al. [120]) uses a Bloom filter to store any n-gram (i.e., a sequence of bytes of a given length) observed during a training phase, without counting the occurrences of n-grams. During the detection phase, Anagram flags as anomalous a succession of previously unseen n-grams. Although not specifically designed for web applications, Anagram has been extensively tested with logs captured from HTTP servers, achieving excellent results. We set the parameters accordingly to authors' suggestions to achieve the best detection and false positive rates. Third, our own POSEIDON, the system we adapted to handle raw-text parameters in Sphinx. POSEIDON, during our previous experiments (see Section 3.2.1), showed a high detection rate combined with a low false positive rate in tests related to web traffic, outperforming the leading competitor.

We divide tests into two phases. We compare the different engines first by considering only the "regular" parameters. Later, we consider full HTTP requests (with both "regular" and "irregular" parameters).

The goal our tests is twofold. Next to the effectiveness of Sphinx, we are also interested in testing the *learning rate*: any anomaly-based algorithm needs to be trained with a certain amount of data before it is able to correctly flag attacks without generating a massive flow of false alerts. Intuitively, the longer the training phase is, the better the IDS should perform. But an anomaly detection algorithm that requires a shorter training phase is easier to deploy than an algorithm that requires a longer training phase.

5.3.2 Testing the Regular-expression Engine

In this first test, we compare our CREG algorithm to WebAnomaly, Anagram and POSEIDON using training sets of increasing size with "regular" requests only (requests where parameters classified as "raw-data" have been previously removed, leaving a total of 55 regular parameters) from DS_A. This test aims to demonstrate the effectiveness of our approach over previous methods when analysing regular parameters. We use training

sets of increasing size to measure the learning rate and to simulate a training phase as it could take place in a real environment, when a system is not always trained thoroughly. As an attack test bed, we use the same set of attacks used to determine the criterion for regular and irregular parameters, and added buffer overflow payloads. We have added also attack mutations, generated using the Sploit framework [116], to reproduce the behaviour of an attacker attempting to evade signature-based systems. The attack test bed contains then 20 attacks in total now. Table 5.8 reports results for tests with "regular" requests.

#training samples		**CREG**	**WebAnomaly**	**Anagram**	**POSEIDON**
5000	Attacks	20/20	18/20	20/20	20/20
	FPs	1062	1766	144783	1461
10000	Attacks	20/20	16/20	20/20	20/20
	FPs	1045	1529	133023	1387
20000	Attacks	20/20	16/20	20/20	20/20
	FPs	43	177	121484	1306
50000	Attacks	20/20	14/20	20/20	20/20
	FPs	16	97	100705	1251

Table 5.8: Results for CREG and comparative algorithms on "regular" requests only extracted from DS_A. CREG outperforms in terms of both detection (all 20 possible attacks detected) and false positive rates competitors, due to its better context-aware approach. We note that with a relatively small training set (20000 samples), CREG already raises fewer false positives than the runner-up competitor (WebAnomaly) while detecting any attack.

Our tests show that with a rather small training set (20000 requests, originally collected in less than two days), CREG generates 43 false positives ($\sim$0,009%), less than 2 alerts per day. The "sudden" decrease in FPs shown by CREG (and Webanomaly) when we train it with at least 20000 requests is due to the fact that, with less than 20000 training samples, some parameters are not analysed during training (i.e., some URLs have not been accessed), therefore no model is created for them and by default this event is considered malicious. One surprise is the high number of false positives shown by Anagram [57, 120]. Anagram raises a high number of false positives on specific fields whose content looks pseudo-random, which are common in web applications. Consider for example the following request parameter *sid=0c8026e78ef85806b67a963ce58ba823* (it is a user session ID automatically added by PostNuke in each URL link), this value being randomly generated as a new user comes: such a string probably contains a number of n-grams which were not observed during the training phase, therefore Anagram is likely to flag any session ID as anomalous. Also Song et al. [110] point out such drawback. On the other hand, CREG exploits regularities in inputs, by extracting the syntax of the parameter content (e.g., the regular expression for sid is $(a^+|d^+)^+$), and easily recognizes similar values in the future. WebAnomaly shows (unexpectedly, at least in theory) a worse

detection rate as the training set samples increase. This is due to the fact that the content of new samples is similar to some attack payloads (but contains different symbols), and the system is not able to discern malicious traffic because of its statistical model.

5.3.3 Testing Sphinx on the Complete Input

We show the results of the second test which uses the complete input of the web application (and not only the regular parameters). We use the two data sets DS_B and DS_C: DS_B contains 78 regular and 10 irregular parameters; DS_C respectively 334 and 10. We proceed as before, using different training sets with increasing numbers of samples. To test our system, we have used the attack database presented in [57] which has already been used to assess several intrusion-detection systems for web attacks. We adapted the original attack database and added the same attack set used in our previous test session. We found this necessary because [57] contains some attacks to the platforms (e.g., a certain web server vulnerability in parsing inputs) rather than to the web applications themselves (e.g., SQL Injection attacks are missing). Furthermore, we had to exclude some attacks since they target web server vulnerabilities by injecting the attack payload inside the HTTP headers: although Sphinx could be easily adapted to process header fields, our logs do not always contain a HTTP header information. In total, our attack bed contains 80 vectors, including the 20 attacks previously used to test DS_A (adapted to target the new data set).

# training samples		**Sphinx**		**Web Anomaly**	**Anagram**	**POSEIDON**
		FPs RT	**FPs RD**			
5000	Attacks	80/80		67/80	80/80	80/80
	FPs	162	1955	2593	90301	3478
10000	Attacks	80/80		67/80	80/80	80/80
	FPs	59	141	587	80302	643
20000	Attacks	80/80		53/80	80/80	80/80
	FPs	43	136	451	71029	572
50000	Attacks	80/80		47/80	80/80	80/80
	FPs	29	127	319	61130	433

Table 5.9: Results for Sphinx and comparative algorithms on full requests from DS_B: we report separate false positive rates for Sphinx (RT stands for "regular-text" models and RD for "raw-data" model). Sphinx outperforms competitors in terms of detected attacks and false positive rate. By looking at the FP value of Sphinx reported in column "RD" and at the FP value reported for POSEIDON (the RD model is the same used by POSEIDON), we note that using two different approaches to detect anomalies is the key to improve accuracy.

# training samples		Sphinx FPs RT	Sphinx FPs RD	Web Anomaly	Anagram	POSEIDON
5000	Attacks	80/80		78/80	80/80	80/80
	FPs	36	238	607	16779	998
10000	Attacks	80/80		77/80	80/80	80/80
	FPs	24	109	515	13307	654
20000	Attacks	80/80		49/80	80/80	80/80
	FPs	10	98	459	7417	593
50000	Attacks	80/80		46/80	80/80	80/80
	FPs	3	47	338	4630	404

Table 5.10: Results for Sphinx and comparative algorithms on full requests from DS_C: we report detailed false positive rates for Sphinx (RT stands for "regular-text" models and RD for "raw-data" model). Sphinx outperforms competitors in terms of detected attacks and false positive rate.

The tests show that the presence of irregular parameters significantly influences the false positive rate of Sphinx. We need an extensive training to achieve a rate of 10 false positives per day: this is not surprising, since we observed a similar behaviour during previous tests (see [1, 4]).

5.4 Signature Generation for Signature-based Systems

Commonly the expression "signature generation" refers to the automatic generation of signatures, for an SBS, once an attack has been detected by an ABS. This capability has been extensively investigated recently [65, 68, 91, 120], but results have been limited in their scope. Chung and Mok [31, 32] demonstrate that it is possible to force the generation of signatures that match normal traffic, thereby generating an impressive number of false alerts. Rather than generating a signature for an attack, one should generate a signature for a vulnerability, as attackers can easily craft a new (undetectable) attack payload, yet exploiting the same vulnerability.

Sphinx uses regular expressions inferred from HTTP request parameters to model the normal content of regular parameters: those regular expressions can be exported to an SBS, making it possible to configure the IDS automatically. In this way, we completely reverse the usual signature-based approach, thereby enforcing the detection (at least, for regular parameters) by defining specifically what is considered normal, rather than what is considered malicious. This has a great impact on a SBS: with signatures exported from Sphinx inferred regular expressions, regular parameters are virtually protected by *any* attack with just *one* signature.

Among the freely available SBSs, three are widely used: Snort, Bro [100] and ModSecurity [128]. We chose to export the inferred regular expressions to ModSecurity format because (1) it is a widely-deployed web application firewall, and (2) a good deal of its signatures are based on regular expressions. It basically works as Snort, but processes information at an higher OSI level. In fact, being an additional module for the Apache web server, it has the advantage of processing HTTP requests after the IP and TCP data have been reassembled (immune to common IP and TCP evasion attacks [104]) and decrypted (no need to provide SSL keys) by the web server.

The reason why ModSecurity signatures are mainly based on regular expressions (and a lot of "facilities" have been provided by developers to work with them) is that they can take advantage of the HTTP context and they aim to detect typical attack traces (such as "select from" for a SQL Injection) in requests, rather than matching a straight pattern (unlike Snort). This approach is an attempt to decrease the false negative/positives rates, by introducing a kind of abstraction. Although these signatures heavily based on regular expressions usually perform better than "old" signatures (more attack variations can be detected with a single signature), the approach presents the usual limits of signature-based solutions: the attack payload must be known in advance.

This is where Sphinx becomes particularly powerful in combination with ModSecurity. ModSecurity can be configured with the exported regular expressions inferred by Sphinx (along with its standard set of signatures) to validate regular parameters in HTTP requests. Below, we compare a standard signature used by ModSecurity to detect SQL Injections attacks to a signature exported by Sphinx (using ModSecurity format) for two regular parameters. Sphinx signatures protect those parameters from *any* currently known attack. For the sake of exposition, we report only part (less than a half) of the original signature of ModSecurity.

ModSecurity signature

```
SecRule REQUEST_FILENAME|ARGS|ARGS_NAMES|REQUEST_HEADERS|
!REQUEST_HEADERS:Referer "(?:\b(?:(?:s(?:elect\b(?:.{1,100}
?\b(?:(?:length|count|top)\b.{1,100}?\bfrom|from\b.{1,100}?
\bwhere)|.*?\b(?:d(?:ump\b.*\bfrom|ata_type)|(?:to_(?:numbe|
cha)|inst)r))|p_(?:(?:addextendedpro|sqlexe)c|(?:oacreat|
prepar)e|execute(?:sql)?|makewebtask)|ql_(?:longvarchar|...
```

The signature of ModSecurity cannot be designed for a specific resource, because each deployment site could be different. Hence, in the first part of the signature, the detection engine is instructed to scan any entry of the HTTP request (e.g., REQUEST_HEADERS and REQUEST_FILENAME), and not the URL only. The following part of the signature search for well-known traces of SQL Injection attacks (in this case, to gain control of a Microsoft SQL Server DBMS). The signature cannot specific which parameters could be exploited, therefore the detection engine has to go trough the whole request.

Sphinx signature

```
SecRule REQUEST_FILENAME "/index.php" "chain,log,deny,status:
403,phase:2"
SecRule ARGS_NAMES "!^(id|func)$"
SecRule ARGS QUERY_STRING "!^(id=\d+)"
SecRule ARGS QUERY_STRING "!^(func=(?:\a+)User)"
```

The signature generated by Sphinx for ModSecurity targets a specific resource (i.e., the URL "index.php" in the example). Since Sphinx works by analysing each request parameter singularly, and it can enumerate precisely which parameters the HTTP request should contain ("id" and "func"), thus detecting any attempt to inject a malicious payload in non-existent parameters (a typical behaviour of automatic scanners, which usually do not fingerprint the attacked web site). For each of these parameters, Sphinx inserts a specific regular expression that validates the input (only integer values for the "id" parameter). Hence, the engine requires less computational resources, as only one signature is used per parameter.

Benefits ModSecurity users have several benefits: (1) they do not need to spend time in *ad hoc* configurations and updating (as signatures tailored for each regular parameter are provided and updated by Sphinx), (2) they can easily review generated signatures and possibly edit them on-the-fly (as signatures are simple human-readable regular expressions), (3) custom-developed web applications are protected as well, and (4) new (previously unknown) attacks payloads are detected as well.

5.5 Related Work

Despite the fact that web applications have been widely developed only in the last half-decade years, the detection of web-based attacks has immediately received considerable attention.

Ingham et al. [58] use a deterministic finite automaton (DFA) to build a profile of legal HTTP requests. It works by tokenizing HTTP request parameters, and storing each token type and (optionally) its value. Pre-defined heuristic functions are used to validate and generalize well-known input values (e.g., dates, file types, IP addresses and session cookies). Each state in the DFA represents an unique token, and the DFA has a transition between any two states that were seen consecutively (from a chronological point of view) in the request. A similarity function determines if a request has to be considered anomalous. It reflects the changes (i.e., for each missed token a new transition would have to be added) that would have to be made to the DFA for it to accept the request. The more the changes, the more likely the request is anomalous.

Despite its effectiveness, this approach relies on predefined functions which can be used to analyse only certain (previously known) input types. Furthermore, for some parameters (e.g., blog messages) it could be difficult to find a good function to validate

the content. Sphinx, on the other hand, is able to learn in an automatic way the syntax of most of parameter values and uses a content-based anomaly detector for parameters whose syntax cannot be extracted.

WebAnomaly (Kruegel et al. [73]) analyses HTTP requests and takes significant advantage of the parameter-oriented URL format common in web applications. The system applies up to nine different models at the same time to detect possible attacks, namely: attribute length and character distribution, structural inference, token finder, attribute presence and order, access frequency, inter-request time delay and invocation order. We have compared WebAnomaly to Sphinx in our benchmarks.

Jovanovic et al. [60] present a static-analysis tool (called Pixy) for web applications. The tool detects data flow vulnerabilities by checking how inputs could affect the (intended) behaviour of the web application, leading to an outflow of information. This approach requires the sources code of the web application to be available.

Finally, we should mention that Almgren et al. [12] and Almgren and Lindqvist [13] present similar systems which are based on signature-based techniques and either analyse web server logs ([12]) or are integrated inside the web server itself.

5.6 Conclusion

Sphinx is conceptually simple, and – as our benchmarks show – to detect attacks to web applications it performs better than competing systems.

The systems we compare Sphinx to are the best currently available and the set of benchmarks we have carried out (with 3 different data sets) is extensive. Another aspect we want to stress is that Sphinx presents also a better learning curve than competitors (i.e., it needs a lower number of samples to train). This is important in the practical deployment phase, when changes to the underlying application require that every now and then the system be retrained (and retraining the system requires cleaning up the training set from possible attacks, an additional operation which needs to be done – accurately – off-line).

Instead of using solely mathematical and statistical models, Sphinx takes advantage of the *regularities* of HTTP request parameters and is able to automatically generate, for most of the parameters, human-readable regular expressions (we call them "positive signatures"). This also means that the IT specialist, if needed, can easily inspect and modify/customize the signatures generated by Sphinx, thereby modifying the behaviour of the detection engine. This aspect should be seen in the light of the criticisms that is often addressed to anomaly-based systems: that they are black-boxes which cannot be tuned by the IT specialists in ways other than modifying, e.g., the alert threshold [70]. Sphinx is – to our knowledge – the first anomaly-detection system which relies heavily on signatures which can be seen, interpreted, and customized by users, thereby improving the usability of the system.

Chapter 6

Panacea: Automating the Classification of Attacks for Anomaly-based Network Intrusion Detection Systems*

Today, security teams aim to automate the management of security events, both to optimize their workload and to improve the chance of detecting malicious activities. However, the automation of the security management tasks poses new challenges.

During the years, IDSs have been continuously improved to detect the latest threats. However, some events that were once considered dangerous have become "not-relevant" (e.g., port scans). Malicious activities conducted by automatic scanners, BOTnets, and so-called script-kiddies can generate a large number of security alerts. Although true positives when detected by an IDS, these kinds of activities cannot normally be considered a serious threat. Most of them attempt to exploit old vulnerabilities that have already been fixed. The fact that a remote automatic scanner is attempting to replicate a 5-year old attack against a now-secure PHP script on a certain web server is no longer important. As a result, the number of security alerts, consisting of irrelevant true positives and false positives (which we have already addressed in Chapter 4), has increased over the years. The most harmful attacks currently consist of several stages. Ning et al. [96] observe that "most intrusions are not isolated, but related as different stages of attacks, with the early stages preparing for the later ones".

A number of techniques to perform alert correlation have been proposed (Cuppens and Ortalo [38], Debar and Wespi [44], Ning and Xu [99] and Valeur et al. [113]), in order to detect attacks at an early stage, or lower the false and the non-relevant alert rates.

*This chapter is a minor revision of the paper with the same title published in the Proceedings of 12th Symposium on Recent Advances in Intrusion Detection, Springer, 2009 (TO APPEAR).

Nowadays, several "security information management" (SIM) tools are widely used by security teams (e.g., the well-known OSSIM [137]). They are used to ease the management of the security infrastructure, as they integrate with heterogeneous security systems, and can perform a number of tasks automatically. Among those tasks, SIM tools automate the alert filtering and correlation. However, for the tasks to be effective, the attacks that triggered the alerts must be classified to provide a good deal of information (apart from the usual IP addresses and TCP ports). In fact, by classifying the attack (e.g., buffer overflow, SQL Injection), it is possible to set in a more precise way an action the system has to execute to handle a certain alert. The alert could (1) trigger automatic countermeasures, e.g., either because an early attack stage has been detected or because the attack class is considered to have a great impact on the security. Alternately, the alert (2) could be forwarded for manual handling or (3) filtered and stored for later analysis (i.e., correlation) and statistics.

Determining the class of an attack is trivial for an alert generated by an SBS. Each signature is the result of an analysis of the corresponding attack conducted by experts: the attack class is manually assigned during the signature development process (i.e., the alert class is included in the signature). Thus, usually security teams do not need to further process the alert to assign a class, and they can set precisely a standard action for the system to execute when such an alert is triggered.

Problem When an ABS raises an alert, it cannot associate the alert with an attack class. The system detects an anomaly, but it has too little information (typically only source and destination IP addresses and TCP ports) to determine the attack class. No automatic or semi-automatic approach is currently available to classify anomaly-based alerts. Thus, any anomaly-based alert must be manually processed to identify the alert class, increasing the workload of security teams. A solution to automate the classification of anomaly-based alerts is to employ some heuristics (e.g., see Robertson et al. [106]) to analyse the ABS alert for features of well-known attacks. Although this approach could lead to good results, it totally relies on the manual implementation of the heuristics (which could be a labour intensive task), and on the expertise of the operator.

The lack of attack classification affects the overall usability of an ABS, because it makes it difficult (if not impossible) for security teams both to employ alert correlation techniques and to activate automatic countermeasures for anomaly-based alerts, and in general to integrate an ABS with a SIM tool.

Contribution We present Panacea, a simple, yet effective, system that uses machine learning techniques to automatically and systematically classify attacks detected by a payload-based ABS (and consequently the generated alerts as well). The basic idea is the following. Attacks that share some common traits, i.e., some byte sequences in their payloads, are usually in the same class. Thus, by extracting byte sequences from an alert payload (triggered by a certain attack), we can compare those sequences to previously collected data with an appropriate algorithm, find the most similar alert payload, and then infer the attack class from the matching alert payload class.

To the best of our knowledge, Panacea is the first system proposed that:

- Automatically classifies attacks detected by an ABS, without using pre-determined heuristics.
- Does not need manual assistance to classify attacks (with some exceptions to be described in Section 6.1.1.2).

Panacea requires a training phase for its engine to build the attack classifier. Once the training phase is complete, Panacea classifies any attack detected by the ABS automatically.

Limitation of the approach Panacea analyses the generated alert payload to build its classification model. Thus, any alert generated by attacks/activities that do not involve a payload (e.g., a port scan or a DDoS) cannot be automatically classified. As most of the harmful attacks inject some data in target systems, we do not see this as a serious limitation. However, Panacea cannot work with an ABS that detects attacks without analysing the packet payloads. Here we consider only attacks that target networks, however it is possible to extend the approach to include anomaly-based HIDSs too.

6.1 Architecture

Panacea consists of two interacting components: the Alert Information Extractor (AIE) and the Attack Classification Engine (ACE). The AIE receives alerts from the IDS(s), processes the payload, and extracts significant information, outputting alert meta-information. This meta-information is then passed to the ACE that automatically determines the attack class. The classification process goes through two main stages. First, the ACE is trained with several types of alert meta-information to build a classification model. The ACE is fed alert meta-information and the corresponding attack class. The attack class information can be provided in several ways, either manually by an operator or automatically by extracting additional information from the original alert (only when the alert has been raised by an SBS). Secondly, when the training is completed, the ACE is ready to classify new incoming alerts automatically. We now describe each component and the working modes of Panacea in detail. Figure 6.1 depicts Panacea and its internal components.

6.1.1 Alert Information Extractor

The first component we examine is the AIE. The extraction of relevant information from alert payloads is a crucial step, as it is the basis for attack class inference. Requirements for this phase are that the extraction function should capture enough features from the original information (i.e., the payload) to distinguish alerts belonging to different

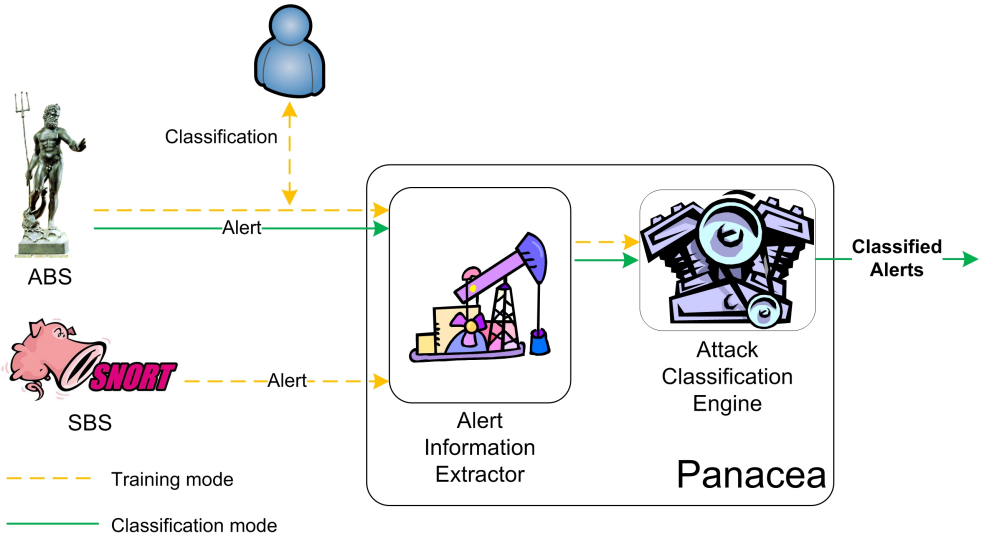


Figure 6.1: An overview of the Panacea architecture and the internal components.

classes, and it should be efficient w.r.t. the required memory space. We now describe the analysis techniques we have chosen.

6.1.1.1 Extracting and Storing Relevant Information

In Section 3.1.2 we introduced the n-gram analysis technique for the detection of network attacks. N-gram analysis is a suitable technique to capture data features also for the problem of attack classification, and the AIE employs such a technique to extract relevant information from alert payloads.

As Wang et al. note [120], by using higher order n-grams (i.e., n-grams where $n > 1$) it is possible to capture more data features and to achieve a more precise analysis. One has to consider that the whole feature space size of a higher-order n-gram is 256^n (where n is the n-gram order). The comparison of byte frequency values becomes quickly infeasible, also for values of n such as 3 or 4, because the space needed to store average and standard deviation values for each n-gram grows exponentially (e.g., 640GB would be needed to store 5-grams statistics). Although a frequency-based n-gram analysis may seem to model data distribution accurately, Wang et al. experimentally show that a binary-based n-gram analysis is more precise in the context of network data analysis. In practice, the fact that a certain n-gram has occurred is stored, rather than computing average byte frequency and standard deviation statistics. The reason why the binary approach performs better is that high-order n-grams are more sparse than low-order n-grams, thus it is more difficult to gather accurate byte-frequency statistics as the order increases. This approach has an additional advantage, other than being more precise. Because less information is required, it requires less space in memory, and we can consider higher-order n-grams (such as 5).

Bitmap The ideal data structure to store binary-based n-gram information is a bitmap. A bitmap is a type of memory organization used to store information as spatially mapped arrays of bits. In our case, each map entry (a bit) maps a certain n-gram: thus the bitmap size depends on the n-gram order. For 3-grams the bitmap size is 2MB, and for 5-grams the size goes up to 128GB. Here we follow Wang et al. and we use Bloom filters to overcome the space dimension problem.

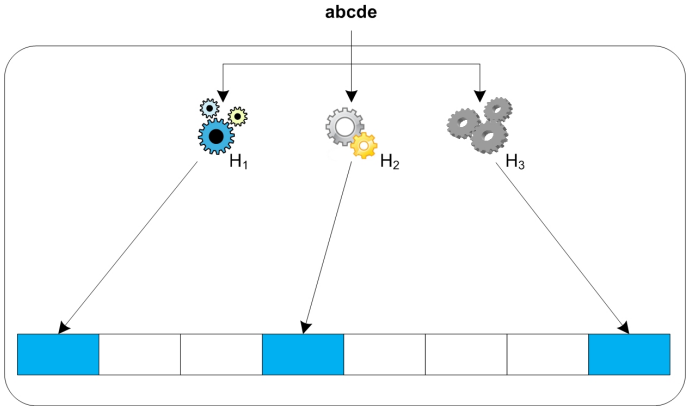
Bloom filter A Bloom filter [24] (BF) is a method to represent a set of S elements (n-grams in our embodiment) in a smaller space. Formally, a BF is a pair $\langle b, H \rangle$ where b is a bit map of l bits, initially all set to 0, and H is a set of k independent hash functions $h_1 \dots h_k$. H determines the storage of b in such a way that, given an element s in S: $\forall h_k$, $b_i = 1 \iff h_k(s)\ mod\ l = i$. In other words, for each n-gram s in S, and for each hash function h_k, we compute $h_k(s)\ mod\ l$, and we use the resulting value as index to set to 1 the bit in b corresponding to it. When checking for the presence of a certain element s, the element is considered to be stored in the Bloom filter if: $\forall h_k$, $b_{h_k(s)\ mod\ l} = 1$. A BF with a size of 10KB is sufficiently large to store the alert meta-information resulting from 5-grams analysis. Figure 6.2 shows an example of insertion in a BF.

A BF employs k different hash functions at the same time to decrease the probability of a false positive (the opposite situation, a false negative, cannot occur). False positives occur when all of the bit positions calculated for a given element have been set to 1 when inserting previous elements (as depicted in Figure 6.3), due to the collisions generated by hash functions. The false positive rate for a given BF is $(1-e^{\frac{kn}{l}})^k$, where n is the number of elements already stored.

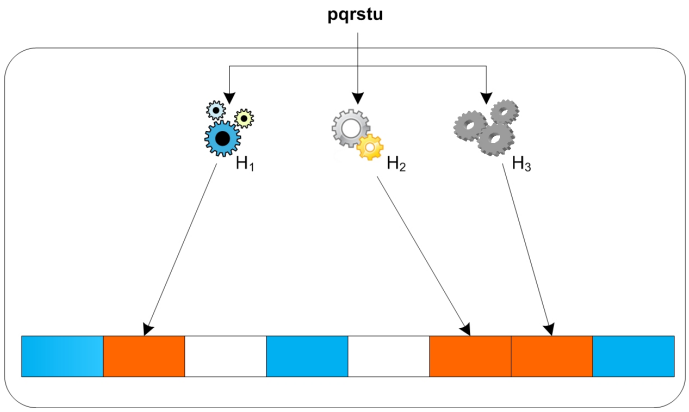
6.1.1.2 Operational Modes

The AIE not only extracts information from alerts as described above, but it is also responsible for forwarding the attack class information to the classification engine, when the latter is in training mode. The attack class can be provided either automatically or manually. In case an SBS is deployed next to the ABS and it is monitoring the same data, it is possible to feed the ACE during training both the payload and the attack class of any alert generated by the SBS. We define this as the *automatic* mode, since no human operator is required to carry out the attack classification. A human operator can classify the alerts raised by the ABS (in a manner consistent with the SBS classification), hence integrating those with the alerts raised by the ABS during the ACE training. We call this the *semi-automatic* mode. The last possible operative mode is the *manual* mode. In this case, *any* alert is manually classified by an operator.

Each mode presents advantages and disadvantages. In automatic mode, the workload is low, but on the other hand the classification accuracy is likely to be low as well. In fact, the SBS and the ABS are likely to detect different attacks, hence the classification engine could be trained to correctly classify only a subset of the ABS alerts. The manual mode requires human intervention but it is likely to produce better results, since each alert



(a) Inserting n-gram “abcde”



(b) Inserting n-gram “pqrst”

Figure 6.2: Examples of inserting two different 5-grams. H_1, H_2 and H_3 represent different hash functions.

is consistently classified (the classification that comes out-of-the-box with an SBS could not be suitable). We assume that the alerts raised by the SBS and ABS have already been verified and any false positive alert has already been purged (e.g., using ALAC [101] or our ATLANTIDES).

6.1.2 Attack Classification Engine

The ACE includes the algorithm used to classify attacks. Since we are aware of the attack class information, we consider only *supervised* machine learning algorithms. These algorithms generally achieve better results than unsupervised algorithms (where the algorithm, e.g. K-medoids, deduces classes by measuring inter-data similarity). The classification algorithm must meet several requirements, namely:

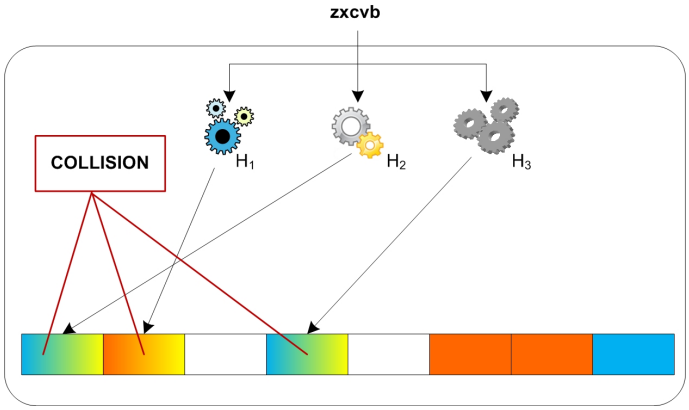


Figure 6.3: Example of a false positive. The element "zxcvb" has not been inserted in the Bloom filter. Due to the collisions generated by the hash functions, the test for its presence returns "true".

- Support for multiple classes, as alerts fall in several classes.
- Classification of high-dimensional data, since each bit of the BF data structure the ACE receives in input is seen as a dimension of analysis.
- fast training phase (the reason for this will be clarified later);
- Estimate classification confidence when in classification phase (optional).

We consider the last requirement optional, as it does not directly influence the quality of the classification, though it is useful to improve the system usability. Confidence measures the likelihood of having a correct classification for a given input. Users can instruct the system to forward any alert whose confidence value is lower than a given threshold for manual classification, hence reducing the probability of misclassification (at the price of an increased workload).

We chose two algorithms for our experiments: (1) Support Vector Machines (SVM) and (2) the RIPPER rule learner. These algorithms implement supervised techniques, their training and classification phases are fast and handle high-dimensional data. Both algorithms perform non-incremental learning. A non-incremental algorithm iterates on samples several times to build the best classification model by minimizing the classification error. The whole training set is then needed at once, and additional samples cannot be incorporated in the classification model unless the training phase is run from scratch. On the other hand, an incremental algorithm can modify the model after the main training phase as new samples become available. An incremental algorithm usually performs worse than a non-incremental algorithm, because the model is not re-built. Thus, a non-incremental algorithm is the best choice to perform an accurate classification. However, because it is highly unlikely that we can collect all alerts for training at once the choice of non-incremental algorithms could be seen as a limitation of our system.

In practice, thanks to the limited BF memory size, we can store a huge number of samples and, by applying a "batch training", we can simulate incremental learning in non-incremental algorithms. As new training samples become available, we add them to the batch training set and build the classifier using the entire set only when a certain number of samples is reached. Then, the classifier is re-built with the set of "batches" available at that time. Because both SVM and RIPPER are fast in training, there are no computational issues.

We chose SVM and RIPPER, not only because they meet the requirements, but for two additional reasons. First, they yield high-quality classifications. Meyer et al. [88] test the SVM against several other classification algorithms (available from the R project [139]) on real and synthetic data sets. An SVM outperforms competitors in 50% of tests and ranks in the top 3 in 90% of them. RIPPER has been used before in the context of intrusion detection (e.g., on data relative to system calls and network connections [76, 77]) with good results. Secondly, because they approach the classification problem differently (geometric for SVM, and rule-based for RIPPER), the algorithms are supposed to compensate for each others weaknesses. Hence, we can evaluate which algorithm is more suitable in different contexts. We will now provide some detail on the algorithms.

6.1.2.1 Support Vector Machines

An SVM (Vapnik and Lerner [115]) is a set of supervised learning methods used for classification. In the original formulation, an SVM is a binary classifier. It uses a non-linear mapping to transform the original training data into a higher dimension. Then, it searches for the linear optimal separating hyperplane, i.e., a plane that separates the samples of one class from another. An SVM uses "support vectors" and "margins" to find the optimal hyperplane, i.e., the plane with the maximum margin. Lets us consider Figure 6.4. By selecting an appropriate non-linear mapping to a sufficiently high dimension, data samples from two classes can always be separated by a hyperplane. In fact, there are a number of separating hyperplanes. However, we expect the hyperplane with the larger margin to be more accurate at classifying future data samples, because it gives the largest separation "distance" between classes. The margin is calculated by constructing two parallel hyperplanes, one on each side of the hyperplane, and then by "pushing them up against" the data sets. Any data samples that fall on these side hyperplanes are called support vectors.

The original SVM algorithm has been modified to classify non-linear data and to use multiple classes. Boser et al. [26] introduce non-linear data classification by using kernel functions (i.e., non-linear functions). The resulting algorithm is similar, but every dot product is replaced with a non-linear kernel function. Then, the algorithm proceeds to find a maximal separating hyperplane in the transformed space.

To support multiple classes, the problem is reduced to multiple binary sub-problems. Given m classes, m classifiers are trained, one for each class. Any test sample is assigned to the class corresponding to the largest positive distance.

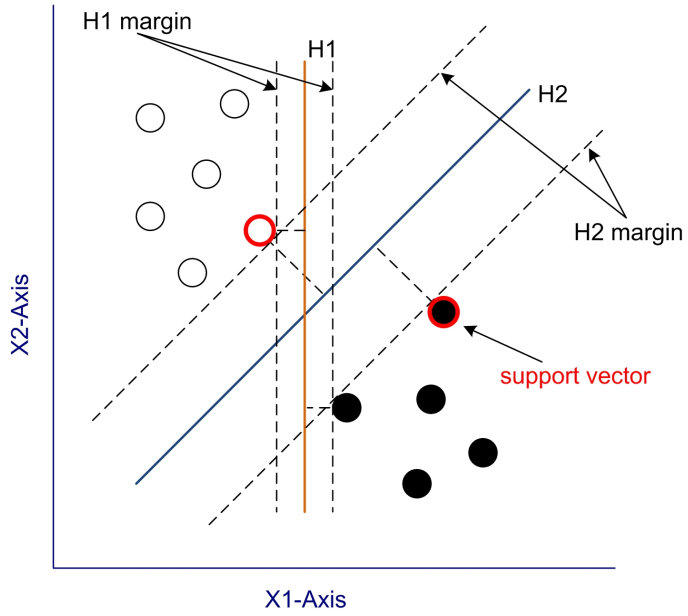


Figure 6.4: Hyperplanes in a 2-dimensional space. H1 separates samples sets with a small margin, H2 does that with the maximum margin. The example refers to linearly separable data. The support vectors are shown with a thicker red border.

6.1.2.2 RIPPER

RIPPER (Cohen [34]) is a fast and effective rule induction algorithm. RIPPER uses a set of IF-THEN rules. An IF-THEN rule is an expression in the form **IF** <*condition*> **THEN** <*conclusion*>. The IF-part of a rule is called the rule antecedent. The THEN-part is the rule consequent. The *condition* consists of one or more attribute tests, that are logically ANDed. A test t_i is in the form $t_i = v$ for categorical attributes (where v is a categorical label) or either $t_i \geq \theta$ or $t_i \leq \theta$ for numerical attributes (where θ is a numerical value). The conclusion contains a class prediction. If, for a given input, the condition (i.e., all of the attribute tests) holds true, then the rule antecedent is satisfied and the corresponding class in the conclusion is returned (the rule is said to "cover" the input). Since RIPPER employs ordered rules, when a match occurs, the algorithm does not evaluate other rules. Some examples of rules are:

IF $bf[i] = 1$ *AND* $\ldots$ *AND* $bf[k] = 1$ **THEN** *class = cross-site scripting*
IF $bf[l] = 1$ *AND* $\ldots$ *AND* $bf[m] = 1 \ldots$ *AND* $bf[n] = 1$ **THEN** *class = sql injection*

RIPPER builds the rule set for a certain class SC_i as follows. The training data set is split into two sets, a pruning set and a growing set. The classifier is built using these two sets by repeatedly inserting rules starting from an empty rule set (the growing set). The algorithm heuristically adds one condition at a time until the rule has no error rate on the growing set.

RIPPER implements also an optimisation phase, in order to simplify the rule set. For each rule r_i in the rule set, two alternative rules are built; the replacement of r_i and the revision of r_i. The replacement of r_i is created by growing an empty rule $r_i^{'}$, and then pruning it in order to reduce the error rate of the rule set including $r_i^{'}$ on the pruning data set. The revision of r_i is constructed similarly, but the revision rule is built heuristically by adding one condition at a time to the original r_i rather than to an empty rule. Then the three rules are examined on the pruning data to select the rule with the least error rate.

When multiple classes $C_1 \ldots C_n$ are used, RIPPER sorts classes on a sample frequency basis and induces rules sequentially from the least prevalent class SC_1 to the second most prevalent class SC_{n-1}. The most prevalent class SC_n becomes the *default* class, and no rule is induced for it (thus, in case of a binary classification, RIPPER induces rules for the minority class only).

6.1.3 Implementation

We have implemented a prototype of Panacea to run our experiments. The prototype is written in Java, since we link to the libraries provided by the Weka platform [136]. Weka is a well-known collection of machine learning algorithms, and it contains an implementation of both SVM and RIPPER. Weka provides also a comprehensive framework to run benchmarks on several data sets under the same testing conditions. In our benchmarks we have used the Weka algorithms with standard settings (e.g., SVM employs a polynomial kernel function). The attacks samples generated by NIDSs, in the form of alerts, are stored in a database that the system fetches to extract the alert payload information.

6.2 Benchmarks

Public data sets for benchmarking IDSs are scarce. It is even more difficult to find a suitable data set to test Panacea, since no research has systematically addressed the problem of (semi)automatically classifying attacks detected by an ABS before. Hence, we have collected three different data sets (referred to as DS_A, DS_B and DS_C, see below for a description of the data sets) to evaluate the accuracy of Panacea. These data sets are used to evaluate the accuracy of Panacea in different scenarios: (1) when working in automatic mode (DS_A), (2) when using an ad hoc taxonomy and the manual mode (DS_B) and (3) when classifying unknown attacks (e.g., generated by two ABSs), having trained the system with alerts from known attacks (DS_B and DS_C).

In the literature there are several taxonomies and classifications of security events. For instance, Howard [56], Hansman and Hunt [54], and the well-known taxonomy used in the DARPA 1998 [81] and 1999 [83] data sets. Only the latter classification has been used in practice (in spite of its course granularity, as it contains only four classes which are unsuitable to classify modern attacks). A modern IDS employs its own attack classification, which is usually non-standard and – according to Andersson et al. [15] – difficult to translate into a standardized classification, e.g., the XML-based Intrusion Detection

Message Exchange Format [41]. In our experiments, we use the Snort classification for benchmarks with DS_A (see [145] for a detailed taxonomy) and the Web Application Security Consortium Threat Classification [149] for benchmarks with DS_B and DS_C, as they contain alerts related to web attacks.

To evaluate the accuracy of the classification model, we use two approaches. For test (1) and (2), we employ cross-validation. In cross-validation, samples are partitioned into sub-sets. The analysis is first performed on a single sub-set, while the other sub-set(s) are retained to validate the initial analysis. In k-fold cross-validation, the samples are partitioned into k sub-sets. A single sub-set is retained as the validation data for testing the model, and the remaining $k - 1$ sub-sets are used as training data to build the model. The process is repeated k times (the "folds"), using each of the k sets exactly once to validate the model. Usually the k fold results are combined (e.g., averaged) to generate a single estimation. The advantage of this method is that all of the samples are used for both training and validation, and each sample is used for validation exactly once. We use 10 folds in our experiments, which is a standard value, used in the Weka testing environment too.

For test 3), we use one of DS_B and DS_C for training and the other for testing. The accuracy is evaluated by counting the number of correctly classified attacks.

Attack Class	**Description**	**# of samples**
attempted-recon*	Attempted information leak	1379
web-application-attack*	Web application attack	1032
web-application-activity*	Access to a potentially vulnerable web application	599
unknown	Unknown traffic	66
attempted-user*	Attempted user privilege gain	45
misc-attack	Miscellaneous attack	44
attempted-admin	Attempted administrator privilege gain	32
attempted-dos	Attempted Denial of Service	14
bad-unknown	Potentially bad traffic	13

Table 6.1: DS_A (alerts raised by Snort): attack classes and samples. It is not surprising that web-related attacks account for more than 50%, since most Snort signatures address web vulnerabilities. * marks classes that contain web-related attacks.

DS_A contains alerts raised by Snort (see Table 6.1 for alert figures). To collect the largest number of alerts possible, we have used several tools to automatically inject attack payloads (Nessus and a proprietary vulnerability assessment tool). Attacks have been directed against a system running some virtual machines with both Linux- and Windows-based installations, which expose several services (e.g., web server, DBMS, web proxy,

SMTP and SSH). We collected more than 3200 alerts in total, classified in 14 different (Snort) attack classes. However, some classes have few alerts, thus we select only classes with at least 10 alerts. This data set (and DS_B as well) is synthetic. We do not see this as a limitation since the alerts cover multiple classes and trigger a large number of different signatures. We test how the system behaves in automatic mode, the whole set being generated by Snort.

DS_B contains a set of more than 1400 Snort alerts related to web attacks (Table 6.2 provides alert details). To generate this data set, we have used Nessus, Nikto [132] (a web vulnerability scanner), and we have manually injected attack payloads collected from the well-known site Milw0rm, that hosts a large collection of web exploits [135]. The attack classification has been performed manually (manual mode), since Snort does not provide a fine-grained classification of web-related attacks (alerts are allocated to different classes with other alerts, see Table 6.1). Attacks have been classified according to the Web Application Security Consortium Threat Classification [149].

Attack Class	**# samples**
Path Traversal	931
Cross-site Scripting	399
SQL Injection	73
Buffer Overflow	8

Table 6.2: DS_B: attack classes and samples. Attacks have been classified according to the Web Application Security Consortium Threat Classification.

DS_C is a collection of alerts generated over a period of 2 weeks by two ABSs, i.e., POSEIDON and Sphinx. We recorded network traffic directed to a main web server of the university network, and did not inject any attack. Afterwards, we processed this data with POSEIDON and Sphinx to generate alerts. The inspection of alerts and the classification of attacks has been performed manually (using the same taxonomy we apply for DS_B). The data set consists of a set of 100 alerts, and Table 6.3 reports attack details.

Attack Class	**# samples**
Path Traversal	53
Cross-site Scripting	27
SQL Injection	16
Buffer Overflow	4

Table 6.3: DS_C: attack classes and samples. Attacks have been classified according to the Web Application Security Consortium Threat Classification.

6.2.1 Tests with DS_A

We use DS_A to validate the general effectiveness of our approach. There are three factors which influence the classification accuracy, namely: (1) the number of alerts processed during training, (2) the length of n-grams used, and (3) the classification algorithm selected. This preliminary test aims to identify which parameter combination(s) results in the most accurate classification.

Testing methodology We proceed with a 3-step approach. First, we want to identify an adequate number of samples required for training: in fact, a too low number of samples could generate an inaccurate classification. On the other hand, while it is generally a good idea to have as many training samples as possible, after some point the benefit from adding additional information could become negligible. Secondly, we want to identify the best n-gram length. Short n-grams are likely to be shared among many attack payloads, and the attack diversification would be poor (i.e., a number of different attacks contains the same n-grams). On the other hand, long n-grams are unlikely to be common among attack payloads, hence it would be difficult to predict a class for a new attack that does not share a sufficient number of long n-grams. Finally, we analyse how the classification algorithms work by analysing the overall classification accuracy (i.e., considering all of the attack classes) and the per-class accuracy. The two algorithms approach the classification problem in two totally different ways, and each of them could be performing better under different circumstances.

To avoid bias by choosing a specific attack, we randomly select alerts in the sub-sets. In fact, by selecting alerts for training in the same order they have been generated (as opposed to random), we could end up with few (or no) samples in certain classes, hence influencing the accuracy rate (i.e., a too good, or bad, value). To enforce randomness, we also run several trials (five) with different sub-sets and calculate the average accuracy rate. The total amount of time spent in training mode varies between 8.9 seconds (RIPPER, 1000 alerts) and 39.7 (SVM, 3000 alerts). Thus, the requirement of a fast training is met. Table 6.4 reports benchmark results (the percentage of correctly classified attacks) for SVM and RIPPER.

	SVM n-gram length				**RIPPER n-gram length**			
# samples	1	2	3	4	1	2	3	4
1000	62.6%	76.8%	**77.3%**	76.7%	66.1%	75.9%	**76.2%**	75.7%
2000	65.9%	78.6%	**78.9%**	77.7%	69.4%	76.7%	**76.9%**	76.4%
3000	66.3%	79.4%	**79.6%**	78.6%	72.7%	77.2%	**77.5%**	76.9%

Table 6.4: Test results on DS_A with SVM and RIPPER. We report the average percentage of correctly classified attacks of five trials. As the number of samples in the testing sub-set increases, the overall effectiveness increases as well. Longer n-grams generally produce better results, up to length 3. SVM performs better than RIPPER by a narrow margin.

Discussion Tests with DS_A indicate that the approach is effective in classifying attacks. As the number of training samples increases, accuracy increases as well for both algorithms. Also the n-gram length directly influences the classification. The number of correctly classified attacks increases as n-grams get longer, up to 3-grams. N-grams of length 4 produce a slightly worse classification, and the same happens for 1-grams (which achieve the worst percentages). SVM and RIPPER present similar accuracy rates on 3-grams, with the former being slightly better. However, if we perform an analysis based on per-class accuracy (see Table 6.5), we observe that, although both classification algorithms score high on accuracy level for the three most populated classes, RIPPER is far more precise than SVM (in once case, the "web-application-activity" class, by nearly 15%).

When we look at the overall accuracy rate, averaged among the 9 classes, for DS_A, SVM performs better because of the classes with few alerts. If we zoom into the classes with a significant number of samples, we observe an opposite behaviour. This means that, with a high number of samples, RIPPER performs better than SVM.

In Table 6.5, a sub-set with fewer samples seems to achieve better results (although percentages differ by a narrow margin), when considering the same algorithm. This happens for SVM once when using 1000 training samples ("attempted-recon" class) and twice when using 2000 training samples ("web-application-attack" and "web-application-activity" classes). When using 2000 training samples, RIPPER performs best in the "web-application-activity" class. The reason for this is that alerts in the sub-sets are randomly chosen, thus a class could have a different number of samples among trials.

	SVM # of samples			**RIPPER # of samples**		
Attack Class	1000	2000	3000	1000	2000	3000
attempted-recon	**90.9%**	90.5%	90.7%	90.4%	93.9%	**94.0%**
web-application-attack	79.8%	**89.0%**	88.8%	97.4%	98.8%	**99.1%**
web-application-activity	80.8%	**81.2%**	80.9%	93.7%	**96.1%**	95.8%

Table 6.5: Per-class detailed results on DS_A, using 3-grams. We report the average percentage of correctly classified attacks of five trials. RIPPER performs better than SVM in classifying all attacks, regardless the attack class or the number of samples.

6.2.2 Tests with DS_B

DS_B is used to validate the manual mode and the use of an ad hoc classification. To perform the benchmarks, we use the same n-gram length that achieves the best results in the previous test. Table 6.6 details our findings for SVM and RIPPER.

Discussion The test results on DS_B show that Panacea is effective also when using a user-defined classification, regardless of the classification algorithm is chosen. Regarding

Attack Class	SVM	RIPPER
Path Traversal	98.6%	**99.1%**
Cross-site Scripting	97.5%	**98.4%**
SQL Injection	**97.6%**	96.2%
Buffer Overflow	37.5%	37.5%
Percentage of total attacks correctly classified	**98.0%**	97.7%

Table 6.6: Test details (percentage of correctly classified attacks) on DS_B with SVM and RIPPER. RIPPER achieves better accuracy rates for the two most numerous classes, although by a narrow margin. We observe the same trend for the rates reported in Table 6.5.

accuracy rates, RIPPER shows a higher accuracy for most classes, although SVM scores the best classification rate (by a narrow margin).

Only the "buffer overflow" class has a low classification rate. Both algorithms have wrongly classified most of buffer overflow attacks in the "path traversal" class. This is because (1) the number of samples is lower than for the other classes, which are at least 10 times more numerous, and 2) a number of the path traversal attacks present some byte encoding that resembles byte values typically used by some buffer overflow attack vectors. In the case of RIPPER, the "path traversal" class has the highest number of samples, hence no rule is induced for it and any non-matching samples is classified in this class.

6.2.3 Tests with DS_C

An ABS is supposed to detect previously-unknown attacks, for which no signature is available yet. Hence, we need to test how Panacea behaves when the training is accomplished using mostly alerts generated by an SBS but afterwards Panacea processes alerts generated by an ABS. For this final test we simulate the following scenario. A user has manually classified alerts generated by an SBS during the training phase (DS_B) and she uses the resulting model to classify unknown attacks, detected by two different ABSs (POSEIDON and Sphinx). Since we collected few buffer overflow attacks, we use the Sploit framework [116] to mutate some of the original attack payloads and increase the number of samples for this class, introducing attack diversity at the same time. Thus, we obtain additional training samples with a different payload. Table 6.7 shows the percentage of correctly classified attacks by SVM and RIPPER. For the buffer overflow attacks, we report accuracy values for the original training set (i.e. representing real traffic) and the "enlarged" training set (in brackets).

Discussion Tests on DS_C show that the SVM performs better than RIPPER when classifying attack instances that have not been observed before. The accuracy rate for the "buffer overflow" class is the lowest, and most of the misclassified attacks have been classified in the "path traversal" class (see the discussion of benchmarks for DS_B). How-

Attack Class	**SVM**	**RIPPER**
Path Traversal	**98.1%**	94.4%
Cross-site Scripting	**92.6%**	88.9%
SQL Injection	**100.0%**	87.5%
Buffer Overflow	**50.0%** (75.0%)	25.0% (50.0%)
Percentage of total attacks correctly classified	**92.0%** (93.0%)	89.0% (91.0%)

Table 6.7: Test details (percentage of correctly classified attacks) on DS_C with SVM and RIPPER. SVM perform better than RIPPER in classifying any attack class. For the "buffer overflow" class and the percentage of total attacks correctly classified we report (in brackets) the accuracy rates when Panacea is trained with additional samples generated using the Sploit framework.

ever, with a higher number of training samples (generated by using Sploit), the accuracy rate increases w.r.t. previous tests. This suggest that, with a sufficient number of training samples, Panacea achieves high accuracy rates.

6.2.4 Summary of Benchmark Results

From the benchmarks results, we can draw some conclusions after having observed the following trends:

- The classification accuracy is always higher than 75%.
- SVM performs better than RIPPER when considering the classification accuracy for all classes, when not all of them have more than 50-60 samples (DS_A, DS_B and DS_C).
- RIPPER performs better than SVM when the class has a good deal of training samples, i.e., at least 60-70 in our experiments (DS_A and DS_B).
- SVM performs better than RIPPER when the class presents high diversity and attacks to classify have not been observed during training (DS_C).

We can conclude that SVM works better when a few alerts are available for training and when attack diversity is high, i.e., the training alert samples differ from the alerts received when in classification phase. On the other hand, RIPPER shows to be more accurate when trained with a high number of alerts.

Evaluating confidence However good Panacea is, the system is not error-free. The consequences of a misclassification can have a direct impact on the overall security. Think

of a buffer overflow attack, for which usually countermeasures must take place immediately (because of the possible consequences), that is misclassified as a path traversal attack, for which the activation of countermeasures can be delayed (e.g., after other actions taken by the attacker). This event occurs often in our benchmarks when the system selects the wrong class. Both SVM and RIPPER can generate a classification confidence value for each attack. This value can be used to evaluate the accuracy of the classification. The lower the classification value is (in a range from 0.0 to 1.0), the more likely the classification is to be wrong (see Table 6.8 for average confidence values for DS_C).

	SVM	**RIPPER**
Average confidence value for correctly classified attacks	0.75	0.62
Average confidence value for misclassified attacks	0.37	0.43
Percentage of total attacks correctly classified without confidence evaluation	92.0%	89.0%
Percentage of total attacks correctly classified with confidence evaluation	95.0%	94.0%
# of alerts forwarded for manual classification	10/100	13/100
# of forwarded attacks that were actually wrongly classified	3/10	5/13
# of forwarded attacks that were actually correctly classified	7/10	8/13

Table 6.8: Effects of confidence evaluation for DS_C, when Panacea is trained with the standard DS_B. When considering the classification confidence to forward alerts for manual classification, the human operator classification increases by 3% and 5% the overall accuracy rate by inspecting 10 and 13 alerts, out of 100, when Panacea uses SVM and RIPPER respectively.

The confidence value can be taken into consideration to detect possible misclassification. Users can set a minimum confidence value (e.g., 0.5). Any alert with a lower confidence value is forwarded to a human operator for manual classification. With this additional check, we are able to increase the percentage of total attacks correctly classified up to 95% for SVM and 94% for RIPPER (when using the standard training set, without additional training samples generated with Sploit). The additional workload involves also the manual classification of alerts which have been correctly classified by the system but whose confidence value is lower than the set threshold. However, less than 10 alerts (out of 100) have been forwarded for manual classification when this action was not needed. Table 6.8 reports the details regarding the evaluation of the confidence value.

6.2.5 Usability in Panacea

Panacea aims not only to provide automatic attack classification for an ABS, but to improve usability as well. In automatic mode, Panacea performs an accurate classifi-

cation (more than 75% of correctly classified attacks). In semi-automatic and manual modes, users actively take part in the classification process: however, users are requested to provide a limited input (i.e., a class label). Panacea classifies attacks systematically and automates (1) the extraction of relevant information used to distinguish an attack class from another and (2) the update of the classification model. These tasks are usually left to the user's experience and knowledge, thus they can be error-prone and not comprehensive. Table 6.9 reports actions that users have to take with and without the support of Panacea.

	User actions	
	Without Panacea	**With Panacea**
DS_A	Classify any alert	No action to take
DS_B	Classify any alert	Classify alerts used during training
DS_C	Classify any alert	No action to take (alerts have been previously classified)

Table 6.9: Actions that users have to take with or without Panacea w.r.t. alert classification for each data set we use during benchmarks.

6.3 Related Work

Although the lack of attack classification is a well-known issue in the field of anomaly-based intrusion detection, little research has been done on this topic.

Robertson et al. [106] suggest to use heuristics to infer the class of (web-based) attacks. This approach has several drawbacks. Users have to generate heuristics (e.g., regular expressions) to identify attack classes. They have to enumerate all of the possible attack variants, and update the heuristics each time a new attack variation is detected. This is a time consuming task. Panacea can operate in an automatic way, by extracting attack information from any SBS, or employ an *ad-hoc* classification, with the user providing only the attack class.

Wang and Stolfo [121] use a "Z-String" to distribute among other ABSs attack payloads to enhance detection. A Z-String contains the information resulting from the n-gram analysis of the attack payload. Once a certain payload has been flagged as malicious, the corresponding Z-String can be distributed to other IDSs to detect the attack also, and stop it at an early stage (think of a worm). If some traffic matches a certain Z-String, that data is likely to be a real attack. Although a Z-String is not used for attack classification, by attaching a class label it would be possible to classify each attack. However, this approach is not systematic, as each attack that does not exactly match any Z-String would have to be manually classified. A Z-String is based on a frequency-based n-gram analysis, thus an exact match could be difficult to achieve. On the other hand, Panacea applies a systematic

classification using the more precise binary-based n-gram analysis. Panacea can also use as a source of information the alerts generated by an SBS, and not only by an ABS.

6.4 Conclusion

In this chapter we present Panacea, a system that automatically and systematically classifies attacks detected by a payload-based ABS (and consequently the generated alerts). Panacea extracts information from alerts during a training phase, then predicts the attack class for new alerts. The alerts used to train the classification engine can be generated by an SBS as well as an ABS. In the former case, no manual intervention is requested (the system operates in automatic mode), as Panacea automatically extracts the attack class from the alert. In the latter case, the user is required to provide the attack class for each alert used to train the classification engine.

Panacea improves the usability and makes it possible to integrate anomaly-based with signature-based IDSs, for instance by using security information management tools. Benchmarks show that the approach is effective in classifying attacks, even those that have not been detected before (and not used for training). Although Panacea works in an automatic way, users can employ ad-hoc classifications, and even manually tune the engine for more precise classifications.

Chapter 7

Concluding Remarks

We now summarize the contribution of this thesis, in relation to the main Research Question discussed in Chapter 1. We also highlight future research directions in the area of anomaly- network-based intrusion detection.

In the introductory chapter we formulate the following research question:

"How can we extend current anomaly- network-based IDSs to improve their usability, yet delivering an effective IDS?"

We argue that current anomaly-based systems are less usable than signature-based systems in daily operations, thus preventing users to benefit fully from the deployment of anomaly-based detection technologies. We see in these limitations the main reason why anomaly-based systems have not been widely deployed, despite research that has been conducted for more than a decade.

Our primary aim is in particular to develop methods that provide practical, efficient and effective ways to decrease user burden and in general enhance the overall user experience of managing network intrusions.

We achieve the aim successfully, allowing us to state, as main conclusion, that the development of effective ways to improve usability for anomaly-based network intrusion detection systems is feasible. We support this conclusion by describing our explorations, which span a range of different results:

- We improve the well-known detection algorithm PAYL (for the HTTP protocol, from 90% up to 100% detection rate and from 0,17% down to 0,0016% false alert rate) by adding a neural network that pre-processes network traffic (POSEIDON, Chapter 3).

- We reduce the number of false positive alerts (between 50% and 100% fewer alerts, for both Snort and POSEIDON) by correlating alerts generated by an intrusion detection system monitoring the incoming traffic with a content-based analysis of the outgoing traffic (ATLANTIDES, Chapter 4).
- We develop a method to detect web attacks by automatically generating regular expressions, that users can edit to tune the anomaly-based detection engine, to validate incoming HTTP requests (Sphinx, Chapter 5).
- We automate the classification of alerts generated by an anomaly-based intrusion detection system by extracting the payload of previous alerts which can be classified using both standard and user-defined taxonomies (Panacea, Chapter 6).

We call this comprehensive set of tools SilentDefense: POSEIDON is the pivotal and basic component of a "pluggable" architecture (see Figure 7.1). Nevertheless each component can operate stand-alone, as well as in cooperation with the other components.

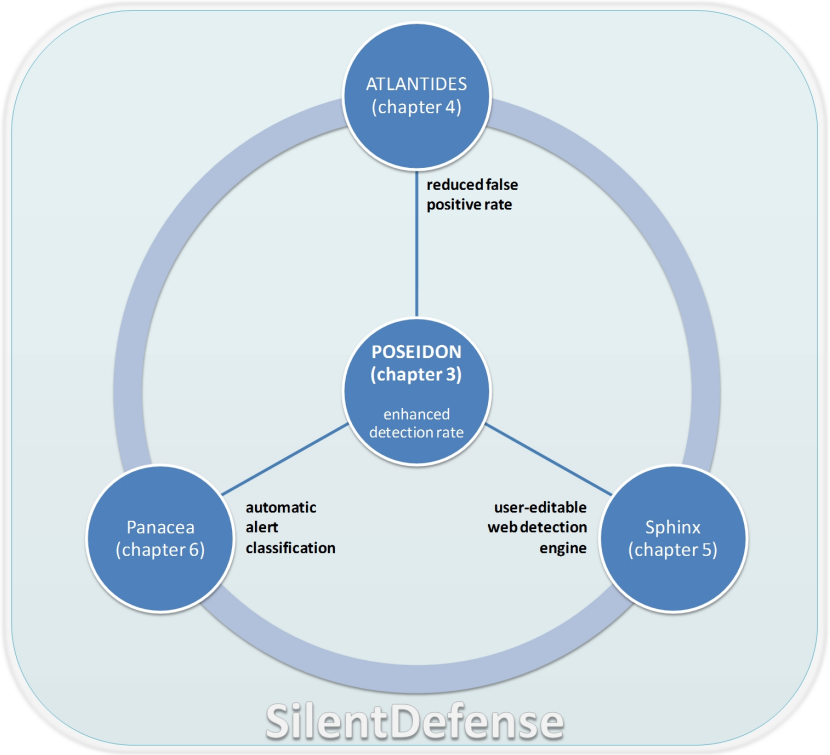


Figure 7.1: An overview of the contributions

We want to stress that SilentDefense (and in general anomaly detection) is not an alternative to signature-based systems. The best chance to detect an attack is provided by

the combination of the two approaches. In particular, a signature-based system works better for well-known applications and systems (e.g., the Windows operating systems), while SilentDefense can detect zero-day and targeted attacks. The latter attack type usually targets custom-developed software. Think of web applications developed by corporations for their internal users, or critical infrastructures that make use of proprietary SCADA systems [140]. Figure 7.2 shows a comparison between Snort (a signature-based system) and SilentDefense.

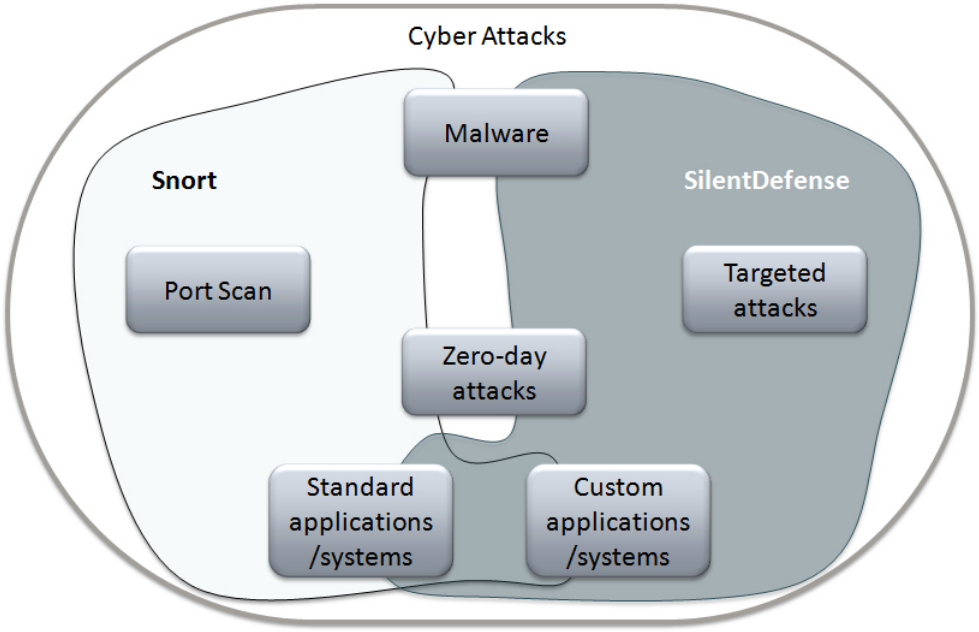


Figure 7.2: A comparison of the relative strengths of Snort (signature-based, left side) and SilentDefense (right side). Snort performs a more accurate detection of attacks against standard and well-known software (e.g., operating systems), port scans and some malware. SilentDefense is more suitable to detect zero-day and targeted attacks, and it can protect custom-developed applications or systems.

Guidelines for building effective payload- anomaly-based intrusion detection systems

The ultimate goal of research is to provide a comprehensive theory, in order to support practitioners in predicting the effectiveness of the systems they develop, without having to run extensive benchmarks. In the intrusion detection field there are some works which make use of information theory to model and detect attacks (Evans et al. [47], Goel and Bush [52]). However, no theoretical framework is available to form the basis of an effective anomaly-based intrusion detection system.

The estimate of the error rates an intrusion detection system will generate is impossible to achieve through theory alone. To illustrate this point, consider the field of biometrics. A biometric authentication device shows some similarities with an intrusion detection system. Such a device usually stores the templates of legitimate users into a database (equivalent to the normal traffic model in an anomaly-based system). There are two possible error types for this device: a legitimate user is not recognized (false reject) and an illegitimate user is wrongly recognized (false accept). In the intrusion detection terminology, these errors are a false positive and a false negative respectively. A biometric device, similarly to an anomaly-based system, usually employs a distance function and a threshold to discriminate legitimate and illegitimate users: it is possible to draw ROC curves (see Section 2.3.4) to analyse the trend of the two error rates as the threshold value changes. Bolle et al. [25] state that it is not possible to pre-compute, or establish theoretically, the error rates a biometric device will generate because such rates will depend on the noise inside the normal data. The only way to determine error rates is by benchmarking the device. Because of the similarities in the approaches, we can infer that estimating the error rates of an intrusion detection system is possible only by performing benchmarks.

To make things worse for intrusion detection systems, the digital world is relatively young and it has been constantly adding new complex technologies (think of the transition from static HTML web pages to AJAX-enriched web applications). Thus, an intrusion detection system is not likely to collect a comprehensive set of normal inputs during its training phase, due to the complexity of the monitored environment (on the other hand, a biometric authentication device usually has a well-defined set of user profiles). Attack techniques have been constantly changing too, and attackers can use different techniques to exploit the same vulnerability. Researchers develop protections in response to a new attack technique, or a new class of vulnerabilities.

Although this work does not provide new theoretical results, it is possible to give some guidelines on how to build effective anomaly-based intrusion detection systems, based on the experienced we gained. We identify three phases of the development of an IDS:

- Acquiring knowledge of the context
- Choosing the abstraction to model input data
- Diversifying data analysis

Context Knowledge There are a number of different approaches that can be chosen to analyse data in order to detect intrusions. One approach can perform better than others, depending also on the input data set. Therefore, the selection of the input data source is an important issue. To select the input source, several factors should be taken into account. First, the environment the system will monitor. The system could be monitoring a specific network application (that uses a certain protocol), or heterogeneous hosts that offer a variety of network services. Secondly, each application can be targeted by a different set of attacks. Thus, the knowledge of attacker techniques plays an important role to select the input source to be analysed. For instance, network flows are the primary source of

information to detect Distribute Denial of Service (DDos) attacks, while HTTP request parameters should be analysed in order to detect SQL Injection attacks. We call the pair $\langle environment, attack\ techniques \rangle$ the "context". Among our tools, Sphinx performs better than POSEIDON because it takes advantage of the knowledge of the syntax of an HTTP request, the way web applications are developed, and the knowledge of (web) attack techniques. The output of this phase is used to create an abstraction model during the following phase.

Data Abstraction Because it is infeasible to store all valid inputs for later comparison with a new incoming input, the system has to extract significant information from input data in order to run its analysis, and abstract the information into a model. The abstraction model has to 1) comprise unique/prominent features of (normal) inputs, 2) allow users to perform modifications in order to change the system behaviour, 3) store information in an efficient way, and 4) allow a verification function to determine in real time if a given input is either normal or malicious. The abstraction model should also allow the verification of future inputs as well, although this feature is difficult to achieve. Requirements 1) and 4) affect the effectiveness and the usability of the system. A model that does not abstract prominent/unique input features is likely to show a low detection rate and a high false alert rate (which impact on final users). The possibility to change the system behaviour is a key component for users, especially in a (complex) enterprise environment.

In Chapter 2 we show that payload-based systems can detect the most harmful cyber-attacks. We believe that the payload information is the most significant input source, and it should be the first source taken into account (when applicable). POSEIDON employs n-gram analysis, a technique which captures a good deal of information from the data payload in a compact way, but which requires continuous updates to reflect the input changes. However, it is impossible to remove from the model used by POSEIDON arbitrary data (e.g., malicious traffic incorporated during the training). The behaviour of the POSEIDON detection engine can be modified only by changing the threshold value. These are strict limitations for users. On the other hand, the positive signatures used by Sphinx allows users to add/delete data processed during the training, and to modify the behaviour of the detection engine. A positive signature not only models the current input, but to some extent also the future input of the web application (thanks to the per-parameter analysis). However, selecting a good abstraction for modelling the payload information is a time consuming task. We have conducted several experiments using only a neural network (a SOM) to detect attacks by analysing the payload of network packets. This approach fails to achieve completeness and accuracy rates comparable to the performances achieved by POSEIDON. Current neural networks fail to handle the packet payloads of modern network applications, and tailored analysis to the application protocol (e.g., Sphinx) should be preferred.

Diversification of Analysis A high quality (and quantity) of the input information is usually not sufficient to cope with increasingly sophisticated new attacks. As we mentioned earlier in this section, attackers develop diverse techniques to exploit similar vulnerabilities. Hence, a single method of analysis is likely to be ineffective with some attacks (or effective with some instances only). Therefore, it is not possible to develop an

"ultimate" detection algorithm that is suitable for any attack. However, the combination of diverse analysis techniques can improve the overall detection. Kruegel et al. [73] employ several detection models to analyse HTTP requests and Sphinx differentiates HTTP parameters into regular and irregular, and applies a different detection technique for each parameter type.

Future Work

The results in this thesis open several possible future research directions:

- ATLANTIDES (Chapter 4) can be extended to include additional information to make the detection of anomalies in the outgoing traffic more precise. This information (e.g., the usual number of bytes sent back from the server and the communication duration) could be included in the model and evaluated as well. The architecture has been designed to work with TCP-based network services: although it could be easily adapted to work with UDP-based services, there are some issues related to this protocol. Since UDP is a connection-less protocol, this presents some difficulties to distinguish real connections from the ones using spoofed IP addresses.

- Sphinx (Chapter 5) detects data-flow attacks, i.e., attacks which exploit vulnerabilities in the way parameter content is handled by the web application. However, we observe that work-flow attacks are becoming common [36]. This type of attack exploits erroneous or inconsistent state management mechanisms inside the web application, in order to bypass authentication and authorization checks. Because the attack does not require to inject suspicious data, Sphinx cannot currently detect such attacks.

- Panacea (Chapter 6) can use different algorithms to classify alerts. The benchmarks with SVM and RIPPER, which approach the classification problem in two different ways, show that each algorithm has its advantages and disadvantages, depending on the circumstances. A possible extension is to use a cascade of SVM and Ripper. We would then use SVM for early classification (when the number of samples is low, and when RIPPER does not perform well), then, when the number of alerts increases, we can train RIPPER, thanks to the batch training mode, and use it for classification as well (RIPPER performs better than SVM when the number of training samples is high). By applying a voting schema to the classification results provided by both algorithms for a given alert (for instance, by considering the confidence value and the number of training samples in a certain class), we might be able to increase the overall accuracy.

- Setting the threshold value is usually a trial-and-error task. We propose heuristics (in Chapter 4) to set such values in a general way, so that users do not need to spend too much time to tune threshold values. However, we have not verified whether this

approach is systematic or suitable only in the context of ATLANTIDES. Because most anomaly-based IDSs require users to set up a threshold value or employ similar mechanisms, more research would be needed to formalize this task (in order to automate it, or at least make it less human-dependent).

Alongside with possible developments and improvements for the SilentDefense components, there are a number of open issues related to anomaly-based intrusion detection systems.

Training data sanitation As we mention in Chapter 2, the training data the IDS engine uses to build the model is crucial to detect as many attacks as possible with a low false alert rate. The "sanitation" of the training data set to remove noise and malicious traffic is a time consuming task, and researchers have not fully addressed this topic to automate it. Cretu et al. [37] present an approach to sanitize a training data set: they employ what they call micro-models. The general idea behind this approach is that in a training set spanning a sufficiently large time interval, an attack instance will appear only in short time frames. A micro-model is an instance of the ABS model built during a time frame. By using a voting scheme, authors eliminate models which are considered anomalous, and use the remaining models to build the final detection model. However, this approach present some drawbacks. First, during benchmarks, authors use 300 hours of data to build micro-models: the authors do not benchmark the approach for shorter time, thus it is unclear if that is the minimum amount of time requested to achieve significant improvements. Should that be the case, a 300 hour training (almost 13 days) might not be acceptable in certain contexts, where network data constantly changes (e.g., think of a web application). Secondly, the main assumption the authors make is that a certain attack instance will not appear (too) often in several micro-models. Think, for instance, of certain malware software that performs requests to a web server for non-existent paths, in order to generate a Denial of Service (we have observed this phenomenon during live experiments in our network). Malicious payloads can be continuously injected for days, hence a good deal of micro-models would be tainted and the sanitation could turn out to be ineffective. Additional research is then needed for this topic.

Interaction with host-based intrusion detection systems However effective an IDS is (be it either signature- or anomaly-based), false positives and negatives are always possible. A combination of network- and host-based systems should be investigated in order to enhance the overall detection (and prevention) of malicious events. So far, research has mainly addressed the problem of correlating alerts generated by heterogeneous systems. What we envisage is that, for certain attacks (e.g., buffer overflow), countermeasures can be activated "on-demand" inside the targeted host. Think of systems like Argos [102], which offer high-end protection but impact host performance. By being able to activate the host-based IDS, the network-based IDS can use, e.g., a lower threshold value, and let suspicious traffic flow to the targeted host where the host-based IDS is now running. Then, based on the host-based IDS analysis, the suspicious activity can

be reported as an attack in case of a real threat (that the host-based IDS can also stop in real-time).

Revisiting public data sets Testing the effectiveness of an intrusion detection engine (and comparing it to other engines) is performed through benchmarks and using data sets. The data sets put together by DARPA a decade ago are the only comprehensive public source of network traffic to test anomaly-based detection engines. These data sets are outdated, when compared to the modern Internet, the data carried and the attack techniques used. Thus, it is difficult for researchers to asses the real effectiveness of a new detection engine. Due to privacy issues, there is no a public and comprehensive data set available that represents the modern Internet traffic data, i.e., web application traffic, and the major threats, i.e., malware, Cross-site Scripting and SQL Injection attacks. Hence, every researcher needs to collect her own data set, whose "quality" (i.e., the diversity of data and attacks) is not easy to evaluate, since most data sets are private and thereby not available to a large audience. The collection of new modern data sets for testing purposes should be a main goal of future research.

Author References

Refereed Conferences

[1] D. Bolzoni, B. Crispo, and S. Etalle. ATLANTIDES: An Architecture for Alert Verification in Network Intrusion Detection Systems. In *LISA '07: Proc. 21th Large Installation System Administration Conference*, pages 141–152. USENIX Association, 2007. **(Subsumed by Chapter 4 of this thesis)**.

[2] D. Bolzoni and S. Etalle. Boosting Web Intrusion Detection Systems by Inferring Positive Signatures. In *Confederated International Conferences On the Move to Meaningful Internet Systems (OTM '08)*, volume 5332 of *LNCS*, pages 938–955. Springer, 2008. **(Subsumed by Chapter 5 of this thesis)**.

[3] D. Bolzoni, S. Etalle, and P.H. Hartel. Panacea: Automating the Classification of Attacks for Anomaly-based Network Intrusion Detection Systems. In *RAID '09: Proc. 12th Symposium on Recent Advances in Intrusion Detection*, LNCS. Springer, 2009. TO APPEAR. **(Subsumed by Chapter 6 of this thesis)**.

[4] D. Bolzoni, E. Zambon, S. Etalle, and P.H. Hartel. POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System. In *IWIA '06: Proc. 4th IEEE International Workshop on Information Assurance*, pages 144–156. IEEE Computer Society Press, 2006. **(Subsumed by Chapter 3 of this thesis)**.

[5] X. Su, D. Bolzoni, and P. van Eck. Understanding and Specifying Information Security Needs to Support the Delivery of High Quality Security Services. In *SECUREWARE '07: Proc. International Conference on Emerging Security Information, Systems, and Technologies*, pages 107–114. IEEE Computer Society, 2007.

[6] E. Zambon, D. Bolzoni, S. Etalle, and M. Salvato. A Model Supporting Business Continuity Auditing & Planning in Information Systems. In *ICIMP '06: Proc. 2nd International Conference on Internet Monitoring and Protection*, pages 33–41. IEEE Computer Society, 2007.

Book Chapters

[7] D. Bolzoni and S. Etalle. Approaches in anomaly-based network intrusion detection systems. In R. Di Pietro and L.V. Mancini, editors, *Intrusion Detection Systems*, volume 38 of *Advances in Information Security*, pages 1–15. Springer, 2008. **(Subsumed by Chapter 2, Sections 2.3.1 and 2.3.2 of this thesis**).

International Workshops

[8] X. Su, D. Bolzoni, and P. A. T. van Eck. A Business Goal Driven Approach for Understanding and Specifying Information Security Requirements. In *EMMSAD '06: Proc. 11th International Workshop on Exploring Modeling Methods in Systems Analysis and Design*, pages 465–472. Presses Universitaries de Namur, 2006.

[9] X. Su, D. Bolzoni, and P. A. T. van Eck. Specifying Information Security Needs for the Delivery of High Quality Security Services (poster session). In *BDIM '07: Proc. 2nd IEEE/IFIP International Workshop on Business-Driven IT Management*, pages 112–113. IEEE Computer Society, 2007.

[10] E. Zambon, D. Bolzoni, S. Etalle, and M. Salvato. Model-based mitigation of availability risks. In *BDIM '07: Proc. 2nd IEEE/IFIP International Workshop on Business-Driven IT Management*, pages 75–83. IEEE Computer Society, 2007.

General References

[11] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99TR-028, Carnegie-Mellon University - Software Engineering Institute, jan 2000.

[12] M. Almgren, H. Debar, and M. Dacier. A lightweight tool for detecting web server attacks. In *NDSS '00: Proc. 7th ISOC Symposium on Network and Distributed Systems Security*, 2000.

[13] M. Almgren and U. Lindqvist. Application-integrated data collection for security monitoring. In *RAID '01: Proc. 4th Symposium on Recent Advances in Intrusion Detection*, volume 2212 of *LNCS*, pages 22–36. Springer, 2001.

[14] J.P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James P. Anderson Co., Fort Washington, PA, April 1980.

[15] D. Andersson, M. Fong, and A. Valdes. Heterogeneous Sensor Correlation: A Case Study of Live Traffic Analysis, 2002.

[16] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security*, 3(3):186–205, 2000.

[17] S. Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. Technical Report 99-15, Chalmers University, mar 2000.

[18] R. Bace. *Intrusion detection*. Macmillan Publishing Co., Inc., 2000.

[19] D. Balzarotti, M. Cova, V.V. Felmetsger, and G. Vigna. Multi-Module Vulnerability Analysis of Web-based Applications. In *CCS '07: Proc. 14th ACM Conference on Computer and Communication Security*, pages 25–35. ACM Press, 2007.

[20] D. Barbará, J. Couto, S. Jajodia, and N. Wu. ADAM: a testbed for exploring the use of data mining in intrusion detection. *SIGMOD Record*, 30(4):15–24, 2001.

[21] D. Barbará, J. Couto, S. Jajodia, and N. Wu. ADAM: Detecting Intrusions by Data Mining. In *IAW '01: Proc. 2nd IEEE SMC Information Assurance Workshop*, 2001.

[22] D. Barbará, J. Couto, S. Jajodia, and N. Wu. Detecting Novel Network Intrusions using Bayes Estimators. In *SIAM '01: Proc. 1st SIAM International Conference on Data Mining*, 2001. `http://www.siam.org/meetings/sdm01/pdf/sdm01_29.pdf` (last accessed: April 2009).

[23] S.D. Bay, D. Kibler, M.J. Pazzani, and P. Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *SIGKDD Exploration: Newsletter of SIGKDD and Data Mining*, 2(2):81–85, 2000.

[24] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[25] R.M. Bolle, J.H. Connell, S. Pankanti, N.K. Ratha, and A.W. Senior. *Guide to Biometrics*. Springer, 2003.

[26] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In *Proc. 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[27] J.D. Cannady. Artificial neural networks for misuse detection. In *NISSC '98: Proc. 21st National Information Systems Security Conference*, pages 443–456, 1998.

[28] J.D. Cannady. *An adaptive neural network approach to intrusion detection and response*. PhD thesis, Nova Southeastern University, 2000.

[29] J.D. Cannady. Next Generation Intrusion Detection: Autonomous Reinforcement Learning of Network Attacks. In *NISSC '00: Proc. 23rd National Information Systems Security Conference*, 2000. `http://csrc.nist.gov/nissc/2000/proceedings/papers/033.pdf` (last accessed: April 2009).

[30] D.J. Chaboya, R.A. Raines, R.O. Baldwin, and B.E. Mullins. Network Intrusion Detection: Automated and Manual Methods Prone to Attack and Evasion. *IEEE Security and Privacy*, 4(6):36–43, 2006.

[31] S.P. Chung and A.K. Mok. Allergy Attack Against Automatic Signature Generation. In *RAID '06: Proc. 9th International Symposium on Recent Advances in Intrusion Detection*, volume 4219 of *LNCS*, pages 61–80. Springer, 2006.

[32] S.P. Chung and A.K. Mok. Advanced Allergy Attacks: Does a Corpus Really Help? In *RAID '07: Proc. 10th International Symposium on Recent Advances in Intrusion Detection*, volume 4637 of *LNCS*, pages 42–62. Springer, 2007.

[33] C. Clifton and G. Gengo. Developing custom intrusion detection filters using data mining. In *MILCOM '00: Proc. 21st Century Military Communications Conference*, volume 1, pages 440–443. IEEE Computer Society Press, 2000.

[34] W.W. Cohen. Fast effective rule induction. In *Proc. 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[35] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: end-to-end containment of Internet worms. In *SOSP '05: Proc. 20th ACM Symposium on Operating Systems Principles*, pages 133–147. ACM Press, 2005.

[36] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna. Swaddler: An approach for the anomaly-based detection of state violations in web applications. In *RAID '07: Proc. 10th International Symposium on Recent Advances in Intrusion Detection*, volume 4637 of *LNCS*, pages 63–86. Springer, 2007.

[37] G.F. Cretu, A. Stavrou, M.E. Locasto, S.J. Stolfo, and A.D. Keromytis. Casting out Demons: Sanitizing Training Data for Anomaly Sensors. In *S&P '08: Proc. 29th IEEE Symposium on Security and Privacy*, pages 81–95. IEEE Computer Society Press, 2008.

[38] F. Cuppens and R. Ortalo. LAMBDA: A Language to Model a Database for Detection of Attacks. In *RAID '00: Proc. 3rd International Symposium on Recent Advances in Intrusion Detection*, pages 197–216. Springer, 2000.

[39] O. Dain and R. Cunningham. Fusing Heterogeneous Alert Streams into Scenarios. In *Proc. Workshop on Data Mining for Security Applications, 8th ACM Conference on Computer Security (CCS' 01)*, pages 1–13. ACM Press, 2002.

[40] M. Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.

[41] H. Debar, D. Curry, and B. Feinstein. The intrusion detection message exchange format (IDMEF). RFC 4765, 2007.

[42] H. Debar, M. Dacier, and A. Wespi. Towards a Taxonomy of Intrusion-Detection Systems. *Computer Networks*, 31(8):805–822, 1999.

[43] H. Debar, M. Dacier, and A. Wespi. A Revised Taxonomy of Intrusion-Detection Systems. *Annales des Télécommunications*, 55(7–8):361–378, 2000.

[44] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *RAID '00: Proc. 4th International Symposium on Recent Advances in Intrusion Detection*, pages 85–103. Springer, 2001.

[45] D.E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, February 1987.

[46] M.O. Depren, M. Topallar, E. Anarim, and K. Ciliz. Network Based Anomaly Intrusion Detection using Self Organizing Maps. In *SIU '04: Proc. 12th IEEE National Conference on Signal Processing and Applications*, pages 76–79, 2004.

[47] S.C. Evans, S.F. Bush, and J. Hershey. Information assurance through kolmogorov complexity. In *DISCEX '00: Proc. DARPA Information Survivability Conference and Exposition II*, volume 2, pages 322–331, 2001.

[48] H. Fernau. Algorithms for Learning Regular Expressions. In *ALT '05: Proc. 16th International Conference on Algorithmic Learning Theory*, volume 3734 of *LCNS*. Springer, 2005.

[49] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999.

[50] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proc. 15th USENIX Security Symposium*, pages 241–256. USENIX Association, 2006.

[51] S. Forrest and S.A. Hofmeyr. A Sense of Self for Unix Processes. In *S&P '96: Proc. 17th IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society Press, 2002.

[52] S. Goel and S.F. Bush. Kolmogorov complexity estimates for detection of viruses in biologically inspired security systems: a comparison with traditional approaches. *Complex.*, 9(2):54–73, 2003.

[53] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley. Worm Detection, Early Warning and Response Based on Local Victim Information. In *ACSAC '04: Proc. 20th Annual Computer Security Applications Conference*, pages 136–145. IEEE Computer Society, 2004.

[54] S. Hansman and R. Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2004.

[55] A. Hinneburg, C.C. Aggarwal, and D.A. Keim. What Is the Nearest Neighbor in High Dimensional Spaces? In *VLDB '00: Proc. 26th International Conference on Very Large Data Bases*, pages 506–515. Morgan Kaufmann, 2000.

[56] J.D. Howard. *An analysis of security incidents on the Internet 1989-1995*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1998.

[57] K.L. Ingham and H. Inoue. Comparing Anomaly Detection Techniques for HTTP. In *RAID '07: Proc. 10th International Symposium on Recent Advances in Intrusion Detection*, volume 4637 of *LNCS*, pages 42–62. Springer, 2007.

[58] K.L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning DFA representations of HTTP for protecting web applications. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51(5):1239–1255, 2007.

[59] H.S. Javitz and A. Valdes. The NIDES Statistical Component Description and Justification. Technical Report A010, SRI, 1994.

[60] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: s Static Analysis Tool for Detecting Web Application Vulnerabilities. In *S&P '06: Proc. 26th IEEE Symposium on Security and Privacy*, pages 258–263. IEEE Computer Society, 2006.

[61] K. Julisch. Mining Alarm Clusters to Improve Alarm Handling Efficiency. In *ACSAC '01: Proc. 17th Annual Computer Security Applications Conference*, pages 12–21. ACM Press, 2001.

[62] K. Julisch. Data Mining for Intrusion Detection: A Critical Review. Research Report RZ 3398, IBM Zurich Research Laboratory, 8803 Ruschlikon, Switzerland, February 2002.

[63] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security*, 6(4):443–471, 2003.

[64] C. Kahn, P.A. Porras, S. Staniford-Chen, and B. Tung. A common intrusion detection framework. *Journal of Computer Security*, 1998.

[65] H. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *Proc. 13th USENIX Security Symposium*, pages 271–286. USENIX Association, 2004.

[66] D.V. Klein. Defending Against the Wily Surfer-Web-based Attacks and Defenses. In *Proc. Workshop on Intrusion Detection and Network Monitoring*, pages 81–92. USENIX Association, 1999.

[67] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, 1995. (Second Extended Edition 1997).

[68] C. Kreibich and J. Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.

[69] C. Kruegel and W. Robertson. Alert Verification: Determining the Success of Intrusion Attempts. In *DIMVA '04: Proc. 1st Workshop on the Detection of Intrusions and Malware and Vulnerability Assessment*, 2004.

[70] C. Kruegel and T. Toth. Using Decision Trees to Improve Signature-based Intrusion Detection. In *RAID '03: Proc. 6th Symposium on Recent Advances in Intrusion Detection*, volume 2820 of *LNCS*, pages 173–191. Springer, 2003.

[71] C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *SAC '02: Proc. 2002 ACM Symposium on Applied Computing*, pages 201–208. ACM Press, 2002.

[72] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *CCS'03: Proc. 10th ACM Conference on Computer and Communications Security*, pages 251–261, 2003.

[73] C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5):717–738, 2005.

[74] K. Labib and V.R. Vemuri. NSOM: A Tool to Detect Denial of Service Attacks Using Self-organizing Maps. Technical report, University of California, Davis, 2002. `http://ailab.das.ucdavis.edu/~klabib/docs/nsom1.9.final.pdf` (last accessed: April 2009).

[75] B.W. Lampson. Computer Security in the Real World. *Computer*, 37(6):37–46, 2004.

[76] W. Lee. *A data mining framework for constructing features and models for intrusion detection systems*. PhD thesis, Columbia University, New York, NY, USA, 1999.

[77] W. Lee, W. Fan, M. Miller, S.J. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1-2):5–22, 2002.

[78] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proc. 7th USENIX Security Symposium*, pages 79–94. USENIX Association, 1998.

[79] W. Lee and S.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.

[80] W. Lee, S.J. Stolfo, and K.W. Mok. A Data Mining Framework for Building Intrusion Detection Models. In *S&P '99: Proc. 20th IEEE Symposium on Security and Privacy*, pages 120–132. IEEE Computer Society Press, 1999.

[81] R.P. Lippmann, R.K. Cunningham, D.J. Fried, S.L. Garfinkel, A.S. Gorton, I. Graf, K.R. Kendall, D.J. McClung, D.J. Weber, S.E. Webster, and M.A.Z̃issman D.W̃yschogrod. The 1998 DARPA/AFRL off-line intrusion detection evaluation. In *RAID '98: Proc. 1st International Workshop on the Recent Advances in Intrusion Detection*, 1998.

[82] R.P. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. Evaluating Intrusion Detection Systems: the 1998 DARPA Off-line Intrusion Detection Evaluation. In *DISCEX '00: Proc. 1st DARPA Information Survivability Conference and Exposition*, volume 2, pages 12–26. IEEE Computer Society Press, 2000.

[83] R.P. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 34(4):579–595, 2000.

[84] M.V. Mahoney and P.K. Chan. Learning non-stationary models of normal network traffic for detecting novel attacks. In *KDD '02: Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 376–385. ACM Press, 2002.

[85] M.V. Mahoney and P.K. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In *RAID '03: Proc. 6th Symposium on Recent Advances in Intrusion Detection*, volume 2820 of *LNCS*, pages 220–237. Springer, 2003.

[86] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz. A Data Mining Analysis of RTID Alarms. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 34(4):571–577, 2000.

[87] J. 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, 2000.

[88] D. Meyer, F. Leisch, and K. Hornik. The support vector machine under test. *Neurocomputing*, 55(1-2):169–186, 2003.

[89] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A Formal Data Model for IDS Alert Correlation. In *RAID '02: Proc. 5th Symposium on Recent Advances in Intrusion Detection*, volume 2516 of *LNCS*, pages 115–127. Springer, 2002.

[90] P.G. Neumann and P.A. Porras. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *NCSC '97: Proc. 20th NIST National Information Systems Security Conference*, pages 353–365, 1997.

[91] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *S&P '05: Proc. 25th IEEE Symposium on Security and Privacy*, pages 226–241. IEEE Computer Society, 2005.

[92] B.V. Nguyen. Self Organizing Map (SOM) for Anomaly Detection. Technical report, Ohio University, 2002.

[93] J. Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., 1995.

[94] P. Ning and Y. Cui. An Intrusion Alert Correlator Based on Prerequisites of Intrusions. Technical Report TR-2002-01, North Carolina State University, 26 2002.

[95] P. Ning, Y. Cui, and D. Reeves. Analyzing intensive intrusion alerts via correlation. In *RAID '02: Proc. 5th Symposium on Recent Advances in Intrusion Detection*, volume 2516 of *LNCS*, pages 74–94. Springer, 2002.

[96] P. Ning, Y. Cui, and D.S. Reeves. Constructing attack scenarios trough correlation of intrusion alerts. In *CCS '02: Proc. 9th ACM Conference on Computer and Communication Security*, pages 245–254. ACM Press, 2002.

[97] P. Ning, Y. Cui, D.S. Reeves, and D. Xu. Techniques and tools for analyzing intrusion alerts. *ACM Transactions on Information and System Security*, 7(2):274–318, 2004.

[98] P. Ning, D. Reeves, and Y. Cui. Correlating Alerts Using Prerequisites of Intrusions. Technical Report TR-2001-13, North Carolina State University, 13 2001.

[99] P. Ning and D. Xu. Learning attack strategies from intrusion alerts. In *CCS '03: Proc. 10th ACM conference on Computer and Communications Security*, pages 200–209. ACM Press, 2003.

[100] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[101] T. Pietraszek. Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection. In *RAID '04: Proc. 7th Symposium on Recent Advances in Intrusion Detection*, volume 3224 of *LNCS*, pages 102–124. Springer, 2004.

[102] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. *SIGOPS Oper. Syst. Rev.*, 40(4):15–27, 2006.

[103] Niels Provos. A Virtual Honeypot Framework. In *Proc. 13th USENIX Security Symposium*, pages 1–14. USENIX Association, 2004.

[104] T.H. Ptacek and T.N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., 1998.

[105] M. Ramadas, S. Ostermann, and B.C. Tjaden. Detecting Anomalous Network Traffic with Self-Organizing Maps. In *RAID '03: Proc. 6th Symposium on Recent Advances in Intrusion Detection*, volume 2820 of *LNCS*, pages 36–54. Springer, 2003.

[106] W. Robertson, G. Vigna, C. Kruegel, and R.A. Kemmerer. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *NDSS '06: Proc. 13th ISOC Symposium on Network and Distributed Systems Security*, 2006.

[107] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *LISA '99: Proc. 13th USENIX Conference on System Administration*, pages 229–238. USENIX Association, 1999.

[108] R.S. Sandhu and P. Samarati. The computer science and engineering handbook. In *Authentication, Access Controls, and Intrusion Detection*, pages 1929–1948. CRC Press, 1997.

[109] R. Sommer and V. Paxson. Enhancing byte-level network intrusion detection signatures with context. In *CCS '03: Proc. 10th ACM conference on Computer and Communications Security*, pages 262–271. ACM Press, 2003.

[110] Y. Song, S.J. Stolfo, and A.D. Keromytis. Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic. In *NDSS '09: Proc. 16th ISOC Symposium on Network and Distributed Systems Security*, 2009.

[111] S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1-2):105–136, 2002.

[112] A.D. Todd, R.A. Raines, R.O. Baldwin, B.E. Mullins, and S.K. Rogers. Alert verification evasion through server response forging. In *RAID '07: Proc. 10th Symposium on Recent Advances in Intrusion Detection*, volume 4637 of *LNCS*, pages 256–275. Springer, 2007.

[113] F. Valeur, G. Vigna, C. Kruegel, and R.A. Kremmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Trans. Dependable Secur. Comput.*, 1(3):146–169, 2004.

[114] H.L. van Trees. *Detection, Estimation and Modulation Theory. Part I: Detection, Estimation, and Linear Modulation Theory*. John Wiley and Sons, Inc., 1968.

[115] V.N. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

[116] G. Vigna, W.K. Robertson, and D. Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. In *CCS '04: Proc. 11th ACM Conference on Computer and Communications Security*, pages 21–30. ACM Press, 2004.

[117] D. Wagner and D. Dean. Intrusion Detection via Static Analysis. In *S&P '01: Proc. 22nd IEEE Symposium on Security and Privacy*, page 156. IEEE Computer Society, 2001.

[118] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *CCS '02: Proc. 9th ACM conference on Computer and Communications Security*, pages 255–264. ACM, 2002.

[119] K. Wang, G. Cretu, and S.J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *RAID '05: Proc. 8th International Symposium on Recent Advances in Intrusion Detection*, volume 3858 of *LNCS*, pages 227–246. Springer, 2006.

[120] K. Wang, J.J. Parekh, and S.J. Stolfo. Anagram: a Content Anomaly Detector Resistant to Mimicry Attack. In *RAID '06: Proc. 9th International Symposium on Recent Advances in Intrusion Detection*, volume 4219 of *LNCS*, pages 226–248. Springer, 2006.

[121] K. Wang and S.J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In *RAID '04: Proc. 7th Symposium on Recent Advances in Intrusion Detection*, volume 3224 of *LNCS*, pages 203–222. Springer, 2004.

[122] R. Werlinger, K. Hawkey, K. Muldner, P. Jaferian, and K. Beznosov. The challenges of using an intrusion detection system: is it worth the effort? In *SOUPS '08: Proc. 4th Symposium on Usable Privacy and Security*, pages 107–118. ACM Press, 2008.

[123] K. Yamanishi, J. Takeuchi, G.J. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *KDD '00: Proc. 6th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 320–324. ACM Press, 2000.

[124] S. Zanero. Analyzing TCP Traffic Patterns using Self Organizing Maps. In *ICIAP '05: Proc. 13th International Conference on Image Analysis and Processing*, volume 3617 of *LNCS*, pages 83–90. Springer, 2005.

[125] S. Zanero and S.M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *SAC '04: Proc. 19th Annual ACM Symposium on Applied Computing*, pages 412–419. ACM Press, 2004.

[126] J. Zhou, A.J. Carlson, and M. Bishop. Verify Results of Network Intrusion Alerts Using Lightweight Protocol Analysis. In *ACSAC '05: Proc. 21st Annual Computer Security Applications Conference*, pages 117–126. IEEE Computer Society, 2005.

[127] H. Zimmermann. OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.

Web References (Last Accessed: April 2009)

[128] Breach Security. ModSecurity™. `http://www.modsecurity.org`.

[129] CERT. Vulnerability in NCSA/Apache CGI example code. Technical report, CERT Coordination Center, 1996. `http://www.cert.org/advisories/CA-1996-06.html`.

[130] CERT. IP Denial of Service Attacks. Technical report, CERT Coordination Center, 1997. `http://www.cert.org/advisories/CA-1997-28.html`.

[131] Check Point Software Technologies. Stateful Inspection Technology, 2005. `http://www.checkpoint.com/products/downloads/Stateful_Inspection.pdf`.

[132] CIRT.net. Nikto web scanner. `http://www.cirt.net/nikto2`.

[133] DEFCON8. Defcon Capture the Flag (CTF) contest, 2000. `http://www.defcon.org/html/defcon-8/defcon-8-post.html`.

[134] SANS Institute – Internet Storm Center web site. `http://isc.sans.org/index.php?on=toptrends`.

[135] Milw0rm. `http://milw0rm.com`.

[136] The University of Waikato. Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/.

[137] Open Source Security Information Management (OSSIM). http://www.ossim.net.

[138] PostNuke. PostNuke Content Managament System. http://www.postnuke.com/.

[139] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. http://www.R-project.org.

[140] Supervisory control and data acquisition. http://en.wikipedia.org/wiki/SCADA.

[141] Tenable Network Security. Nessus Vulnerabilty Scanner. http://www.nessus.org/.

[142] Security Reason. PostNuke Input Validation Error, 2005. http://securitytracker.com/alerts/2005/May/1014066.html.

[143] Sourcefire. Snort Network Intrusion Detection System web site. http://www.snort.org.

[144] Symantec Corporation. Internet Security Threat Report, 2008. http://www.symantec.com/enterprise/threatreport/index.jsp.

[145] Snort Team. Snort user manual. http://www.snort.org/docs/snort_htmanuals/htmanual_2832/node220.html.

[146] The Honeynet Project. Sebek. http://www.honeynet.org/project/sebek/.

[147] The MITRE Corporation. Common Vulnerabilities and Exposures database. http://cve.mitre.org.

[148] The Open Web Application Security Project. OWASP Top Ten Most Critical Web Application Security Vulnerabilities. http://www.owasp.org/index.php/OWASP_Top_Ten.

[149] Web Application Security Consortium. Web Security Threat Classification. http://www.webappsec.org/projects/threat/.

Samenvatting

Intrusion detection systemen (IDS) zijn algemeen bekend en een breed in te zetten beveiligingshulpmiddel voor het opsporen van cyberaanvallen en schadelijke activiteiten in computersystemen en netwerken.

Een IDS op basis van patronen (signature) werkt vergelijkbaar met anti-virus software. Het heeft een database van bekende aanvallen, en een succesvolle match met de huidige invoer veroorzaakt een alarmsignaal. Een IDS op basis van patronen kan geen onbekende aanvallen detecteren, hetzij omdat de database is verlopen of omdat er nog geen patroon beschikbaar is.

Om deze beperking te ondervangen, hebben onderzoekers een IDS op basis van afwijking (anomaly detection) ontwikkeld. Een IDS op basis van afwijking werkt door het bouwen van een model met normale data/gebruik van patronen, vervolgt door het vergelijken van de huidige invoer met het model (gebruik makend van een metrische overeenkomst). Een significant verschil is dan gemarkeerd als afwijking. Een IDS op basis van afwijking is in staat om eerder onbekende (of modificatie van bekende) aanvallen te detecteren zodra ze plaatsvinden, d.w.z. zero-day aanvallen.

Cyberaanvallen en schendingen van informatiebeveiligingen lijken in frequentie en met impact toe te nemen. Het is aannemelijk dat een IDS op basis van patronen een steeds groter deel van de pogingen tot aanval mist, aangezien cyberaanvallen diversifiëren. Je zou dus verwachten dat een groot aantal IDSen op basis van afwijkingen zijn ingezet voor het opsporen van de nieuwste ontwrichtende aanvallen. Echter, de meeste IDSen die vandaag de dag worden gebruikt zijn op basis van patronen, en er zijn maar enkele IDSen op basis van afwijkingen ingezet in productieomgevingen.

Tot nu toe is een IDS op basis van patronen gemakkelijk te implementeren en eenvoudiger te configureren en te onderhouden dan een IDS op basis van een afwijking, d.w.z. het is gemakkelijker en minder kostbaar in gebruik. We zien in deze beperkingen de belangrijkste reden waarom systemen op basis van afwijkingen niet op grote schaal zijn ingezet, ondanks het onderzoek dat voor meer dan een decennium is uitgevoerd.

Om deze beperkingen aan te pakken hebben wij SilentDefense ontwikkeld, een uitvoering op basis van anomaly intrusion detection architecture die beter presteert dan con-

currenten. Niet alleen in termen van aanvallen detectie en vals alarm percentage, maar het kost de gebruiker ook minder inspanning. Verschillende geïntegreerde componenten vormen de architectuur van SilentDefense: elk component kan zelfstandig werken, maar ze kunnen ook worden aangesloten op verschillende configuraties om diverse (geautomatiseerde) faciliteiten aan te bieden aan gebruikers wat de inspanning vermindert. In het bijzonder, SilentDefense:

- Verbetert het bekende algoritme PAYL (voor het http protocol, van 90% tot 100% detectie percentage en van 0,17% tot 0,0016% vals alarm percentage) door toevoeging van een neuraal netwerk dat netwerkverkeer voorbewerkt;
- Vermindert het aantal vals positieve meldingen (tussen 50% en 100% minder meldingen) door het correleren van meldingen die gegenereerd worden door een Intrusion Detection System (hetzij op basis van patronen of afwijkingen) die controleert op het inkomende verkeer met een contentbased analyse van uitgaand verkeer;
- Genereert automatisch de regelmatig te valideren inkomende http-verzoeken, voor gebruikers om de op basis van afwijking detection engine te kunnen tunen;
- Automatiseert de rangschikking van de meldingen op basis van afwijking door de extractie van de lading van eerdere signaleringen, die kunnen worden ingedeeld met behulp van zowel standaard als user-defined taxonomies.

SilentDefense is de eerste die een systematische poging doet om een intrusion detection system op basis van afwijking met een hoge mate van gebruikersvriendelijkheid te ontwikkelen. Echter, SilentDefense (en in het algemeen detectie op basis van afwijking) is geen alternatief voor systemen op basis van patronen. In feite zijn wij van mening dat de beste kans om een aanval op te sporen, gebeurd door middel van een combinatie van de twee benaderingen. Een systeem op basis van patronen werkt beter voor bekende applicaties en systemen (bijvoorbeeld de Windows-besturingssystemen) terwijl SilentDefense zero-day en gerichte aanvallen kan opsporen. De laatste aanval doelt gewoonlijk op maat ontwikkelde software, zoals webapplicaties ontwikkelt door ondernemingen voor interne gebruikers of particuliere systemen die worden gebruikt in kritische infrastructuren.

Titles in the IPA Dissertation Series since 2005

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*.

Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*.

Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathe-

matics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14