UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

ANOMALY DETECTION OF WEB-BASED ATTACKS

Gustavo Miguel Barroso Assis do Nascimento

MESTRADO EM SEGURANÇA INFORMÁTICA

November 2010

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

ANOMALY DETECTION OF WEB-BASED ATTACKS

Gustavo Miguel Barroso Assis do Nascimento

Orientador
Miguel Nuno Dias Alves Pupo Correia

MESTRADO EM SEGURANÇA INFORMÁTICA

November 2010

Resumo

Para prevenir ataques com sucesso, é crucial que exista um sistema de detecção que seja preciso e completo. Os sistemas de detecção de intrusão (IDS) baseados em assinaturas são uma das abordagens mais conhecidas para o efeito, mas não são adequados para detectar ataques *web* ou ataques previamente desconhecidos. O objectivo deste projecto passa pelo estudo e desenho de um sistema de detecção de intrusão baseado em anomalias capaz de detectar esses tipos de ataques.

Os IDS baseados em anomalias constroem um modelo de comportamento normal através de dados de treino, e em seguida utilizam-no para detectar novos ataques. Na maioria dos casos, este modelo é representativo de mais exemplos de comportamento normal do que os presentes nos dados de treino, característica esta a que chamamos generalização e que é fundamental para aumentar a precisão na detecção de anomalias. A precisão da detecção e, portanto, a utilidade destes sistemas, é consideravelmente influenciada pela fase de construção do modelo (muitas vezes chamada fase de treino), que depende da existência de um conjunto de dados sem ataques que se assemelhe ao comportamento normal da aplicação protegida. A construção de modelos correctos é particularmente importante, caso contrário, durante a fase de detecção, provavelmente serão geradas grandes quantidades de falsos positivos e falsos negativos pelo IDS.

Esta dissertação detalha a nossa pesquisa acerca da utilização de métodos baseados em anomalias para detectar ataques contra servidores e aplicações *web*. As nossas contribuições incidem sobre três vertentes distintas: i) procedimentos avançados de treino que permitem aos sistemas de detecção baseados em anomalias um bom funcionamento, mesmo em presença de aplicações complexas e dinâmicas, ii) um sistema de detecção de intrusão que compreende diversas técnicas de detecção de anomalias capazes de reconhecer e identificar ataques contra servidores e aplicações *web* e iii) uma avaliação do sistema e das técnicas mais adequadas para a detecção de ataques, utilizando um elevado conjunto de dados reais de tráfego pertencentes a uma aplicação *web* de grandes dimensões alojada em servidores de produção num ISP Português.

Palavras-chave: detecção de anomalias, detecção de intrusões, ataques web, computadores, redes.

Abstract

To successfully prevent attacks it is vital to have a complete and accurate detection system. Signature-based intrusion detection systems (IDS) are one of the most popular approaches, but they are not adequate for detection of web-based or novel attacks. The purpose of this project is to study and design an anomaly-based intrusion detection system capable of detecting those kinds of attacks.

Anomaly-based IDS can create a model of normal behavior from a set of training data, and then use it to detect novel attacks. In most cases, this model represents more instances than those in the training data set, a characteristic that we designate as generalization and which is necessary for accurate anomaly detection. The accuracy of such systems, which determines their effectiveness, is considerably influenced by the model building phase (often called training), which depends on having data that is free from attacks resembling the normal operation of the protected application. Having good models is particularly important, or else significant amounts of false positives and false negatives will likely be generated by the IDS during the detection phase.

This dissertation details our research on the use of anomaly-based methods to detect attacks against web servers and applications. Our contributions focus on three different strands: i) advanced training procedures that enable anomaly-based learning systems to perform well even in presence of complex and dynamic web applications; ii) a system comprising several anomaly detection techniques capable of recognizing and identifying attacks against web servers and applications and iii) an evaluation of the system and of the most suitable techniques for anomaly detection of web attacks, using a large data set of real-word traffic belonging to a web application of great dimensions hosted in production servers of a Portuguese ISP.

Keywords: anomaly detection, intrusion detection, web attacks, computers, networks.

Acknowledgments

This M.Sc. was an incredibly rich and rewarding experience which culminates with this dissertation. I would like to thank several people for their help and support during this program, as getting here would not have been possible without them.

My sincere thanks to my adviser Miguel Correia, for his time, patience, orientation, expertise and valuable inputs provided. Likewise, to Paulo Sousa, who also advised and gave me the initial guidance, which allowed me to focus and structure the work in order to achieve what was being proposed.

Many thanks to Kenneth Ingham for his invaluable help, insight and readiness to answer my questions.

Thanks to all my class colleagues for their companionship and for the sharing of experiences.

I would like to thank my company TMN and Portugal Telecom, for supporting my presence in this program. Also, thanks to all the colleagues from the ISP team, namely Nuno Loureiro, who helped me with everything I needed from their side in order to write this dissertation.

And now, the most important acknowledgement, special thanks to my family for their love and support, which played the most decisive role for my success during these last sixteen months.

Lisbon, November 2010

Dedicated to my parents, Raquel and José.

Contents

List of Figures

List of Tables

Abbreviations

ASCII	American Standard Code for Information Interchange
CGI	Common Gateway Interface
CRLF	Carriage Return and Line Feed
DR	Detection Rate
FN	False Negative
FP	False Positive
FPR	False Positive Rate
HIDS	Host-Based Intrusion Detection System(s)
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ID	Intrusion Detection
IDS	Intrusion Detection System(s)
IP	Internet Protocol
ISP	Internet Service Provider
NFA	Non-deterministic Finite Automaton
NIDS	Network-Based Intrusion Detection System(s)
OS	Operating System
PHP	PHP: Hypertext Preprocessor
RFI	Remote File Inclusion
ROC	Receiver Operating Characteristic
TCP	Transmission Control Protocol

TN	True Negative
TPR	True Positive Rate
TP	True Positive
UDP	User Datagram Protocol
WAF	Web Application Firewall
WWW	World Wide Web
XSS	Cross Site Scripting

Chapter 1

Introduction

Ever since computers exist, computer networks have not ceased to grow and evolve. It is a known fact that computer networks have become essential tools in the development of most enterprises and organizations in various fields such as banking, insurance, military, etc. The increase in interconnection between several systems and networks has made them accessible by a vast and diversified user population that keeps on growing. The users, known or not, do not always carry good intentions when accessing these networks. They can try to read, modify or destroy critical information or just attack and disturb the system. Since these networks can be potential targets of attacks, making them safe is an important aspect that cannot be ignored. Over the past two decades, the growing number of computer security incidents witnessed in the Internet has reflected the very growth of the Internet itself.

As a facet of the Internet growth, the Hypertext Transfer Protocol (HTTP) has been widely used throughout the years. Applications are nowadays designed to run as a web service and are deployed across the Internet using HTTP as its standard communication protocol. The success and broad acceptance of web applications shaped a trend suggesting that people will heavily rely of web applications in the years to come. The popularity of such web applications has caught the attention of attackers which try to exploit its vulnerabilities. According to the CSI/FBI Computer Crime and Security Survey of 2005 [1], 95% of the respondent organizations reported having experienced more than 10 incidents related to their web sites.

Additionally, as shown in by the CVE vulnerability trends [2] over a period of five years, the total number of publicly reported web application vulnerabilities rose sharply, to the point where they overtook buffer overflows. Three of the most typical web related attacks which contributed to this trend are: SQL-Injection, Cross Site Scripting (XSS) and Remote

File Inclusion (RFI). The main reasons for this tendency can be explained by the ease of discovery and exploitation of web vulnerabilities, combined with the proliferation of low-grade software applications written by inexperienced developers. Figure 1.1, which show the evolution of attack sophistication versus the necessary technical knowledge of the intruders throughout the years, is a solid evidence of this tendency.

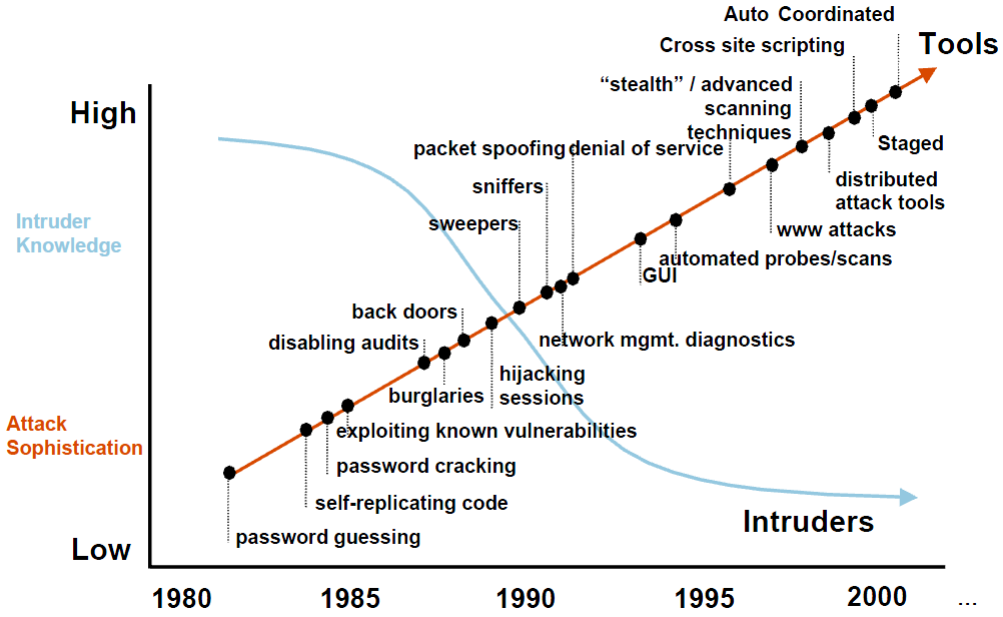

Figure 1.1: Attack Sophistication vs. Intruder Technical Knowledge (source CERT/CC).

Intrusion Detection (ID) refers to the capacity of a computer system to automatically determine a security breach, by analyzing security related events. Intrusion Detection Systems (IDS) have long been a topic for theoretical research and development, and gained mainstream popularity as companies moved many of their critical business interactions to the Internet. Such technology, helps security administrators by alerting them to suspicious activity that may be occurring on their systems and networks in real time. An Intrusion Detection System (IDS) can provide information in advance about attacks, or intrusion attempts, by detecting the actions of intruders. Since many computer system infrastructures are vulnerable to attacks, ID became an important technology business sector, as well as an active area of research [3] that helps enterprises to keep their systems, and consequently their businesses, safe. The main idea behind an IDS is to detect deviations from the normal behavior of a monitored resource. Taking into account that when someone voluntarily attempts to tamper with a system, he or she will alter some activities or parameters of the system itself, the outcomes of such activities are supposed to deviate from the normal behavior and thus be detected by the IDS.

An IDS can be classified according to several features, e.g., the kind of data its engine analyzes (host/network data) and the way it detects anomalies (signature or anomaly-based). In this dissertation, we give particular emphasis to anomaly-based IDS, in the context of detecting attacks against web servers and applications, as these represent a significant fraction of the security problems related to computer networks nowadays.

In order to detect known web attacks, signature-based IDS possess a large number of signatures. Unfortunately, it is hard to keep up with the constant disclosure of web vulnerabilities and thus, signature-based systems are not sufficient and should be complemented by anomaly detection systems, which is the main subject addressed by this work.

1.1 Motivation and Objectives

Anomaly-based intrusion detection systems have the potential to detect novel, or previously unknown attacks (also referred to as zero-day attacks[1]). Nevertheless, these systems typically suffer from high rates of false positives and can be evaded by mimicry attacks (i.e., a variation of attacks that attempts to pass by as normal behavior), for example by using byte substitution and/or padding techniques. In order to address these problems and increase the accuracy of anomaly-based IDS, one of the main challenges lies in controlling model generalization. An anomaly-based IDS learns and builds a model of normal behavior derived from observations in a set of training data. Normal behavior assumes that a service is used according to what their administrators intend for it, in other words, it assumes no attacks are present. After the model is created, consequent observations which deviate from it are categorized as anomalies. In order to achieve more than simply memorizing the training data, an anomaly-based IDS must generalize. By generalizing, an anomaly-based IDS accepts input similar, but not necessarily identical to that of the training data set, which implies the set of instances considered normal will encompass a larger set than those in the training data. An anomaly detection system that under generalizes generates too many false positives, while one that over generalizes misses attacks. Therefore, correct generalization is a necessary condition for the accuracy of an anomaly-based IDS.

Another major challenge, which we explicitly address with our research, is the significant influence the model creation phase (often called training) imposes on the system's overall detection accuracy. This phase depends on the quality of the traffic or data that is used to resemble normal activity of the protected system or application, which should be free

[1] A zero-day attack exploits a software vulnerability that has not been made public yet.

from attacks. In this dissertation, we propose an algorithm to sanitize (i.e., remove attacks) a dataset of web requests collected from captures of network traffic, in order to make it suitable for training the anomaly detection system. The main goal of this process is to enable anomaly-based systems to build their statistical models using as input data without attacks.

Another objective of our research was to devise an anomaly-based intrusion detection system comprising multiple anomaly detection techniques apt to recognize and indentify attacks against web servers and applications, based on models that allow the distinction between normal and anomalous requests.

As a final and main goal of our research, we evaluate the usage of the anomaly-based IDS with a large data set of real-world traffic from a production environment hosting a web application of great dimensions, as well as the accuracy for each of the anomaly detection techniques it includes.

1.2 Thesis Overview

In Chapter 2 we contextualize our work by discussing intrusion detection, providing a taxonomy of Intrusion Detection Systems alongside with their role and requirements. Then we discuss the World Wide Web (WWW) and the Hypertext Transfer Protocol (HTTP) protocol as well as the typical classes of attacks that target it. Next, we discuss underlying issues related to the anomaly detection of attacks using the HTTP protocol. To conclude, we present previous work in the area of anomaly detection for web-based attacks.

Chapter 3 details our research on the usage of anomaly-based IDS with data from a production environment hosting a web application of great dimensions. We discuss the concept of generalization and its importance in anomaly detection and explain how models should be created. Next, we propose an algorithm that can be used to sanitize a dataset of requests in order to remove attacks contained within so that it can be used to train anomaly detection systems. Moreover, we explain how similarity metrics are used to detect anomalies and then describe nine different algorithms suggested in previous literature to detect anomalies in web traffic, discussing their virtues and limitations. Furthermore, we present our anomaly-based IDS, devised to encompass all those algorithms, as well as the generalization heuristics that we utilized.

Chapter 4 presents the results of our experiments and an evaluation of the algorithms for anomaly detection of web attacks. We explain how we obtained and defined our datasets,

and provide a brief description of how our IDS works. Then we depict the requirements in terms of space needed for each model, running time for creating the model (training) and using it to test for anomalies (detection). Finally, we analyze each of the models by discussing their results and by illustrating and comparing the detection accuracy between them.

In Chapter 5 we conclude and point out some of the issues encountered during our research, highlight our contributions and discuss future work.

1.3 Terminology

Along this dissertation, the acronym IDS (Intrusion Detection System) is used in the singular and plural to identify Intrusion Detection System(s) network-based (NIDS) and host-based (HIDS). The terms *web* and *http* are used interchangeably when referring to requests or attacks that target World Wide Web servers or applications using the HTTP protocol.

Chapter 2

Context and Related Work

In this chapter, we discuss Intrusion Detection, introduce definitions, evaluation methods, and present a taxonomy of Intrusion Detection Systems outlining their role and requirements. Then we discuss the World Wide Web and the HTTP protocol, web vulnerabilities as well as the typical classes of attacks against web servers and applications. Next, we discuss underlying issues related to the anomaly detection of attacks using the HTTP protocol. To conclude, we present previous work in the field of anomaly detection for web-based attacks.

2.1 Intrusion Detection

Before discussing intrusion detection, we have to define what an intrusion is. Intrusions are defined in relation to a security policy, which defines what is permitted and what is denied on a system, and unless you know what is allowed or not on your system, attempting to detect intrusions has little consequence.

Definition (Intrusion) An intrusion is a set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource [4].

Definition (Intrusion Detection) Intrusion detection is the process of identifying and responding to malicious activity targeted at computing and networking resources [5].

Definition (Intrusion Detection System) An intrusion detection system monitors computer systems and/or network activity to identify malicious or suspicious events (i.e., intrusions). Each time such an event occurs, the IDS raises an alert.

2.1.1 Evaluation

In order to evaluate an IDS, we first need to choose good metrics to be able to ascertain its quality and detection capabilities. Then, we need to run it on some previously collected data, such as the DARPA 1999 IDS dataset [6], which is the most popular publicly available dataset used to evaluate IDS. A model for an IDS can be created by using data collected from different sources, like capturing network packets or log files entries from applications, which can be seen as objects for the IDS. The IDS will then decide whether each of these objects is an attack or not. In signature-based IDS, the decision is made upon the signatures present in the IDS's rule-set database, whereas anomaly-based IDS decide relying on statistical models. In either case, the IDS will label each object as normal or as an attack. Therefore, the main idea behind evaluating an IDS is to identify how many of those objects are labeled correctly and how many are not.

For assessing the quality of IDS systems, we first need to define some terms. When an IDS is looking at the network traffic or events, it tries to decide whether the traffic or event is malicious or not. When the IDS indicates an intrusion, this is called a positive. If the corresponding alert refers to a real intrusion attempt, then we have a True Positive (TP), that is, the positive assertion from the IDS should be trusted, as it is a correct assertion. However, the IDS can also indicate an intrusion even though no attack has happened. In this case, we have what we call a False Positive (FP).

Definition (True Positive) A true positive is a real alert, raised in response to an intrusion attempt. It indicates that the intrusion detection system detects precisely a particular attack having occurred.

Definition (False Positive) A false positive is a false alert, raised in response to a non-malicious behavior, i.e., an event incorrectly identified by the IDS as being an intrusion when none has occurred. It indicates that the intrusion detection system detects an attack despite no real attack having occurred.

On the other hand, if the IDS indicates the traffic or event is harmless, this is called a negative. When the IDS indicates there is no intrusion and this is a correct assertion, is it called a True Negative (TN). On the contrary, when the IDS does not indicate an intrusion, but an intrusion attempt actually exists, we have a False Negative (FN).

Definition (True Negative) A true negative is the event when no alert is raised and no intrusion attempt takes place. It indicates that the intrusion detection system has not made a mistake in detecting a normal condition.

Definition (False Negative) A false negative is the event when no alert is raised but a real intrusion attempt takes place, i.e., an event that the IDS fails to identify as an intrusion when one has in fact occurred.

Definition (True Positive Rate) The true positive rate (TPR) or detection rate (DR) of an IDS is defined as:

$$TPR = \frac{TP}{TP + FN}$$

Where TP is the number of true positives and FN is the number of false negatives. The perfect IDS is such that $TPR = 1$. The TPR measures the amount of malicious events correctly classified and reported as alerts.

Definition (False Positive Rate) The false positive rate is defined as:

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

Where FP is the number of false positives and TN is the number of true negatives. The False Positive Rate (FPR) measures the amount of legitimate events that are incorrectly classified and reported as alerts.

2.1.2 ROC Curves

Receiver Operating Characteristic (ROC) curves are often used to evaluate the quality of an IDS (see [7] and [8] for more information on ROC Curve analysis).

A ROC curve is a two-dimensional depiction of the accuracy of a signal detector, as it arises on a given set of testing data. Two dimensions are required to show the whole story of how the true-positive rate of detection decreases as the false-positive rate of error increases. The purpose of a ROC curve is to indicate the accuracy of the corresponding signal detector. This accuracy information, revealed in the shape of the curve, is two-dimensional because there are two kinds of events, and hence two kinds of accuracies possible. The first dimension is the success rate of detecting signal events, which is shown along the y-axis (the vertical axis). The second dimension is the error rate of falsely identifying noise events, which is shown along the x-axis (the horizontal axis). Since success is good and error is bad, a good ROC curve will have y-values which grow at a faster rate than its x-values, resulting in a curve shape which rises swiftly upward. Later on, as the decision threshold changes to become more and more lenient, the error values for noise (x-values) must also

grow large, catching up with the success values for signals (y-values). This makes the curve bend over to the right, until it touches the point (0,1)[7].

The ROC curve for an IDS is basically the plot between the TPR and the FPR rates as the threshold value is varied, in order to show the tradeoff between them. It is obtained by tuning the IDS to tradeoff false positives (FP) against true positives (TP), where each point of the ROC curve corresponds to a fixed amount of TPR and FPR calculated under certain sensitivity parameters (threshold). The ROC curve provides a compact and easy way to visually understand how the tradeoff varies for different values of the threshold. The way ROC curves indicate the accuracy of the detector, is by the rise of the curve.

In Figure 2.1, (a) shows a perfect ROC curve that passes through the point (0,1) while a ROC curve no better than chance lies along the positive diagonal. Shown in 2.1 (b), each point along the ROC curve corresponds to a different threshold. Points closer to the origin (0,0) indicate a more exclusive criterion; points closer to the upper-right (1,0) indicate a more inclusive one (source [7]).

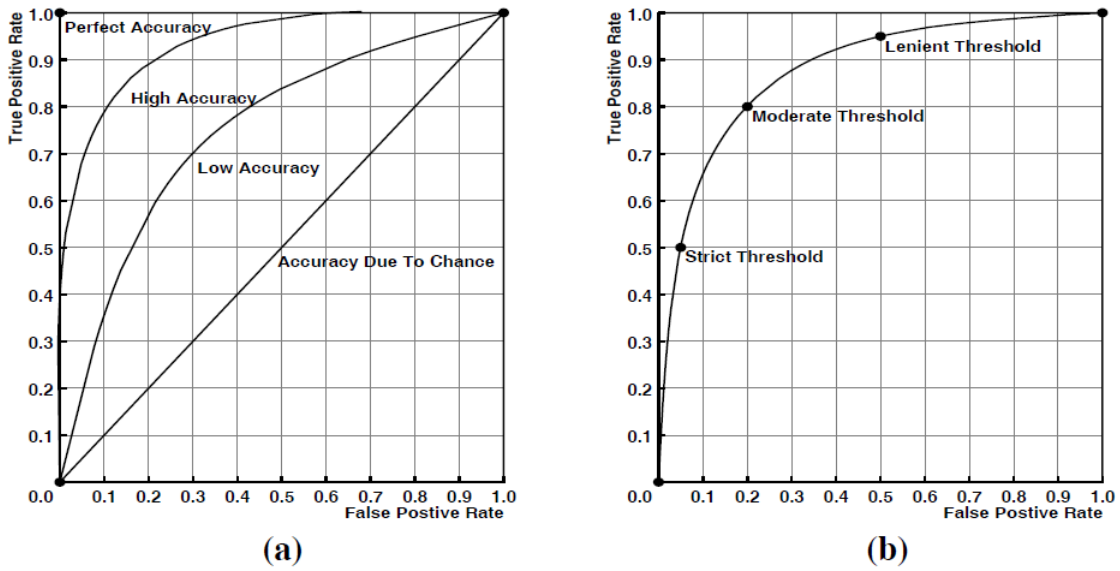

Figure 2.1: ROC Curve examples.

2.2 IDS Taxonomy

Debar et al. [9] were the first to introduce a systematic and taxonomic approach of Intrusion Detection Systems (later revised in [10]). Axelsson [11] provided a comprehensive survey

and taxonomy of IDS, which addressed some aspects in more depth, namely the detection principles.

In this section, we present a taxonomy of IDS, both for Host and Network-based as well as for Signature and Anomaly-based approaches. We introduce the basic definitions and discuss typical advantages and disadvantages by comparing the different approaches and also note some examples of real systems.

2.2.1 Characterization of IDS based on Data Source

An IDS can be classified according to the type of activity it monitors, more specifically, according to the source of the data that is used to build its detection model. Usually, this classification falls in one of two categories, either Host or Network-based.

2.2.1.1 Host-based Systems

Intrusion detection has been an active field of research for three decades. In 1980, Anderson [12] published an influential paper entitled "Computer Security Threat Monitoring and Surveillance", later followed by an early abstract model of a typical IDS proposed by Denning [13] in 1987, which inspired the research community and was the catalyst for many of the existing IDS products nowadays. The earliest forms of these programs focused on discovering security incidents by reviewing system audit logs and accounting files - resembling many modern log analyzers. Since then, a number of these IDS were designed and deployed as surveyed by Mukherjee et al. [14]. These programs then started evolving by adding real-time log monitoring and more elaborate system checks, and are nowadays most commonly known as Host-based Intrusion Detection Systems (HIDS).

A Host-based IDS (HIDS) monitors a single host (or application) and audits data traced by the hosting Operating System (OS) or application, such as system logs or resource usage. An known example of a HIDS is OSSEC[1].

However, due to the growth of the Internet and the increase in complexity of modern operating systems HIDS are not as popular as they once were, as it is very difficult to achieve a complete coverage of the system being monitored. Since an HIDS typically runs on the same hardware and shares the same resources with the actual system or application that is being monitored, the eventual hit on performance of the overall system has also made this kind of IDS less popular.

[1] `http://www.ossec.net/`

Table 2.1 summarizes some of the advantages and disadvantages of Host-based IDS.

Advantages	Detect local attacks that do not involve the network
	Can monitor and analyze what an application or operating system is doing
	Do not need additional hardware to be installed
Disadvantages	Have to be installed on every host you want to monitor
	Can impact or eventually degrade system performance

Table 2.1: Advantages and disadvantages of Host-based IDS.

2.2.1.2 Network-based Systems

Later on, we witnessed the birth and generalization of Network-based Intrusion Detection Systems (NIDS), which were primarily packet sniffers that captured data according to a set of rules or filters, flagging those which appeared to be malicious in nature. A NIDS can be considered an effective second line of defense against attacks directed at computer systems and networks [15]. Due to the increasing severity and likelihood of network-based attacks from the Internet, NIDS are nowadays employed in almost all large-scale IT infrastructures [3].

A Network-based IDS (NIDS) monitors a network segment and analyzes the traffic which flows through it. Network intrusion detection systems are driven by the interpretation of raw network traffic. They examine the contents of actual packets transmitted on the network and parse those packets, analyzing the protocols used on the network, and extracting relevant information from them. This is typically achieved by listening to the network passively, and capturing copies of packets that are transmitted by other machines.

Passive network monitoring takes advantage of promiscuous mode access, where a promiscuous network device, or sniffer, obtains copies of packets directly from the network media, regardless of their destination (normal devices only read packets addressed to them). NIDS then attempt to detect attacks by watching for patterns of suspicious activity in this traffic, and they are typically good at discerning attacks that involve low-level manipulation of the network. This type of IDS, can also easily correlate attacks against multiple machines on a network [16].

The main advantage of NIDS is their ability to monitor data and events without affecting the host performance. Regardless, NIDS lack the ability of determining what exactly is occurring inside a computer system, and that is their main disadvantage, whereas HIDS have this information in great abundance. For example, a NIDS correctly analyze applications or protocols which use data encryption, unless the encryption key is made available to them.

Table 2.2 lists some of the advantages and disadvantages of Network-based IDS.

Advantages	Can monitor multiple hosts at the same time
	Can correlate attacks against multiple hosts
	Do not affect host performance
	Can detect attacks that are not locally visible from hosts
Disadvantages	Must be able to keep up with the network speed
	May have problems with encrypted channels

Table 2.2: Advantages and disadvantages of Network-based IDS.

Two typical and well known examples of a NIDS are Snort[2] and Prelude[3].

2.2.2 Characterization of IDS based on Model of Intrusions

2.2.2.1 Signature-based systems

A signature-based IDS (also known as misuse-based), e.g. Snort [17], includes a signature database of known attacks and works much like anti-virus software, raising an alert when it matches an attack in its database. Those signatures (also known as rules) typically address widely used systems or applications for which security vulnerabilities exist and are widely advertised. Nevertheless, similarly to anti-virus software that fails to identify unknown viruses when there is yet not signature available, or the virus database is out of date, a signature-based IDS also fails to detect unknown attacks.

Since signatures are specifically designed to match known attacks, this type of IDS normally has a very low rate of false alarms (false positives). However, also due to that specific and static nature of signatures, signature-based IDS are not likely to detect even slight modifications of a known attack. This is a serious disadvantage because zero-day and polymorphic attacks (where a change in the attack payload does not affect the attack effectiveness) will go unnoticed by the IDS until the signature database is updated.

Signature-based IDS are ineffective against zero-day attacks because those attacks are unknown and as such, no signature that exactly matches them can exist in advance. Given that new attacks appear often, if not daily, and that a gap normally exists between the time a new attack is first detected and the time a signature is ready for it (first someone has to write the signature, and then IDS administrators have add it to their signature database),

[2] `www.snort.org`
[3] `www.prelude-ids.org`

this means that systems remain exposed to the attack for the entire duration of that period of time, giving attackers a window of opportunity to gain control of the system. Due to the dynamic and impromptu nature of web traffic, this limitation severely impairs the usability of signature-based IDS for the detection of attacks against web servers and applications.

An example of a signature-based system is ModSecurity[4], a web application firewall (WAF), which runs as an Apache web-server module that attempts to provide generic protection from unknown vulnerabilities often found in web applications (which are in most cases custom coded), but it still relies on rules (signatures).

2.2.2.2 Anomaly-based systems

To overcome the limitation inherent to signature-based systems being unable to detect previously unknown attacks, researchers have sought other ways to detect intrusions, namely by using anomaly-based methods.

An anomaly-based IDS works by building a statistical model of usage patterns describing the normal behavior of the monitored resource (which is nothing more than a set of characteristics observed during the its normal operation), and to this process we usually define it as the training phase. Then, after the statistical model for the normal and expected behavior is created, the system uses a similarity metric to compare new input requests with the model, and generates alerts for those deviating significantly, considering them anomalous. Basically, attacks are detected because they produce a statistically different, i.e. anomalous, behavior than what was observed when creating the model.

Anomaly detection assumes that intrusions are highly correlated to abnormal behavior exhibited either by a user or an application. The basic idea is to baseline normal behavior of the object being monitored and then flag behaviors that are significantly different from the baseline as abnormalities, or possible intrusions [18].

The main advantage of an anomaly-based system is its ability to detect previously unknown (or modifications of well-known) attacks as soon as they occur. Since anomaly-based systems are able to detect novel attacks that bypass signature-based systems, that information can later be used to develop new rules for signature-based systems. Therefore, using a combination of both IDS approaches, signature and anomaly-based, ensures a wider coverage when it comes to detecting attacks, as well as a means to keep signature databases up to date.

[4] `www.modsecurity.org`

Unfortunately, most of the IDS used nowadays are still signature-based, and few anomaly-based IDS have been deployed in production environments, mainly due to the fact that signature-based IDS are easier to implement, simpler to configure and less effort in needed to maintain them.

The high rates of false alarms associated with anomaly-based IDS have always impaired its acceptance and popularity, namely for cases where the monitored resource is not stationary and changes frequently like in the case of web applications. If the statistical models are too strict and the system is unable to generalize and account for variations, then it is likely that many false alarms are generated. Nevertheless, several products that employ anomaly detection mechanisms already exist.

Examples of anomaly-based IDS related to the web are some Web Application Firewalls. These products use anomaly-based techniques to detect web-based attacks. Artofdefence Hyperguard[5] and F5 BIG-IP Application Security Manager[6] are two existing commercial products that employ such techniques.

Table 2.3 shows a comparison between some of the advantages and disadvantage of signature versus anomaly-based intrusion detection.

Detection model	Advantages	Disadvantages
Signature	Low false positive rate	Cannot detect new, previously unknown attacks
	Training is not necessary	Requires continuous updates of signature databases
	Alerts are thoroughly typified and classified	Difficult to tune for custom resources
Anomaly	Can detect new, previously unknown attacks	Prone to raise false positives
	Self-learning	Alters are not classified or typified
		Requires initial training

Table 2.3: Advantages and disadvantages signature vs. anomaly-based systems.

[5] `http://www.artofdefence.com/`

[6] `http://www.f5.com/products/big-ip/application-security-manager.html`

2.3 The Role and Requirements of an IDS

McHugh et al. [19] presented a concise paper describing the role intrusion detection technology should play in an overall security architecture, and stressing the importance of Intrusion Detection Systems.

Ideally, an IDS should be able to detect, report and prevent a wide range of security events including intrusion and penetration attempts, execution of unauthorized privileged programs, unauthorized network connections and other relevant security issues. An ideal IDS should be able to perform its actions in real time, resist denial of service attacks, detect known and unknown intrusion methods, and generate no false positives. There is still no IDS that can conform to all these characteristics, but a few can optimize some of these parameters quite closely to ideal. Also, by using a mixture of approaches between signature and anomaly-based IDS, more of these characteristics can be achieved.

According to Crosbie and Spafford [20], an IDS should address the following issues, regardless of what mechanism it is based on:

- It has to run continually without human supervision. The system has to be reliable enough to allow it to run in the background of the system being observed. However, it should not be a "black box". That is, its internal workings should be examinable from outside.
- It has to be fault tolerant in the sense that it has to survive a system crash and not have its knowledge-base rebuilt at restart.
- On a similar note to above, it has to resist subversion. The system can monitor itself to ensure that it has not been subverted.
- It has to impose minimal overhead on the system. A system that slows a computer to a crawl will simply not be used.
- It has to observe deviations from normal behavior.
- It has to be easily tailored to the system in question. Every system has a different usage pattern, and the defense mechanism should adapt easily to these patterns.
- It has to cope with changing system behavior over time as new applications are being added. The system profile will change over time, and the IDS has to be able to adapt.
- Finally, it has to be difficult to fool.

2.4 World Wide Web and HTTP Protocol

The World Wide Web, abbreviated as WWW and commonly known as the Web, is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages that may contain text, images, videos, and other multimedia and navigate between them by using hyperlinks. Using concepts from earlier hypertext systems, Berners-Lee and Cailliau [21] wrote a proposal in March 1989 at CERN in Geneva to use "HyperText to link and access information of various kinds as a web of nodes in which the user can browse at will".

The Hypertext Transfer Protocol (HTTP) is a stateless networking protocol described in the RFC 2616 Internet Standard [22] which has been in use by the World-Wide Web global information initiative since 1990.

Like many network protocols, HTTP uses the client-server model where an HTTP client opens a connection and sends a request message to an HTTP server, which then returns a response message, usually containing the resource that was requested. After delivering the response, the server closes the connection (hence its stateless nature, i.e. not maintaining any connection information between transactions). Clients can be either browsers (e.g. Mozilla or IE) or web robots (e.g. Google bot).

```
GET /example.html HTTP/1.1
Accept: */*
Accept-charset: ISO-8859-1,utf-8
Accept-encoding: gzip,deflate
Accept-language: en-us,en;q=0.5
Connection: keep-alive
Host: example.org
Referer: http://example.org/index.html
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv
    :1.9.2.10) Gecko/20100914 Firefox/3.6.10
```

Figure 2.2: Example HTTP request from a Mozilla browser.

```
GET / HTTP/1.1
Host: example.org
User-Agent: Googlebot/2.1 (+http://www.google.com/bot.html)
Content-Type: text/html
Connection: close
```

Figure 2.3: Example HTTP request from Google Bot.

The requested resource can either be a path or a filename, which has to be interpreted by the web server. Some paths are meant to be interpreted by programs, like for example Common Gateway Interface (CGI) scripts described in RFC 3875 [23], or PHP. The use of server side interpreted scripts allowed websites to evolve from early and very static HTML only contents, into dynamic contents which nowadays provide users with a completely different and richer browsing experience of web sites. Typically, the path to the script is part of the requested resource path, and the parameters follow a ***?*** in the path, with the form *key*=*value*, separated by a ***&*** when there is more than one parameter to be passed to the script. Figure 2.4 shows an example where the resource is */example.php*, and there are two parameters, *user* with a value of *johndoe*, and *id* with a value of 1.

```
GET /example.php?user=johndoe&id=1 HTTP/1.1
Host: example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
Cookie: PHPSESSID=7dd1d5d1471fa8be2fea8f163cce3257
```

Figure 2.4: Example HTTP request calling a PHP script and passing parameters.

2.4.1 HTTP Request Structure

An HTTP request is a collection of text lines (separated by a CRLF) sent to a web server which includes a request line, request header fields and the body of the request.

Request Line has three parts, separated by spaces, and specifies: the method name which must be applied, the local path of the requested resource, and the version of the protocol being used. The method is the first word that appears in an HTTP request, the majority of HTTP requests being of the GET type, but others exist, such as POST or HEAD. Following the method, comes the resource path (URI) that usually represents a file, a directory in the file system, or a combination of both. The last part, indicates the version of the protocol used by the client (generally HTTP/1.0 or 1.1) .

A typical request line is shown in Figure 2.5:

```
GET /path/to/file/index.html HTTP/1.1
```

Figure 2.5: Example request line.

Following the initial request line in a HTTP request, we have the request header fields, which provide information about the request. The header field lines are in the usual text header format, which is: one line per header, of the form "Header-Name: value", ending with CRLF. HTTP 1.0 [24] defines 16 headers, though none are required. HTTP 1.1 [22] defines 46 headers, and one *(Host:)* is required in requests.

Request Headers is a collection of optional lines allowing additional information about the request and/or the client to be given (browser, operating system, etc.). Each of these lines is composed of a name describing the header type, followed by a colon (:) and the value of the header.

Figure 2.2 shows an example of the structure an HTTP request with several header lines.

Finally, we have the request body, which basically contains the data bytes transmitted immediately after the headers.

Request Body is a collection of optional lines which must be separated from preceding lines by an empty line and for example allowing data to be sent by a POST command during the sending of data to the server using a HTML form.

In our experiments, we only took into account requests with the method GET, thus we did not delve into request bodies. Note, however, that it is straightforward to include POST requests and to parse the body of a request to perform the same analysis that we present in Chapter 4. We simply did not do it, mainly due to the large volumes of data we were working with, but this is planned for future work.

2.5 Web Vulnerabilities

The Open Web Application Security Project (OWASP) has tracked and studied the trends of web vulnerabilities throughout the years, and periodically publishes their findings on this subject.

Table 2.4 shows the OWASP Top 10 web vulnerabilities for the year 2010 [25], ranked by the risk they represent to organizations. The risk ranking methodology used by OWASP encompasses and accounts for several factors, like threat agents, attacks vectors, security weakness, technical and business impacts.

Position	Vulnerability
1	Injection
2	Cross site Scripting (XSS)
3	Broken Authentication and Session Management
4	Insecure Direct Object References
5	Cross Site Request Forgery (CSRF)
6	Security Misconfiguration
7	Insecure Cryptographic Storage
8	Failure to Restrict URL Access
9	Insufficient Transport Layer Protection
10	Unvalidated Redirects and Forwards

Table 2.4: OWASP Top 10 Application Security Risks - 2010.

We now briefly explain each of the OWASP Top 10 vulnerabilities as described in [25]:

1-Injection Injection flaws (e.g. SQL) occur when untrusted data is sent to an interpreter as part of a query, tricking it to execute unintended commands or to access unauthorized data.

2-Cross Site Scripting (XSS) XSS flaws occur whenever an application sends untrusted to a web browser without proper validation and escaping. This type of flaws allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites (such as phishing or malware).

3-Broken Authentication and Session Management Application functions related to authentication and session management are often not implemented correctly, which allows attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.

4-Insecure Direct Object References When a direct reference to an internal implementation object, such as a file, directory, or database key is exposed, is done without access control checks or other protection mechanisms, attackers can manipulate these references to access unauthorized data.

5-Cross Site Request Forgery (CSRF) A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

6-Security Misconfiguration Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application.

7-Insecure Cryptographic Storage Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

8-Failure to Restrict URL Access Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.

9-Insufficient Transport Layer Protection Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.

10-Unvalidated Redirects and Forwards Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

2.6 Attacks against Web Servers and Applications

Attacks against web servers or applications can be highly diversified, both in terms of the particularity they target as well as the appearance they can assume. Typically, most attacks seen to date have focused on the resource path and only a few have targeted other parts of the request. The diversity of attacks against web servers or applications is high, which illustrates the amount of bugs that programmers introduce into code.

Alvarez and Petrovic [26] provided a taxonomy for web attacks using the notion of attack life cycle, i.e., the succession of steps followed by an attacker to carry out some malicious activity on the web server. Figure 2.6 (source [26]) illustrates that sequence of steps:

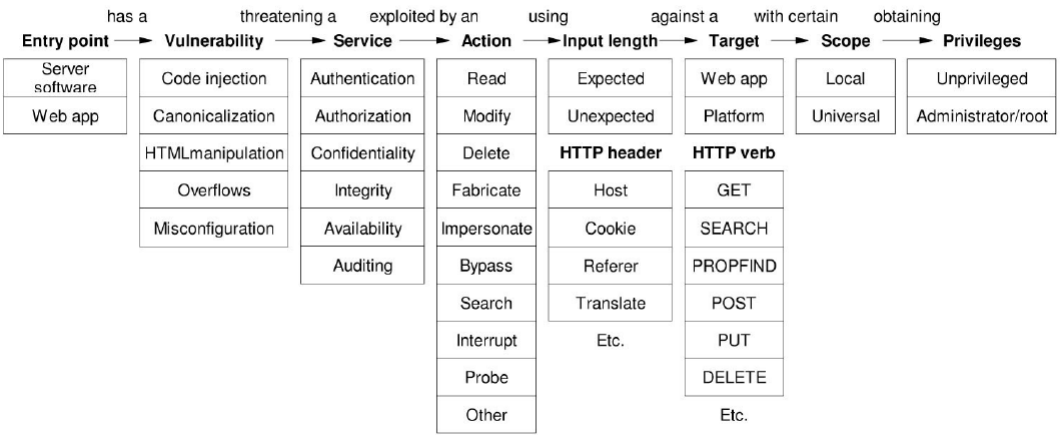

Figure 2.6: Taxonomy of web attacks.

1. Entry point: where the attack gets through.
2. Vulnerability: a weakness in a system allowing unauthorized action.
3. Service (under threat): security service threatened by the attack.
4. Action: actual attack against the Web server exploiting the vulnerability.
5. Length: the length of the arguments passed to the HTTP request.
6. HTTP element**:** verbs and headers needed to perform the attack.
7. Target: the aim of the attack.
8. Scope: impact of the attack on the Web server.
9. Privileges: privileges obtained by the attacker after the successful completion of the attack.

Given this sequence of steps that are typically followed to perform a web attack, we now briefly describe some of the most frequent classes of attacks against web applications.

2.6.1 SQL Injection

SQL injection is a class of attacks where unsanitized user input is able to change the structure of an SQL query so that when it is executed it has an unintended effect on the database [27]. This type of attack is made possible because SQL queries are typically assembled by

performing a series of string concatenations of static strings and variables. Should those variables, used in the creation of the query, be under the control of the user, he or she might be able to change the meaning of the query in an undesirable way. Considering an example of a web-based application that lets the user list all his registered credit cards of a given type (taken from [27]), the pseudo code could be as shown in Figure 2.7.

```
uname = getAuthenticatedUser()
cctype = getUserInput()
result = sql("SELECT nb FROM creditcards WHERE user='" + uname +
   "' AND type='" + cctype +"';")
print(result)
```

Figure 2.7: Example pseudo code illustrating vulnerable to SQL Injection.

If the user Bob did a search for all his VISA cards the following query would be executed:

SELECT nb FROM creditcards WHERE user = 'Bob' AND type = 'VISA';

This example code contains an SQL injection vulnerability, as Bob could manipulate the structure of the SQL query to view all the credit cards belonging to user Alice, simply by asking for a list of cards of type *' OR user ='Alice*. The attack query would be as follows:

*SELECT nb FROM creditcards WHERE user = 'Bob' AND type = '' **OR user = 'Alice';***

This query would return a list of Alice's credit cards to the attacker, as the user input is not being sanitized. A correct implementation of the application shown above should not have allowed data supplied by the user to change the structure of the query. User-supplied parts of SQL queries should never be interpreted as SQL keywords, table names, field names or operators by the SQL server. The remaining parts of the SQL query, which we will refer to as constants, consist of quoted strings and numbers. Prior to using user-supplied strings as constants, one must ensure that all quotes are escaped before inserting them into the SQL query. Similarly, user-supplied numbers also have to be checked to verify they are numbers and not strings. In the example above, the SQL injection attack is possible because the string *cctype* is not properly escaped before being inserted into the query.

2.6.2 Cross Site Scripting - XSS

Cross site scripting (XSS) occurs when a web application gathers malicious data from a user, usually in the form of a hyperlink which contains malicious content within. The user

will most likely click on this link from another website, instant message, or simply by reading a forum post or email message.

Three different types of Cross Site Scripting attacks exist, namely Reflected XSS (or non-persistent), Stored XSS (or persistent) and DOM based XSS (Document Object Model).

A XSS attack exploits trust relationships between web servers and web browsers by injecting a script (often JavaScript which is embedded in HTML code) into a server that is not under the control of the attacker. Generally, a Cross Site Scripting attack (we use the stored XSS attack here to exemplify) works as follows:

First, the attacker uploads HTML code containing JavaScript code to a web service (e.g. a web server forum) as shown in Figure. Next the malicious script will be executed in the victims' browsers and since the script originates from the web server, it is run with the same privileges as legitimate scripts originating from the server. Assuming the victim has a trust relationship with the domain hosting the web server (which typically does), the malicious script can access sensitive data associated with that domain.

```
<a
href="http://www.legitsite.com/forum/error.aspx?err=
<script>document.location.replace(
'http://www.hackersite.com/HackerPage.aspx?Cookie=' +
document.cookie); </script>">Click here to view
recent posts</a>
```

Figure 2.8: Example XSS attack code.

This type of attacks is normally used to steal login credentials or other personal information from users.

2.6.3 Remote File Inclusion - Shellcode Injection

Remote File Inclusion (RFI) is an attack technique used to exploit dynamic file include mechanisms in web applications. When web applications take user input (URL, parameter value, etc.) and pass them into file include commands, the web application might be tricked into including remote files with malicious code. RFI can be used for running malicious code on the server which can lead to serious system compromise, as well as for running malicious code on clients by manipulating the content of the responses (for example by embedding malicious javascript code to steal client session cookies) sent back to the clients by the server.

This type of attacks is dangerous as it allows the execution of remote code in the web server such as shellcode injection attacks, which typically exploit Buffer Overflow vulnerabilities (see [28] for a detailed explanation).

2.7 Issues with Anomaly Detection of Web Attacks

Many of the existing IDS solutions, like Snort [17] for example, have rules to detect HTTP attacks. However, those rules are very limited mainly because HTTP is a difficult and challenging protocol for an IDS to handle. Those challenges include:

- the stateless nature of HTTP
- the nonstationary nature of web servers
- HTTP requests can vary in length
- training data for anomaly detection without attacks is hard to obtain
- there are ”harmless” attacks which are part of the normal Internet web traffic, thus an IDS should tolerate them without producing alarms, but yet, still be able to detect novel attacks
- since HTTP is stateless, the IDS cannot rely on sequence relationship between requests, and in fact, most existing attacks are only one request long

Due to the existence of botnets that can be used to perform these attacks, an attacker and a legitimate user might be using the same computer. Thus, a web-server anomaly detection method must be able to analyze isolated transaction requests and determine whether or not they represent an attack on the server.

Apart from the stateless nature of HTTP, web site content changes often. Newspaper sites, change their news and headlines every day. Sites which sell products on the Internet, add new products and remove obsolete ones all the time. Users with personal blogs, add new entries often.

In order to be successful, any IDS must be able to cope with this dynamic and non-stationary environment.

2.8 Anomaly Detection for Web Applications

Several anomaly web-based detection techniques have been proposed in the last decade. All those techniques are based on HTTP, which means the modeling of normal activity is constructed either by observing decoded network frames up to the HTTP layer, or by functioning as reverse HTTP proxies.

Anomaly-based detectors, which use these techniques to protect web applications, were first suggested in [29], where the authors describe a system that uses Bayesian parameter estimation to analyze web access logs and detect anomalous session in web applications. Their technique assumed that malicious activity expressed itself in the parameters found on HTTP requests, which turned out to be the basic idea for the research that would follow in these matters.

In [30] and [31], a number of different anomaly detection techniques to detect attacks against web servers and web-based applications is proposed. These techniques focused on the analysis of parameters in HTTP requests and basically consisted of a combination of different detection models, namely attribute length, attribute character distribution, structural inference, token finder, attribute presence or absence, and attribute order. A significant part this dissertation relies on the models proposed by Kruegel and Vigna [30].

Wang and Stolfo [32] proposed an anomalous payload-based network intrusion system that uses the Mahalanobis distance as a way to detect anomalous requests in data sets with multiple attributes, scaling each variable based on its standard deviation and covariance and taking into account how the measured attributes change in relation to each other.

The approach by Robertson et al. [33], suggests using heuristics to infer the class of web-based attacks. It attempts to address the limitations of anomaly-based intrusion detection systems by using both generalization and characterization techniques. By using generalization, a more abstract description of an anomaly can be created which enables one to group similar attacks. As for characterization, it is used to infer the class of attack that is associated with a group of anomalies.

Wang et al. [34], propose a content anomaly detector based in n-gram analysis, which uses bloom filters and offers resistance to mimicry and polymorphic attacks.

As for Ingham et al. [35], it is shown how to construct a system using a DFA induction algorithm combined with automata reduction heuristics that can minimize false-positive risk by avoiding overgeneralization. It describes a learning algorithm capable of handling arbitrary-length non-stationary data.

In [36], a comparison of anomaly detection techniques for HTTP is presented, whereas in [37], a methodology for designing accurate anomaly detection systems is presented and shown how it can be applied to the problem of detecting malicious web requests.

Next, in [38], Ingham presents a thorough study and compares existent anomaly detection methods for HTTP intrusion detection, which was the most influential reference for our work. In his research, he suggests two new grammar based anomaly detection models, DFA and n-grams, describes a framework for testing anomaly detection algorithms and presents his findings using data from four different web sites.

In [39], Bolzoni et al. propose an anomaly-based network intrusion detection system which is later compared with other approaches in [40]. In [41], an anomaly-based web intrusion detection system is presented. Next, in [42], the authors suggest a system to reduce false positives raised by their anomaly-based systems. All this research is presented in detail in [43].

Vigna et al. [44] propose a technique to increase the detection rate of web-based intrusion detection systems through a combined use of a web request anomaly detector and a SQL query anomaly detector. The also attempt to reduce false positives by suggesting an anomaly-driven reverse proxy that provide differentiated access to a web site based on the anomaly score associated with web requests.

Criscione et al. [45], suggest an anomaly detector which models HTTP responses and Document Object Model (DOM) to enhance detection capabilities against SQL injection and XSS attacks.

More recently, Song et al. [46], proposed a machine learning based statistical tool for defense against web-layer code-injection attacks. This approach utilizes a mixture of Markov chains to model legitimate payloads at the HTTP layer and derive the corresponding training algorithm.

Maggi et al. [47] address the problem of web application concept drift in the context of anomaly detection, whereas Robertson et al. [48] show that scarcity of training data is a problem that can be overcome in web-based anomaly detection systems by using global knowledge bases of well-trained, stable profiles.

This dissertation uses several of the anomaly-based intrusion detection techniques previously proposed in [30, 32, 36, 38]. However, it aims to present an IDS for a real reasonably complex web application, so not only we implemented several of this techniques, but studied the problem of obtaining good training data and had to deal with several practical issues not considered in these previous works.

Chapter 3

Anomaly Detection of Web-based Attacks

The main goal of our research was to evaluate the usage of anomaly-based IDS in a production environment hosting a web application of great dimensions. In this chapter, we start by discussing the concept of generalization and its importance in anomaly detection and how anomaly detection models should be created. Next, we propose an algorithm that can be used to sanitize a dataset of requests obtained from real-world traffic of an application (hence likely to contain attacks) in order to remove anomalous requests contained within, with the goal of producing a dataset, free from attacks, which can safely be used to train anomaly detection systems. Moreover, we explain how similarity metrics are used to detect anomalies. We then describe nine different algorithms suggested in previous literature to detect anomalies in web traffic and discuss their virtues and limitations. Furthermore, we present the anomaly-based intrusion detection system, devised to encompass all those algorithms, as well as the generalization heuristics that we utilized to reduce the false alarms and improve its accuracy.

3.1 Generalization

The behavior models for intrusion detection are generally built from knowledge of the protected resource. Thus, the quality of a model is determined by the knowledge as well as the technique used to build it. In reality, it is hard to guarantee the completeness of knowledge under most scenarios as we might not know the future behaviors of a computing resource. To account for the incompleteness of normal behaviors of computing resource, model gen-

eralization is used to infer more behaviors besides the known and observed behaviors ([10], [49], [50], [51]).

In principle, model generalization can improve the detection rate of an IDS but may also degrade its detection performance. Therefore, the relation between model generalization and detection performance is critical for intrusion detection. In general, it is useful to generalize the normal behavior model so that more normal behaviors can be identified as such. However, model generalization may lead to model inaccuracy, and it cannot fully solve the incompleteness problem under most scenarios where it infers part but not all of the unknown behaviors. In Section 3.7 we explain how we addressed generalization, specifically how we employed generalization heuristics with the goal of improving the overall accuracy of the IDS.

3.2 Model Generation

Having a good dataset during the training phase is a crucial aspect for the accuracy and effectiveness of the system. A model used by an anomaly-based system should reflect the behavior of the system under normal conditions, that is, without the presence of attacks. Otherwise, during the detection phase, the system might fail to recognize an attack and classify it as normal.

Due to this fact, the system should be trained with a clean (i.e. without attacks) dataset. However, in practice such a dataset is difficult to obtain, as a network capture is bound to contain attack attempts and the more traffic it captures, the higher is the chance that it will include such attempts. This raises an issue, because its know that clean training datasets are required for anomaly based systems, and yet, there is no easy way to obtain those. The way to deal with this is by relying on the expertise of security administrators to analyze a large amount of that. Since the model should be regularly updated to cope and adapt to changes in web applications, this process can turn out to be a very intensive and time consuming task, likely bound to contain (human) errors.

Thus, manual inspection can be complemented with automatic means such as running the dataset though a signature-based system like Snort, which can pre-process the training data and detect some of the well-known attacks like scans, old exploits, etc. Regardless, signature-based systems will not be able detect all the attacks in the training data.

Regarding the duration of the training phase, two different factors have to be accounted for. First, it should last long enough in order to allow the system to construct a trustworthy

model, as opposed to having a very short training phase which could lead to having a very small model that does not capture the essence of normal behavior for the application or web site in question. During the detection phase, this would most likely result in having many legitimate requests wrongly flagged as anomalous (false positives). However, having a longer training phase is bound to induce a great deal of noise into the model. On the other hand, web applications are nowadays very dynamic and change regularly, so this might suggest that every time there is a software change in the application which translates into a noticeable change to the normal input of the application, then the model would have to be rebuilt and the system retrained. The larger the training set, the harder it is to maintain such an IDS.

As such, we now outline some desired properties for model generation:

- The quality of the model is fundamental to achieve a high detection rate and low false positives and false negatives.
- To avoid false positives, the model should contain all the possible non-malicious inputs
- To avoid false negatives, the model should not contain any of the possible attacks.
- The model should be simple to build, the shorter the training phase required building a trustworthy model, the better.

3.3 Algorithm to Sanitize a Dataset

The objective of this algorithm we are proposing is to define a set of requests for training T without anomalous traffic (*i.e.* without attacks). Typically, obtaining a data for training without attacks is hard and in Sections 1.1 and 3.2 we described the relevance of having such data for training anomaly-based intrusion detection systems.

Our algorithm is called *Sanitize-Dataset*, as its purpose it to transform (sanitize) a given dataset of requests in order to remove those that are anomalous. Starting with a set of requests collected from network traffic captures or server logs representing the usage of the service, the idea is to divide this data in slots corresponding to specific time periods. Then, each of these slot sets is run through the sanitization algorithm in order to remove attacks from the set, until the number of detected anomalies drops below a certain threshold. Defining the threshold depends on the environment and data in question, but this algorithm

can also easily be adapted to perform the sanitization using a fixed number of iterations (simply by changing the *until* condition).

Before presenting the pseudo code for the algorithm, we define the input data, formulate the hypothesis and list the variables.

Data A set of requests obtained from usage of the service, divided in message slots corresponding to a specific time period (e.g. 1 day), $d_0, d_1, ..., d_{n-1}$where d_0 is the first slot, and d_{n-1} the last.

Hypothesis There are no attacks distributed throughout the period in which the training traffic is collected. This means that given a certain attack A it does not appear in all the slots d_i.

Variables Gi is the set of requests manually verified and given as good (clean); Bi is the set of requests manually verified and given as bad, that is, with attacks; Ai is the number of anomalous requests detected in the last iteration

Algorithm 3.1 Obtaining a dataset without attacks

INITIALIZATION

$\forall\, i \in \{0, .., n-1\}\,, G_i = B_i = \{\}\,,\ A_i = \infty$

SANITIZE-DATASET(n)

```
1  i ← 0
2  repeat
3        TRAIN using A(i mod n) ∖ B(i mod n)
4        DETECT over A(i + 1 mod n) ∖ G(i + 1 mod n)
5        for all requests r detected as anomalous
6        if manual analysis shows that r = attack
7          then B(i + 1 mod n) ← r
8        else G(i + 1 mod n) ← r
9        i ← i + 1
10   until ∑ A_i < threshold
11 return T ← (G_i − A_i − B_i)
```

Prior to starting the algorithm, we initialize G_i and B_i and A_i as described above. Then the algorithm works as follows:

- In line 1, the i variable is initialized to 0.

- In line 2, the loop begins and runs until the number of detected anomalies (sum of the Ai) drops below a certain threshold in line 10.
- In line 3, the anomaly detection system is executed, in training mode, using the set of requests from $A(i \bmod n) \setminus B(i \bmod n)$.
- In line 4, the anomaly detection system is executed, in detection mode, over the set of requests from $A(i+1 \bmod n) \setminus G(i+1 \bmod n)$.
- In lines 5 to 8, for every request *r* which the anomaly detection system considered anomalous, manual verification is performed to confirm if it is really an attack. In case it is an attack, *r* gets added into the set $B(i+1 \bmod n)$. Otherwise, in case it is not an attack, *r* gets added into the set $G(i+1 \bmod n)$.
- In line 9, the *i* variable is incremented
- Finally, in line 11, the filtered set of requests for training T without anomalous traffic (*i.e.* without attacks) is returned.

3.4 Similarity Metric

An anomaly-based IDS learns and builds a model of normal behavior derived from observations in a set of training data. Then, in order to evaluate whether a certain request is anomalous or not, the system compares the similarity of the request to the model build from the training data.

Most of the existing anomaly-based systems referred in previous literature are based on statistical models and the similarity is calculated based on a function, usually a distance function whose output is compared to a pre-defined threshold value. The choice of such function depends on the model that is being used and the number of objects taken into account. For example the Length of the contents of a parameter inside an HTTP request, can be defined using Chebyshev inequality as seen in Kruegel and Vigna [30]. The Mahalanobis distance in Wang and Stolfo [32] considers multiple correlated mean and standard deviation values at once.

Setting the value of the threshold is a very important aspect that influences the accuracy of the system. Setting a low threshold, means having a high number of alarms, and therefore a low false negative rate, but also a high false positive rate. On the other hand, setting a high

threshold value, means having a low number of alarms, and a low number of false positives at the expense of a high number of false negatives.

As such, determining the correct value for the threshold is difficult and depends on the system being monitored and its environment, as well as the quality of the data used for training.

3.5 Detection Models

In this section, we briefly describe nine models that have been proposed for anomaly detection of web attacks, notably by Kruegel and Vigna [30], Ingham [38] and Wang and Stolfo [32], which were included in the IDS described in Section 3.6 and later evaluated in Chapter 4.

3.5.1 Length

Kruegel and Vigna [30] noted that the length of an attribute can, in many cases, be used to detect anomalous requests. Since parameters are usually either fixed-size tokens or short strings obtained from user input, the length of parameter values should not vary much between requests for the same web application. However, when malicious input is passed to the web application inside parameters, the length of the values is likely to be different from that of normal requests.

This model aims at approximating the parameter lengths and detecting instances that significantly deviate from the observed normal behavior.

During the training phase, the mean μ and the variance σ^2 of attribute length strings are measured. As for the detection phase, using Chebyshev's inequality, the system calculates the probability p that an attribute would have the observed length l by:

$$p = \frac{\sigma^2}{(l-\mu)^2}$$

Trained on clean data (without attacks), this measure is likely to be accurate in detecting cross-site scripting (XSS) and buffer overflow attacks, since the requests related to those attacks are typically much longer than normal requests, but will however miss attacks with length similar to normal requests. Since this model does not distinguish between parts of a request, if attacks are present in the training data, this measure unlikely to be as useful.

Moreover, there are some types of attacks (such as Apache chunked transfer error or variants of Nimda[1]) which are short enough to pass as normal, or if they are too short, padding can be used to increase their length in order to pass as normal. Consequently, imposing a minimum length will not stop every attack, and additionally, since this model basically accepts strings, some of which that might not even be legal in HTTP, an attacker has some flexibility in crafting his attack.

This model employs generalization by classifying strings with length close the mean μ as normal. Considering for example the number of valid sentences with n words in the English language, should this model be applied to tokens instead, it would over generalize. Consequently, this model alone is unlikely to be very helpful, but it can be useful in combination with other models, considering normal requests have a strict upper bound on their length.

In our testing, we evaluated the Length model both as standalone and as part of the Combination of models (Section 3.5.7).

3.5.2 Character Distribution

Two different approaches have been proposed to compare the character distribution of new requests to the distribution of those in the training data.

Kruegel and Vigna [30] used a character distribution model of the 256 ASCII characters by grouping them into 6 possible segments: 0, 1-3, 4-6, 7-11, 12-15, and 16-255, and then used a χ^2 test to calculate the anomaly score of new requests. Wang and Stolfo [32] used a character distribution metric to detect anomalous payload on network packets, by using Mahalanobis distance during the detection phase to calculate the similarity of new data against a pre-computed profile.

Character distributions generalize by accepting any string having a distribution of characters matching the learned distribution, more precisely, they allow similar instead of identical, character distributions. When trained on filtered data, these measures are likely to detect cross-site scripting (XSS) and buffer-overflow attacks as they often have a distinctive character distribution. Since these measures are unable to discern between parts of a request, they are unlikely to perform well when trained on unfiltered data containing attacks. However, the flexibility of the HTTP protocol narrows the effectiveness of these methods since attackers can use padding to produce character distributions close to normal, particularly due to the multitude of standardized existing ways to encode data. Unless web

[1]These attacks are present in the dataset described in Chapter4 which we used to test the IDS.

canonicalization is performed on HTTP requests before validation (or detection in case of an anomaly-based IDS), that is, reducing the request to its simplest or standard form, attacks can pose as normal to character distribution models using the mentioned techniques.

3.5.2.1 Mahalanobis Distance

Mahalanobis distance is a standard distance metric used to compare two statistical distributions, which provides a useful way to measure the similarity between the (unknown) new payload sample and the previously computed model. Wang and Stolfo [32] computed the distance between the byte distributions of the newly observed payload against the profile from the model computed for the corresponding length range. The higher the distance score, the more likely the payload is abnormal.

The Mahalanobis distance d is calculated as follows:

$$d^2(x, y)^T = (x - \overline{y})\, C^{-1}(x - \overline{y})$$

Where x is the feature vector of the new observation, and $\overline{y}$ is the averaged feature vector computed from the training examples. $C^{-1}$is the inverse of the covariance matrix, with $C_{ij} = Cov(y_i, y_j)$. Then, to further speed up the computation, they derived the simplified Mahalanobis distance, where the variance is replaced by the standard deviation. Here n is 256 for the ASCII character set:

$$d(x, y) = \sum_{i=0}^{n-1} \frac{|x - \overline{y}|}{\overline{\sigma_i}} < \infty$$

However, it is possible the standard deviation equals zero, so they introduced a smoothing factor α:

$$d(x, y) = \sum_{i=0}^{n-1} \frac{|x - \overline{y}|}{\overline{\sigma_i} + \alpha}$$

As for the smoothing factor α, since Wang and Stolfo did not specify how to define it, a value of 0.001 was used in our testing (but it can be changed since it is a runtime parameter).

To compute the incremental version of the Mahalanobis distance, the mean and the standard deviation of each ASCII character seen for each new sample observed must be computed. In order to implement the Mahalanobis distance, the update technique described by Knuth [52] was used. Since the standard deviation is the square root of the variance, the variance computation can be rewritten using the expected values E as follows:

$$Var(X) = E[X - E(X^2)] = E(X^2) - [E(X)]^2$$

The actual implementation for this metric is described by Ingham [38]. Given an instance, relative character frequencies ($f(c)$ for character c) of a request are calculated. Using

Knuth's method, the mean and squared mean for each character seen from the training requests (two 256- element arrays: $\overline{x_c}$ and $\overline{\sigma_c}$) are recorded. The distance is then calculated as follows:

$$d = \sum_{c=0}^{255} \frac{f(c) - \overline{x_c}}{\overline{\sigma_c} + \alpha}$$

Next, since d is a finite map, it is mapped into the interval [0, 1]:

$$s = \begin{cases} 1 & d \in [0, 1] \\ \frac{1}{ln(d+e-1)} & d > 1 \end{cases}$$

Where e is the base of the natural logarithms. The mapping for values of $d \leq 1$ makes sense, because the larger the distance, the more abnormal the data item. Experiments shown that requests from the training dataset have a distance $d > 1$, so the few requests with a small distance are clearly close to the learned distribution.

In our testing, this measure was applied to the entire HTTP request.

3.5.2.2 Chi-Square of Idealized Character Distribution

The approach by Kruegel and Vigna [30] is based on the observation that attributes have a regular structure, are mostly human-readable, and almost always contain only printable characters. Therefore, this method attempts to model normal behavior of a query parameter by looking at its character distribution.

Typically, the characters in those attributes are mainly letters and numbers, along with a few special characters. As in English text, those characters are not uniformly distributed, but do actually occur with different frequencies. Despite not following the same distribution as English text, there are similarities between the character frequencies of query parameters.

They used a measure of relative character frequency by producing a sorted list of character frequencies f_c containing the relative frequency of the character c, and noted that relative frequencies decrease slowly for non-attack requests, but have a much steeper decline for buffer overflows, and no decline for random data.

Their character distribution induced from the training data was denominated as the idealized character distribution (ICD), where $\sum_{i=0}^{255} ICD(i) = 1.0$. The ICD is sorted so that the most common frequency is ICD(0) and the least common is ICD(255). The calculation of the ICD is done during training as the average over the character distributions of the requests in the training data.

For testing, they grouped the ICD (the expected distribution) and the distribution of the test request (observed distribution) into six[2] bins as follows:

bin	0	1	2	3	4	5
i	0	1-3	4-5	7-11	12-15	16-255

Then, once grouped they used a variation of the Pearson χ^2-test to ascertain whether the character distribution of CGI parameter values is similar to that of the training data as follows:

$$\chi^2 = \sum_{i=0}^{i<6} \frac{(O_i - E_i)^2}{E_i}$$

Here E_i is bin i for the ICD, and O_i is bin i for the observed distribution. Then, χ^2 is compared to values from a table and the corresponding probability is the returned.

The derived value p is used as the return value for this model. When the probability that the sample is drawn from the idealized character distribution increases, p increases as well.

This model was tested both as part of the Combination (Section 3.5.7) and as a standalone model.

3.5.3 Ordering of Parameters

Another method from Kruegel and Vigna [30] was based in the order of CGI parameters in requests.

Legitimate requests to server-side programs, are typically generated by client-side scripts or HTML forms as opposed to having users hand-typing the input parameters into the URIs themselves. Therefore, since program logic is generally sequential, these legitimate requests usually contain the parameters in the same fixed order, and even if some of those parameters are omitted for a particular request, the relative order of parameters is preserved. However, should requests be hand-crafted by a human, their order can be totally arbitrary, which might suggest a plausible attack.

This method determines whether a certain ordering of parameters is consistent with that observed during training. During the training phase, a directed graph, representing the order of the attribute values, is constructed. As for the detection phase, an attempt to traverse the graph is made, and in case of success this method returns an anomaly score of 1, otherwise it returns 0.

[2] According to Tan and Maxion [53], the number six apparently has some particular relevance in the field of anomaly detection

This model is able to generalize by allowing orderings not originally seen in the training data. Should the training data consist of *abcd* and *dbac*, the resulting directed graph would be as shown in Figure 3.1 (source [38]).

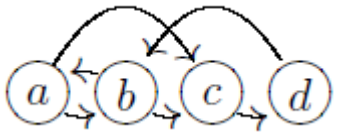

Figure 3.1: Example directed graph induced by abcd and dbac.

This method has the potential to distinguish regions of the request, but was only tested as part of the Combination model (Section 3.5.7), as it was unclear how to map an entire HTTP request into an order relationship that differed from the n-gram methods.

3.5.4 Presence or Absence of Parameters

As explained above for the Ordering model, URI parameters are normally provided by a script or HTML form, and not hand-filled by a human. Thus, as noted by Kruegel and Vigna [30], a regularity in the number, name, and order of the parameters exists for legitimate requests.

During the training phase, this approach models normal behavior by learning the parameters present for a given program path. For the detection phase, the return value is 1 if the same parameters were observed during training and 0 otherwise.

The generalization of this method is limited, since observing various requests to a certain resource with certain individual parameters, means that any combination of those parameters would also be considered normal. Unless the Ordering of parameters model is used to identify the combination had not been seen before, this method alone is not particularly useful.

Due to the reason explained in the previous paragraph, this method was only tested as part of the Combination model (Section 3.5.7*)*.

3.5.5 Token Finder

The idea behind this model is to determine whether the values of a certain query attribute are selected from a finite set of possible alternatives (enumeration), or are in fact random.

Since web applications usually require specific values for certain attributes (like flags or indexes), when an adversary attempts to use these attributes to pass illegal values, the attack can be detected.

During the training phase, to decide if an argument a is an enumeration, the statistic correlation p between the values of two functions f and g for increasing numbers 1, . . . , i of occurrences of a is calculated:

$$f(x) = x$$

$$g(x) = \begin{cases} g(x-1)+1 & \text{if the value is new} \\ g(x-1)-1 & \text{if the value was seen before} \\ 0 & \text{if } x = 0 \end{cases}$$

Note that f is strictly increasing, whereas g increases only when new values appear in the training data. The correlation parameter ρ is calculated from f and g with their respective variances and covariance as shown below:

$$\rho = \frac{Covar(f,g)}{\sqrt{Var(f) * Var(g)}}$$

If $\rho < 0$, then f and g are negatively correlated, and an enumeration is assumed. Otherwise, if $\rho > 0$, the values of a are presumed to be random. Since Kruegel and Vigna did not specify the case where $\rho = 0$, we assumed random for $\rho \geq 0$. When an enumeration is assumed during training, the values are stored for use in the detection phase.

During the detection phase, if the training data indicates the values were random, then the similarity is 1. If the training indicates the data is enumerated, and the value is contained in the learned set, the similarity is also 1. On the other hand, if the value is not in the learned set, the similarity returned is 0.

As for generalization, if a parameter value is considered enumerated no generalization is performed, whereas if the value is random, everything is accepted. What this means is that generalization with this model depends on the specificity of the server side program.

This method was only used as part of the Combination model, since it was unclear how to apply it to a complete HTTP request.

3.5.6 Markov Model

Normally, most web exploits can immediately be perceived in query attributes with unusually long values or parameters containing several repetitions of non-printable characters.

Such attacks can be easily detected by the Order (Section 3.5.3) and Character Distribution (Sections 3.5.2.1, 3.5.2.2) models explained above, but there are cases where adversaries are able to craft the attacks in a more concealed manner, making them seem more regular (e.g. replacing non-printable characters with groups of printable characters). For this type of situations, a more detailed model of the query attribute that contains evidence of the attack is necessary.

The Markov model can be created by analyzing the parameter's structure, that is, the regular grammar that describes all of its normal, legitimate values. The basic approach is to generalize the grammar as long as it seems to be reasonable and stop before too much information is lost. To specify this idea of reasonable generalization, Markov Models are used.

A Markov model is a non-deterministic finite automaton (NFA) which has probabilities associated to the transitions. The fundamental idea with this model is to build such an automaton that matches exactly the training data and gets compressed through a series of state merging operations, thus achieving generalization.

The output of the Markov model consists of all paths from its start state to its terminal state, and the probability of a given word/string w (sequence of output symbols), is calculated as the sum of the probabilities of all distinct paths through the automaton that produce w. As for the probability of a single path, it is calculated as the product of the probabilities of each of the taken transitions and it is used as the similarity measure for the detection phase.

A Markov Model represents the structure of the HTTP request through a directed graph. Kruegel and Vigna [30] used a Markov model as part of their system that detected attacks only in CGI program parameters, but due to the high diversity of HTTP requests, since the probability of any given request is small, they used it only to see whether or not the model was capable of generating the request in question. Since we applied this method using complete HTTP requests, the problem is even worse, as there are more places where normal diversity results in lower probabilities for any given HTTP request.

As stated above, generalization in the Markov model is achieved with the compression obtained through a series of state merging operations, which may allow combinations not seen in the training data. Nonetheless, generalization is limited since novel values will result in a probability of 0, which means that in a very dynamic environment, this model is likely to perform poorly unless it keeps being updated.

For the training phase, this method learned normal behavior based on tokens in order for it to be able to distinguish between different regions of a request, and enabling it to later detect attacks in distinct regions of an HTTP request.

The Markov model was tested both as a standalone and as part of the Combination of models (Section 3.5.7).

3.5.7 Combination

Kruegel and Vigna [30] proposed a system to detect web attacks in HTTP CGI requests. Therefore, they only took into account the initial line of an HTTP request, ignoring everything else such as the headers. Their system consisted of a combination of the following models (hence we call it the Combination model):

- Length for CGI parameter values.
- Character distribution for CGI parameter values.
- Markov model for learning the structure of parameter value strings.
- Enumerated or random parameter values for CGI parameter values.
- Presence or absence of parameters.
- Order of parameters.

In order to dynamically determine the normal threshold for each model, two passes over the training data were performed and the threshold was calculated to be 10% above the highest value obtained.

For the detection phase, each measure is equally weighted and the system returns a binary indication of whether the request is considered normal or not. The overall result of the combination for each request is the average of the six measures. If the combination was an AND operation instead of the average, then one single model could cause a request to be rejected, whereas if the combination was an OR operation, then acceptance by any model would result in acceptance by the combination.

The generalization of the Combination method is that of the various individual models, and since those generalize differently, using the average allows the combination to account for problems introduced by the over or undergeneralization of a particular model.

In order to evaluate the Combination of the six models in conformity to how it was originally proposed in [30], this method was tested using the initial line of HTTP requests as seen for example in Figure 2.5, and ignoring the remaining header data lines.

3.5.8 *N*-Grams

An *n*-gram [54] is subsequence of tokens, strings in this case, that we get by sliding a window of length n across text. For example, given the text *abcdef* and $n = 4$, the resulting 4-grams are: *abcd*, *bcde*, *cdef*. In order to build a saved *n*-gram set, it is necessary to break every string into n-grams and store every new gram into the set.

Since the length of the string from the training dataset can vary, if a string length is shorter than *n* in the process of transforming a string into *n*-grams, the entire string is treated as an *n*-gram and is then saved or tested. The parameter n controls the granularity of the model, and when it decreases to 1, the model loses the structural information of the grammar and is reduced to an alphabet of the grammar which accepts any input string that is composed of its characters (from the alphabet). Otherwise, if *n* is higher than the lengths of all input strings, the model will save trained strings only, accepting nothing more and thus under generalizing.

The detection phase for this model simply checks if the *n*-grams in question were observed during training, that is, if they are present in the set of *n*-grams learned from the training data.

It returns a similarity value s as follows:

$$s = \frac{\text{number of n-grams from the request also in the training data}}{\text{number of n-grams in the HTTP request}} \in [0, 1]$$

N-grams can encode the entire structure of an HTTP request and consequently distinguish between regions in a HTTP request. Using tokens, the structure of a request is modeled by encoding sequences of tokens representative of the input as a directed graph. The *n*-grams induce a directed graph and the generalization is similar to that described for the order of parameters model (Section 3.5.3).

During our experiments, it was applied to the complete HTTP requests and tested in five variants, with $n = 3, 4, 5, 6, 7$, using tokens.

3.6 Anomaly-based IDS

Our work consisted in devising an IDS comprised of a collection of the anomaly-based models described in the sections above, in order to evaluate and assess the accuracy for each of those models, which can be seen as a proof of concept for a fully fledged anomaly-based IDS solution.

The system implements nine anomaly-based models, namely, Length (Section 3.5.1), Mahalanobis distance (Section 3.5.2.1), χ^2 of idealized character distribution (Section 3.5.2.2), Ordering of parameters (Section 3.5.3), Presence of Absence of parameters (Section 3.5.4), Token-Finder (Section 3.5.5), Markov Model (Section 3.5.6), Combination (Section 3.5.7) and *N*-grams (Section 3.5.8). All the models are implemented in Perl using *IDS::Test*, an IDS test framework developed by Kenneth Ingham, available from CPAN[3]. The system is capable of running each model independently (as well as the combination of 6 models from Kruegel and Vigna [30]), under identical conditions, datasets and in the same environment.

To represent the HTTP request data, depending on the model, the IDS can use a string of characters or a stream of tokens, which can contain the associated values. By using tokens, the system is able to learn the structure of a request and its meaning.

For each of the anomaly detection models, the system operates in two different phases, training and detection. During the training phase, it determines the characteristics of requests and creates a statistical model describing normal behavior. The detection phase is where the new request instances are tested against the model in order to determine the likelihood of them being attacks. In the detection phase, the system tests a request and returns a similarity value, $s \in [0, 1]$, which represents the similarity of the request being tested to what is expected from the trained model. Values of $s = 0$ represent a novel request , that is, a completely new request that was not observed nor derived from the training data. When $s = 1$, it represents a request with no differences from the observations during training. Any values of s in between, represent intermediate similarity. For all the results presented in Chapter 4, and according to what was suggested in [38] to allow a better comparison, the required accuracy was a relatively generous true positive fraction of 0.852, which means that all results below this value were considered anomalous (those equal or above were considered normal).

For any of the models that originally returned a similarity value outside of that interval, i.e. $s \notin [0, 1]$, their similarity result was mapped into this range.

3.7 Generalization Heuristics

Several heuristics were employed in the detection system in order to increase the generalization for specific areas in the request. In order to use these heuristics, the IDS checks if the specific area values in the request that display a high degree of variability are well

[3]http://search.cpan.org/~ingham/IDS-Test-1.00/

formed, and then replaces them with an indication of whether or not those values met the standard.

After analyzing our datasets we assessed which areas of an HTTP request were plausible for applying generalization, and then decided to use the following heuristics:

IP Addresses and Hostnames Instead of learning every IP address and hostname in the Internet, the system will validate whether the form of the IP address or the hostname meets the Internet standard.

Hashes Hashes exist in HTTP request for example in the form of entity tags or sessions IDs in Cookies (e.g. Content-MD5 or PHPSESSID). Since the purpose of a hash is to be unique, it does not make sense for an anomaly-based IDS to learn every hash. Thus, the system just validates the form (character set and length) of the hash and returns if the hash is valid or not.

Dates Dates present a problem for an IDS attempting to learn the structure of an HTTP request, as they change every day. Thus, instead of learning every day, instead the system validates whether the date in question meets the RFC 2616 standard [22].

File Types Most requests to a web server are for files of a few types (e.g., HTML files, JPEG files, PHP scripts, etc). Rather than learning every file on the web site, and since our test data shown that many of those files are photos or pages ”dynamically” created by the server, the system attempts to identify the file type for the last part of the resource path based on the file name, and will return an identification, which in fact corresponds to an extension, and not the entire filename.

Filename Lengths This heuristic complements the previous one, File Types, and is applied when the file is of unknown type. Should this be the case, the system returns the length of the filename.

Lowercase Only In the HTTP RFC 2616 standard [22], it is stated that each header field consists of a name followed by a colon (“:”) and the field value, and that field names are case-insensitive. Due to this fact, and since different clients use different schemes for the same headers, the system needs to treat all the header field names as case insensitive as standardized. This heuristic increases generalization because it maps the header (both names and values) to lowercase, which reduces the amount of instances the system has to lean as normal.

Email user Length The ”From:” header field HTTP can include as value the email address of the user making the request, in the form of username@hostname. The hostname part of the address can be verified as explained for the IP addresses and hostnames heuristics, but for the username, short of sending an email, there is no simple way to validate it. Still, in order to prevent buffer overflow attacks in the username, its length can be measured for the system to learn it and that is what this heuristic does.

Try Alternates We observed that several HTTP header lines were causing the system to generate many false positives. Since these lines are not critical for the web server to identify a requested resource, the system tries to delete the following header lines: Referer, Cookie, Accept-Language, Accept-Charset, and Accept, one at a time, and if after deleting a line, the request passes the similarity test, then it is accepted and processed without the anomalous header lines. Should the IDS be implemented as a reverse proxy, deletion of these lines can protect the web server from attacks present in those parts of the HTTP request, but deleting the cookies (which can encode state, e.g., PHPSESSID) could eventually prevent a user from using browsing the web site correctly. Apart from the cookies, the worst that could happen was that client would receive the default version of a web page instead of one customized to its preferred language, character set or file format.

Chapter 4

Experimental Results and Analysis

This chapter presents the results of our experiments and an evaluation of the algorithms for anomaly detection of web attacks. We explain how we obtained and defined our datasets, and provide a brief description of how our IDS works. Then we depict the requirements in terms of space needed for each model, running time for creating the model (training) and using it to test for anomalies (detection). Finally, we analyze each of the models by discussing their results and by illustrating and comparing the detection accuracy between them.

4.1 Obtaining the Datasets

Our data sets were built from a series of network captures of HTTP requests delivered to a web server of a Portuguese ISP during the period of 14 weeks. To collect these requests, the web server traffic was port mirrored to a server where we captured the data with Tshark. Tshark is the terminal-based (non-GUI) counterpart of Wireshark [55], a network packet analyzer formerly know as Ethereal, which enables capturing of packet data from a live network, or reading packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. Its native capture file format is libpcap, which is also the format used by Tcpdump [56] and various other tools.

The web servers in question host an application which has more than 500000 requests on average per day. When we began this research, we did not use any particular filters with tshark. However, and due to the high volume of traffic, associated with the disk space limitations on our capture server, we soon realized that we needed to restrict what was

being captured, as the early traces averaged about 8 GB with bzip2 best compression (-9) per day, which meant around twice the size uncompressed, that is 16 GB of daily traffic.

4.1.1 Network Captures and Filtering

We started to apply capture filters to tshark in order to reduce the size of those traces. First we used only one filter, to capture traffic involving TCP port 80, which is the standard HTTP port where the web servers were listening on.

Initially, tshark was run like this:

tshark -s 65535 -i eth1 -n -q -t a -w cap_file -b filesize:100000 -b duration:21600 -f "tcp port 80"

Here the flag *-s 65535* represents the length of the packet that should be captured (also referred to as snaplength), that is the first 65535 bytes of each packet. The flag *-i eth1* indicates on which interface the capture should be done, in our case it was eth1 receiving the mirrored traffic. Next, the parameter *-n* is used to disable network object name resolution (such as hostname, TCP and UDP port names) in order not to delay the capture. As for the *-q* flag, we use it so that at the end of a trace, tshark displays a count of the packets captured, rather than displaying the continuous count of packets during runtime. The *-t a* sets the format of the packet timestamp in the trace file to absolute, that is, the actual time the packet was captured. The parameter *-w cap_file* indicates where should the files be written to, whereas, *-b filesize:100000* means that the size of the capture file should be at most 100MB and when that size is reached, it should start writing to a new file. The parameter *-b duration:21600* makes tshark switch to the next file after value 21600 seconds (6 hours) have elapsed, even if the current file is not completely filled up. Finally, the flag *-f "tcp port 80"* is a capture filter that records only traffic on tcp port 80.

But since we only wanted to capture the requests issued to the web server and not the responses returned to the clients, we applied one extra filter:

tshark -s 65535 -i eth1 -n -q -t a -w cap_file -b filesize:100000 -f "tcp port 80 and dst net x.x.x"

Here the filter dst net x.x.x included the network addresses of the web servers, so we were only recording the traffic destined to that network, which in reality meant only incoming requests (to the web servers). However, then we had to run tshark a second time over the capture files, and apply a read filter to extract only HTTP requests with the method GET, and generate new trace files. We achieved this as follows:

for i in $(ls cap_file); do tshark -r $i -T fields -R "http.request.method == "GET" " -w $i.cap; done*

Here used a loop to go over each of the previously captured files and read it through tshark with the *-r $i* flag, and then applied a read filter *-R "http.request.method == "GET" "* making tshark parse only GET http.request packets.

The fundamental reason for choosing only HTTP GET requests was that with this method, data is encoded (by a browser) into a URL, whereas in a HTTP POST request the data appears within a message body as explained in Section 2.4.1. Besides, we also observed the majority of the web requests in our captured traffic used the GET method.

Nonetheless, this required doing two passes over the capture files, and due to disk space limitations in our capture server, we later decided to start capturing straight from the network only HTTP requests with method GET. For this purpose, we needed to use a capture filter that would achieve the same as in the previous two steps. However, since tshark does not allow using the same syntax as we used in the read filters, we devised a new filter and ran tshark as follows:

tshark -s 65535 -i eth1 -n -q -t a -w cap_file -b filesize:100000 -f "port 80 and tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420"

What this filter does is to capture HTTP GET requests by looking for the bytes 'G', 'E', 'T', and ' ' (hex values 47, 45, 54, and 20) just after the TCP header. The "tcp[12:1] & 0xf0) >> 2", is used to figure out the TCP header length.

Finally, as we wanted to record traffic for the longest period possible, we made a decision to apply an extra filter in order to record only traffic destined to one of the three machines in total that were serving the web site, so we appended a *dst host x.x.x.1* capture filter where we set the IP address for one specific machine.

tshark -s 65535 -i eth1 -n -q -t a -w cap_file -b filesize:100000 -f "port 80 and tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420 and dst host x.x.x.1"

4.1.2 Snort and Application-layer Extraction

After having Tshark running as we explained, we began by building only Common Log Format (CLF [57]) logs containing the HTTP GET requests and the resource paths. However, given that we had other useful information in the capture files other than just the GET and resource path requests typically seen in CLF logs, we decided to use Snort [17] in order to process the capture files from Tshark so that we could extract the application-layer

data from the individual network packets. The main idea here was not using Snort as an IDS, but instead as a tool to extract the application-layer data from our network captures. Regardless, we also used Snort as one of our several filtering actions that will be explained further ahead.

By taking advantage of Snort's ability to decode the application layer of a packet, perform TCP stream reassembly and handle IP fragmentation, we were able to obtain data records which included the entire HTTP request sent by the clients to the server, allowing us to make use of the HTTP header lines to test for attacks not contained in the requested resource path. Snort is able to process data with approximately a 99% accuracy rate, as long as the machine running it is fast enough to keep up with network traffic, otherwise packets can get lost and the resulting stream becomes incomplete. We used Snort with preprocessors frag3 and stream5 active, and only in log mode for TCP port 80 as show in Figure 4.1:

```
# IP defragmentation
preprocessor frag3_global
preprocessor frag3_engine
# Stream reassembly
stream5_global: max_tcp 8192, track_tcp yes, track_udp no, track_icmp no
preprocessor stream5_tcp
# just inbound packets for the web server
log tcp !x.x.x.x/32 any -> any 80 (session: printable; sid:0;)
```

Figure 4.1: Configuration file for snort.conf

However, probably due to Snort's stream reassembly preprocessor, or due to incomplete packets during the capture with tshark, likely related to file rotation in the middle of a stream being captured, we detected several requests that were incomplete. Therefore, we had to develop some custom shell scripts with regular expressions to filter these incomplete requests from our datasets.

We decided to aggregate the output streams processed by Snort into weekly datasets. Since our capture files from tshark had timestamps, and given that Snort creates directories with the IP address for each logged stream, and then places logs of the streams there (e.g. /var/log/snort/1.2.3.4/SESSION:56789-80), we created a method to obtain a set of daily ASCII files, each containing all the corresponding streams including the complete HTTP requests as decoded by Snort. From then on, having daily files, it was just a matter of concatenating them to obtain weekly sets.

This is how we processed the files through snort (starting in the directory where the capture files were located, in the example below "~/caps/"):

for i in $(ls); do mkdir ~/log/snort/$i; snort -c ~/snort.conf -C -O -A none -r $i -l ~/log/snort/$i > /dev/null 2>&1; cd ~/log/snort/$i; cat */* >> ../$i.log; cd ..; rm -rf $i; cd ~/caps/; done

We used a loop to list the capture files and then created a directory with the name of each capture file, which contained the time when it was created [1]. Then we ran *snor*t with the flag *-c ~/snort.conf* indicating where the configuration file seen in Figure 4.1 was located (in this case our $HOME directory, or ~/). The flag *-C* means Snort should print only the character data from the packet payload whereas the *-O* is used to obfuscate the IP addresses when in ASCII packet dump mode. As for the flag *-A none*, it refers to the alert-mode which we set to none (recall we only use Snort to process the packets from tshark captures and extract the complete HTTP requests). The flag *-r* is used for reading tcpdump-formatted files, in our case the files created with tshark which were passed in the shell variable *$i* in for loop. Then, for each file processed we concatenated the contents of all the streams for all the IP addresses with the commands *cd ~/log/snort/$i; cat */* >> ../$i.log* and finally we removed the daily directory after it had been processed.

As stated above, the result of this process was a set daily ASCII files, which contained all the complete HTTP requests (including headers), which then we grouped into weekly sets.

4.1.3 Unfiltered and Filtered Datasets

We decided to produce two different datasets. The first one contained all the requests resulting from the captures done with tshark after being processed by Snort and parsed by our custom scripts to remove incomplete requests. Aside from the filters with tshark, the processing done by Snort and the parsing done with our custom scripts, we did not verify this dataset for attacks. We refer to it as the unfiltered dataset.

The second dataset, to which we refer as the filtered dataset, was obtained by selecting the five most accessed server side application components observed in the captured web server traffic and by applying read filters in tshark to match only the requests directed at those applications. This is an example of how we ran tshark to achieve it (the application names and extensions have been changed here just to exemplify):

tshark -r cap_file -R "(http.request.uri matches \"(?i)^/App1.php\") or (http.request.uri matches \"(?i)^/App2.cgi\") or (http.request.uri matches \"(?i)^/App3.asp\")" -w cap_file_filtered

[1] e.g. cap_file_00001_20100712185927

The filter here is the same for each application, that is, it filters only packets with HTTP requests whose *http.request.uri*, case insensitive (*?i*) starts with (^) the resource path name of the application (e.g. /App1.php). The flag *-w cap_file_filtered*, means that we created new capture files containing only the filtered packets corresponding to requests directed to the five most accessed server side application components.

Then we used Snort to process and parse these new filtered capture files created with tshark and to extract the full application-layer data from the HTTP requests. Then, we used our custom scripts to filter for any incomplete requests and once we had all the requests in this filtered format, we selected four different weeks of traffic, each one from a different month, and then applied the Algorithm presented in the previous chapter (Section 3.3), using the Combination of models (Section 3.5.7) to obtain a sanitized dataset that was free from attacks. Then, from this dataset we decided to select four weeks of traffic for training, one from each month since we believe this would better capture the normal usage of the web site, rather than selecting for example one consecutive month (since we performed captures during the summer, seasonal vacations and other tendencies could eventually be more noticeable).

Table 4.1 shows the number of HTTP GET requests present in each of the weeks of capture and parsed traffic, for the unfiltered and filtered datasets. The rows in emphasis represent the portion of data that was used to train the anomaly detection models. The remaining rows of the filtered column, represent the traffic considered normal and that was used for testing the models.

dataset	unfiltered size	filtered size
2010-07-12	*729058*	*42659*
2010-07-19	751170	38682
2010-07-26	805296	37017
2010-08-02	549256	41598
2010-08-09	630795	40020
2010-08-16	*555708*	*48533*
2010-08-23	583201	47086
2010-08-30	538670	42552
2010-09-06	615254	43922
2010-09-12	572264	53634
2010-09-20	*661150*	*53766*
2010-09-27	681352	38851
2010-10-04	608760	43145
2010-10-11	*657060*	*39607*

Table 4.1: The web server dataset sizes (in number of requests).

4.2 Attack Dataset

In order to test the detection capabilities of each model, we produced another dataset, consisting solely of HTTP attack requests, using the compilation from [36] available at

`http://www.i-pi.com/HTTP-attacks-JoCN-2006`.

This compilation consists of 63 attacks collected from different sources, namely BugTraq and the SecurityFocus archives `http://www.securityfocus.com/`; the Open Source Vulnerability Database `http://www.osvdb.org/`; the Packetstorm archives `http://packetstorm.widexs.nl/`; and Sourcebank `http://archive.devx.com/sourcebank/`.

It contains the following categories of attacks: buffer overflow; input validation error (other than buffer overflow); signed interpretation of unsigned value; and URL decoding error. The attacks targeted different web servers: Active Perl ISAPI; AltaVista Search Engine; AnalogX SimpleServer; Apache with and without mod php; CERN 3.0A; FrontPage Personal Web Server; Hughes Technologies Mini SQL; InetServ 3.0; Microsoft IIS; NCSA; Netscape FastTrack 2.01a; Nortel Contivity Extranet Switches; OmniHTTPd; and Plus-Mail. The target operating systems for the attacks include the following: AIX; Linux (many varieties); Mac OS X; Microsoft Windows; OpenBSD; SCO UnixWare; Solaris x86; Unix; VxWorks; and any x86 BSD variant.

4.3 Model Considerations

The training phase produces different files containing the state for each trained model. Follows an example of how the system runs the training phase for producing the 3-gram model:

gentest.pl HTTP Ngram –length=3 –generate –gensave=3gram.save
–normal_threshold=0.852 < training.files

Here *gentest.pl* is the name of the Perl program which implements the core functionality of the IDS, *HTTP* is the name of the protocol being used, *Ngram –length=3* means we are using the N-Gram model with $n = 3$. Then, *–generate* means we are running the system in training mode and thus we are generating a new model and *–gensave=3gram.save* defines the file where the model state will be saved. The flag *–normal_threshold=0.852* sets the

value below which results will be considered abnormal as explained in Section 3.6, whereas < *training.files* is the list of input files containing the requests that will be used for training the system and thus producing the model.

Note however that for the Combination of models from [30] (Section 3.5.7), since six different models are used, it runs slightly different as six saved model files have to be created to compose the overall model. As referred multiple times in the previous chapter, the Combination of models only looks at the initial request line in HTTP requests and ignores the headers. Apart from those details, everything else is works the same as with any other model.

Regarding the detection phase, it is also run once for each model, here is an example of how it works for the Length model:

gentest.pl HTTP Length –test –ids_state=Length.save –normal_threshold=0.852 –try_alternates < test.files > Length.test

Just like in the training phase, the same program and protocol are used. The flag *–test* indicates the system is to be run in detection mode, whereas *–ids_state=Length.save* loads the file with the model created in the training phase. The flag *–normal_threshold=0.852* indicates the threshold value of the normal requests, whereas *–try_alternates* turns on the generalization heuristic described in Section 3.7. The redirection < *test.files* is the list of files containing requests to be tested for anomalies and > *Length.test* is where the output indicating the results for each request will be written to.

Figure 4.2 shows a brief example of part of the output file from the detection phase using the Length model:

```
[...]
week2.log request 16: 1
week2.log request 17 alternate Without Cookie orig 0.590058689134289 now
    1
week2.log request 17: 1
[...]
```

Figure 4.2: Example output from detection phase.

The example output reads as follows. First, the request number 16 from the file week2.log resulted in a similarity of 1, which means that according to what was observed during training it is a normal request. Next, the 17th request from the week2.log file resulted in a similarity of 0.59, which is below the normal threshold we defined, hence it should be

considered anomalous. However, given that we were using the Try Alternates generalization heuristic, the same request was tested without the Cookie header and the similarity result was changed to 1. What this means is that the system was considering this request anomalous just because of the added length introduced by the Cookie header. This is a clear example of the importance of having generalization heuristics in place.

4.3.1 Space Requirements

One of the requirements for an IDS presented in Section 2.3, stated the imposed overhead in the system it monitors should be minimal. Therefore, one measure to ascertain that overhead is the amount of memory required for each method to construct its model, in other words, the disk space size occupied by the files containing the state of each model created after the training phase is completed.

Table 4.2 presents a comparison of the storage requirements for each algorithms in terms of disk space in bytes, when trained with the filtered and the unfiltered datasets. The n-grams are the models which require more space, with the *7*-grams requiring 27MB when trained on the filtered dataset, and 93MB when trained on the unfiltered dataset. As for the remaining algorithms, the Markov Model is the one that takes more space, 1.9MB with the filtered dataset and 6MB with the unfiltered dataset, while the other models all take little space.

Model	trained on filtered dataset	trained on unfiltered dataset
3-grams	4,306,864	13,277,971
4-grams	8,053,974	25,294,592
5-grams	13,344,542	42,723,133
6-grams	20,097,675	66,762,546
7-grams	28,293,212	96,863,131
Token	2164	2,609
Combination	41,004	220,444
χ^2ICD	3,327	3,350
Length	198	200
Mahalanobis distance	7,896	8,198
Markov Model	1,915,325	6,011,577
Order	61,124	144,317

Table 4.2: Size in bytes of each model.

4.3.2 Duration of Training Phase

A second measure we used to evaluate the models was the time taken by the training phase. Table 4.3 shows the time in seconds, taken by each of the methods to construct its model using the filtered and unfiltered datasets.

These results were obtained on the same server machine we used to perform our captures, an HP Proliant DL360 server, with two quad-cores Intel (R) Xeon(R) E5410 2.33GHz CPU, 16Gb of RAM, running Debian GNU/Linux 5.0.4 with Linux Kernel 2.6.26-2-amd64.

We can observe that using the filtered dataset effectively shortened the time spent in building the models, which was to be expected as the unfiltered dataset contained a much larger number of requests that the filtered one as seen in Table 4.1.

All the n-gram models took approximately the same time to build, whereas the Length model was the fastest and the Order the slowest. All the remaining training phases for the models took more or less the same time.

Model	with filtered dataset	with unfiltered dataset
3-grams	395.547	5518.076
4-grams	401.072	5517.658
5-grams	401.211	5811.334
6-grams	411.502	5717.185
7-grams	415.059	5747.996
Token	461.071	6305.021
Combination	426.673	5391.923
χ^2 ICD	416.014	5669.225
Length	365.045	5123.164
Mahalanobis distance	560.152	7965.284
Markov Model	419.877	5774.333
Order	664.763	8377.648

Table 4.3: Time in minutes taken to build each model.

4.3.3 Duration of Detection Phase

A third measure we used, was the time taken by the detection phase. Table 4.4 presents these results, which were obtained using the same machine as in the training phase.

This measure, is particularly important because it shows the performance of each model in respect to detecting anomalies.

For the *n*-gram models, the detection phase took at maximum around 18 minutes to complete, which is not bad considering four weeks of requests, adding up to 184565 complete HTTP requests (recall than when we say complete HTTP requests, we are not just referring to the method and request URI, but also to several header lines as well), were being analyzed, which meant parsing approximately 2 million lines of logged streams.

The Token model performed a little slower, around 26 minutes. The Combination model, when trained on the filtered dataset, performed well, around 22 minutes, but when trained on the unfiltered dataset, it took over 33 minutes to run the detection phase. As for the χ^2 ICD, the detection phase took around 19 minutes to complete with either dataset. The Length model was the fastest to run the detection phase with the filtered dataset in approximately 16 minutes, but that value rose to 20 minutes when the detection was done using the model trained on the unfiltered dataset. As for the Mahalanobis distance, it performed slowly, taking around 42 minutes to complete. The Markov model was the slowest to complete the detection phase. Using the model trained on the filtered dataset, it took over 1 hour and 37 minutes to run the detection phase, whereas that value rose to 4 hours and 32 minutes when the model trained on unfiltered dataset was used. This is absolutely unacceptable when you think about detecting attacks to your web servers and applications and need to act on it promptly. Finally, the Order model took approximately 27 minutes to run the detection phase.

Model	when trained on filtered dataset	when trained on unfiltered dataset
3-grams	1011.612	1022.045
4-grams	1045.231	1042.683
5-grams	1053.087	1057.952
6-grams	1069.335	1073.219
7-grams	1104.207	1098.394
Token	1556.975	1545.800
Combination	1320.763	2002.591
χ^2 ICD	1143.606	1045.060
Length	985.872	1230.509
Mahalanobis distance	2545.163	2540.387
Markov Model	5824.736	16369.686
Order	1645.202	1621.774

Table 4.4: Time in seconds taken to analyze the testing dataset.

4.4 Model Accuracy

In this section, we present the results of our testing and compare the accuracy between the different models. The accuracy of a model is the ratio of correctly classified requests to the total number of requests. Testing a model means running the IDS in detection mode (after it has been trained) and using as input a set of requests for it to analyze and detect those which are anomalous according to a pre-defined normal threshold.

Each of the following models was tested on sets of data described in Section 4.1. In the ROC curves in this chapter, the TPR and FPR are represented as the fraction of the attack dataset and/or the test dataset properly or improperly identified. To produce the ROC plots, first we run the IDS in detection mode using as input a set of data that contained only legitimate (normal) requests, and from there we obtained the FPR, which is the ratio of incorrectly classified normal requests (false alarms) to the total number of normal requests. Then another set of data, that we knew to contain only attack requests, was used as input for running the IDS in detection mode, and from there we obtained the TPR, which is the ratio of correctly classified attack requests to the total number of attack requests. These results, demonstrating which requests were properly or improperly identified according to each dataset, were then used to plot the ROC curves. Each line in the plots represents a different dataset or a different model and each point represents a different similarity threshold for distinguishing normal from abnormal.

4.4.1 Length

The results for the Length testing are presented in Figure 4.3. When trained with the filtered dataset, the detection accuracy for the Length model was surprisingly worse than when trained on the unfiltered dataset, but in either case, it was always below 80%. Regardless, we have observed that using the Length model alone, particularly with a full HTTP request containing headers as some limitations has there are many headers that can influence the final lengths of the request. We were able to improve the accuracy of the Length model with the generalization heuristics mentioned in Section 3.7. In relation to the better results observed with the unfiltered dataset, we believe this was due to the specificity of the web applications present in our datasets, which suggests that with this model the filtering techniques we employed to obtain the filtered dataset did not produce the expected outcome.

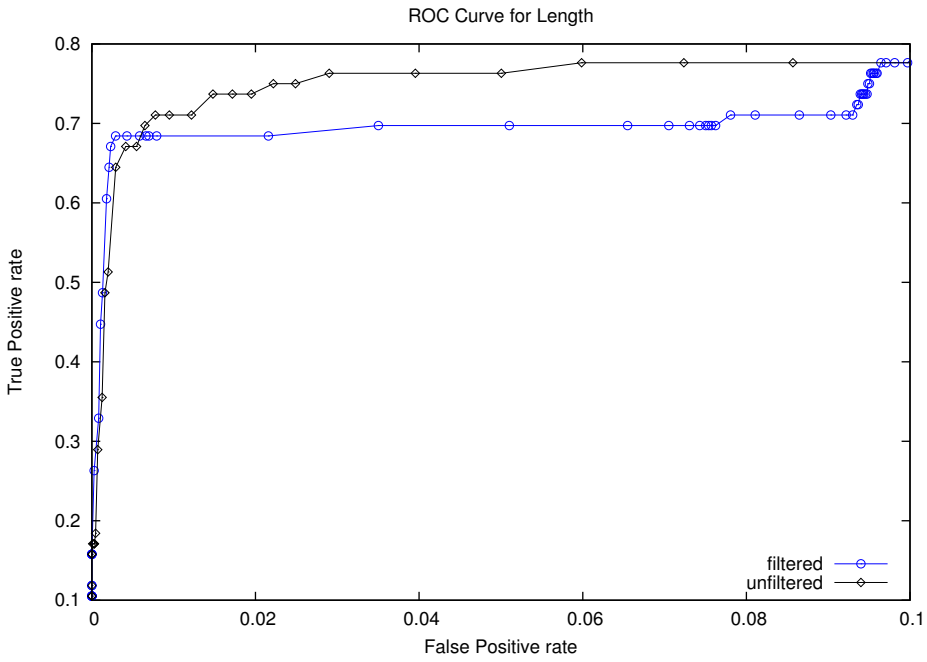

Figure 4.3: ROC curves illustrating the accuracy of the Length model on the filtered and unfiltered datasets.

4.4.2 Character Distribution

4.4.2.1 Mahalanobis Distance

Figure 4.4 shows the results of testing the Mahalanobis distance model. As it was to be expected, when trained on the filtered dataset, this model provided more accurate results than when trained on the unfiltered dataset, which was exactly what we aimed for when performing the filtering techniques described earlier. Nevertheless, the best this model could perform was around a 90% true positive rate while maintaining a false positive rate under 10%, which in our opinion is still not sufficient for a production environment.

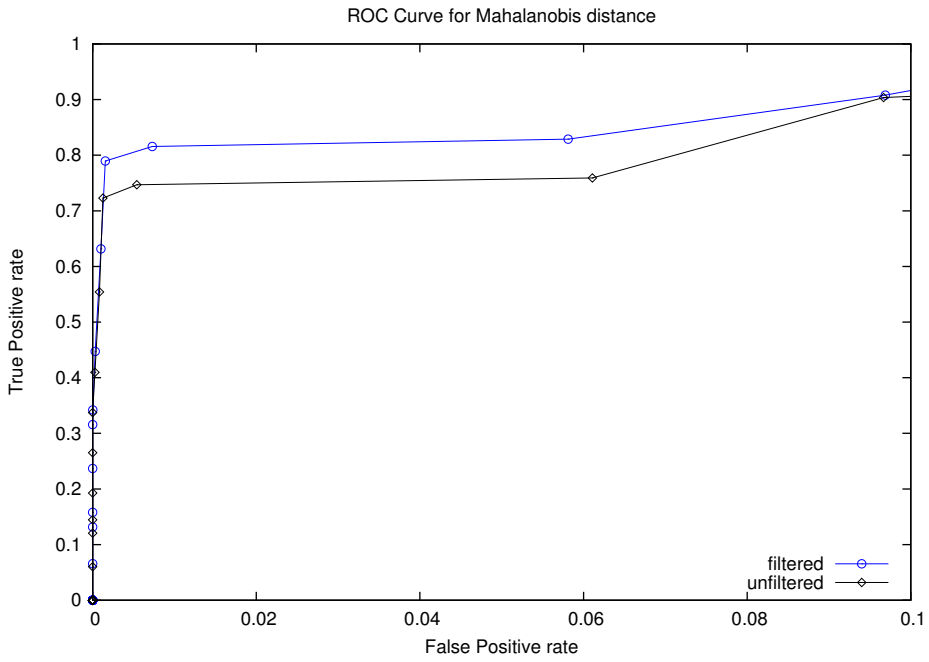

Figure 4.4: ROC curves illustrating the accuracy of the Mahalanobis distance model on the filtered and unfiltered datasets.

4.4.2.2 Chi-Square of Idealized Character Distribution

The results for the testing of the χ^2 of idealized character distribution model are presented in Figure 4.5. Trained on either dataset, filtered or unfiltered, this model performs badly, not even being able to achieve a true positive rate above 40%, which is completely inadequate for use in production servers. These poor results can be explained because the HTTP protocol is flexible enough to allow padding and as such, attacks can result in character distributions considered close enough to normal, mainly due to the countless ways of encoding data allowed by the standards. Moreover, by grouping characters into only six segments reveals some limitations in handling complete HTTP.

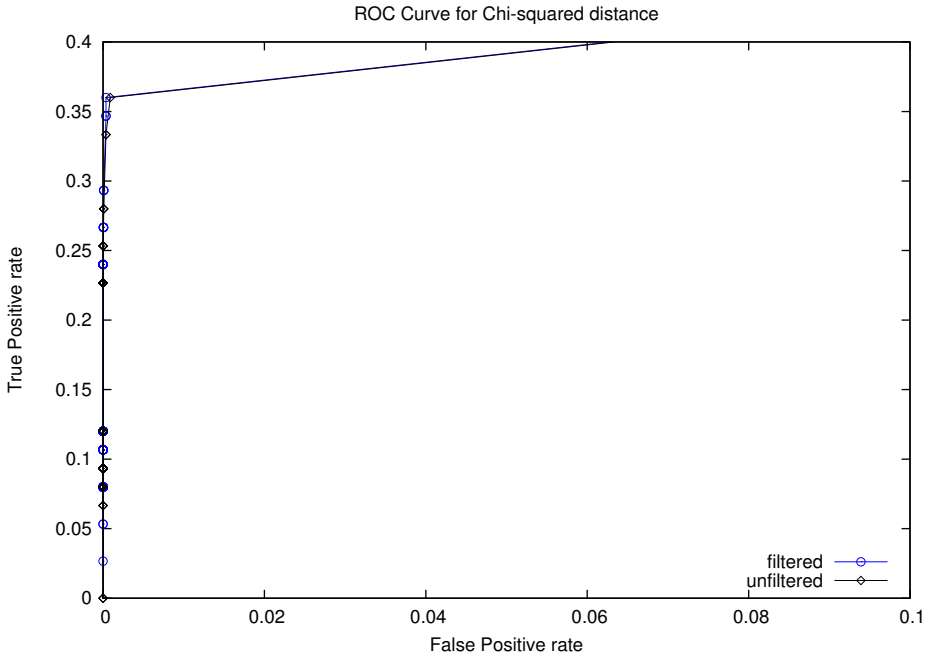

Figure 4.5: ROC curves illustrating the accuracy of the χ^2 of idealized character distribution model on the filtered and unfiltered datasets.

4.4.3 Markov Model

The similarity results from testing with the Markov Model, were very small and close to 0, which suggested the model identified everything (both normal and attack requests) as anomalous. Therefore, as suggested in [38], in order to better understand these results, Figure 4.6 was plotted by transforming the similarity value from the Markov Model m into a new similarity value s by:

$$s = \frac{1}{|\log_e(m)|}$$

In order to better see the data, the scale of the plot has also been changed, which means it is not comparable to the rest of the ROC Curve plots in this chapter. The log transformed Markov Model shows a 100% accuracy on filtered data, but with a false positive rate higher than 85%, which is totally unacceptable for production use. We did not test this model with the unfiltered dataset.

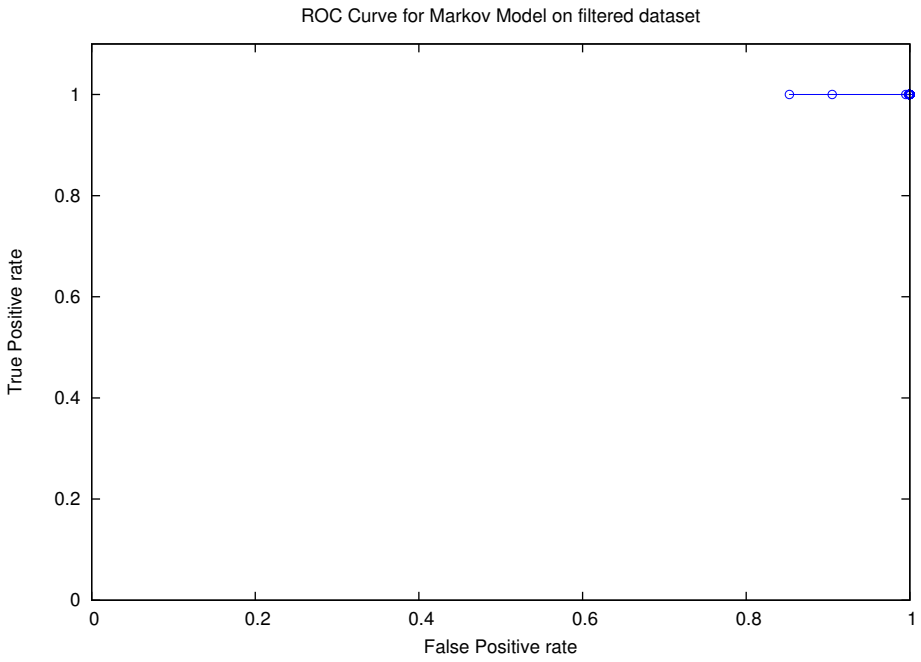

Figure 4.6: ROC curves illustrating the accuracy of the log transformed Markov Model on the filtered dataset.

4.4.4 Combination

Testing the combination of the six different models mentioned in Section 3.5.7, produced the results in Figure 4.7. As expected, in general the combination model is more accurate when trained on the filtered dataset. Regardless of the dataset used for training, filtered or unfiltered, it made no significant difference in terms of overall accuracy, which was always below 50%, which suggests the Combination model to be inadequate for use in production environments. These results were very surprising as we had expected it to perform better given the results presented in all the works from Kruegel et al. ([30, 31, 33, 44, 47]). The reasons why we obtained such poor results with this model can be explained by the particularities of the web applications in our captured traffic, but also due to the equal weights we used for each model and the average measure described in Section 3.5.7. Moreover, since the normal threshold for each of the individual models of the Combination is determined dynamically (as explained in Section 3.5.7) this can also contribute to the poor results that we experienced with this model. Possibly, with different variations of weights for the model we could have obtained similar results to those reported in [30], which demonstrates that combining several anomaly detection models requires careful consideration.

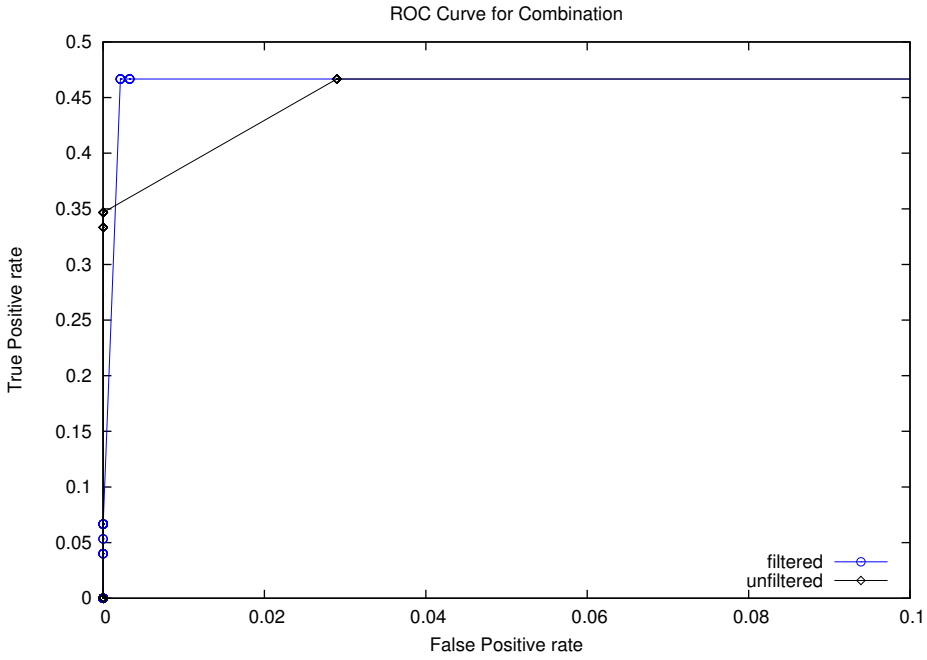

Figure 4.7: ROC curves illustrating the accuracy of the Combination on the filtered and unfiltered datasets.

4.4.5 *N*-Grams

Figures 4.8, 4.9, 4.10, 4.11, 4.12 show the results of testing the *n*-grams model for *n = 3, 4, 5, 6, 7* respectively. Note the range of the x-axis, i.e. the scale of the False Positive rates, has been changed from 10% to 1%, in order to better visualize the plots. This makes little difference in terms of comparison with the remaining models as all *n*-gram variations achieve a true positive rate of 100% before the false positive rate reaches 1%.

Against our expectations, for all the *n*-grams models, the detection was more accurate when the unfiltered dataset was used for training, rather than when the filtered dataset was used, despite the differences being almost insignificant to even mention. The *3*-grams model was also the most accurate model when it came to testing, but in general, for any of the five different *n*-grams models that were used, the detection rate was always very high and the false positive rate almost nonexistent. However, it was surprising that smaller the *n*, the better this model performed. The reason for this lies in the fact that most malicious parameters involve rarely seen characters rather than a different combination of usually seen characters, which can also explain why it drastically outperforms the Length model.

According to our experiments, *n*-gram modeling of web requests shown to be a very promising and accurate model to detect attacks, with a very high detection rate and very

low false positive rate. Clearly, this was the most appropriate model for detecting attacks with the real-world data we used to build our models and train our system.

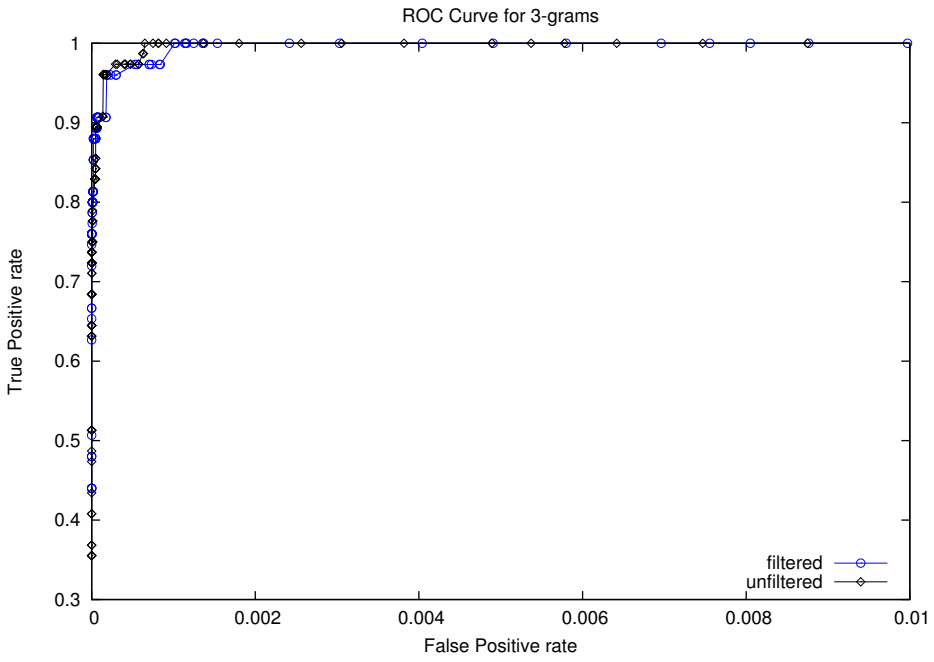

Figure 4.8: ROC curves illustrating the accuracy of *3*-grams on the filtered and unfiltered datasets.

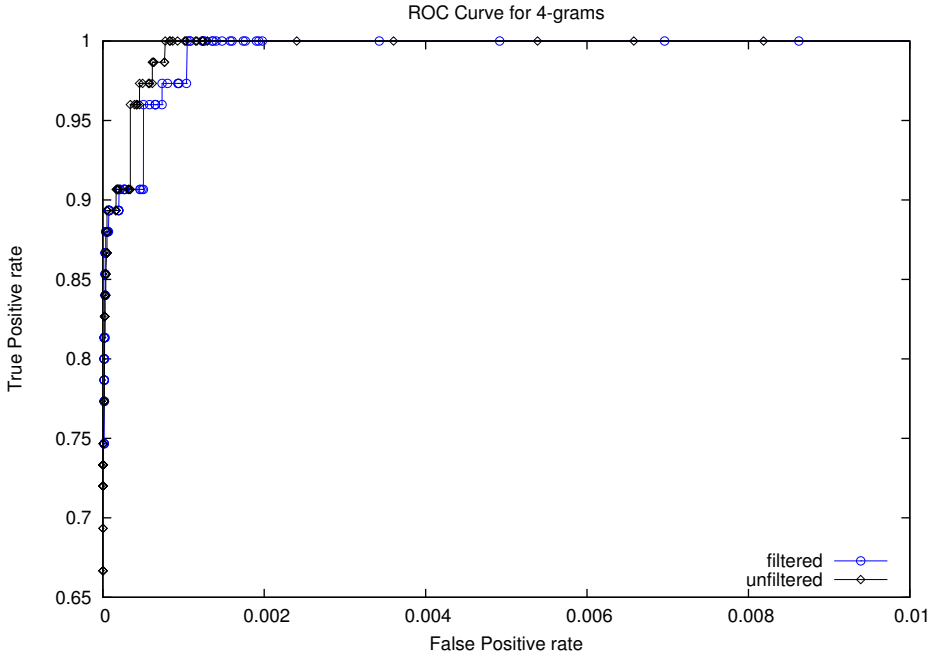

Figure 4.9: ROC curves illustrating the accuracy of *4*-grams on the filtered and unfiltered datasets.

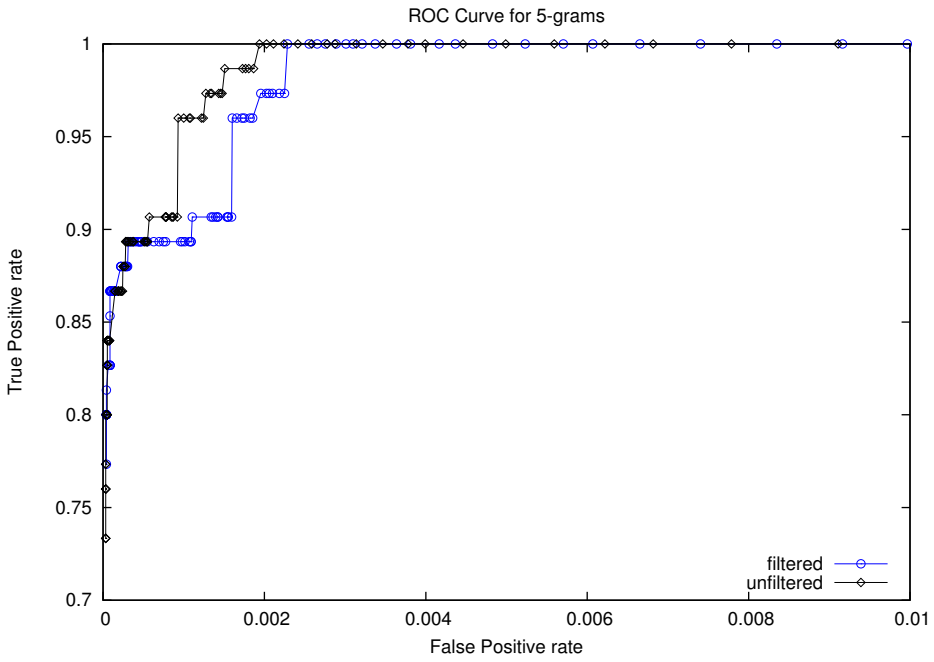

Figure 4.10: ROC curves illustrating the accuracy of *5*-grams on the filtered and unfiltered datasets.

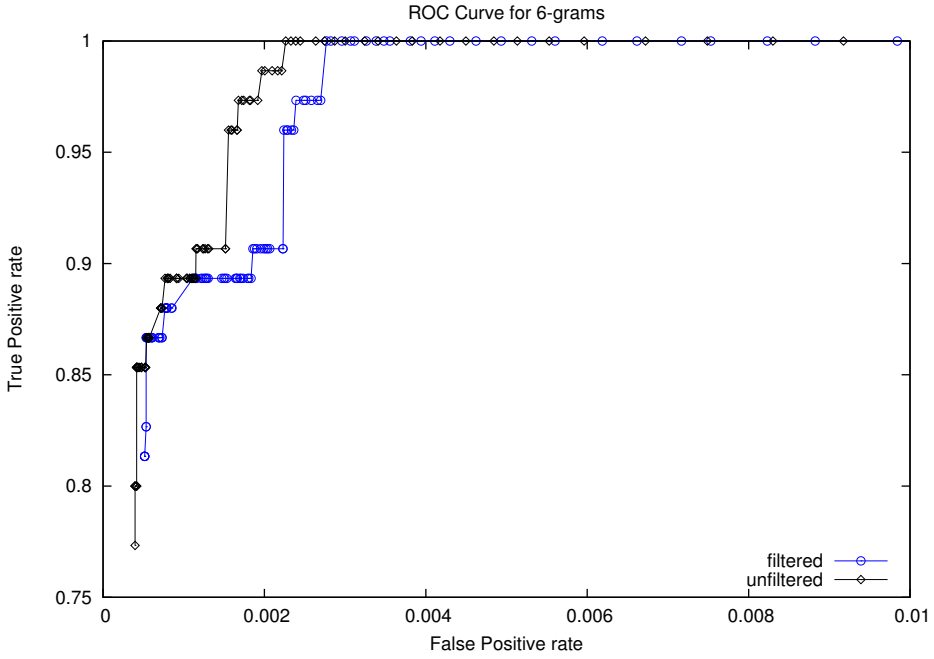

Figure 4.11: ROC curves illustrating the accuracy of *6*-grams on the filtered and unfiltered datasets.

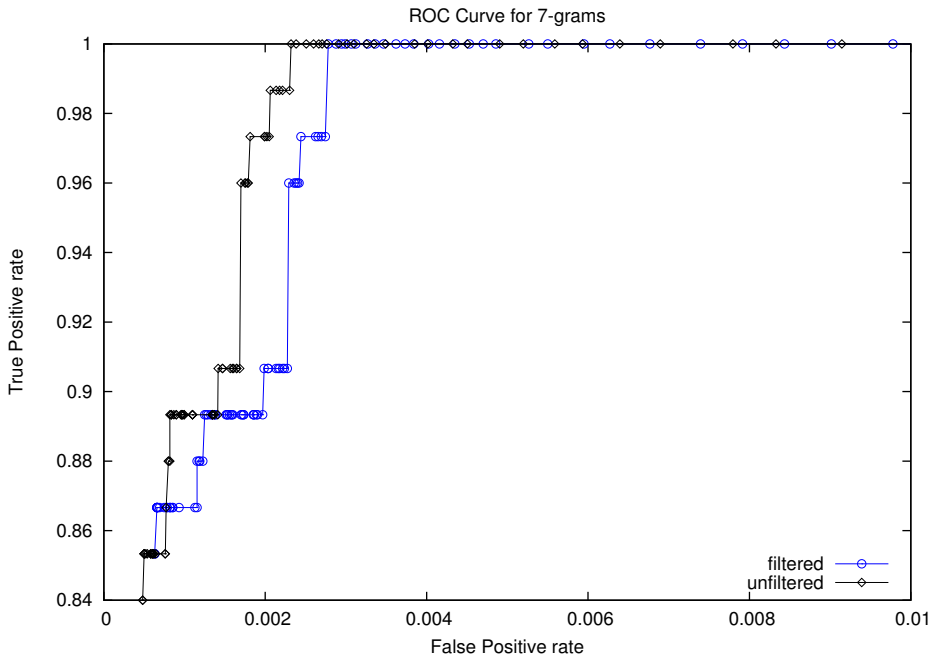

Figure 4.12: ROC curves illustrating the accuracy of *7*-grams on the filtered and unfiltered datasets.

Figures 4.13 and 4.14, present a comparison between the different *n*-grams testing, when trained with the filtered and unfiltered datasets respectively.

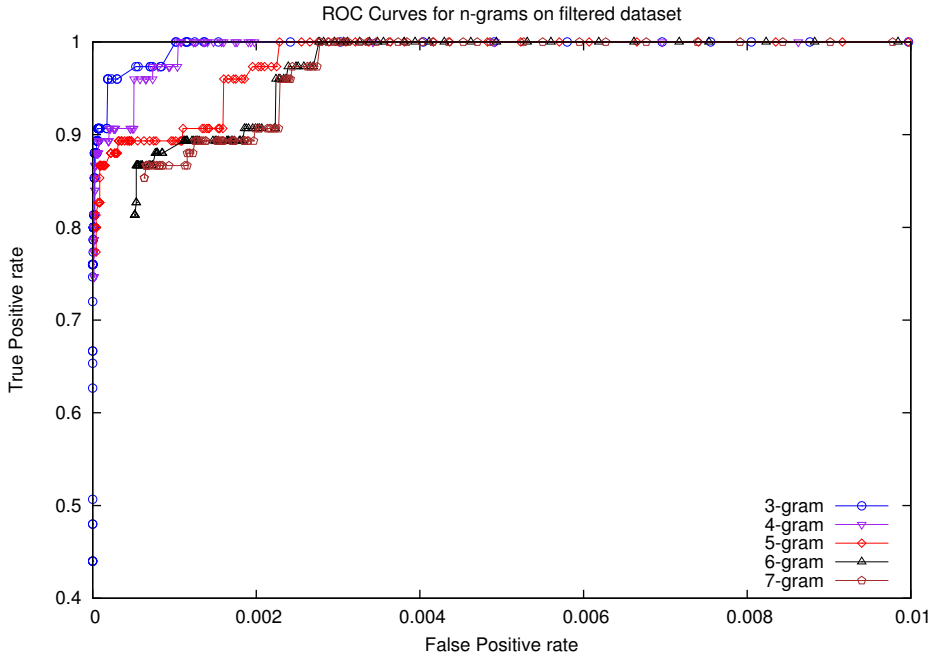

Figure 4.13: Comparison of *n*-grams ROC curves on the filtered dataset.

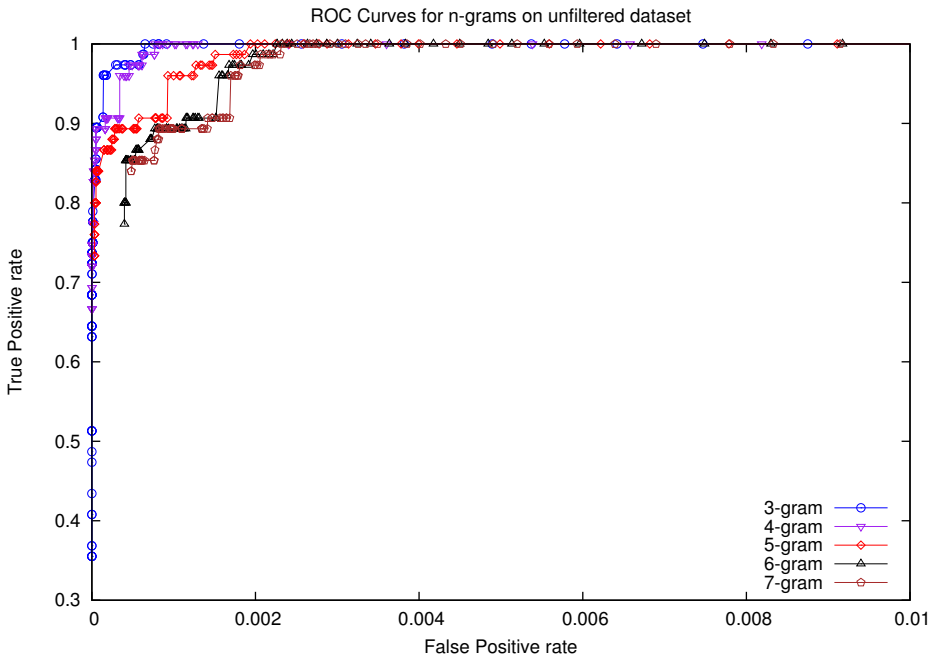

Figure 4.14: Comparison of *n*-grams ROC curves on the unfiltered dataset.

4.5 Model comparison

Ordering models by accuracy depends on the acceptable false positive rate, but in generally the ROC curves of the models are well-separated throughout the graph.

In order to depict the difference in terms of accuracy between the models presented in the previous section, Figure 4.15 presents a comparison between the 3-grams, 7-grams, Mahalanobis distance, Length, Combination and χ^2 distance, when using the filtered dataset to construct the models.

The *n*-grams models (shown for $n = 3$ and $n = 7$) are clearly the ones that show the highest true positive and lowest false positive rates. In regards to the remaining models, the Mahalanobis distance is the one that performs better, followed by the Length. There is however a considerable gap between the n-grams and Mahalanobis distance and the remaining ones, Length, Chi-Squared distance and Combination.

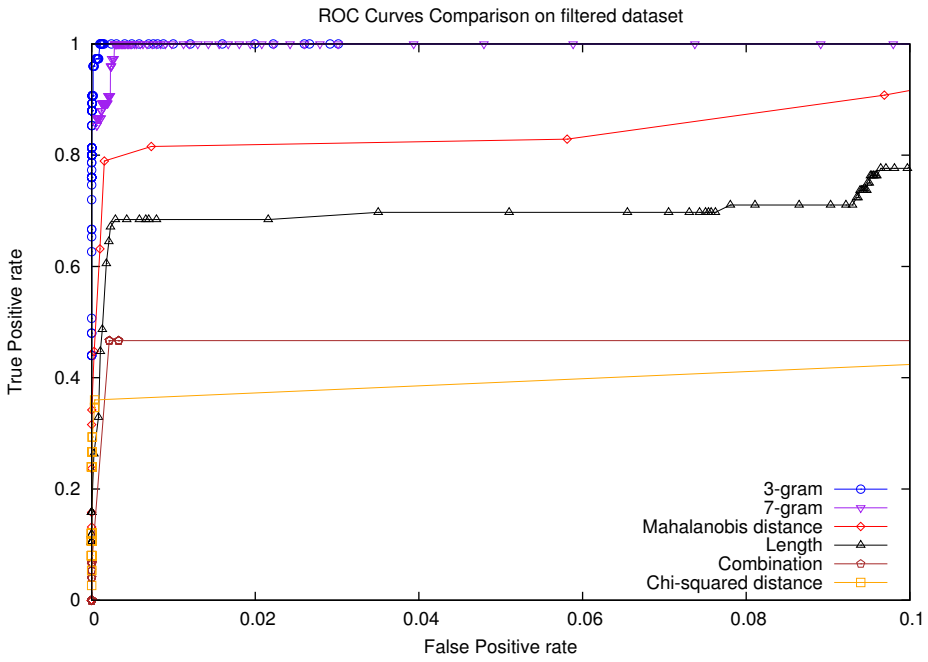

Figure 4.15: ROC curves comparing the accuracy of the 3-grams, 7-grams, Mahalanobis distance, Length, Combination and χ^2 distance when trained on the filtered dataset.

Figure 4.16, shows the same comparison between the models, but in this case, the unfiltered dataset was used for training.

The *n*-gram models were still the only ones that shown acceptable accuracy to be used in production environments, and the efforts we spent of filtering and obtaining a clean dataset for training the system did not seem to make that much of a difference when using the *n*-gram models. Regarding the Mahalanobis distance, as expected it performed worse with this dataset than with the filtered dataset, and in some cases even the Length model was slightly better than it, which suggests that filtering and cleaning datasets, depending on the web application and the environment in question, can in some cases be useful on production environments when there is the need to monitor only some critical applications that might be targeted by attackers.

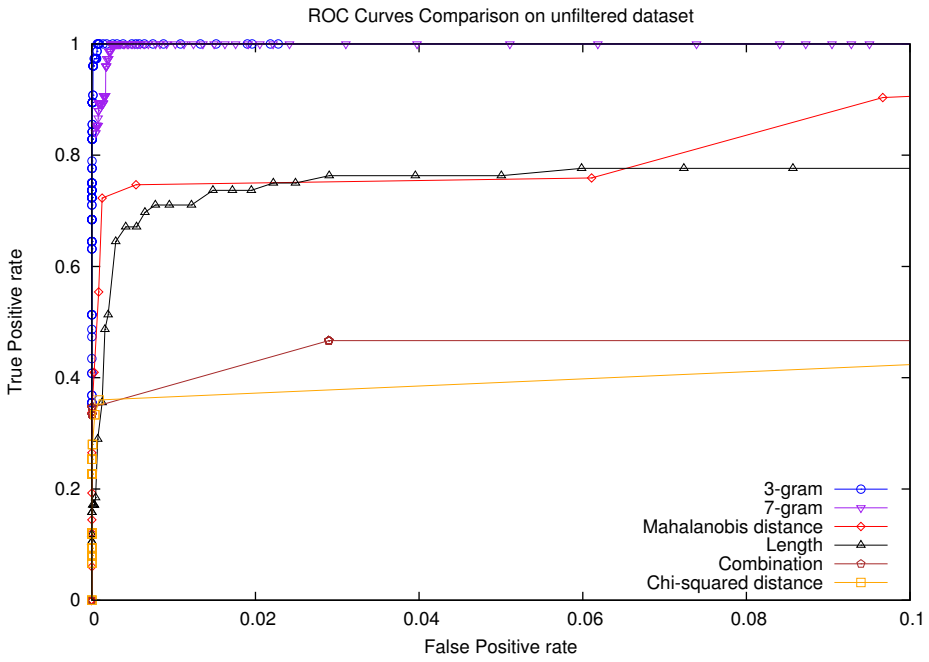

Figure 4.16: ROC curves comparing the accuracy of the 3-grams, 7-grams, Mahalanobis distance, Length, Combination and χ^2 distance when trained on the unfiltered dataset.

Chapter 5

Conclusion

Intrusion Detection is an important security asset that helps organizations to detect attacks against their infrastructures. Different approaches can be used to implement Intrusion Detection Systems. Signature (or misuse) based IDS work well at detecting previously known attacks as long as a tailored rule for each of those attacks exists in the signature database, which needs to be frequently updated in order to cope with new attacks. Nevertheless, signature-based systems are unable to detect novel (previously unknown) attacks and that is where anomaly-based IDS come into play. Anomaly-based IDS are able to build models from a set of training data, and if this model can correctly match the normal usage data, it gives them the ability to detect novel attacks.

Since many applications are rapidly being developed and pushed to the Internet, the World Wide Web and its HTTP protocol are two very interesting fields of research for anomaly detection. Regardless, there are several challenges related to intrusion detection for the web, which were addressed in this dissertation.

Our test results have demonstrated that anomaly-based models are a promising avenue to detect web-based attacks. We note that *n*-gram modeling of web requests shown to be a very promising and accurate model to detect attacks, with a very high detection rate and very low false positive rate. However, in terms of memory required to store the models, the *n*-grams were the ones that required more space, which is understandable as their models have to encode all the resulting possibilities from a given grammar. Compared with the Length model for example, that only saves statistical measures like the mean and standard deviation, the difference is considerable. Nevertheless, given the contrast in terms of accuracy, this is an acceptable tradeoff for a production environment, since nowadays memory and storage space are not that scarce anymore. In terms of performance for the training phase, the *n*-grams are quite fast when compared with the rest and nearly match the time

of the Length model, which is also an advantage in its favor. The same can be said for the detection phase, where the *n*-grams, together with the Length are the fastest in processing a dataset of requests and detecting those which attacks. The considerable amount of time taken by the Markov model and Mahalanobis distance models during the detection phase makes them inadequate for use in production scenarios. Regarding the Combination of models, the results we obtained were very poor as opposed to what we expected and to what other researchers had reported. We speculate the particularities of the web applications in our captured traffic data sets might justify these poor results, but we also note that using equal weights for each of the individual models and determining the normal threshold dynamically could be the reasons for the low accuracy of the Combination of models. Regardless, we note that no model is perfect and can miss attacks. Additionally, we also reported results to understand the overhead and requirements necessary to use these models for detecting anomalies and discussed their limitations which means further research is still necessary to improve anomaly detection systems.

We have experienced several issues that attest the difficulty of using an anomaly-based IDS to detect attacks using real-world traffic of web applications in production environments. First, the amount of data that we had to work with was considerable and we had to make a compromise and start filtering some of the data for sake of performing a thorough analysis during several months. Apart from the high volumes of data and tasks related to the network captures that we described in this dissertation, we also faced some underlying issues related to applying our IDS to that data. For once, we were working only with the web and the HTTP protocol which is stateless in nature. Then, the web applications present in our captured data were highly complex and presented a very dynamic behavior (alongside with web site content that changes and gets added often according to the nature of the services) that even led us to use some aggressive filters for the traffic at first, which made us believe we could easily model the traffic, when in fact we were only taking into account referral header values and confusing them with actual requests to the web applications. Moreover, getting training data that was adequate to build the detection models was very hard to obtain, and a lot of manual work had to be done to obtain it. The fact that we performed the analysis over complete HTTP requests, other than only the initial request lines like nearly all the previous researchers have done, also prompted some challenges which led us to discover some issues within our datasets representative of problems in client applications (namely in Nokia Symbian devices) and in HTTP Cookies, and also on requests coming through proxies (mostly old versions of squid).

We also used and experimented with two other systems that were proposed to detect anomalies in web requests as a term of comparison. First, we tested Webanomaly which per-

forms anomaly detection on HTTP access logs in the Common Log Format (CLF) and uses libAnomaly[1]. We spent a considerable amount of time testing this software using our datasets, but faced several issues mostly related to encoding of chars (in UTF-8 and iso-8859-1) and the impact that had on creating the models. We actually developed some custom scripts in php to convert between different charsets, but ended up not being able to produce results to compare with the ones produced by our system. Then we also experimented with HMM-Web[2], which is a framework for the detection of attacks against Web applications using hidden markov models. This framework was very interesting and worked relatively well for small amounts of data, but given the amount of requests we wanted to model, the machines where we tested it were simply unable to build the models as they ran out of memory.

5.1 Contributions

In this dissertation we start by discussing Intrusion Detection and then we provide a taxonomic review on Intrusion Detection Systems. We discuss the requirements and role of an IDS in the overall security architecture of an organization. Furthermore, we introduce the context surrounding our research, the World Wide Web, and then discuss vulnerabilities and attacks against web servers and applications. Then we briefly explain some of the problems related to the use of anomaly detection in the web context and present previous work related to this subject.

As for our major contributions, we present our research on anomaly detection of web-based attacks starting with a discussion on the concept of generalization and its importance in anomaly detection. Then we describe how anomaly-based systems create their models and how those are used to detect anomalies through similarity metrics. Moreover, we propose an sanitization algorithm that can be used to remove attacks from a dataset in order to make it appropriate for training anomaly-based detection systems. Then, we discuss in detail nine anomaly-based models that had been proposed for detecting web-attacks. Next, we describe our anomaly-based IDS that includes all these models and explain how it was used in to evaluate them. Finally, we discuss the generalization heuristics employed in the IDS to achieve more accurate detection results.

The main goal of our research was to study the usage of anomaly-based IDS in a production environment hosting a web application of great dimensions. We present an evaluation on

[1] `http://www.cs.ucsb.edu/~seclab/projects/libanomaly/`
[2] `http://sourceforge.net/projects/hmm-web-attacks/`

the usage of anomaly-based IDS with data from a production environment hosting a web application of great dimensions, and compare the accuracy of the different methods it employs. We start by discussing how our datasets were collected and processed, describe how our IDS works and then present results, requirements and accuracy for each of evaluated models, as well as a comparison between them.

5.2 Future Work

The results from this dissertation provide a solid view on the use of anomaly-based systems for detection of web attacks. However, this research area is relatively young and still has much to offer.

As future work, we would like to extend our approach to include other types of HTTP methods, namely the POST method which, after the GET method we described in this dissertation, is the most relevant for detection of web attacks. We would also like to build a larger database of attacks, and use it to re-evaluate our experiments.

Given that controlling generalization is necessary for accuracy, a direction for future work could involve investigation of new methods of generalization and ways of combining measures to better model the set of normal requests.

Another point that was not pursued in our research was the use of a DFA (deterministic finite automaton) to model web requests and detect attacks, which was left for future work. Previous research has shown this method is capable of reaching similar accuracy to that of the *N*-Grams.

Finally, we believe this dissertation can also serve as a foundation for applying the mechanisms we described to other protocols rather than HTTP, such email protocols like SMTP or IMAP, or even to DNS, which is also something we would like to explore as future work.

Anomaly-detection is a very interesting research problem which can be applied to a number of different areas related to computer and network security.

Bibliography

[1] L. Gordon, M. Loeb, W. Lucyshyn, and R. Richardson, "CSI/FBI computer crime and security survey," *Computer Security Institute*, vol. 25, 2005.

[2] S. Christey and R. Martin, "Vulnerability type distributions in CVE, version 1.1," 2007. [Online]. Available: http://cwe.mitreorg/documents/vuln-trends/indexhtml

[3] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, J. Ellis, E. Hayes, J. Marella, and B. Willke, "State of the practice of intrusion detection technologies," Carnegie Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-99-TR-028, 2000.

[4] R. Heady, G. Luger, A. Maccabe, and M. Servilla, "The architecture of a network level intrusion detection system," Los Alamos National Lab., New Mexico Univ., Albuquerque, Dept. of Computer Science, Tech. Rep., 1990.

[5] E. G. Amoroso, *Intrusion Detection*. Intrusion.Net Books, 1999.

[6] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, 2000.

[7] R. Maxion and R. Roberts, "Proper Use of ROC Curves in Intrusion/Anomaly Detection," Newcastle University, Tech. Rep. CS-TR-871, 2004.

[8] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[9] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[10] H. Debar, M. Dacier, and A. Wespi, "A revised taxonomy for intrusion-detection systems," *Annals of Telecommunications*, vol. 55, no. 7, pp. 361–378, 2000.

[11] S. Axelsson, “Intrusion detection systems: A survey and taxonomy,” 2000.

[12] J. P. Anderson, “Computer Security Threat Monitoring and Surveillance,” James P Anderson Co., Tech. Rep. 98–17, 1980.

[13] D. Denning, “An intrusion-detection model,” *Software Engineering, IEEE Transactions on*, vol. 13, no. 2, pp. 222–232, 1987.

[14] B. Mukherjee, L. Heberlein, and K. Levitt, “Network intrusion detection,” *Network, IEEE*, vol. 8, no. 3, pp. 26–41, 2002.

[15] R. Bace, *Intrusion detection*. Sams, 2000.

[16] T. Ptacek and T. Newsham, “Insertion, evasion, and denial of service: Eluding network intrusion detection,” 1998.

[17] M. Roesch *et al.*, “Snort-lightweight intrusion detection for networks,” in *Proceedings of the 13th USENIX conference on System administration*, 1999, pp. 229–238.

[18] A. Ghosh, J. Wanken, and F. Charron, “Detecting anomalous and unknown intrusions against programs,” in *Proceedings of the 14th Annual Computer Security Applications Conference*. IEEE, 1998, pp. 259–267.

[19] J. McHugh, A. Christie, and J. Allen, “Defending yourself: The role of intrusion detection systems,” *Software, IEEE*, vol. 17, no. 5, pp. 42–51, 2002.

[20] M. Crosbie and G. Spafford, “Active defense of a computer system using autonomous agents,” 1995.

[21] T. Berners-Lee and R. Cailliau, “WorldWideWeb: Proposal for a HyperText project,” *European Particle Physics Laboratory (CERN)*, 1990.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1,” RFC 2616 (Draft Standard), Internet Engineering Task Force, 1999, updated by RFCs 2817, 5785. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[23] D. Robinson and K. Coar, “The Common Gateway Interface (CGI) Version 1.1,” RFC 3875 (Informational), Internet Engineering Task Force, 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3875.txt

[24] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0," RFC 1945 (Informational), Internet Engineering Task Force, 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1945.txt

[25] J. Williams and D. Wichers, "OWASP top 10–2010," *OWASP Foundation, April*, 2010. [Online]. Available: http://www.owasp.org/index.php/Top_10_2010-Main

[26] G. Alvarez and S. Petrovic, "A new taxonomy of web attacks suitable for efficient encoding," *Computers & Security*, vol. 22, no. 5, pp. 435–449, 2003.

[27] F. Valeur, D. Mutz, and G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks," in *Intrusion and Malware Detection and Vulnerability Assessment*, ser. Lecture Notes in Computer Science. Springer, 2005, vol. 3548, pp. 123–140.

[28] A. One, "Smashing the stack for fun and profit," *Phrack magazine*, vol. 7, no. 49, pp. 1996–11, 1996.

[29] S. Cho and S. Cha, "SAD: web session anomaly detection based on parameter estimation," *Computers & Security*, vol. 23, no. 4, pp. 312–319, 2004.

[30] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 2003, pp. 251–261.

[31] C. Kruegel, G. Vigna, and W. Robertson, "A multi-model approach to the detection of web-based attacks," *Computer Networks*, vol. 48, no. 5, pp. 717–738, 2005.

[32] K. Wang and S. Stolfo, "Anomalous payload-based network intrusion detection," in *Recent Advances in Intrusion Detection*. Springer, 2004, pp. 203–222.

[33] W. Robertson, G. Vigna, C. Kruegel, R. Kemmerer *et al.*, "Using generalization and characterization techniques in the anomaly-based detection of web attacks," in *Proceedings of the 13th Symposium on Network and Distributed System Security (NDSS)*, 2006.

[34] K. Wang, J. Parekh, and S. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Recent Advances in Intrusion Detection*. Springer, 2006, pp. 226–248.

[35] K. Ingham, A. Somayaji, J. Burge, and S. Forrest, "Learning DFA representations of HTTP for protecting web applications," *Computer Networks*, vol. 51, no. 5, pp. 1239–1255, 2007.

[36] K. Ingham and H. Inoue, "Comparing anomaly detection techniques for HTTP," in *Recent Advances in Intrusion Detection*. Springer, 2007, pp. 42–62.

[37] K. Ingham and A. Somayaji, "A methodology for designing accurate anomaly detection systems," in *Proceedings of the 4th international IFIP/ACM Latin American conference on Networking*, 2007, pp. 139–143.

[38] K. Ingham, "Anomaly detection for HTTP intrusion detection: algorithm comparisons and the effect of generalization on accuracy," Ph.D. dissertation, University of New Mexico, 2007.

[39] D. Bolzoni, E. Zambon, S. Etalle, and P. Hartel, "Poseidon: A 2-tier anomaly-based intrusion detection system," *Arxiv preprint cs/0511043*, 2005.

[40] D. Bolzoni and S. Etalle, "Approaches in anomaly-based intrusion detection systems," *Intrusion Detection Systems*, pp. 1–16, 2008.

[41] D. Bolzoni and E. Zambon, "Sphinx: An anomaly-based web intrusion detection system," in *Workshop on Intrusion Detection Systems*, vol. 14, 2007.

[42] D. Bolzoni and S. Etalle, "Boosting web intrusion detection systems by inferring positive signatures," *Proceedings of the On the Move to Meaningful Internet Systems Conference*, pp. 938–955, 2008.

[43] D. Bolzoni, "Revisiting anomaly-based network intrusion detection systems," Ph.D. dissertation, University of Twente, 2009. [Online]. Available: http://purl.org/utwente/61673

[44] G. Vigna, F. Valeur, D. Balzarotti, W. Robertson, C. Kruegel, and E. Kirda, "Reducing errors in the anomaly-based detection of web-based attacks through the combined analysis of web requests and SQL queries," *Journal of Computer Security*, vol. 17, no. 3, pp. 305–329, 2009.

[45] C. Criscione, G. Salvaneschi, F. Maggi, and S. Zanero, "Integrated Detection of Attacks Against Browsers, Web Applications and Databases," in *European Conference on Computer Network Defense (EC2ND)*. IEEE, 2009, pp. 37–45.

[46] Y. Song, A. Keromytis, and S. Stolfo, "Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic," in *Proc of the 16th Annual Network and Distributed System Security Symposium (NDSS)*, 2009.

[47] F. Maggi, W. Robertson, C. Kruegel, and G. Vigna, "Protecting a moving target: Addressing web application concept drift," in *Recent Advances in Intrusion Detection*. Springer, 2009, pp. 21–40.

[48] W. Robertson, F. Maggi, C. Kruegel, and G. Vigna, "Effective Anomaly Detection with Scarce Training Data," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2010.

[49] W. Lee and S. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 227–261, 2000.

[50] M. Mahoney and P. Chan, "Learning nonstationary models of normal network traffic for detecting novel attacks," in *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 376–385.

[51] Z. Li, A. Das, and J. Zhou, "Model generalization and its implications on intrusion detection," in *Applied Cryptography and Network Security*. Springer, 2005, pp. 222–237.

[52] D. Knuth, *Fundamental algorithms, volume 1 of The art of computer programming*. Addison-Wesley, 1973.

[53] K. Tan and R. Maxion, ""Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector," in *Proceeding of the IEEE Symposium on Security and Privacy*, 2002, pp. 188–201.

[54] M. Damashek, "Gauging similarity with n-grams: Language-independent categorization of text," *Science*, vol. 267, no. 5199, p. 843, 1995.

[55] G. Combs *et al.*, "Wireshark-network protocol analyzer," 2010. [Online]. Available: www.wireshark.org

[56] V. Jacobsen, C. Leres, and S. McCanne, "Tcpdump/libpcap." [Online]. Available: www.tcpdump.org

[57] A. Luotonen, "The common log file format," *CERN httpd user manual*, 1995. [Online]. Available: http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format

[58] D. Ariu, G. Giacinto, and R. Perdisci, "Sensing attacks in computers networks with hidden markov models," *Machine Learning and Data Mining in Pattern Recognition*, pp. 449–463, 2007.

[59] A. Berqia and G. Nascimento, "A distributed approach for intrusion detection systems," in *Proceedings of the International Conference on Information and Communication Technologies: From Theory to Applications*. IEEE, 2004, pp. 493–494.

[60] I. Corona, D. Ariu, and G. Giacinto, "HMM-Web: a framework for the detection of attacks against Web applications," in *IEEE International Conference on Communications*. IEEE, 2009, pp. 1–6.

[61] G. Cretu, A. Stavrou, M. Locasto, S. Stolfo, and A. Keromytis, "Casting out demons: Sanitizing training data for anomaly sensors," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008, pp. 81–95.

[62] Y. Huang, S. Huang, T. Lin, and C. Tsai, "Web application security assessment by fault injection and behavior monitoring," in *Proceedings of the 12th international conference on World Wide Web*, 2003, pp. 148–159.

[63] R. Ierusalimschy, L. De Figueiredo, and W. Celes Filho, "Lua-an extensible extension language," *Software: Practice and Experience*, vol. 26, no. 6, pp. 635–652, 1996.

[64] S. Joshi and V. Phoha, "Investigating hidden Markov models capabilities in anomaly detection," in *Proceedings of the 43rd annual Southeast regional conference-Volume 1*, 2005, pp. 98–103.

[65] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "Secubat: a web vulnerability scanner," in *Proceedings of the 15th international conference on World Wide Web*, 2006, pp. 247–256.

[66] T. Krueger, C. Gehl, K. Rieck, and P. Laskov, "TokDoc: A self-healing web application firewall," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, 2010, pp. 1846–1853.

[67] S. Kumar, "Classification and detection of computer intrusions," Ph.D. dissertation, Purdue University, 1995.

[68] H. Labiod, K. Boudaoud, and J. Labetoulle, "Towards a new approach for intrusion detection with intelligent agents," *Networking and information systems journal*, vol. 2, no. 5-6, pp. 701–739, 1999.

[69] S. Northcutt and J. Novak, *Network intrusion detection: an analyst's handbook*. New Riders, 2002.

[70] A. Patcha and J. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[71] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee, "McPAD: A multiple classifier system for accurate payload-based anomaly detection," *Computer Networks*, vol. 53, no. 6, pp. 864–881, 2009.

[72] D. Scott and R. Sharp, "Abstracting application-level web security," in *Proceedings of the 11th international conference on World Wide Web*, 2002, pp. 396–407.

[73] K. Sequeira and M. Zaki, "ADMIT: anomaly-based data mining for intrusions," in *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 386–395.

[74] M. Tavallaee, N. Stakhanova, and A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 40, no. 5, pp. 516–524, 2010.

[75] E. Tombini, H. Debar, L. Me, and M. Ducasse, "A serial combination of anomaly and misuse IDSes applied to HTTP traffic," in *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2005, pp. 428–437.

[76] A. Valdes and K. Skinner, "Adaptive, model-based monitoring for cyber attack detection," in *Recent Advances in Intrusion Detection*. Springer, 2000, pp. 80–93.

[77] F. Valeur, "Real-time intrusion detection alert correlation," Ph.D. dissertation, University of California Santa Barbara, 2006.

[78] S. Yahalom, "URI Anomaly Detection using Similarity Metrics," Ph.D. dissertation, Tel-Aviv University, 2008.

[79] X. Yang, M. Sun, X. Hu, and J. Yang, "Grammar-Based Anomaly Methods for HTTP Attacks," in *Chinese Conference on Pattern Recognition*. IEEE, 2009, pp. 1–5.

[80] Q. Yin, L. Shen, and H. Wang, "Intrusion detection based on hidden Markov model," in *International Conference on Machine Learning and Cybernetics*, vol. 5. IEEE, 2004, pp. 3115–3118.