Fall 10-18-2016

A Framework for Hybrid Intrusion Detection Systems

Robert N. Bronte
Kennesaw State University

A Framework for Hybrid Intrusion Detection Systems

Master's Thesis

by

Robert Bronte
MSIT Student
Department of Information Technology
Kennesaw State University, USA

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Information Technology

October 18, 2016

A Framework for Hybrid Intrusion Detection Systems

This thesis is approved for recommendation to the Graduate Council.

Hisham M. Haddad
Thesis Co-Advisor

Hossain Shahriar
Thesis Co-Advisor

Jack Zheng
Committee Member

Kennesaw State University
Graduate College

Thesis/Dissertation Defense Outcome

Name Robert Bronte KSU ID

Email rbronte@students.kennesaw.edu Phone Number

Program MSIT

Title
A Framework For Hybrid Intrusion Detection Systems

Thesis/Dissertation Defense: Date 10/18/2016

☑ Passed ☐ Failed ☐ Passed With Revisions (attach revisions)

Signatures

Hossain Shahriar	10/18/16
Thesis/Dissertation Chair	Date
Committee Member	Date
	10/18/2016
Committee Member	Date
Committee Member	Date
Committee Member	Date
	10/19/2016
Program Director	Date
Rebecca Rutherfoord	10/19/2016
Department Chair	Date
David Baugher	10-20-16
Graduate Dean	Date

Last Modified 05/26/16

DEDICATION

This thesis is dedicated to my wife and family,
I love you and thank you for always being there for me.

ACKNOWLEDGEMENTS

I would like to thank Dr. Hisham M. Haddad and Dr. Hossain Shahriar
for their support and encouragement throughout this entire process.
I am extremely thankful for this opportunity to work with each of them.

I would also like to thank my entire thesis committee
for their feedback, comments and suggestions. I would not have made it this far
without my mentors behind me.

This research work is made possible through the assistance and support from
everyone, including my professors, mentors, my wife, family and friends.

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Web application security is a definite threat to the world's information technology infrastructure. The Open Web Application Security Project (OWASP), generally defines web application security violations as unauthorized or unintentional exposure, disclosure, or loss of personal information. These breaches occur without the company's knowledge and it often takes a while before the web application attack is revealed to the public, specifically because the security violations are fixed. Due to the need to protect their reputation, organizations have begun researching solutions to these problems. The most widely accepted solution is the use of an Intrusion Detection System (IDS). Such systems currently rely on either signatures of the attack used for the data breach or changes in the behavior patterns of the system to identify an intruder. These systems, either signature-based or anomaly-based, are readily understood by attackers. Issues arise when attacks are not noticed by an existing IDS because the attack does not fit the pre-defined attack signatures the IDS is implemented to discover. Despite current IDSs capabilities, little research has identified a method to detect all potential attacks on a system.

This thesis intends to address this problem. A particular emphasis will be placed on detecting advanced attacks, such as those that take place at the application layer. These types of attacks are able to bypass existing IDSs, increase the potential for a web application security breach to occur and not be detected. In particular, the attacks under study are all web application layer attacks. Those included in this thesis are SQL injection, cross-site scripting, directory traversal and remote file inclusion. This work identifies common and existing data breach detection methods as well as the necessary improvements for IDS models. Ultimately, the proposed approach combines an anomaly detection technique measured by cross entropy and a signature-based attack detection framework utilizing genetic algorithm. The proposed hybrid model for data breach detection benefits organizations by increasing security measures and allowing attacks to be identified in less time and more efficiently.

TABLE OF CONTENTS

Chapter 1: Motivation, Problem Statement and Contribution

1.1 Background

Even the novice user comprehends that data security breaches are not unfamiliar occurrences in this fast-paced, ever-advancing technological world. To understand what a data security breach is, the definition must be unambiguous. A data breach, specifically in relation to privacy, occurs when an unauthorized person, such as a skilled hacker [1], obtains the personal information of others. In a more general sense, a data security breach can be defined as an organization's unauthorized or unintentional exposure, disclosure, or loss of personal information [2]. Another way to present this is to say that a data breach can be simply defined as the accidental or unintentional loss of sensitive data [3]. Given that such threats exist and are of high concern, it is more than important to have some type of intrusion detection system. An Intrusion Detection System (IDS), is a system that protects computer networks against attacks. These systems work with the network's existing firewalls and anti-virus systems [4].

Currently, there are two common types of IDS: signature-based intrusion detection systems and anomaly-based intrusion detection systems. Signature-based IDS are commonly called misuse-based intrusion detection systems. These systems rely on signatures to recognize the attacks. Signature-based IDS would ideally identify 100% of the attacks with no false alarms as long as signatures are specified ahead of time. However, each signature, even if it leads to the same attack, has the potential to be unique from any other signatures. This is the most commonly implemented IDS [5, 6, 7].

The above explanation can be contrasted with the other common type of IDS: an anomaly detection system. This type of IDS focuses on the system's normal behaviors instead of focusing on attack behaviors, as seen with signature-based intrusion detection systems. To implement this type of IDS, the approach is to use two phases. The first phase is the training phase where the system's behavior is observed in the absence of any type of attack. Normal behavior for the system is identified into a profile. After this, the second phase or detection phase, begins. In this phase, the stored profile is compared to the way the system is currently behaving and deviations from the

profiles are considered potential attacks on the system. This can lead to several false negative alarms [8, 9, 10]. In an anomaly-based IDS, the system watches for changes from the expected behavior of the system. Currently, entropy and/or KLD have shown promising results in the literature surrounding mobile malware application detection [11].

1.2 Motivation

Each type of system has its own benefits and drawbacks. A hybrid model can optimize the benefits and minimize the drawbacks of the two systems. Hybrid models are essentially the combination of a signature-based detection approach and an anomaly-based detection system. To explain the need for a hybrid model, the various attacks that can be used to create a data security breach must be considered. Common attacks that are utilized to carry out data breaches are SQL Injection (SQLI) [12], brute-force [13], buffer overflows [14], Cross-Site Scripting (XSS) [15], Remote File Inclusion (RFI) [70], Directory Traversal (DT) [71] and Cross-Site Request Forgery (CSRF) [16] to name a few. In addition, hybrid models have a central focus on the newest threat: polymorphic attacks [17]. A limited amount of work has been done on hybrid models due to the nature of aforementioned polymorphic attacks [18].

This research work identifies numerous types of attacks that can result in data breaches detected by the proposed hybrid model. The model is based on analyses of logs that are generated by the web server and database server, which is explained a bit later in this proposal. The results of this study benefit and influence decisions of network and system administrators at companies who use an existing IDS and those who may be looking for a more advanced IDS model. End users may not see an exact benefit, but the security of their personal data may be exponentially increased.

In this work, we have replicated four specific web application attacks by using open source web applications. For the study, we employed an Apache web server and a MySQL database server in a virtual environment with a Windows operating system. Simultaneously, the study was run with an identical set up and design, but through the use of a Linux operating system. Examples of what the URLs may look like for the specified attacks under study are presented in the Table below.

Table 1: Examples of Attacks in URLs

Attack	Example URL
SQLI	http://www.xyz.com/login.php?uid=jonh&pwd=' or 1=1 --
XSS	http://www.xyz.com/index.php?uid=><body onload=alert('test1')>&pwd=test
RFI	http://www.xyz.com/test.php?uid=www.badsite.com/a.php
DT	http://www.xyz.com/index.php?uid=../../../../ dir/pwd.txt

1.3 Problem Statement

Log data analysis is a common practice used for the detection of data security breaches. This practice, however, can pose challenges for companies and organizations due to the vast number of data security breaches that are not detected. Several issues concerning log data analyses currently exist. One main problem is the creation of logs from multiple sources. The logs from all of the different sources do not necessarily contain the same information, creating the need for a universal log collector and analyzer. For instance, when a user has access to a company website, the company's web application server and database server both create separate logs. If a universal log analyzer were in place, any potential data security breaches would be easier to identify. Another problem with logs on different servers is that they may be located in different geographical regions or time zones. The timestamp that servers automatically place on the log may not match thus making the detection of a data security breach that much more strenuous.

Current literature shows little effort on combining anomaly-based and signature-based methods of identifying data security breaches and new types of attacks at the application layer. The reason for combining both methods is to detect even more attacks than either approach could detect alone. By compiling each of the approaches, additional false positive and false negative attacks can be discovered as well. In this thesis, the priority is to understand if current anomaly-based and signature-based approaches are sufficient to detect application layer attacks and how to improve upon these techniques.

We are applying genetic algorithm to pre-existing signatures to generate mutant signatures to detect attack violations. Similarly, since information theoretic metrics have not appeared much in the literature, this thesis is aimed at incorporating information theory as a computational approach to develop a new anomaly-based approach. There are two key research questions we plan to answer in this thesis:

- Could we develop an anomaly-based IDS to mitigate attacks on web applications? How effective the new approach would be compared to other existing anomaly-based approaches?
- How do we overcome the limited list of attack signatures in existing signature-based IDS with the goal of detecting new attacks? How effective is the new approach compared to existing signature-based approaches?

1.4 Research Methodology

The research methodology involved multiple literature reviews of more than 40 articles. Content within the articles varied. For example, articles were included that discuss anomalies, signatures, intrusion detection systems, and web application attacks. The methods used for this research include the following activities:

1) Conduct literature searches on attacks related to data breaches, anomaly-based attack detection, signature-based attack detection, hybrid models and log analysis.
2) Explore the logs based on attack detection types, categorizing them as signature-based or anomaly-based detection approaches, to create an offline analysis framework for storing and indexing the data from the various logs.
3) Develop a set of pre-defined signatures from an established database and a model that represents a normal data (training data) access pattern to make an abnormal data (testing data) access pattern identifiable.
4) Calculate the result of the test log data to measure the performance of the proposed solution at detecting false positives and false negatives.
5) Compare the results using information theoretic metrics with other available prototype tools on performance.

1.5 Overview of Research Tasks

This work addresses the stated research by performing the following tasks:

1) Conduct Literature Search
    i. Conduct literature search on existing log combination techniques, specifically aimed at methods used by intrusion detection systems. Since there are two main types of intrusion detection systems, the literature search will include explanations and techniques of each type.

ii. Compose a survey of compiled papers related to this topic and document the findings.

2) Develop Virtual Environment
   i. Profile available pre-defined signature and anomaly-based intrusion detection systems and identify common attacks within each IDS and which logs are generated by these attacks.
   ii. Use the resulting profile to develop a centralized log server that is connected to a web server and a database server collecting logs from the generated attacks.
3) Develop Hybrid Detection Model
   i. Propose detection techniques for both types of attacks in one model. This purpose of this is to detect any type of attack so prevention methods can be deployed.
   ii. Use the hybrid model to demonstrate the proposed approach within controlled environment.
4) Conduct Testing and Evaluation
   i. Deploy web applications in XAMPP in the Linux and Windows environments. XAMPP is an integrated web platform consisting of a web server and a database server.
   ii. Generate normal traffic and attack traffic data and MySQL logs and Apache logs to detect the application layer attacks.
   iii. Detect anomalies using information theoretic metrics with an expected false alarm rate below the average of 8.4% seen in the literature [8, 18].
   iv. Detect attacks that have unique signatures based on a signature database as seen in other studies conducted in the past [19-20]. The expected accuracy level of signature detection is 100%.
5) Disseminate Work Results
   i. Disseminate the log analyzer and dataset with those in the field of technology.
   ii. Prepare and submit one or more papers for publication at relevant venues.

In the next Chapter, we discuss the findings from literature surveys surrounding our research goals and objectives.

Chapter 2: Literature Review

2.1 Overview

This Chapter elaborates on the surrounding literature searches and explains the findings from the literature research. Due to the level of detail that is required for this thesis, multiple literature reviews were carried out, but only five directly apply to the case studies and fit within the context of this work. In subsequent sections, first, anomaly-based intrusion detection is explained, followed by a short description of how entropy is used to detect attacks. Then, a section discussing related literature on signature based intrusion detection is provided. Next, genetic algorithms are illustrated as a new avenue for attack detection. Finally, this Chapter concludes with a discourse about benchmarks used for evaluation of datasets in the literature.

2.2 Anomaly-based Intrusion Detection

Literature surrounding this topic is not scarce. In fact, in [21] the authors study the development of an IDS by a training dataset collected from a large scale web application. The work only considered GET requests and did not consider POST types of requests or response pages. They captured logs from a TShark tool and converted them to Common Log Format. The filtered data was generated by accessing sub-applications. They manually inspected every single request to gather a filtered (good) dataset. Their detection used nine models. Cho *et al.* [22] develop a Bayesian parameter estimation-based anomaly detection approach from web server logs and showed that it outperformed signature-based tools such as Snort. They assume a user visits a set of pages in a certain order (denoted as a session). Their approach is effective when the order is maintained. Ariu [23] develop a host-based IDS to protect web applications against attacks by employing the Hidden Markov Model (HMM). HMM is used to model a sequence of attributes and their values received by web applications. To account for various parameters and their values, they employ multiple HMMs and combine them to generate an output for a given request on likelihood that it would be generated from the training dataset.

Park *et al.* [24] analyze both GET and POST request data and capture the profiles of the data for each parameter. Then they apply the Needleman-Wunch algorithm for a new parameter value to see if the new value would be accepted or not as part of the alarm generation process. Le *et al.* [25]

develop the DoubleGuard framework that examines both web server and database server logs to precisely detect attacks leaking confidential information. They report 0% false positive rate for static web pages, and 0.6% false positive rate for dynamic web pages. A similar approach has been proposed by Vigna *et al.* [26] earlier. Their work reduces false positive warnings in a web-based anomaly IDS by combining web log anomaly detection and a SQL query anomaly detector. In their approach, a request that was found to be anomalous, based on logs, would still be issued to a database server if it is found that the request is not accessing sensitive data from the server.

Ludinard *et al.* [27] profile web applications by learning invariants (e.g., a user session should have same value as the login value). Then source code is instrumented to check violation of invariants. If an invariant is violated, it indicates an anomalous input has been supplied. Li *et al.* [28] develop an anomaly-based IDS by first decomposing web sessions into workflows. A workflow consists of a set of atomic requests which may access one or more data objects. They apply HMM to model the sequence of the data access of workflows. Gimenez *et al.* [29] develop a web application firewall (as an anomalous request detector) and captured its behavior through an XML file, which specifies the desired attributes of parameter values. An input value deviating from the expressed profile is considered an attack. However, the approach would generate false positive warnings as it does not consider page and path information to be more precise.

Our work [30] is focused on web-based anomaly detection analyzing log files, based on the outlined work of Robertson *et al.* [31]. Both studies consider similar resource names to compare a new request with profiled requests to reduce false positive warnings. However, we apply information theoretic measures to compare entropy levels for parameter combinations and values. A comparison of each aforementioned study to our own objectives is provided in Table 2.

Table 2: Related Works on Anomaly-based IDS

Author(s)	Study summary	Contrast with our study [30]
Nascimento *et al.* [21]	The work only considered GET requests and did not consider POST types of requests or response pages; Captured logs from a TShark tool and converted them to Common Log Format; The filtered data was generated by accessing sub-applications	We employ both server and client side tools to collect GET and POST data, combined them to form unified log files and processed them for defining good and bad datasets
Cho *et al.* [22]	Develop a Bayesian parameter estimation based anomaly detection approach from web server logs and showed that it outperformed signature-based tools such as SNORT; Assume a user visits a set of pages in a certain order	Our approach relies on path resources and does not need to rely on the order in which a user visits different pages
Ariu [23]	Create a host-based IDS to protect web applications against attacks by employing the Hidden Markov Model (HMM). HMM is used to model a sequence of attributes and their values received by web applications	Our approach is free from the state explosion problem that the HMM approach suffers from
Park *et al.* [24]	Analyze both GET and POST request data and capture the profiles of the data for each parameter; Apply the Needleman-Wunch algorithm for a new parameter value to see if it would be accepted as part of the alarm generation process	We employ entropy levels of parameter values for profiling and attack detection
Le *et al.* [25]	Create the DoubleGuard framework that examines web server and database server logs to detect attacks leaking confidential information	Our study uses a similar framework, but aims to identify all potential attacks
Vigna *et al.* [26]	Reduce false positive warnings by combining web log anomaly detection and a SQL query anomaly detector; A request that was found to be anomalous would still be issued to a database server if the request is not accessing sensitive data	We focus on web server logs and apply entropy measures to detect anomalous requests
Ludinard *et al.* [27]	Profile web applications by learning invariants (e.g., a user session should have same value as the login value); Then source code is instrumented to check for violation of invariants, which indicate anomalous input	Our work does not rely on source code instrumentation
Li *et al.* [28]	Develop an anomaly-based IDS by decomposing web sessions into workflows of a set of atomic requests which may access one or more data objects; Apply the Hidden Markov Model	We apply cross entropy of the parameter name, value, and types

Author(s)	Study summary	Contrast with our study [30]
	(HMM) to model the sequence of the data access of workflows	
Gimenez *et al.* [29]	Use a web application firewall as an anomalous request detector and specifies the desired attributes of parameter values; This generates false positive warnings since it does not consider page and path information	Our work does not rely on firewall policies and applies entropy measures to detect anomalous requests
Robertson *et al.* [31]	Similar resource names are used to compare new requests with profiled requests to reduce false positive warnings	We apply information theoretic measures to compare entropy levels for parameter combinations and values

2.3 Related Works: Information Theoretic Metrics

The earliest work we are aware of in the literature is from Lee *et al.* [32]. In their work, the authors applied several metrics (entropy, relative entropy and conditional entropy) to model network log data to demonstrate anomalies. Similar to their work, we apply entropy to model web request parameter values. However, we explore the application of information theoretic metrics for web-based anomaly detection. Shahriar *et al.* [33] apply entropy to detect vulnerable SQL queries in PHP web applications. Later they explore an information theory based dissimilarity metric (Kullback-Leibler Divergence) to detect XSS attacks in web applications [34]. The KLD measures have been explored to detect repackaged Android malware [36] and content provider leakage vulnerabilities [35]. Ozonat *et al.* [37] detect anomalies in performance metric behavior in large-scale distributed web services applying information theoretic metrics. Below, Table 3 presents these works with additional details.

Table 3: Detecting Anomalies through Information Theoretic Metrics

Author(s)	Main Objective	Metrics Used
Lee *et al.* [32]	Suggest how to build the correct anomaly IDS for audit datasets and measure the performance	Entropy, Conditional Entropy, Relative Entropy, Information Gain and Information Cost
Shahriar *et al.* [33]	Discover PHP web applications that are vulnerable to SQL injection without relying on attack input	Entropy
Shahriar *et al.* [34]	Implement malicious JavaScript code intentionally to find XSS attacks	Kullback-Leibler Divergence
Cooper *et al.* [35]	Identify the source and the source code behind specific malicious functions of interest on Google's Android mobile operating system	Kullback-Leibler Divergence
Shahriar *et al.* [36]	Detect repackaged malware on an Android operating system to avoid end users from downloading applications with unexpected, malicious functionalities	Kullback-Leibler Divergence
Ozonat [37]	Model the temporal and spatial relationships between various web services to find anomalies	Relative Entropy

2.4 Signature-Based Intrusion Detection

A representative sample of literature works have developed signature-based IDSs [20, 38, 39, 40, 41, 42]. The vast amount of research on this topic has focused on the network layer and multiple Denial of Service attacks. Each of the previously mentioned works has its own strengths, but all of the approaches have universal underlying limitations. First, in the literature, there is little wide-spread coverage of the known web application layer attacks (see Table 4 for the levels we follow in this Chapter). Each study identified at least one of these types of attacks. Second, each of the related works have yet to attain zero false positive and false negative rates with only one exception.

Finally, regardless of the type of attack the authors were searching for, each study only considered one type of log data for analysis. This paper addresses these limitations by discussing a signature-based IDS framework to detect certain application layer attacks by analyzing data from multiple logs generated by web applications. The included signature-based IDS is meant to protect web applications. Table 4 shows various attack types at different levels and related work that proposed IDS. This Table was created based on industry-level data [44]. Our approach analyzes web server log data, trains an IDS using a GA, and detects three common web application attacks (Cross-Site

Scripting, SQL Injection and Remote File Inclusion attacks). The definitions and explanations of these attacks can be found in Section 2 of Chapter 4. In addition, the evaluation of previously collected normal data and newly created malicious data via this approach would provide a detailed view of the results.

Table 4: Signature-based Attack Detection

App Layer	Attack Type	Work(s)
Application	XSS	[15, 19, 30, 41, 43]
	RFI	[30, 43, 46]
	SQLI	[12, 30, 43, 45]
	Brute-Force	[13, 15, 40]
	Buffer Overflow	[5]
	CSRF	[53]
	Zero-days	[38]
Transport	SYN/ACK	[40]
	XMAS Scans	[20]
	DoS	[5, 39]
	Apache2	[42]
Network	Smurfs	[20]
	Ping-of-Death	[20]
	SYN flooding	[40]

For our purposes, the network layer and the transport layer attacks in the Table 4 above are irrelevant. However, it serves to illustrate that attacks are diverse and can occur in other contexts outside of this case study. From the vast body of literature about signature-based IDSs, we classify the works in multiple ways. We look at benchmark data sources, the attributes of examined data, metrics employed by the IDS, the environment where implementation and evaluation was conducted, the types of attacks being covered, and reported performance rates (false positive and false negative rates). Among all the works, one study achieved detection of zero-day attacks by analyzing web and database server logs, and examining attack code, command payload and traffic generated by the payload [38]. Neelakantan *et al.* [39] used protocol information, headers, and packet payloads of packet captures to reduce the total number of false alarms from DDoS attacks. The rules defining the source address, destination address and destination port were used to increase the speed of signature detection in another study [40].

In [20], the signature of a priori algorithm from the MySQL database logs was proposed to detect known network level attacks. Based on PHP source code, Gupta *et al.* [41] used a controlled VM to detect XSS attacks achieving 0% FP and 0% FN rate. The authors of [42] used known SNORT and ClamAV signatures to detect signature-based attacks. Additionally, the researchers utilized a honeypot to collect data for a character frequency exclusive signature matching scheme and the Boyer-Moore algorithm was applied to the dataset. Through the use of Mail Exchange (MX) records on Windows servers, the authors of were able to brute-force into numerous Hotmail addresses [13]. In [5], Vigna *et al.* examined Apache logs to collect data on string length and sequence and exploited mutations to detect buffer overflows, directory traversals and other attacks.

Chou [12] used web servers that were hosted in a Cloud environment to detect SQL injections, XSS, and brute-force attacks. Finally, in [15] the researchers looked at innerHTML properties such as GET, HTTP header and cookies to determine the presence of mutation-based XSS attacks, denoted as mXSS. This work, in contrast, uses logs collected from both the web server and the MySQL database for analysis. The environment is configured with a single host running multiple virtual machines (VMs) within a virtual cluster. Some of the VMs are running Windows while others are running Linux. We decided to use a signature matching scheme and added genetic operators to introduce changes into the signatures. Such mutation allowed for the GA to detect all of the known attack input. These include XSS, SQLI and RFI attacks and result in 0% false positive rate and 0% false negative rate. We outline the major points of each of these studies and compare them with our proposed approach, as shown in Table 5 below.

Table 5: Related Work on Signature Based Attack Detection Approaches

Author	Source of data/ benchmark	Attributes of data examined	Metrics used in their model	Environment, configuration, virtual machines	Attacks detected	Effort to reduce alarm rates (FP and FN)
Holm [38]	Web servers and database servers	Attack code, command payload, traffic generated by the payload	% of known attacks detected	Windows 2000, XP, 2003, Vista and Ubuntu 10.04	Zero-day Attacks	False positives are mentioned, but no % is provided due to the variety of employed OS

Author	Source of data/ benchmark	Attributes of data examined	Metrics used in their model	Environment, configuration, virtual machines	Attacks detected	Effort to reduce alarm rates (FP and FN)
Neelakantan *et al.* [39]	Apache, OpenSSH, SMTP	Protocol information, headers, and packet payloads of packet captures	% of critical alarms generated	Linux (Red-Hat), Windows 2003 and Windows 2000	DDoS	The article mentions that the total number of false alarms were reduced
Krugel *et al.* [40]	Web servers	Rules defining the source address, destination address, and destination port	Average increase in speed of signature-detection	Red-Hat Linux	Brute-Force, Synscan, Portscan	Not reported
Modi *et al.* [20]	MySQL database log	Signature A priori Algorithm	Proposed solution, not implemented thus far	Eucalyptus on Ubuntu	Known network attacks, DoS derivatives	Propose a low FP rate
Gupta *et al.* [41]	PHP Source Code	HTML Context including styles, body tag names, etc.	Number of safe vs. unsafe files	Controlled VM	XSS attacks	0% FP and 0% FN rate
Meng *et al.* [42]	Honeypot	Character-frequency exclusive signature matching scheme and the Boyer-Moore algorithm	Maximum execution time variance using incoming payload or signature set partitions	VMs in Cloud Environment	Known SNORT and ClamAV signature-based attacks	Not mentioned; focused on reducing TIME to process signatures, not FP/FN rates
Parwani *et al.* [13]	MX records on Windows servers	Expired Hotmail addresses	Number of accounts that were hacked	Windows	Brute Force Attacks	Not discussed
Vigna *et al.* [5]	Web servers, such as Apache	String length and sequence, exploited mutations	Number of signature-based attacks detected by either SNORT or IIS RealSecure	Linux, Windows, OpenBSD	Buffer Overflows, DoS, Stack Overflow, DT, Double decoding (code execution),	Did not consider FP/FN results because the goal was to provide useful indication about the

Author	Source of data/ benchmark	Attributes of data examined	Metrics used in their model	Environment, configuration, virtual machines	Attacks detected	Effort to reduce alarm rates (FP and FN)
			before and after mutating data		Non-exhaustive Signatures	average quality of signatures under testing
Chou [12]	Web servers in the Cloud	Data models categorized as SaaS, PaaS or IaaS	Increased frequency of identified SQL injection attacks in SaaS, PaaS, and IaaS Cloud settings measured in %	Linux, Solaris, Windows VMs in Cloud environment	Malware (SQL) injection, other attacks detected with Cloud systems: DDoS, brute-force, session hijacking, XSS, etc.	Not discussed; focus was more on cloud security
Heiderich *et al.* [15]	Web servers	innerHTML properties: GET, HTTP header, cookies, etc.; mXSS vectors	Page load time in milliseconds versus the page size with and without the performance penalty introduced to users by TrueHTML	VMs running Ubuntu	Mutation-based XSS attacks (mXSS)	Briefly suggested as potential future research, but not directly addressed
This study [43]	Web servers MySQL database	Signature matching scheme with added genetic mutations	Mutation, selection, chromosome cross-overs	VMs running Linux or Windows	XSS, SQLI, RFI	0% FP and 0% FN rate

2.5 Genetic Algorithm

Data sources and types of detected attacks vary greatly across the body of signature-based IDS literature. For example, in [48], Avancini *et al.* examined parameters and values of PHP code to find XSS attack vectors. These authors use static analysis of the Genetic Algorithm (GA) to automate the log analysis procedure. They also minimized the false positive and false negative alarms through path sensitization. Authors of [49] created their own set of suspicious data and used

the DARPA dataset as the normal dataset for their study. This study was the only study in the literature we encountered to use the DARPA dataset as a normal dataset. When comparing this GA-based IDS to a known signature database called Snort, the authors found that the GA-based IDS outperformed Snort by detecting a higher number of attacks and having a lower false alarm rate than Snort. Danana *et al.* [50] obtained all of their data from the KDD99 dataset, and were able to find attacks including Denial of Service, Probing, User-to-root and Remote-to-local attacks. A fuzzy genetic algorithm utilized in [51] pulled data from a six-by-six matrix of response-resource entries to measure the parameters of the fitness function.

A multivariate statistical clustering algorithm was suggested to detect web application attacks in [52]. The discrete variables in the study were measured by frequency and the number of similar characters between two separate activities (attacks) was suggested as a way to lower the number of false alarms. Liu and Fang genetically modified two sets of real numbers to shorten the lengths of the chromosomes to optimize the GA [53]. In another study, network attacks like smurf, teardrop, neptune, portsweep and others were identified in offline, normal audit data as well as in real time, processed data [53]. Normal data and attack data were compared by the authors looking for Denial of Service, Probes, User-to-root and Remote-to-local network-level attacks [54]. Authors of [55] lowered the false alarm rates by implementing an optimal genetic feature selection process and a support vector machine. Despite the potential of the existing signatures in the literature to detect patterns across any operating system, this work will use genetic operators to mutate such patterns for detection. Table 6 summarizes these related works and the methods used by the authors to reach their conclusions.

Table 6: Comparison of Related Work on Genetic Algorithms

Author	Source of data	Metrics	Types of attacks	Effort to reduce alarm rates (FP and FN)
Avancini *et al.* [48]	Parameters and values of PHP code	Integrating a static analysis of the genetic algorithm and taint analysis	Cross-Site Scripting attack vectors	FP and FN were minimized through path sensitization
Barati *et al.* [49]	Normal dataset (DARPA) and suspicious dataset	Number of scan attacks detected/missed by this GA-based IDS versus Snort	Horizontal and vertical scan attacks	Overall false alarm rate was 10%

Author	Source of data	Metrics	Types of attacks	Effort to reduce alarm rates (FP and FN)
Danane *et al.* [50]	KDD99 dataset	Accuracy, execution time, memory allocation	DoS, Probe, U2R, R2L	Rules added in the testing phase to reduce FP and FN
Fessi *et al.* [51]	6 by 6 binary matrix of response-resource entries	Rules that only match anomalous connections to show signatures, parameters of fitness function	Not specified; focused on the fitness value: attack impact ratio	FP rate was "low" but not specified numerically
Zhou *et al.* [52]	Multivariate statistical clustering algorithm is suggested to detect attacks; No real data is used in the study	Frequency for discrete variables; the number of similar characters between 2 activities	Web-layer attacks under study, but none are specified	Not disclosed but states the goal is "a very low rate"
Liu *et al.* [53]	Two sets of real numbers are genetically modified to shorten chromosome length	Delphi method to determine Figure 1 in this article; fitness value of each chromosome, total fitness values, selection probability	No attacks; discusses modified genetic algorithm for optimization	No FP or FN; studied detection reliability (R), time of detection (T), and threshold time (S)
Narsingyani *et al.* [54]	Offline data for normal traffic dataset [audit data]; real time data for attack detection [Processed data]	Src_bytes, land, wrong_fragment, [all numerical] and service [nominal]	DoS attacks: smurf, pod, teardrop, Neptune, back, portsweep	Specifically focused on FP rate to improve performance by increasing the number of rules
Senthilnayaki *et al.* [55]	Attack data and normal data	Protocol type, service, src_bytes, flag, num_failed_logins, Logged_in, srv_diff_host_rate, dst_host_srv_count, is_guest_login, and num_shells	DoS, Probe, U2R, R2L	False alarms reduced by using optimal genetic feature selection and a support vector machine

2.6 Benchmarking and Evaluation

Work completed by Alhamazani *et al.* [56] proposes a benchmark named the Cross-Layer Multi-Cloud Application Monitoring- and Benchmarking-as-a-Service (CLAMBS). This study used an Apache web server, a Tomcat web server and a MySQL database. The attack detection approach

worked across Windows and Linux environments, and was implemented to establish the baseline performance of applications while also monitoring each application's quality of service (e.g., round trip time, packet loss). In this study, datasets of 50, 100 and 200 MB were generated on a virtual machine as a proof-of-concept to test Amazon Web Services and Windows Azure. However, this benchmark also had a heavy reliance on JAVA and specific reliability on cloud services. As described by [57], a study by Champion *et al.* [58] utilized an attack detector titled Ebayes by the authors. This detector was able to detect more attacks at the application layer than the commercially available intrusion detection system (IDS) in 2001. Despite this, Ebayes still only detected up to 50% of known attacks in the in-house generated dataset. Athanasiades *et al.* [57] also describe a study carried out in 1997 [59]. Through the use of customized software based on the Tool Command Language Distributed Program (TCL-DP) package, these authors simulated users performing FTP and/or Telnet procedures. A script was then created to record and replay the user actions to generate their dataset. These authors used a very controlled environment to ensure that the results of the study could be replicated. Aside from this precaution, the authors neglected to test their dataset for attack detection in a normal network environment. Instead, attack detection was only under study in the stress tests of the data [59].

Using a self-named benchmark, Ballocca *et al.* [60] created a fully integrated web stressing tool. The benchmark, called the Customer Behavior Model Graph (CBMG), relies on the stressing tool that is composed of a script recorder and a load generator. This allowed the traffic from the workload characterization to be automated and begin from the web log files. Generating this workload, on the other hand, is time consuming and involves multiple processes. The authors in [61] developed a unique algorithm to generate the Research Description Framework (RDF) benchmark. Generating datasets would no longer be an issue if RDF was adopted as a universal benchmark because the authors state that this generator can convert any dataset (real or fake) into a benchmark dataset. They can even make sure the user-specific data properties are generated. While this sounds like a potential solution, the authors also noted that the initial input data must first be cleaned and normalized. Neto *et al.* [62] took a trust-based approach to application-layer attack detection. By defining how likely vulnerabilities were to be found rather than determining a specific number of attacks that would be found, these authors measured the trustworthiness of the relationship between the application and the developer. As a new approach, this approach may

sound simple, but it is full of complex coding and involves a three step process. Anyone wishing to use this benchmark would require a fundamental understanding of how to read complex computer code.

In [63], Neto *et al.* implemented Static Code Analysis as a benchmark for attack detection. Four metrics were applied to real web applications to determine the trustworthiness of the application. An application with a high mean score across each of the metrics was deemed untrustworthy. Despite these efforts, the benchmark relied on the TCP-App standard for web application code and JAVA. Stuckman *et al.* [64] crafted a modular benchmark on a testbed that automated the evaluation of an intrusion prevention system. This benchmark was a collection of modules and each module had an intentionally vulnerable application installed in an environment that would allow the application to run and simulate an attack. Each testbed was a deliverable virtual machine, so anyone could easily deploy the benchmark on any system running Debian Linux. The benchmark was limited in that it had to be customized for each individual developer if the developer wanted to generate their own attacks. Another benchmark for attack detection was made by Zhang *et al.* [65] in 2009. Known as WPBench, or Web Performance Benchmark for Web 2.0 applications, this benchmark utilized a replay mechanism that was able to simulate user interactions with applications and characteristics of networks and servers. The benchmark worked well with Internet Explorer, Firefox, and Google Chrome browsers.

Ultimately, this benchmark was intended to measure the responsiveness of each of the browsers to page loading times and event response times. The main disadvantage of this proposed benchmark was that is required users to run the benchmark in the background of their daily browsing activities and recorded their actions. This benchmark would then take more time to replay the actions in order to learn the user's environment and preferences. A Model Driven Architecture (MDA) approach was proposed in [66] and allowed for the generation of repetitive and complicated infrastructure code by the benchmark tool. The MDA approach included a core benchmark application, a load testing suite and performance monitoring tools for the user. However, the approach did not include any type of tool to collect information regarding data performance. Yet another benchmark suggested in the literature is a web server benchmark named *servload* by the authors of the work [67]. This benchmark supports load balancing, can replay web server logs,

tells users the number of requests and sessions, as well as provide the connection time and error counts to the user. All of this information is very useful when trying to establish a standard for application-layer attack detection, but *servload* only supports GET requests and has to analyze web server logs. Varying log formats bring *servload* to a halt, impeding this benchmark from being universally adopted. We show the comparison and contrast among these literature works in Table 7. Coupled with this summary, Table 8 highlights the specific data attributes that other authors measured to evaluate their datasets.

Table 7: Summary of Related Literature on Benchmarking

Author(s)	Description of proposed new model or benchmark	Advantages of method	Disadvantages of method	Size of the Dataset
Alhamazani *et al.* [56]	CLAMBS-Cross-Layer Multi-Cloud Application Monitoring- and Benchmarking-as-a-Service	Monitors QoS of application QoS information of application components is shared across cloud layers Baseline performance established by B-a-a-S	Study a proof-of-concept on a VM testing Amazon AWS and Windows Azure Heavy reliance on JAVA	Datasets of 50MB, 100 MB and 200MB
Athanasiades *et al.* [57]	Environment similar to DARPA 1998 Ebayes detector [58]	Detected more attacks than the commercially available IDS [58]	Not publicly available (Privacy issues at Georgia Tech would not allow researchers to access their own subnet)	Did not disclose the size of the dataset
Same as above [57]	Custom Software based on the Expect and Tool Command Language Distributed Program (TCL-DP) package [59]	Environment was very controlled to make sure the results could be replicated	Attack identification only took place during stress tests	Did not disclose the size of the dataset
Ballocca *et al.* [60]	Customer Behavior Model Graph (CBMG)	Traffic from the workload characterization is automatic The characterization process begins from the web log files	Creating a workload takes a lot of time and involves four different processes: merging and filtering web logs, getting sessions, transforming sessions, and CMBGs clustering	No size given

Author(s)	Description of proposed new model or benchmark	Advantages of method	Disadvantages of method	Size of the Dataset
Duan *et al.* [61]	Research Description Framework (RDF)	This generator can convert any real or fake dataset into a benchmark dataset and make data with similar characteristics as the real dataset with user-specific data properties	Must perform data cleaning and normalization of the dataset before using this method	User can indicate dataset size
Neto *et al.*[62]	Trust-based benchmark with 5 metrics: Code Average Code Prudence, Code Average Code Carelessness, Quality, Hotspot Prudence Discrepancy and Hotspot Carelessness Discrepancy	Defining how likely vulnerabilities are to be found rather than the number of vulnerabilities	Anyone using this benchmark method would have to understand how to read code	No set size of data
Neto *et al.*[63]	Static Code Analysis	Applies all 4 metrics to real web applications; higher metric values mean the product is less trustworthy	Relies on TCP-App standard for code on web applications instead of developing their own JAVA heavy	Not disclosed
Stuckman *et al.* [64]	Run a modular benchmark on a testbed that automates the evaluation of the IPS	Testbed can be given out as a VM, so anyone can deploy it with Debian Linux	Need to make this customizable for individual developers to generate their own attacks	Resulting size of code; not specified
Zhang *et al.* [65]	WPBench: Web Performance Benchmark for Web 2.0 applications	Replay mechanism simulates user interactions with applications and characteristics of servers and networks	Requires users to run the benchmark in the background of daily browsing to create a recording of steps to replay so the benchmark learns the environment and user preferences	38MB

Author(s)	Description of proposed new model or benchmark	Advantages of method	Disadvantages of method	Size of the Dataset
Zhu *et al.* [66]	Model Driven Architecture (MDA) approach	Generates repetitive and complicated infrastructure code	No tools are included to collect data performance	Claim a large amount of data, but not specific
Zinke *et al.* [67]	Web Server Benchmark named *servload*	Can replay web server logs, tells users the # of requests, sessions, connect time, and error counts; error counts may be connection errors, HTTP codes, or # of timeouts	Web server logs have to be analyzed and log formats can limit this feature Only supports GET requests	No dataset size

Table 8: Measured Data Attributes for Benchmarking

Author(s)	Attributes discussed
Alhamazani *et al.* [56]	Worked on Windows and Linux Monitoring agent used SNMP, HTTP, SIGAR and custom built APIs Benchmarking component measured QoS parameters like network bandwidth, download and upload speeds, and latency Web server: Apache Tomcat Database Server: MySQL
Athanasiades *et al.* [57]	Generated traffic like DARPA 1998 [58] FTP server was the "victim" Used attack injection programs and in-house tools Attack effectiveness measured by number of hung connections at the victim server Percentage of detected hosts were measured (ranged from 25-50%) [58]
Same as above [57]	Simulated users performing Telnet and/or FTP operations [59] Script was used to record and reply the user actions to generate data Some attacks used: password files being sent to remote hosts, password cracking, elevating user access, password dictionary
Ballocca *et al.* [60]	Fully integrated web stressing tool Workloads were extracted from web log files Stressing tool was made up of a script recorder and a load generator
Duan *et al.* [61]	TPC Benchmark H (19 GB) was used as the baseline for this generator The authors created a unique algorithm to generate a benchmark
Neto *et al.* [62]	Measured the trustworthiness of the relationship between the application and the developer

Author(s)	Attributes discussed
	A 3 step process: user sent parameters (i.e., session token) to the server and identified a target resource, server processes code, server sent back output like a form or html text
Neto *et al.* [63]	Raw number of vulnerabilities reported Calibrated number of vulnerabilities reported Normalized raw number of vulnerabilities reported Normalized calibrated number of vulnerabilities reported
Stuckman *et al.* [64]	Benchmark was a collection of modules that were each within a vulnerable application in an environment that let the application run and simulated an attack against the application
Zhang *et al.* [65]	Worked with Internet Explorer, Firefox or Chrome Measured responsiveness of browsers to page loading times and event response times
Zhu *et al.* [66]	Included a core benchmark application, a load testing suite, and performance monitoring tools
Zinke *et al.* [67]	Supported load balancing Did not ignore think times or different user sessions Generated higher workloads than SURGE with similar statistical characteristics through 1 of 3 methods: multiply, peak, or score method

In the next Chapter, we discuss anomaly intrusion detection, and our proposed technique to increase the attack detection rate.

Chapter 3: Anomaly-Based Intrusion Detection System Development

3.1 Overview

This Chapter introduces the anomaly Intrusion Detection System (IDS) development and discusses some relevant work on anomaly IDS development in Section 3.2. Next, in Section 3.3 we introduce the idea of detecting web application attacks by using cross-entropy metrics. Section 3.4 explains our proposed approach to detect such attacks using web application log data based on our previous publication [30]. Finally, Section 3.5 shows how we compared our measures to other accepted measures.

3.2 Related Explanation of Anomaly-Based IDS Development

A recent report from Imperva [68] shows many applications have been targeted to exploit known vulnerabilities such as SQL Injection (SQLI), Remote File Inclusion (RFI), Directory Traversal (DT), and Cross Site Scripting (XSS). SQL injection attacks attempt to provide part of a SQL query in a web request URL (parameter value) where the query part is intended to change the structure of the query to introduce anomalous behaviors [69]. Remote File Inclusion [70] adds arbitrary server-side source files to introduce unwanted application behaviors. A directory traversal attack [71] supplies arbitrary traversing of directory commands in supplied URLs. XSS attacks inject arbitrary JavaScript code and occur when unsanitized inputs are passed within request URLs and are accepted by applications and processed or stored [72]. The vulnerability may arise from plugins the application uses during runtime, for example. One million Wordpress websites have been reported to be vulnerable to SQLI due to leak of secret keys from associated plugins [73]. Similarly, security protocols used by web applications can play the role for successful exploitation (e.g., heart bleed bug exploited to reveal secret information from web servers [74]).

IDS is a popular approach to prevent attacks. IDS can be classified into two types based on the location of deployment: host-based (where one host or computer is protected) and network-based (where a set of hosts connected to a network is protected). This work considers development of a host-based IDS. Depending on the type of detection, an IDS can apply signatures of known attacks.

For example, Snort and Bro are two popular signature-based IDS. Both of these signature-based IDS have currently available signatures to detect web-based attacks such as SQLI and XSS [38, 75]. However, the limitation of an established signature-based IDS is that they are not suitable for detecting new attacks and it is common to see attackers devise new signatures to bypass IDS detection [39, 40]. To address this limitation, anomaly-based IDS have been getting much attention from the research community [5, 12, 15, 39, 40].

An anomaly-based IDS has learning and detection phases. During the learning phase, it learns profiles of normal web requests and then compares with a new request to find the dissimilarity level. If the level exceeds a certain threshold level, an attack is detected. Anomalous IDS has the advantage of detecting new attacks, but at the cost of a high number of incorrect detections. Thus, it is important to explore approaches to reduce the number of warnings. Most of the anomaly-based IDS analyzing web logs from the literature [5, 12, 15, 39, 40] primarily analyzes GET requests, and do not consider POST requests. These POST requests include parameter and value information that should be considered for profiling of requests. Some existing approaches require the knowledge of source code level information to reduce the number of false warning [5, 20, 39, 43]. However, source code may not be accessible while developing an IDS.

In contrast to earlier works, our approach relies on path resources (e.g., a request page in a certain path) and does not need to rely on similar assumptions of the other works. We place the emphasis on web server logs and apply entropy measures to detect anomalous requests. Our work employs cross entropy levels of parameter name, value, and types for profiling and attack detection.

3.3 Detection of Web Application Attacks with Cross-Entropy

Our approach is motivated by earlier works that apply information theoretic measures. For our study, we created the framework of the web anomaly IDS for the learning phase. A browser is used to run deployed web applications and access resource pages with benign input, as illustrated in Figure 1. All GET requests from the browser get logged into the web server log files. For POST, we deploy a suitable browser extension (Firebug for Firefox [76]). We combine the POST request data with GET request data during offline analysis by the anomaly detector. The anomaly detector learns request profiles based on resource paths.

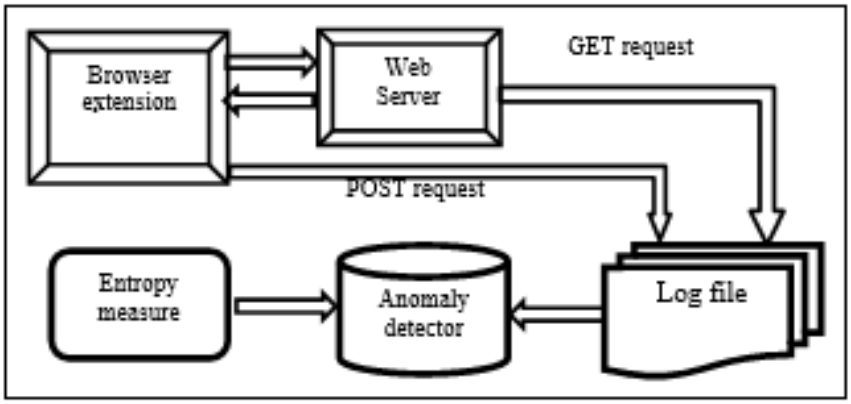

Figure 1: Information-theoretic IDS Framework

Figure 2 shows two example requests (we display part of the log due to space constraint) that we gather during dataset generation by deploying a large scale web application named Joomla [77]. The first request is intended to access the resource /joomla/index.php, and has the list of parameters such as *option*, *view*, *task*, *id*, *timeout* with associated values *com_installer*, *update*, *update.ajax*, *6*, and *3600*. The second request accesses the same resource as the first one. However, it has one parameter (*option*) with the value *com_media*. Therefore, applications may let a user access the resource path with various sets of parameters and values. Solely relying on the parameter sequence would not be enough to detect attacks where the sequence remains the same (e.g., SQL Injection). We also find that the value of id is related to the user account, and the type of the parameter usually remains as numeric with no upper or lower bound.

```
/joomla/index.php?option=com_installer&view=update&task=update.ajax&id=6&skip=700&ti
meout=3600… …
/joomla/index.php?option=com_media …
```

Figure 2: Example of Log Data

For each of the common paths, the anomaly detector profiles three types of information: parameter set, parameter value set and parameter value data type. To account for all of these variations within parameters and their values, our proposed detection approach employs three types of measures. We now present the processing of each request that appears in a log file in Algorithm 1.

Algorithm 1: Processing of URLs from a log file

```
Input: F: Input log file
Output:
  R: set of unique resource path r
  Pr: set of parameter for r
  Tr: type of parameter
  Vr: value set for each parameter

1.  For each line l ∈ F
2.  r = getResourcePath (l)
3.  if  r ∉ R
4.      R = R ∪ r
5.  Pr = getParam (l)
6.  for each pi ∈ Pr
7.      vi = getVal (pi)
8.      ti = getType (vi)
9.      updateValFreq (pi, vi)
10.     updateTypeFreq (pi, ti)
11. UpdateParamFreq(Pr)
```

Line 2 identifies the resource path and adds to the *R* set if not included already (Lines 3-4). Line 5 extracts the list of parameters from the request. For each of the parameters (Lines 6-10), we obtain the value (Line 7) and type of data (Line 8). Then, we update occurrences of the value and type at Lines 9 and 10, respectively. Finally, Line 11 updates parameter occurrence.

3.4 Case Study and Evaluation

We apply entropy as the metric to profile the randomness of parameter occurrences, parameter values, and value types. The entropy (*H*) is calculated using the formula in Equation (i). Here, *Q* is a set of symbols (unique values passed in the parameter), where q_i is the i^{th} element, $p(q_i)$ indicates the occurrence probability of q_i^{th} element.

$$H(Q) = -E[\log P(Q)] = -\Sigma_{q \pounds Q}(q = qi) \log_2 P\ (q=qi) \ldots \ldots (i)$$

Since entropy is useful for a single set of frequency distribution, it cannot be directly applied to compare two distributions (i.e., a new request and a set of earlier observed requests). Instead, we apply a cross-entropy measure between two distributions. Cross entropy (CE) [78, 79] between two distributions p and q is shown in Equation (ii). Here, $p(x_i)$ is the probability of x_ith element's occurrence.

$$CE(p, q) = -\Sigma_i\, p(x_i) * \log_2(q(x_i)) \ldots \ldots \ldots \text{(ii)}$$

In Equation (ii), $p(x_i)$ is the probability of x_ith element from p set, and $q(x_i)$ is the probability of x_ith element from q set. CE becomes minimal when p and q are identical. The CE between two probability distributions measures the average number of bits needed to identify an event from a set of possibilities. We define a threshold level d which, if exceeded, would flag a new request as anomalous. If the CE does not exceed threshold, we consider it normal request. For web anomaly detection, we deploy three measures: cross-entropy of parameter (CEP), cross-entropy of value (CEV), cross-entropy of type (CET). CEP is intended to measure the missing parameter or additional parameters injected as part of attacks or tampering. Equation (iii) shows CE between two parameter sets P_1 and P_2 for a given resource path r. We apply a back off smoothing algorithm [80] to avoid the zero occurrence of any parameter by replacing zero with a very small probability value (as the logarithm of zero probability cannot be computed, otherwise).

$$CEP_r(P_1, P_2) = -\Sigma_i\, P_1(x_i) * \log_2(P_2(x_i)) \ldots \ldots \ldots \text{(iii)}$$

CEV is intended for a given parameter's observed values during the training. It compares the distribution of earlier observed values and the values present in a new request. It can capture any deviation between anomalous attack inputs with an earlier observed normal input. Equation (iv) shows the CEV between V_1 (values observed during profiling) and V_2 (value observed during testing) for a given parameter p.

$$CEV_p(V_1, V_2) = -\Sigma_i\, V_1(x_i) * \log_2(V_2(x_i)) \ldots \ldots \ldots \text{(iv)}$$

CET is intended to reduce false positives as well as increase attack detection. It observes the deviation between data type of the supplied parameter values and a new request parameter value type. Equation (v) shows CET between type set T_1 and T_2 for a given resource path r.

$$CET_r(T_1, T_2) = -\Sigma_i\, T_1(x_i) * \log_2(T_2(x_i)) \ldots \ldots \ldots \text{(v)}$$

These metrics were applied to both of the datasets in our study, the training dataset and the testing dataset. For the training phase, we deploy a large scale PHP Content Management System (Joomla [77]) and perform various functionalities for a four day period. For each day, various types of inputs have been applied to different pages and the logs are stored. We ensured that the data does not contain any malicious input by manually inspecting the logs. We then use the first day of data to build a normal profile of requests and then validate for false positive rates for the subsequent three days of datasets. Table 9 shows the number of GET and POST requests for all four days. We combine all the data, and then choose 25% of the data randomly for building profiles, while keeping the remaining 75% of the data for testing.

Table 9: Good Dataset Characteristics

Request Type	Day 1	Day 2	Day 3	Day 4
GET	1,013	1,556	1,640	1,536
POST	412	517	511	481
Total	1,425	2,073	2,151	2,017

We gather attack input from various sources [81-83] and apply them to the deployed application to generate the attack dataset. Table 10 shows the number of samples we applied in our attack dataset generation. These attack inputs are applied randomly within web requests from the browser.

Table 10: Distribution of Attack Inputs

Attack type	# of samples
SQLI [83]	1000
DT [71, 82]	8
RFI [70]	5
XSS [81]	60
Total	1073

When the IDS generates a warning we call it positive, if it is real, we call it True Positive (TP). If no warning is generated we call it negative, if it is actually not a malicious request, we call it True Negative (TN). If the IDS misses the actual attack detection, we call it False Negative (FN). We follow similar approaches [39, 40] to evaluate the performance of the proposed approach. We use a Receiver Operating Characteristics (ROC) curve to evaluate the performance of the anomaly IDS. It has two measures: True Positive Rate (TPR) on the y axis, and False Positive Rate (FPR) on the x axis. TPR and FPR are defined as follows:

$$TPR = TP/(TP+FN) \quad \ldots\ldots\ldots \text{(viii)}$$

$$FPR=FP/(FP+TN) \quad \ldots\ldots\ldots \text{(ix)}$$

Ideally, we expect IDS to demonstrate a TPR of 100%, while FPR would be 0%. The filtered (normal and attack input free) data is used to evaluate an IDS to produce FPR, whereas the dataset containing only attack requests would be used to obtain TPR. Table 11 shows that the lowest FPR is observed for CEV (0.53%) while the highest FPR is for the CET (3.6%). The lowest TPR observed is 83.66 (CEP) while the highest TPR is obtained for all measures when considering higher threshold levels (d>8). The raw results are presented in Table 11.

Table 11: FPR and TPR for the Proposed Measures

d	CEP		CEV		CET	
	FPR(%)	TPR(%)	FPR(%)	TPR(%)	FPR(%)	TPR(%)
d >2	0.54	83.66	0.53	90.22	1.2	94.21
d >4	1.1	92.45	1.45	94.53	1.45	95.67
d>6	1.26	98.67	1.67	99.33	2.56	98.67
d>8	2.56	100	1.92	100	3.6	100

Data from Table 11 is also below in Figure 3. The Figure shows the ROC curve of performance for the IDS for various distance (*d*) values. In this graph, the x axis shows FPR (%) and the y axis shows TPR (%). We find that a higher detection accuracy is achieved at the cost of a higher FPR. Among CEP, CEV, and CET, the best performance is shown by CEV as it has the lowest FPR.

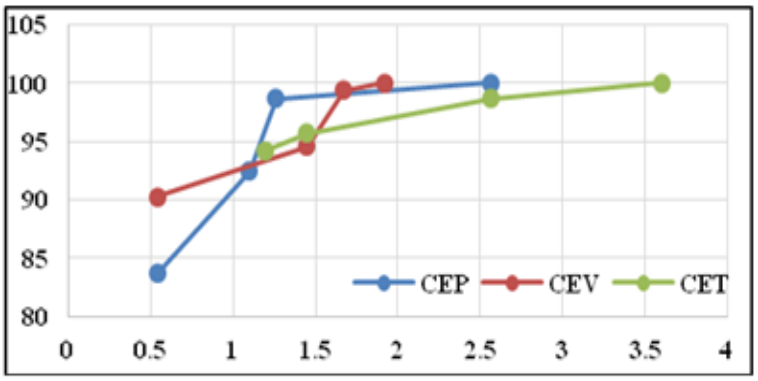

Figure 3: ROC Curve of various Cross Entropy Metrics

3.5 Comparison of Related Metrics

We compare our approach with two earlier proposed approaches: value length and Mahalanobis distance [31]. The length of a parameter value should be limited in size. However, for an attack request, it may be higher. We compute mean and variance of length during training and testing. We measure the deviation based on Chebyshev's inequality (calculating the probability that an attribute would have the observed length) [84]. Let *X* be a random variable with mean μ and standard deviation s >0. Then the Chebyshev Inequality is shown as follows (k>0):

$$P(|X-\mu| >= ks) \leq 1/k^2 \quad \ldots\ldots\ldots \quad \text{(vi)}$$

Mahalanobis Distance (MD) [85] is a metric to compare two statistical distributions. It indicates how close a given distribution is to observed distributions. If we assume the two groups are *x* (x_1, x_2, …x_n) and *y* (y_1, y_2, … y_n). Then MD between x and y is defined as follows:

$$MD(x, y) = sqrt((x-y)^T * S^{-1} * (x-y)) \quad \ldots\ldots\ldots \quad \text{(vii)}$$

Sqrt is the square root operation, $(x-y)^T$ is the transpose of the difference between *x* and *y*, S^{-1} is the inverse of co-variance matrix *S*. We adopt the length measures and consider the length of the parameter name, and the value and consider k=4 for Chebyshev inequality. For MD, we form groups based on parameter, value, and type (*e.g.*, each unique parameter, value, or type is labeled with a numeric value, for example, a data type for number is 1, while string is 2). The following ROC curves (Figures 4, 5 and 6) compare CE measures from our case study with length and MD-based anomaly detection approaches. We find CEP performs better than length and MD.

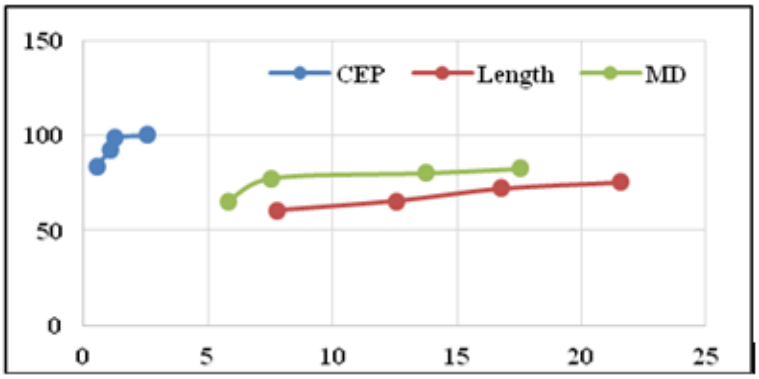

Figure 4: Comparison between CEP, length and MD measures

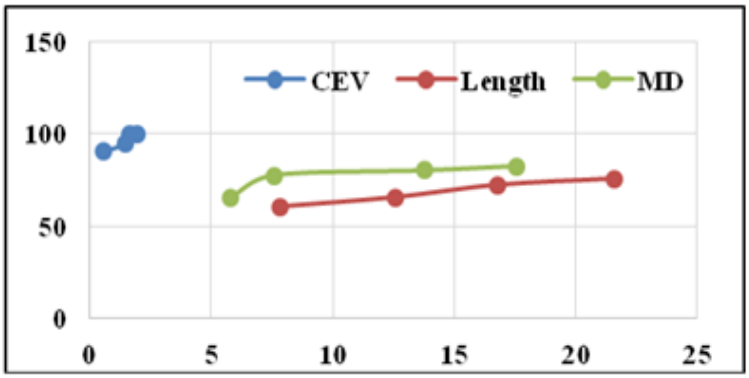

Figure 5: Comparison between CEV, length and MD measures

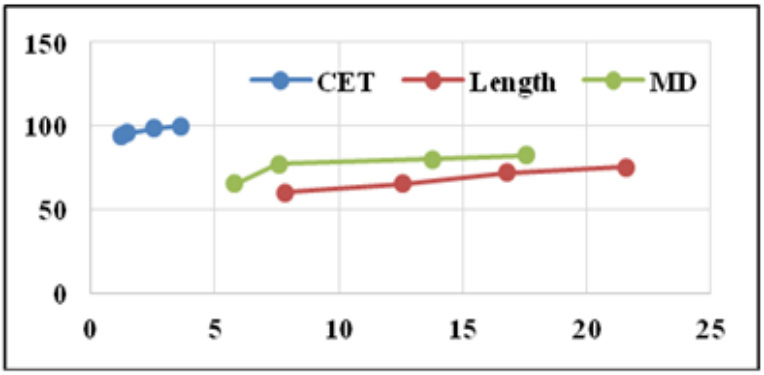

Figure 6: Comparison between CET, length and MD measures

Among the three measures (CEP, CEV, CET), CEV performs best followed by CEP and CET when compared with length and MD measures. CEV accounted for payload diversity more than the other two measures. In all cases, we find that the cross entropy measure performs better than two other existing anomaly detection measures. As anomaly detection measures become more capable of detecting advanced web application attacks, the signature-based approach to attack detection must also be investigated.

The next Chapter discusses how a signature-based IDS can work with a genetic algorithm and improve attack detection. Another case study will also be included.

Chapter 4: Signature-Based Intrusion Detection System Development

4.1 Overview

In this Chapter, we establish our methodology for creating a genetic algorithm that is applicable to signature-based intrusion detection in Section 4.2. In Section 4.3, we describe how datasets are generated and applied. Finally, Section 4.4 presents our case study and results.

Traditional Intrusion Detection Systems (IDSs) use signatures where attacks are defined as a sequence of events to match with network traffic [86]. This approach is accurate as long as the list of attacks is known in advance and signatures are defined before deploying an IDS such as Snort [87] and Bro [88]. There has been little effort to develop signature-based IDS for web applications. Moreover, they rely on regular expressions to detect attacks. For example, a script created to use a PHPIDS [89] allows attack signatures to be expressed using a set of regular expressions. The burden is on the user to keep up with new expressions. To address this limitation of a signature-based IDS, in this paper, we propose to develop a Genetic Algorithm (GA) based IDS. GA-based approaches have gained the attention of the research community in recent years. In a signature-based attack detection approach, the network traffic is monitored and the IDS searches for malicious behaviors that match the known signatures [4]. Any signatures with even minor deviations from the attack descriptions would not set off any security alarms, which may leave a system vulnerable [38]. However, a GA-based approach can address this limitation by generating new signatures from existing signatures. We explored this idea and carried out a case study [42] to exemplify how a GA can improve attack detection rates as well.

4.2 Creation of a Genetic Algorithm

Within this Section, we explain how to create a genetic algorithm, based on previous literature. Generally speaking, a genetic algorithm advances a set of solutions by combining good solutions to craft new ones until the best solution is found. This process is composed of multiple steps [90-93]. In order to generate a genetic algorithm, a general process and set of steps can be followed. The first step is to create an initial population. This population is typically generated in a random

manner and may include as many individuals as preferred, from a few to several thousand. An individual is also called a chromosome in the population. Each chromosome in the initial population is then evaluated for fitness. Next, a new population has to be created. This process consists of repeating the steps that use genetic operators, including selection, crossover and mutation, until the new population is established.

During the selection phase, the main goal is to keep the best individuals in the population and improve the overall population fitness. Two parent chromosomes are selected from the population based on their fitness score value. The better the fitness score is, the more likely that the chromosome will be selected for the population. Crossing over, or the sharing of information, takes place between two parents to create new offspring or children. This occurs in hopes of crossing two chromosomes with a high fitness value that will then create an offspring that has the best traits from each parent chromosome. When mutations occur, there are random changes that happen in individual genes. This increases the diversity among the initial population over multiple generations. All of the new offspring are placed into a new population and serve as the base population for the next iteration of the genetic algorithm. Genetic algorithms are used to create repetitive populations until the optimum solution for the population is found or the population's end condition is reached [91-93]. The genetic algorithm steps are outlined in Figure 7.

1. Begin with a random set of solutions (represented by chromosomes) to form population.
2. Evaluate the fitness of each chromosome in the population.
3. Create new solution by using genetic operators (selection, cross over) by selecting chromosomes having higher fitness level.
4. Apply mutations randomly on newly generated chromosomes.
5. Repeat steps 2-4 until we reach maximum number of iterations, or exceed population size.

Figure 7: Steps of Genetic Algorithm

4.3 Dataset Generation for GA-Based IDS Development and Application

As our goal is to apply the GA to improve a signature-based IDS, we start with an attack dataset that we generated by deploying a large scale PHP web application named Joomla [77]. The applications interacted automatically using scripts, and were provided with malicious inputs. We

collected inputs from the sources such as OWASP [94]. These attack inputs are applied randomly within web requests from a browser. Table 12 displays the number of each type of attack that was distributed into the attack dataset. The Apache web server logs were referenced to manually detect successful attacks. Figure 8 shows an example of log data for a SQL injection attack where an input field (id) has a tautology attack encoded in hexa-decimal format. Similarly, Figure 9 shows an example of log data for an XSS attack where an image source has been supplied with malicious code. Finally, Figure 10 shows an example of log data for a RFI attack, where the *FORMAT* field is included with an include statement pointing to a file source from an attacker-controlled website, followed by an *exit()* command.

Table 12: Distribution of Attack Input Data

Attack type	# of samples
SQLI	1000
RFI	5
XSS	60
Total	1073

"GET
/sqlinj/?id=1%27+or+%271%27+%3D+%271%27%29%29%2F*&Submit=Submit&user_token
=c14e5f424d9f279c19ba507492745d50…

Figure 8: Example Log Data for SQL Injection Attack

"GET
/xss_r/?name=%3CIMG+SRC%3DJaVaScRiPt%3Aalert%28%26quot%3BXSS%26quot%3B%
29%3E&user_token=f37e5a82a994725092fd3155bb8cffba…

Figure 9: Example Log Data for XSS Attack

“GET /?FORMAT={${include("http://www.verybadwebsite.com/hacker.txt")}}{${exit()}}…

Figure 10: Example Log Data for RFI Attack

Step 1: The GA accepts a set of chromosomes as input, and provides another set of chromosomes as outputs after a certain number of iterations while following the fitness evaluation, cross over and mutations. For our contribution, we first convert each of the GET requests to a chromosome, which is a bit string representation. Figure 11 shows an example representation of a chromosome for SQL injection attack (based on the log in Figure 8 above). Here, we have three blocks of

information that include total number of SQL keywords (two of them include OR, =), presence of an encoded character (1=Yes, 0=No), number of input fields that have SQL keywords (one field here has a SQL keyword). The last block is the decision block, which represents attack type, expressed in four bits. In the literature, there are six common types of SQL injection attacks. Hence, we reserve three bits to express various types of attacks.

# of SQL keywords	Presence of encoded character	# of fields with SQL keyword	Attack type
010	1	001	0001

Figure 11: Example of a Chromosome (C1) for SQL Injection

Since each chromosome length should be same across different types of attacks, we define chromosomes for XSS and RFI using three blocks of bit representation, followed by attack type information. Figure 12 shows an example of chromosome for XSS based on the XSS log data described earlier. Here, three script/html words are present (<script>, <img>, </script>), the input is encoded, and one field has XSS keywords. Figure 13 shows an example of RFI chromosome based on the RFI log data presented, where the attack payload includes one URL, and it was not encoded. There is only one command included for this situation. Once each attack has the appropriate binary string, Step 2 can begin.

# of script/html keywords	Presence of encoded character	# of fields with XSS keyword	Attack type
011	1	001	0111

Figure 12: Example of a Chromosome (C2) for XSS Attack

# of URLs	Encoded	# of commands	Attack type
001	0	001	1100

Figure 13: Example of a Chromosome (C3) for RFI Attack

Steps 2 and 3: We define two fitness functions (FF2, FF3) to evaluate chromosome x as follows (Equation (i) to (iii)).

FF1(x): # of attacks detected by x in training dataset/total # of attacks in training data … … (i)
FF2(x): # of attacks detected by x in testing dataset/total # of attacks in testing data … … (ii)
FF3(x): FF1(x) + FF2(x) … … … (iii)

For example, we can apply FF3 to evaluate the fitness value of a SQLI chromosome (C1). If we assume that C1 matches with 1 attack input out of 100 samples, and results in no false positive warning, then its fitness value is 0.01. When we are evaluating the fitness function for a chromosome, we are considering the entire dataset including training and testing. We compare bit level representation of chromosomes from training or testing data to determine how many attacks are detected. The chromosomes are crossed over based on fitness level. We apply one point cross over for this case study. For instance, if we decide to cross over between C1 and C2, the before and after results would mimic those added below. If we assume in C1 (after cross over), the fourth bit gets mutated from 1 to 0, then we have a new signature (01000010111) for XSS, where the attack payload is not encoded. This cross over process is illustrated below as well.

Before cross over (C1, C2):

C1	010	1	00 1	0001
C2	011	1	00 1	0111

After cross over (C1, C2):

C1	010	1	00 1	0111
C2	011	1	00 1	0001

Step 4: Our proposed framework allows for the web log data to be converted to chromosomes, as demonstrated. The GA is then applied to generate more chromosomes which act as new attack signatures until the solution is achieved. The generic framework we applied is shown below in Figure 14.

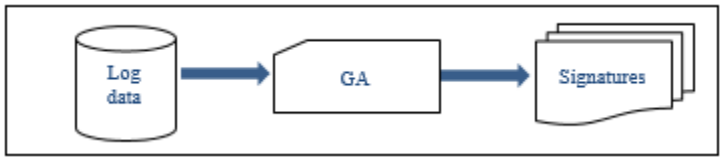

Figure 14: GA-Based IDS Framework

4.4 Case Study and Evaluation

In this Section, we evaluate our approach in multiple ways. First, the GA parameters were evaluated. We divide our attack dataset (web log files) into two parts: training dataset (30%) and testing dataset (70%). This division is based on some earlier literature work that also developed a GA-Based classifier (see Table 3). For each of the training dataset logs, we convert GET or POST

requests into chromosome representations by editing and implementing a number of open source PHP class files [95]. Figure 15 shows a screenshot of the application output used while evaluating this approach on a Windows Computer.

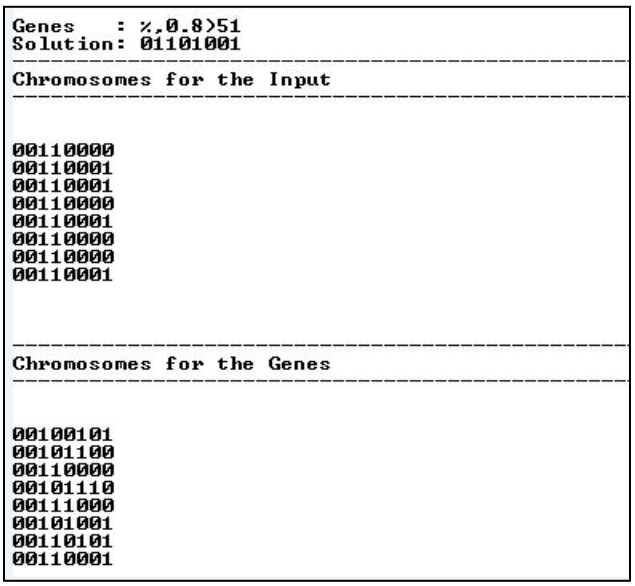

Figure 15: Screenshot of Results from GA-Based IDS

Based on the variables involved, such as the mutation rates, fitness functions and cross overs, multiple scenarios were carried out to evaluate our approach. These results are depicted in this Section. Figure 16 shows attack detection accuracy for various population sizes while using FF2 as the fitness function and keeping the mutation rate at 0.5. We can observe that the higher the selection rate for a chromosome to cross over, the better accuracy for attack detection capability we achieve. Figure 17 shows the attack detection accuracy for various population sizes while using FF3 as the fitness function and keeping the mutation rate at 0.7. Each chromosome had varying selection rates, which makes it easy to see that attacks are detected with more accuracy as the population size increases and the selection rate increases.

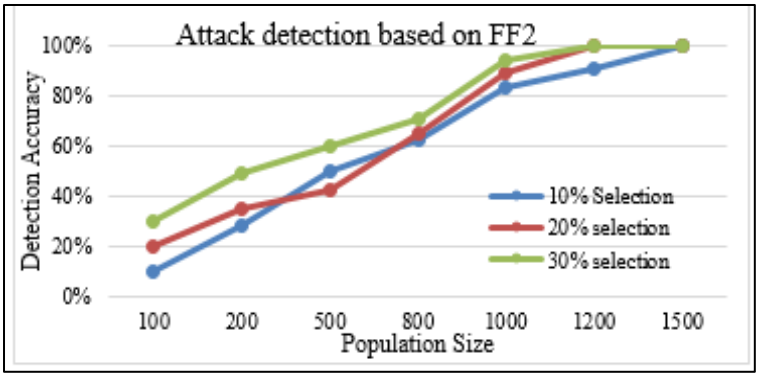

Figure 16: Attack Detection Accuracy vs. Population Size (FF2, mutation rate=0.5)

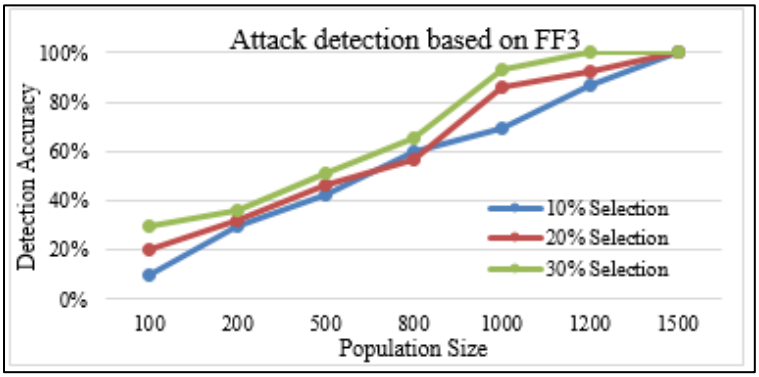

Figure 17: Attack Detection Accuracy vs. Population Size (FF3, mutation rate=0.7)

Figure 18 demonstrates that as mutation rate changes, so does the accuracy of attack detection. Here, the selection rate was set at 10% and FF2 was used as the fitness function. A higher mutation rate implies that the attacks can be detected with more accuracy and within a smaller population. Figure 19 illustrates that as mutation rate increases, so does the attack detection accuracy. For this situation, FF3 was used as the fitness function and the selection rate was set at 20%. A higher mutation rate shows that the attacks can be detected with more accuracy and in a smaller population.

Figure 18: Attack Detection Accuracy vs. Mutation Rate (FF2, selection rate=10%)

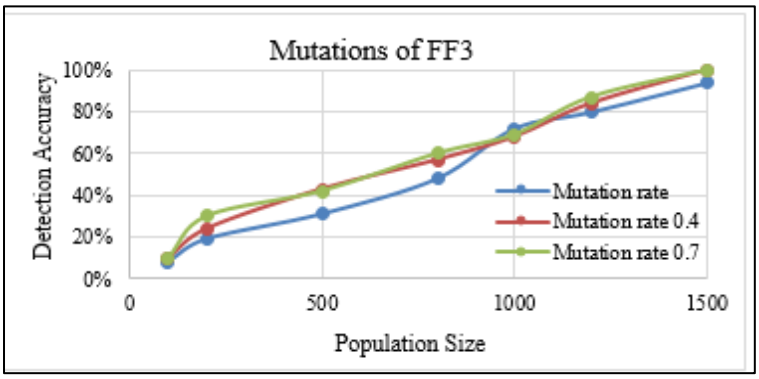

Figure 19: Attack Detection Accuracy vs. Mutation Rate (FF3, selection rate=20%)

Despite obtaining the expected results, we continued with our case study a step further. We compared the GA-Based IDS with PHPIDS [89], which is a popular open source web application level attack detector. PHPIDS relies on a set of regular expressions in a configuration file to detect known signatures. Such regular expressions are the signatures of each attack under study. Therefore, by testing a known attack dataset, we compare GA with PHPIDS. Figures 20-22 below illustrate samples of the regular expressions provided in PHPIDS for XSS, SQLI and RFI. In Figure 20, any script code can be detected that is pre or post pended with an arbitrary string ("(?:\<scri)|(<\w+:\w+)]"). It can also detect data having possible scripts (other than tag). Similarly, Figure 21 shows an example regular expression that is supposed to detect SQL injection attack inputs having specific keywords (e.g., exists, type). Figure 22 shows an example of a regular expression for remote file inclusion that looks for a php file.

```
<![CDATA[(?:\<\w*:?\s(?:[^\>]*)t(?!rong)) | (?:\<scri)|(<\w+:\w+)]]>
```

Figure 20: Regular Expression for Cross-Site Scripting

```
<![CDATA[(?:\[\$(?:ne|eq|lte?|gte?|n?in|mod|all|size|exists|type|slice|or)\])]]>
```

Figure 21: Regular Expression for SQL Injection

```
<![CDATA[(?:@[\w]+\s*\()|(?:]\s*\(\s*["!]\s*\w)|(?:<[?%](?:php)?.*(?:[?%]>)?)|(?:;[\s\w|]*\$\w
+\s*=)|(?:\$\w+\s*=(?:(?:\s*\$?\w+\s*[(;])|\s*".*"))|(?:;\s*\{\W*\w+\s*\()]]>
```

Figure 22: Remote File Inclusion Example

To compare the GA-Based IDS with PHPIDS, we consider the population set generated by GA and then convert back to string representation of attack inputs. We then pass these inputs to PHPIDS and see if all of them can be detected by PHPIDS. Figures 23 and 24 show example performances of the PHPIDS. In both cases, as the GA generated a greater population (of signatures), the PHPIDS failed to detect all of them. Thus, a GA can be complementary to a PHPIDS to detect new attacks using a signature-based approach.

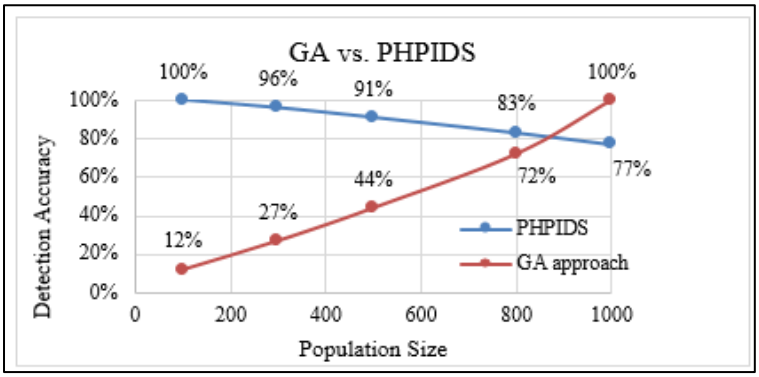

Figure 23: Performance of PHPIDS for GA Generated Signatures (cross over rate=10%, mutation rate=0.5)

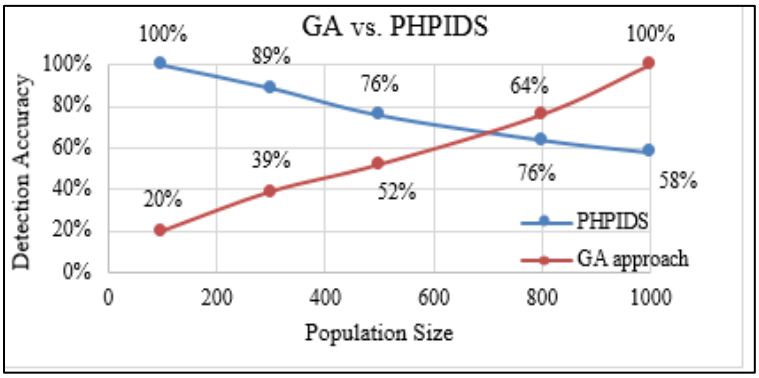

Figure 24: Performance of PHPIDS for GA Generated Signatures (cross over rate=20%, mutation rate=0.7)

The initial results find that the use of a GA is promising and can act as complementary to other existing signature-based IDS approaches. When the population size of chromosomes is increased (representing rules), the better the GA achieves the capability of detecting new attacks. Further, having increased selection rate and mutation rate, we can generate new attack detection rules that can address the limitation of traditional signature-based IDSs, such as the PHPIDS.

In the next Chapter, a broad summary of benchmark evaluation is presented and applied to a selection of the previous log files.

Chapter 5: Benchmark for Evaluation

5.1 Overview

Technologists and computer scientists need a set of standards, called a benchmark, to evaluate the datasets they handle on a daily basis that may be made up of log files generated by user actions, web applications, and login attempts. These types of datasets may vary in size, content, purpose, and many other characteristics. However, all of the datasets should be able to be evaluated by the same benchmark. For the purposes of this chapter, we consider a benchmark to be a set of data obtained from real world applications and that can be used to measure performance of web application attack detection tools. The benchmark could be used to detect how resistant an application is towards detecting attacks and performance changes [96, 97]. Benchmarking can be carried over into almost any domain of technology; however, this chapter focuses on developing a benchmark for detecting attacks against web applications.

5.2 Description of a Benchmark

A benchmark can be applied to nearly any situation in the technology field. Based on the literature dating back to 1997 and as far forward as 2015, this section will explain why the need for one collective benchmark is a relevant issue. Strictly by definition, any attack aimed at the application layer of the Open Systems Interconnection (OSI) model is a web application attack [98]. These application-layer attacks often involve a web server and/or a database server, depending on the specific type of attack. To exemplify this, consider XYZ-WebTech, a technology company located within the United States. At this company, a benchmark would be needed that could be applied to web application security testing. However, this company would also need a separate benchmark to apply to web service performance monitoring. The body of literature surrounding benchmarking discusses the lack of one universal benchmark to detect web application attacks. Currently, the fact that there is no benchmark for web application attack detection has led authors to develop their own benchmarks for their specific datasets [56, 57, 58, 60, 62]. A discussion of why the disadvantages of each approach outweigh their respective advantages is still to come. In addition, the reasons authors in the literature attempted to establish their own benchmarks will be explained in more detail in the next section.

5.3 Motivations for an Application Layer Benchmark

Once a benchmark is created, such as the MIT Lincoln Lab dataset for detecting network-layer attacks from 1998 [99], the attackers find new avenues to explore. This process is nearly cyclic in nature since attackers are continuously looking for different ways to access important information, such as web server or database server logs. Such logs may hold highly sensitive information including company passwords, client credit card data or employee payroll information, for instance. Measures such as intrusion detection systems are in place to prevent such actions, but no benchmark is available to evaluate the efficacy of the detection systems. Relevant characteristics of the numerous benchmarks that independent studies have instituted must be considered when developing a benchmark for detecting application layer attacks.

Among the benchmarks individual authors have proposed in the literature, all authors agree that there is no current benchmark for evaluating datasets for web application layer attack detection. For instance, for authors working with cloud-based datasets, it is stated that existing monitoring frameworks such as Amazon CloudWatch, do not monitor all of an application's components [56]. Since the release of the DARPA dataset in 1998 and similar datasets in surrounding years, no updated datasets have been published as a benchmark. The datasets published in the 1990s are irrelevant now. Specifically for this chapter, the DARPA dataset [99] or the KDD Cup dataset [100] are not only outdated, but also focused on network layer level attacks rather than the web application layer. Two examples of the multiple network layer attacks are depicted in the Figures

below. Figure 25 shows the steps involved in an Apache2 attack, and Figure 26 illustrates how a User-to-Root attack would occur.

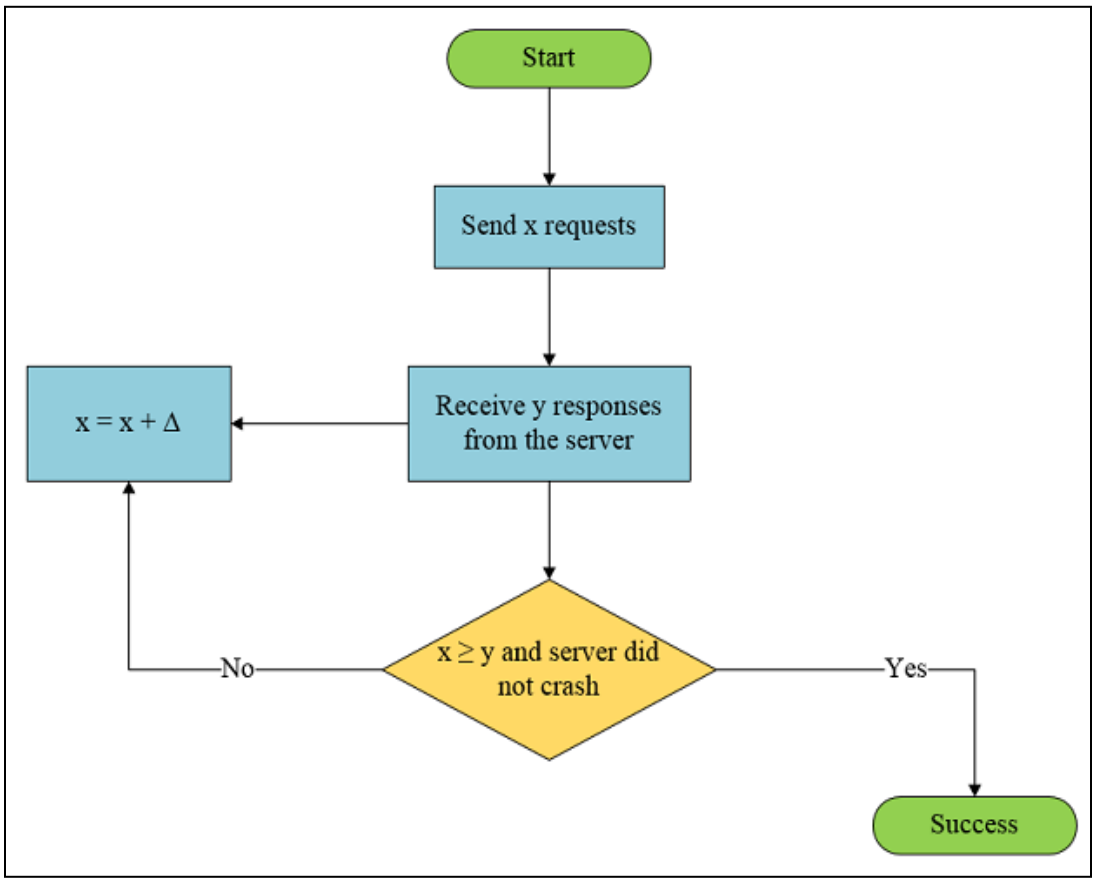

Figure 25: An Apahce2 Attack Illustration

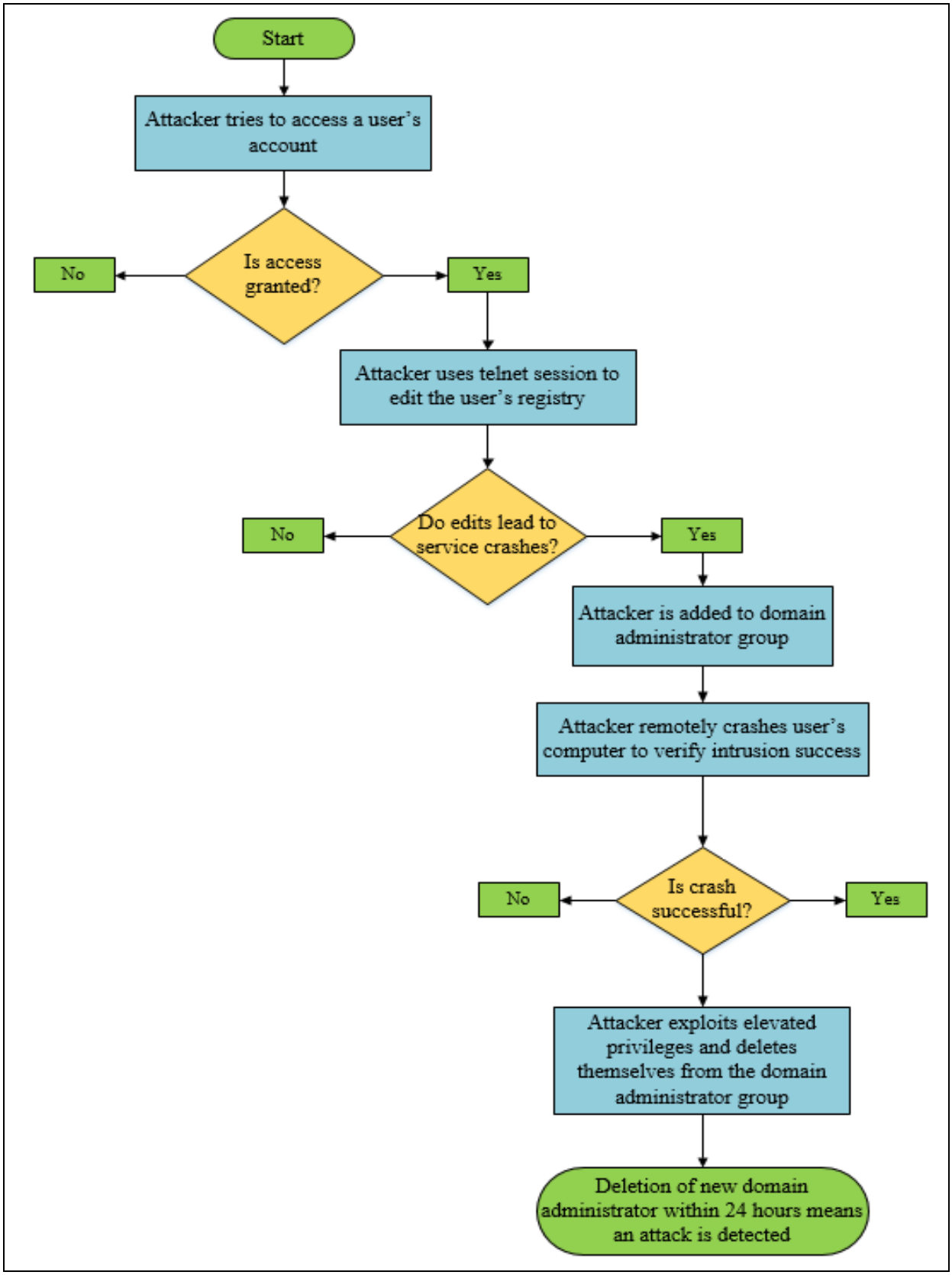

Figure 26: User-to-Root Attack Diagram

Attacks such as the previous examples cause issues at the network layer, but are not the same attacks leading to havoc at the application layer. Due to this situation, authors have transitioned towards crafting their own benchmark in a controlled environment [57]. Common characteristics of benchmarks across the literature included a consistent focus on application-layer attack detection by training and testing their datasets. During a training phase, the researchers would use normal day-to-day logs generated by user activities and regular business actions that were simulated on a web server and/or database server. Testing datasets were often the datasets that contained malicious data that researchers placed into the normal data. This was done so that the researchers, regardless of their objectives, could easily observe if the attacks were detected by their benchmark application or not. Similarly, our case studies used training and testing data as well. The observed dissimilarities can demonstrate what should be the best suitable application and potential scope for the benchmark. For example, a few of the benchmarks proposed to detect application-layer attacks, or data security breaches, were heavily reliant on complex coding schemes [62] and using JAVA platforms posed issues as well [56, 63].

5.4 Generating Data and Setting up a Test Environment

An environment that is used to generate data has to be very controlled to ensure that no attacks can be introduced into the setting. For this chapter, the benchmark datasets were generated through the use of a virtual machine cluster using VMware Workstation 12 on the host machine [101]. The host machine is a 64-bit standalone server running an AMD FX 8350 eight core processor at 4Ghz, contains 32 GB of physical memory and 64 GB of virtual memory. The operating system on the host machine is Windows 7 Ultimate. Figure 27 is a diagram showing the environment that was used for this data generation process. Having a virtual setting for the benchmark generation provides an additional layer of defense against any out-of-network traffic. Thus, the resulting web application traffic was all generated by the user actions and the benchmark was known to be free of application layer attack attempts.

Some of the virtual environments had a Windows 7 operating system while others ran on a Linux operating system. This variation in the operating system was utilized to make sure the benchmark was applicable to machines with Windows and Linux environments. All security features, such as antivirus and firewalls, were deactivated to allow for the generation of attack data. Each virtual

environment had the same baseline software installed including Microsoft Office and Notepad++ and Google Chrome served as the default web browser.

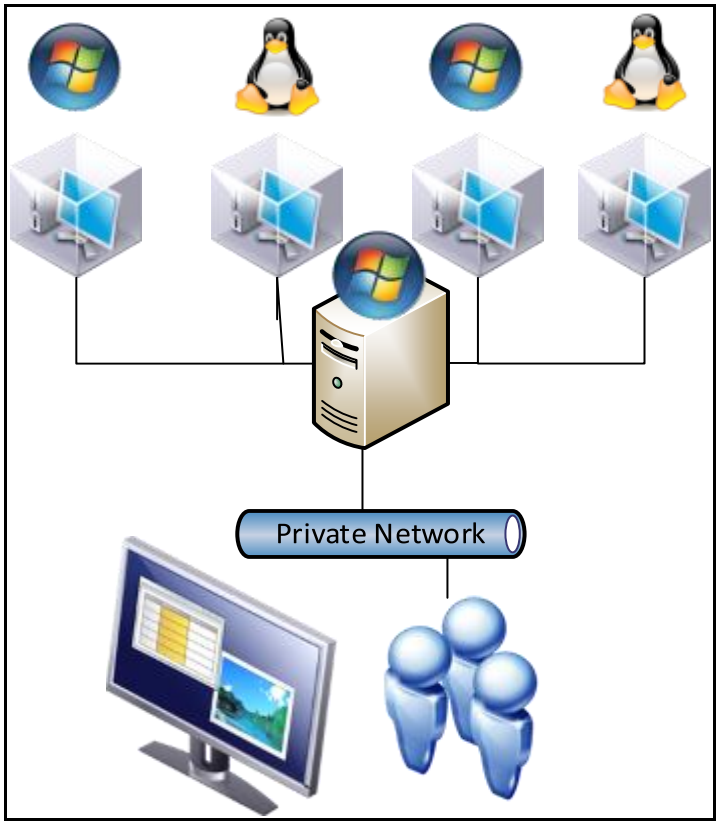

Figure 27: The Environment for Data Generation

In addition to the baseline software, a popular open source software named XAMPP was added to each virtual environment. This web application works across operating systems and incorporates Apache, MySQL, PHP and PERL. To generate datasets in the controlled environment, Apache was used as the web server and MySQL was used as the database management system. Implementation of the PHP and PERL features of the application were beyond the scope of this thesis work. Both the Apache web server and the MySQL database management system kept logs of information about what was occurring on the system while XAMPP was running. A total of five web applications were installed on the virtual machine cluster, and the user was only accessing one web application at a time. The web applications that were installed on the virtual cluster were all open source applications and already integrated with the XAMPP software. These applications had various functions, which led to the creation of different types of data over the course of four days for the final benchmarking dataset.

5.5 Evaluating the Benchmark

Multiple web applications were launched in the virtual environments, but the initial benchmark was used to evaluate the detection capabilities of the IDS. This data, generated solely from the content management system application Joomla, was comprised of basic user action logs and utilized to merge with known attack data. Attack data was generated by executing and re-executing many known web application layer attacks. The specific virtual environment for this case study was running a Windows 7 operating system without any antivirus, firewall, or other known security features. A lack of security software in a virtual, otherwise completely controlled, setting is required to lower the number of false positive results obtained during the evaluation process.

An anomaly-based IDS is used in this case study to determine which data in the combined datasets should be flagged as potential attacks against a web application. As described earlier in Chapter 3, entropy is a measure that falls under the category of information theoretic metrics. The entropy level of normalized traffic from the Content Management System was represented by a limit (X) to create a cut-off point. Any traffic with an entropy level above the pre-determined limit was considered anomalous and thus an attack against the Content Management System web application. A data security breach would be an example of an outcome from this type of attack. For the purposes of comparing the probability distributions of the normal traffic and attack traffic to one another, a cross-entropy measure was used. The normal data is also referred to as the learned profile because this data was utilized to establish the benchmark. In contrast, the attack dataset is also called the new requests because such data was not introduced to the benchmark prior to the evaluation step.

To test a new request, the cross entropy between the learned profile and the new request is measured. A high level of cross entropy is considered malicious for this study. Based on the characteristics of the preliminary dataset from the Content Management System application, three cross entropy measures from Chapter 3 were employed: cross entropy for parameter (CEP), cross entropy for value (CEV) and cross entropy for value data type (CET). The preliminary results of this case study showed that the lowest false positive rate (FPR) was observed for CEV while the highest FPR was for the CET. False positive rates ranged from less than 1% to about 4%. The lowest true positive rate (TPR) observed was for the CEP equating to almost 84 % of those results.

Additionally, the highest TPR was obtained for all measures when considering higher threshold levels. Given that previous literature states the average anomaly detection IDS has a FPR of 8.4% [41, 49], we can report that this benchmark allowed us to reach our objective of lowering the FPR to under 4%. Based on the preliminary evidence, cross entropy was a valid metric for the benchmark datasets.

Since the conclusions contain only results from the case study using the Content Management System logs, these case study results cannot be generalized across all web applications for attack detection approaches. To empirically evaluate the entire set of log files from all five of the deployed web applications, another case study would have to be carried out, allowing the benchmark to be applied to all of the log files that were generated after the final submission of the case study [30]. If additional web application log files are included in benchmark evaluation, the empirical conclusion would be further supported and extended to multiple applications based on the initial findings in the study. Figure 28 shows the set up for the continuation of the case study with examples of open source PHP applications.

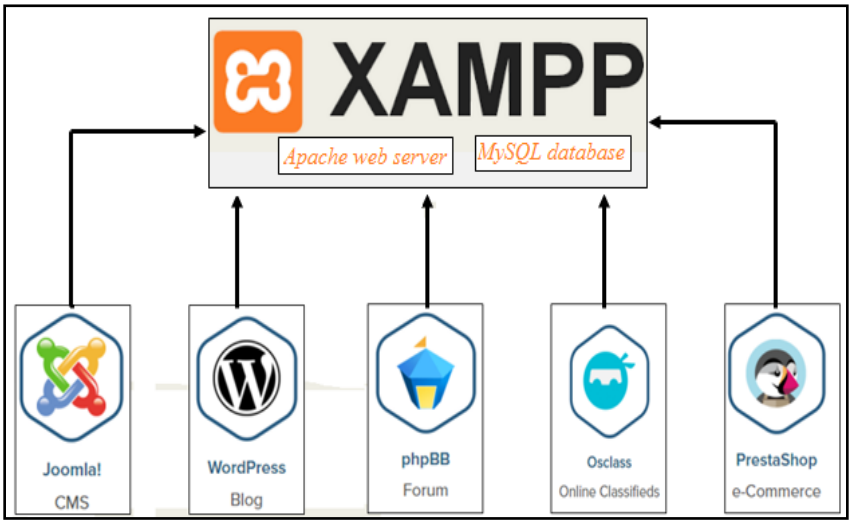

Figure 28: Web Applications Deployed in Apache and Stored in MySQL

The following Chapter demonstrates the output from Apache logs for these web applications and additional tools used during the course of this work.

Chapter 6: Implementation and Testing

6.1 Anomaly Detection

For the context of this section of the Chapter, it is important to revisit Chapter 3 and review the case study we conducted. Our approach was used to detect the occurrence of specific attacks including SQLI, RFI, DT and XSS. We used measures of cross entropy to detect anomalies in the dataset. Cross entropy was calculated for parameters, values and value types, as illustrated in Figures 29 and 30. The resource path in the first line of Figure 29 would be */joomla/index.php*. The parameter of this resource path is *controller* followed by the value *config.display* and the value type is *config*. Similarly, in Figure 30, the last GET request would be parsed into its parameters (*post, action, message)* and values (*207, edit, 6*). Figure 30 would not have any value types.

```
"GET /joomla/index.php/site-settings?controller=config.display.config HTTP/1.1" 200 18484
"GET /joomla/index.php/component/tags/tag/2-joomla HTTP/1.1" 200 12374
"GET /joomla/index.php HTTP/1.1" 200 13484
"GET /joomla/index.php/your-profile HTTP/1.1" 200 43279
```

Figure 29: Content Management System Log Entries

```
"GET /wordpress/wp-content/uploads/2016/04/oooo1.jpg HTTP/1.1" 304
"POST /wordpress/ HTTP/1.1" 200 37339
"POST /wordpress/wp-cron.php?doing_wp_cron=1460232961.9166638851165771484375 HTTP/1.0" 200 - "-"
"GET /wordpress/wp-admin/post.php?post=207&action=edit&message=6 HTTP/1.1" 200 121820
```

Figure 30: Blogging Platform Log Entries

Figure 29 shows an example of normal traffic as GET entries from the Apache server's log file generated by the content management system application named Joomla. Figure 30 shows both GET and POST requests. The last GET request is posting a message by a user. The POST examples are related to the image that the user uploaded to the blog application. Each log entry, including training and testing data, was parsed in this manner through the use of the algorithm we described earlier. Once each entry was broken down into its parameters, values and types, the previously described cross entropy equations were applied to the data. Overall, the cross entropy of the data parameters showed the lowest performance, with a true positive rate of 83.66%. In contrast, the cross entropy of data values was the best measure in the study, producing a false positive rate below 1% (see Table 11). The implementation and testing of this approach outperformed both measures from previous literature as well.

6.2 Signature Detection

The subject matter and discussion in this section is a reflection of the study presented in Chapter 4. We completed the signature-based approach case study with the same datasets as those used in our anomaly case study, but did not have any DT attacks in the attack dataset during that time. Our technique for detecting signatures employed a genetic algorithm. For each log entry, we applied the algorithm and transformed the data into a binary string. These binary strings, or chromosomes, were used to identify the signature of the attacks. A screenshot of the genetic algorithm output was presented earlier (see Figure 15). We utilized methods explained in Chapter 4, such as changing cross over rates and mutation rates, to increase the variability of the attack signatures. Through the use of our genetic algorithm to create new attack signatures, our detection rates were increased relative to other literature in the field.

All of the logs from the Content Management System were used to make new signatures. However, we deployed many applications after the initial case study, as mentioned. Shortened samples of normal traffic from the Apache web server logs of the other applications are presented in the Figures below to illustrate the variability across the final dataset. Specifically, the deployed applications consisted of a content management system, a blogging platform, a bulletin board system, a classifieds marketplace, and an e-commerce platform. In Figure 31, the POST log shows that an entry was deleted from the bulletin board while one of the GET requests demonstrates the user browsing the forum. Figure 32 provides evidence of the user conducting searches and adding content to the classifieds application. Finally, Figure 33 illustrates the user browsing the various pages of the e-commerce application by generating GET requests.

```
"POST /phpbb/posting.php?f=2&mode=soft_delete&p=1&sid=6715823b48f5d89365f2872263781c2d&confirm_key=27WZTFO4L3 HTTP/1.1" 200 11032
"GET /phpbb/index.php?sid=6715823b48f5d89365f2872263781c2d HTTP/1.1" 200 14602
"GET /phpbb/viewforum.php?f=2&sid=6715823b48f5d89365f2872263781c2d HTTP/1.1" 200 20594
"GET /phpbb/styles/prosilver/theme/images/icon_topic_deleted.png HTTP/1.1" 200 1205
```

Figure 31: Bulletin Board System Log Entries

```
"GET /osclass/index.php?page=search&sCategory=83 HTTP/1.1" 200 16357
"GET /osclass/oc-content/uploads/0/28_thumbnail.jpg HTTP/1.1" 200 8744
"POST /osclass/ HTTP/1.1" 200 - "Osclass (v.361)" "-"
"GET /osclass/index.php?page=item&action=item_add HTTP/1.1" 200 40543
```

Figure 32: Classifieds Marketplace Log Entries

```
"GET /prestashop/img/logo_stores.png HTTP/1.1" 200 2962
"GET /prestashop/contact-us HTTP/1.1" 200 18128
"GET /prestashop/themes/default-bootstrap/cache/v_0_e7832ae050fe83ee3aad22771c27a164_all.css HTTP/1.1" 200 224542
"GET /prestashop/modules/blockbanner/img/sale70.png HTTP/1.1" 200 13212
```

Figure 33: E-commerce Platform Log Entries

Attack data was made up of the aforementioned attacks that we successfully simulated. Figure 34 illustrates a small sample of SQL injection attacks that we purposefully introduced to the IDS during testing. Here, the main log information would be the SQL keywords (select-from-where, etc.) and database changes that can be seen in Apache logs. The provided Figure demonstrates an attacker sending a query to discover which users are super users based on the privilege type the user has in the user privilege table.

```
"GET
/sqlinj/?id=SELECT+grantee%2C+privilege_type%2C+is_grantable+FROM+information_sche
ma.user_privileges+WHERE+privilege_type+%3D+%E2%80%98SUPER%E2%80%99%3BSE
LECT+host%2C+user+FROM+mysql.user+WHERE+Super_priv+%3D+%E2%80%98Y%E2%
80%99%3B+%23+priv&Submit=Submit&user_token=7c075e53d4f53712cd11fd828dc78aa2
HTTP/1.1" 200 5219
```

Figure 34: Malicious Data Composed of SQL Injections

In the next Chapter, we present the dissemination of our research results thus far into the work.

Chapter 7: Dissemination of Research Results

This chapter illustrates the dissemination of results, such as published conference papers and other works completed for this thesis. Below we list the title, abstract, and venue for each dissemination.

7.1 *Information Theoretic Anomaly Detection Framework for Web Applications*

Robert Bronte, Hossain Shahriar and Hisham Haddad. Conference Proceedings. *Proc. of 40th IEEE International Computer and Software Application* (COMPSAC), Atlanta, GA June 10-14, 2016, pp. 394-399. Doi: 10.1109/COMPSAC.2016.139 ***[30]***

Abstract

Intrusion Detection System (IDS) is a popular approach to detect attacks in web applications. Signature-based IDS may not know all possible attack signatures in advance, thus a complementary anomaly-based IDS is deployed to and detect new attacks. In this paper, we propose an anomaly detection approach that utilizes three measures: cross entropy for parameter, value, and data type. The measures are intended to compare the deviation between learned request profiles and a new web request. To reduce the number of incorrect detections, we consider requests accessing similar resource paths to learn entropy parameter's value. We evaluate this approach by generating log datasets from a large scale web application (Content Management System). The initial results show that the proposed approach can detect all malicious web requests and demonstrate lower false positive rates. It outperformed when comparing two other approaches: length of parameter value and Mahalanobis Distance.

7.2 *A Signature-Based Intrusion Detection System for Web Applications based on Genetic Algorithms*

Robert Bronte, Hossain Shahriar and Hisham Haddad. Conference Proceedings. *Proceedings of the 9th International Conference on Security of Information and Networks* (SIN '16), July 2016, NJ, USA, ACM, pp. 32-39. Doi:10.1145/2947626.2951964 ***[43]***

Abstract

Web application attacks are an extreme threat to the world's information technology infrastructure. A web application is generally defined as a client-server software application where the client uses a user interface within a web browser. Most users are familiar with web application attacks. For instance, a user may have received a link in an email that led the user to a malicious website. The most widely accepted solution to this threat is to deploy an Intrusion Detection System (IDS). Such a system currently relies on signatures of the *predefined* set of events matching with attacks. Issues still arise as all possible attack signatures may not be defined before

deploying an IDS. Attack events may not fit with the pre-defined signatures. Thus, there is a need to detect new types of attacks with a mutated signature-based detection approach. Most traditional literature works describe signature-based IDSs for application layer attacks, but several works mention that not all attacks can be detected. It is well known that many security threats can be related to software or application development and design or implementation flaws. Given that fact, this work expands a new method for signature-based web application layer attack detection. We apply a genetic algorithm to analyze web server and database logs and the log entries. The work contributes to the development of a mutated signature detection framework. The initial results show that the suggested approach can detect specific application layer attacks such as Cross-Site Scripting, SQL Injection and Remote File Inclusion attacks.

7.3 *Benchmark for Empirical Evaluation of Web Application Anomaly Detectors*

Robert Bronte, Hossain Shahriar and Hisham Haddad. Book Chapter. Benchmark for Empirical Evaluation of Web Application Anomaly Detectors. Empirical Research for Software Security: Foundations and Experience *(under review)*, [Editors: L. Othmane, M. Jaatun and E. Weippl]. ***[101]***

Abstract

Designing a benchmark that is applicable to a wide range of datasets is not a simple task. Before any benchmark can be established, the training and testing data has to be generated. The generation of a dataset is accomplished in controlled environments in most studies [65-67]. Within such datasets, the data consists of typical user actions on web applications that represent normal traffic on the network. By normal data, we mean that no attacks are occurring on the network during the data generation process. An attack-free environment is crucial in order to generate normal data, which creates the need for controlled study environments. Some examples of common user actions on web applications that are logged by the web server and/or database server are login attempts, edits to a Table in an existing database, uploading files or images, and updating user profiles to name a few. Normal datasets serve as referent or baseline datasets to evaluate the benchmark. Once a benchmark is established based on normal traffic, the benchmark can then be applied to data in a controlled environment with known attack inputs. This process allows individuals to determine the number and types of attacks the benchmark can detect. The detection of data security breaches in particular may rely on detecting certain types of application-layer attacks. For instance, the emerging threats against web applications are generally methods used to exploit vulnerabilities that attackers target in the datasets of web applications.

Chapter 8: Conclusions and Future Work

8.1 Conclusions

In this thesis, we proposed an IDS framework based on an information theory metric to detect web application attacks. The focus was on detecting four types of web-based attacks SQLI, XSS, RFI, and DT. We proposed three cross entropy measures on parameter, value, and data type. We evaluated our approach with a generated dataset by deploying a large scale content management web application. The evaluation suggests that the proposed measures can be applied to detect all the introduced attacks with a 100% detection rate, while the lowest false positive rate is below 1%. Further, all three measures can perform better than two related detection approaches we included: length of value and Mahalanobis Distance.

Next, we contributed to the development of a GA-Based IDS where a set of web logs were converted to chromosomes and new attack signatures were generated. The approach addresses the current limitations of signature-based IDS. A limited number of known attack signatures poses a problem, as does the lack of variation of attack signatures. This may cause a system to miss an attack in the traffic log. We evaluated our approach with a generated attack dataset from a large scale PHP application. The results find that the use of a GA is successful and can act as a complement to other existing signature-based IDS approaches. The larger the population size of chromosomes becomes (representing numerous rules), the better the GA achieves the capability to detect new attacks. Further, by increasing the selection rate and mutation rate, we can generate new attack detection rules that can address the limitation of traditional signature-based IDS such as the referenced PHPIDS.

Finally, we discussed how a benchmark is defined, the advantages and disadvantages of the existing benchmarks and the data attributes that previous authors have analyzed within Chapter 5. The metrics and characteristics previously applied to other datasets for application-layer attack detection were explained. Additionally, an in-depth description of the host environment was provided and samples of log files that would be used to evaluate the benchmark were included. We explained entropy and cross entropy measures taken from information theory concepts and

how those metrics were applied to the present dataset with a case study. The methodology of the case study was compared to other existing application-layer attack detection approaches to demonstrate its performance. We intend to deploy even more web applications to validate this benchmark approach while also continuing to compare the approach against others in the literature.

Analysis of logs is a common source of detailed information about what occurs on a network or host system. Logs with differing content can cause conflicts when trying to present specific findings. By developing a framework for a hybrid intrusion detection system, the log data can be used to identify attacks and increase detection rates. A hybrid model can be implemented by the combination of an anomaly detector that is based on cross entropy measures and a signature detection method that incorporates the proposed genetic algorithm. Due to the results we obtained in our two case studies, it can be concluded that our approaches are valid and may be useful for others to reference.

8.2 Future Work

The results of our work have shown that improving attack detection rates was a feasible goal. However, obstacles were also introduced into the work. This happened with the web applications we intended to use for the studies. We plan to deploy more open source web applications to evaluate the IDS approach. We will expand our work to include those four web applications and repeat the studies as one large study that implements each approach into the finalized hybrid model. In our future work, we will compare our proposed techniques based on any additional methods seen in the literature. For instance, we can compare our genetic algorithm approach to other algorithms seen in related works. We also plan to implement our combined approaches is through the use of a rule-based technique that combines the cross entropy measures used earlier into one metric. Developing a hybrid intrusion detection system derived from our methods thus far seems to be the beginning of the ever-changing attacks that hackers continuously execute.

References

[1] Peretti, K. Data Breaches: What the Underground World of Carding Reveals. *Santa Clara High Technology Law Journal 25,* 2, 375–413.

[2] General Accounting Off., Personal Information: Data Breaches Are Frequent, But Evidence of Resulting Identity Theft is Limited; However, the Full Extent is Unknown, *at 2 (GAO-07-737 June 2007),* available at http://www.gao.gov/new.items/d07737.pdf?source=ra

[3] Schmidt, M., Fahl, S., Schwarzkopf, R. and Freisleben, B. 2011. TrustBox: A Security Architecture for Preventing Data Breaches. *2011 19th Euromicro International* Conference on Parallel, *Distributed and Network-Based Processing (PDP),* pp.635-639. Doi: 10.1109/PDP.2011.44

[4] Massicotte, F. and Labiche, Y. 2012. On the Verification and Validation of Signature-Based, Network Intrusion Detection Systems. *IEEE 23rd International Symposium on Software Reliability Engineering (ISSRE),* pp.61-70. Doi:10.1109/ISSRE.2012.16

[5] Vigna, G., Robertson, W. and Balzarotti, D. 2004. Testing network-based intrusion detection signatures using mutant exploits. *In Proceedings of the 11th ACM conference on Computer and communications security (CCS '04).* ACM, New York, NY, USA, 21-30. Doi: 10.1145/1030083.1030088

[6] Accorsi, R., Stocker, T. and Müller, G. 2013. On the exploitation of process mining for security audits: the process discovery case. *ACM Symposium of Applied Computing (SAC),* Coimbra, Protugal, pp. 1462-1468.

[7] King J. and Williams, L. 2014. Log your CRUD: design principles for software logging Mechanisms. *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, Article 5, 2014.

[8] Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H. and Zhou, S. 2002. Specification-based anomaly detection: a new approach for detecting network intrusions. *In Proceedings of the 9th ACM conference on Computer and communications security,* Vijay Atluri (Ed.). ACM, New York, NY, USA, pp. 265-274.

[9] Mashima D. and Ahamad, M. 2009. Using identity credential usage logs to detect anomalous service accesses. *Proceedings of the 5th ACM workshop on Digital identity management (DIM),* Chicago, Illinois, USA, pp. 73-79. Doi: 10.1145/586110.586146

[10] Liu, Y., Zhang, L. and Guan, Y. 2009. A distributed data streaming algorithm for network-wide traffic anomaly detection. *ACM SIGMETRICS Performance Evaluation Review, 37,* 2, pp. 81-82.

[11] Shahriar, H. and Haddad, H. 2014. Content Provider Leakage Vulnerability Detection in Android Applications. *Proceedings of the 7th International Conference on Security of Information and Networks (SIN '14).* ACM, New York, NY, USA, pp. 359. Doi: 10.1145/2659651.2659716

[12] Chou, T. 2013. Security Threats on Cloud Computing Vulnerabilities. International *Journal of Computer Science & Information Technology, 5,* 3, pp. 79–88. Doi:

10.5121/ijcsit.2013.5306

[13] Parwani, T., Kholoussi, R. and Karras, P. 2013. How to hack into Facebook without being a hacker. *In Proceedings of the 22nd International Conference on World Wide Web (WWW '13 Companion).* Republic and Canton of Geneva, Switzerland, 751-754.

[14] Geer, D. 2004. Just How Secure Are Security Products? *Computer, 37,* 6, pp. 14-16. Doi: 10.1109/MC.2004.28

[15] Heiderich, M., Schwenk, J., Frosch, T., Magazinius, J. and Yang, E. 2013. mXSS attacks: attacking well-secured web-applications by using innerHTML mutations. *In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13).* ACM, New York, NY, USA, pp. 777-788. Doi: 10.1145/2508859.2516723

[16] Siddiqui, M. and Verma, D. 2011. Cross site request forgery: A common web application weakness. *Proceedings of the 3rd IEEE International Conference on Communication Software and Networks (ICCSN),* Xi'an, China. pp. 538-543. Doi: 10.1109/ICCSN.2011.6014783

[17] Fogla, P. and Lee, W. 2006. Evading network anomaly detection systems: formal reasoning and practical techniques. *In Proceedings of the 13th ACM conference on Computer and Communications Security (CCS '06).* ACM, New York, NY, USA, pp. 59-68. Doi: 10.1145/1180405.1180414

[18] Moftah, R.A., Maatuk, A. M., Plasmann, P. and Aljawarneh, S. 2015. An Overview about the Polymorphic Worms Signatures. *Proceedings of the International Conference on Engineering & MIS 2015 (ICEMIS '15).* ACM, New York, NY, USA, Article 29. Doi: 10.1145/2832987.2833031

[19] Kruegel, C. and Vigna, G. 2003. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security (CCS '03).* ACM, New York, NY, USA, pp. 251-261. Doi: 10.1145/948109.948144

[20] Modi, C. N., Patel, D. R., Patel, A. and Rajarajan, M. 2004. Integrating Signature Apriori based Network Intrusion Detection System (NIDS) in Cloud Computing. *Procedia Technology, 62,* 12, pp. 905-912.

[21] Nascimento, G. and Correia, M. 2011. Anomaly-based Intrusion Detection in Software as a Service. *Proceeding of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*, Hong Kong, China, pp. 19-24.

[22] Cho, S. and Cha, S. 2004. SAD: web session anomaly detection based on parameter estimation. *Computes & Security, 23*, pp. 312-319.

[23] Ariu, D. 2010. Host and Network based Anomaly Detectors for HTTP Attacks, PhD Thesis, University of Cagliari.

[24] Park, Y. and Park, J. 2008. Web Application Intrusion Detection System for Input Validation Attack. *Proceedings of the 11th International Conference on Computer and Information Technology*, pp. 497-504.

[25] Le, M. and Stavrou, A. 2012. DoubleGuard: Detecting Intrusions in Multitier Web Applications. *IEEE Transactions of Dependable and Secure Computing, 9,* 4, pp. 512-525.

[26] Vigna, G., Valeur, F., Balzarotti, D., Robertson, W., Kruegel, C. and Kirda, E. 2009. Reducting Errors in The Anomaly-based Detection of Web-based Attacks Through the Combined Analysis of Web Requests and SQL Queries. *Journal of Computer Security*, *17,* pp. 205-329, IOS Press.

[27] Ludinard, R., Totel, E., F. Tronel, V. Nicomettee, and Kaaniche, M. 2012. Detecting Attacks Against Data in Web Applications. *Proceedings of the 7th International Conference on Risks and Security of Internet and Systems*, Cork, Ireland, pp. 1-8.

[28] Li, X., Xue, Y. and Malin, B. 2012. Detecting Anomalous User Behaviors in Workflow Driven Web Applications. *Proceedings of the 31st IEEE International Symposium on Reliable Distributed Sysytems (SRDS)*, Irvine, CA, USA, pp. 1-10.

[29] Gimenez, C., Villaegas, A. and Alvarez, G. 2010. An Anomaly-Based Approach for Intrusion Detction in Web Traffic. *Journal of Information Assurance Security, 5,* 4, pp. 446-454.

[30] Bronte, R., Shahriar H. and Haddad, H. 2016. Information Theoretic Anomaly Detection Framework for Web Application. *Proceedings of the 40th IEEE International Computer and Software Application Workshop (COMPSAC),* Atlanta, GA, USA, pp. 394-399.

[31] Robertson, W., Vigna, G., Kruegel, C. and Kremmer, R. 2006. Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks. *Proceedings of the 13th Network and Distributed System Security Symposium (NDSS),* San Diego, California, USA.

[32] Lee, W. and Xiang, D. 2001. Information-theoretic measures for anomaly detection. *In Proceedings of the IEEE Symposium on Research in Security & Privacy,* Oakland, California, USA, pp. 130-143.

[33] Shahriar, H. and Zulkernine, Z. 2012. Information Theoretic Detection of SQL Injection Attacks. *Proceedings of the 14th IEEE International Symposium on High-Assurance Systems Engineering*, Omaha, Nebraska, USA, pp. 40-47.

[34] Shahriar, H., North, S., Chen, W. and Mawangi, E. 2014. Design and Development of Anti-XSS Proxy. *Proceedings of the 8th IEEE International Conference for Internet Technology and Secured Transactions (ICITST),* London, UK, pp. 489-494.

[35] Cooper, V., Haddad, H. and Shahriar, H. 2014. Android Malware Detection using Kullback-Leibler Divergence. *Advances in Distributed Computing and Artificial Intelligence Journal*, *3*, 2, pp. 1-8, University of Salamanca Press. Doi: 10.14201/ADCAIJ2014391725

[36] Shahriar, H. and Clincy, V. 2014. Detection of repackaged Android Malware. *Proceedings of the 9th IEEE International Conference for Internet Technology and Secured Transactions (ICITST),* London, UK, pp. 349-354.

[37] Ozonat, K. 2008. An information-theoretic approach to detecting performance anomalies and changes for large-scale distributed web services. *Proceedings of the IEEE Dependable Systems and Networking (DSN),* Anchorage, AK, USA, pp. 522-531.

[38] Holm, H. 2014. Signature-based Intrusion Detection for Zero-Day Attacks: (Not) A Closed

Chapter?. *In the 47th Hawaii International Conference on System Sciences,* Washington, DC, USA, pp.4895-4904. Doi: 10.1109/HICSS.2014.600

[39] Neelakantan, S. and Rao, S. 2008. A Threat-Aware Signature-based Intrusion-Detection Approach for Obtaining Network-Specific Useful Alarms. *In The Third International Conference on Internet Monitoring and Protection,* pp.80-85, Doi: 10.1109/ICIMP.2008.24

[40] Kruegel, C. and Toth, T. 2003. Using Decision Trees to Improve Signature-Based Intrusion Detection. *In Recent Advances in Intrusion Detection.* Pittsburgh, Pennsylvania: Springer Link, pp. 173–191.

[41] Gupta, M., Govil, M., Singh, G. and Sharma, P. 2015. XSSDM: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications. *In the 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI).* Doi: 10.1109/ICACCI.2015.7275912

[42] Meng, Y., Li, W. and Kwok, L. 2013. Design of Cloud-Based Parallel Exclusive Signature Matching Model in Intrusion Detection. *In the 10th IEEE International Conference on High Performance Computing and Communications & Embedded and Ubiquitous Computing (HPCC_EUC),* pp.175-182. Doi: 10.1109/HPCC.and.EUC.2013.34

[43] Bronte, R., Shahriar H. and Haddad, H. 2016. A Signature-Based Intrusion Detection System for Web Applications based on Genetic Algorithm. *In Proceedings of the 9th International Conference on Security of Information and Networks (SIN '16).* ACM, New York, NY, USA, 32-39. Doi:10.1145/2947626.2951964

[44] MacVittie, L. 2013. The Application Delivery Firewall Paradigm. F5 Networks, Inc. *White Paper.* https://f5.com/fr/resources/white-papers/the-application-delivery-firewall-paradigm

[45] Buja, G., Jalil, K., Ali, F. and Rahman, T. 2014. Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack. *Proceedings of the IEEE Symposium of Computer Applications and Industrial Electronics (ISCAIE),* pp.60-64. Doi: 10.1109/ISCAIE.2014.701021

[46] Robledo, H. 2008. Types of Hosts on a Remote File Inclusion (RFI) Botnet. *Electronics, Robotics and Automotive Mechanics Conference, (CERMA '08),* Morelos, Mexico, pp. 105-109. Doi: 10.1109/CERMA.2008.60

[47] The Open Web Application Security Project. Top Ten 2013.

[48] Avancini, A. and Ceccato, M. 2010. Towards security testing with taint analysis and genetic algorithms. *In Proceedings of the 2010 International Conference Workshop on Software Engineering for Secure Systems (SESS '10).* ACM, New York, NY, USA, pp. 65-71. Doi: 10.1145/1809100.1809110

[49] Barati, M., Faez, K. and Hakimi, Z. 2013. A novel threshold-based scan detection method using genetic algorithm. *In Proceedings of the 6th International Conference on Security of Information and Networks (SIN '13).* ACM, New York, NY, USA, pp. 436-439. Doi: 10.1145/2523514.2523580

[50] Danane, Y. and Parvat, T. 2015. Intrusion detection system using fuzzy genetic algorithm. *International Conference on Pervasive Computing (ICPC),* Pune, India, pp. 1-5. Doi: 10.1109/PERVASIVE.2015.7086963

[51] Fessi, B., BenAbdallah, S., Hamdi, M. and Boudriga, N. 2009. A new genetic algorithm approach for intrusion response system in computer networks. *IEEE Symposium on Computers and Communications (ISCC),* Sousse, Tunisia, pp. 342-347. Doi: 10.1109/ISCC.2009.5202379

[52] Zhou, L. and Liu, F. 2003. Research on computer network security based on pattern recognition. *IEEE International Conference on Systems,* Man and Cybernetics, 2, pp. 1278-1283. Doi: 10.1109/ICSMC.2003.1244587

[53] Liu, S. and Fang, Y. 2012. Application research in computer network security evaluation based on genetic algorithm. *International Symposium on Instrumentation & Measurement, Sensor Network and Automation (IMSNA),* Sanya, China, pp. 468-470. Doi: 10.1109/MSNA.2012.6324623

[54] Narsingyani, D. and Kale, O. 2015. Optimizing false positive in anomaly-based intrusion detection using Genetic algorithm. *IEEE 3rd International Conference on Innovation and Technology in Education (MITE),* Amritsar, India, pp. 72-77. Doi: 10.1109/MITE.2015.7375291

[55] Senthilnayaki, B., Venkatalakshmi, K. and Kannan, A. 2015. Intrusion detection using optimal genetic feature selection and SVM based classifier. *3rd International Conference on Signal Processing, Communication and Networking (ICSCN),* Chennai, India, pp. 1-4. Doi: 10.1109/ICSCN.2015.7219890

[56] Alhamazani, K., Ranjan, R., Jayaraman, P., Mitra, K., Rabhi, F., Georgakopoulos, D. and Wang, L. 2015. Cross-Layer Multi-Cloud Real-Time Application QoS Monitoring and Benchmarking As-a-Service Framework. *IEEE Transactions Cloud Computing, 99,* pp.1 Doi: 10.1109/TCC.2015.2441715

[57] Athanasiades, N., Abler, R., Levine, J., Owen, H. and Riley, G. 2003. Intrusion detection testing and benchmarking methodologies in Information Assurance. *Proceedings of 1st IEEE International Workshop on Information Assurance (WIAS),* pp.63-72. Doi: 10.1109/IWIAS.2003.1192459

[58] Champion, T. and Denz, M. 2001. A benchmark evaluation of network intrusion detection systems. *IEEE Proceedings of the 2001 Aerospace Conference,* pp. 2705-2712. Doi: 10.1109/AERO.2001.931291

[59] Puketza, N., Chung, M., Olsson, R. and Mukherjee, B. 1997. A Software Platform for Testing Intrusion Detection Systems. *IEEE Software,* pp. 43-51.

[60] Ballocca, G., Politi, R., Russo, V. and G. Ruffo, G. 2002. Benchmarking a site with realistic workload. *IEEE International Workshop in Workload Characterization,* pp.14-22, Doi: 10.1109/WWC.2002.1226490

[61] Duan, S., Kementsietsidis, A., Srinivas, K. and Udrea, O. 2011. Apples and oranges: A comparison of RDF benchmarks and real RDF datasets. In *Proceedings of the 2011 ACM*

SIGMOD International Conference on Management of data (SIGMOD '11). ACM, New York, NY, USA, pp. 145-156. Doi:10.1145/1989323.1989340

[62] Neto, A. and Vieira, M. "Towards benchmarking the trustworthiness of web applications code," *Proceedings of the 13th European Workshop on Dependable Computing* (EWDC '11), 2011. ACM, New York, NY, USA, 29-34. doi:10.1145/1978582.1978589

[63] Neto, A. and Vieira, M. 2011. Trustworthiness Benchmarking of Web Applications Using Static Code Analysis. *Proceedings of 6th International Conference in Availability, Reliability and Security (ARES)*, pp.224-229. Doi: 10.1109/ARES.2011.37

[64] Stuckman, J. and Purtilo, J. 2011. A testbed for the evaluation of web intrusion prevention systems. *Proceedings of the 3rd International Workshop in Security Measurements and Metrics (Metrisec)*, pp.66-75. Doi: 10.1109/Metrisec.2011.14

[65] Zhang, K., Wang, L., Guo, X., Pan, A. and Zhu, B. 2009. WPBench: a benchmark for evaluating the client-side performance of web 2.0 applications. *Proceedings of the 18th International Conference on World Wide Web* (WWW). Madrid, Spain, pp. 1111-1112. Doi:10.1145/1526709.1526882

[66] Zhu, L., Gorton, I., Liu, Y. and Bui, N. 2006. Model driven benchmark generation for web services. *Proceedings of the International Workshop on Service-oriented Software Engineering* (SOSE '06). ACM, New York, NY, USA, pp. 33-39. Doi: 10.1145/1138486.1138494

[67] Zinke, J., Habenschuss, J. and Schnor, B. 2012. Servload: Generating representative workloads for web server benchmarking. *Proceedings of the 2012 International Symposium in Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pp.1-8.

[68] Imperva Report, October 2014, Accessed from http://www.imperva.com/docs/hii_web_application_attack_report_ed5.pdf

[69] OWASP- SQL Injection, Accessed from https://www.owasp.org/index.php/SQL_Injection

[70] Server Side Injection, https://www.owasp.org/index.php/Server-Side_Includes_(SSI)_Injection

[71] Path Traversal, https://www.owasp.org/index.php/Path_Traversal

[72] OWASP-XSS, Accessed from https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[73] Santillan, M. 2015. One Million WordPress Websites Vulnerable to SQL Injection Attack, http://www.tripwire.com/state-of-security/latest-security-news/one-million-wordpress-websites-vulnerable-to-sql-injection-attack/

[74] Netcraft News Report. 2014. Half a million widely trusted websites vulnerable to Heartbleed bug, http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html

[75] Varadarajan G. and Santander Peláez, M. 2012. Web Application Attack Analysis Using Bro IDS, Accessed from https://www.sans.org/reading-room/whitepapers/detection/web-

application-attack-analysis-bro-ids-34042
[76] Firebug, Accessed from http://getfirebug.com/
[77] Joomla, Accessed from https://www.joomla.org
[78] Lin, J. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory, 37,* 1, pp. 145-151.
[79] Rao C. and Nayak, T. 1985. Cross entropy, dissimilarity measures, and characterizations of quadratic entropy. *IEEE Transacation of Information Theory, IT-31,* 5, pp. 589-593.
[80] Mei, Q. and Church, K. 2008. Entropy of search logs: how hard is search? with personalization? with backoff?. *In Proceedings of the 2008 International Conference on Web Search and Data Mining* (WSDM '08). ACM, Palo Alto, California, USA, pp. 45-54. Doi:10.1145/1341531.1341540
[81] OWASP XSS Cheat Sheet, https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
[82] Path Traversal, Accessed from https://www.owasp.org/index.php/Relative_Path_Traversal
[83] SQLI cheat sheet, https://information.rapid7.com/rs/rapid7/images/R7%20Injection%20CheatSheet.v1.pdf
[84] Meng, Y., Li, W. and Kwok, L. 2013. Design of Cloud-Based Parallel Exclusive Signature Matching Model in Intrusion Detection. *IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC),* pp.175-182, Doi: 10.1109/HPCC.and.EUC.2013.34
[85] Zhou, H., Wen, Y. and Zhao, H. 2007. Detecting early worm propagation based on entropy. *In Proceedings of the 2nd international conference on Scalable information systems* (InfoScale '07). Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST), Brussels, Belgium, pp. 1-2.
[86] Nielsen, F. and Sérandour, A. 2009. Accuracy of distance metric learning algorithms. *In Proceedings of the 2nd Workshop on Data Mining using Matrices and Tensors* (DMMT '09). Paris, France, pp. 1-8. Doi: 10.1145/1581114.1581115
[87] Snort 2.8.9.0, Accessed from https://www.snort.org
[88] The Bro Network Security Monitor, Accessed from https://www.bro.org/
[89] Gaucher, R. 2008. PHPIDS: Scalp!, GitHub repository, https://github.com/nanopony/apache-scalp
[90] Obitko, M. 1998. Genetic algorithm. Courses.cs.washington.edu, https://courses.cs.washington.edu/courses/cse473/06sp/GeneticAlgDemo/gaintro.html
[91] Malhotra, R., Singh, N. and Singh, Y. 2011. Genetic Algorithms: Concepts, Design for Optimization of Process Controllers. *Computer and Information Science, 4,* 2, pp. 39-54.
[92] Jacobson, L. 2012. Creating a genetic algorithm for beginners. *The Project Spot*. http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3.
[93] Zaman, S., El-Abed, M. and Karray, F. 2013. Features selection approaches for intrusion

detection systems based on evolution algorithms. *In Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC '13).* Kota Kinabalu, Malaysia, Article 10, pp. 1-5. Doi: 10.1145/2448556.2448566

[94] OWASP, https://www.owasp.org/index.php/Main_Page

[95] Brandao, T. 2015. Genetic algorithms in PHP code, an example of evolutionary programming, *Personal programming blog*, http://www.abrandao.com/2015/01/simple-php-genetic-algorithm/

[96] Joshi, A., Eeckhout, L., Bell, R. & John, L. 2008. Distilling the essence of proprietary workloads into miniature benchmarks. *ACM Transactions on Architecture and Code Optimization (TACO), 5*, 2, pp. 769-782. Doi: 10.1145/1400112.1400115

[97] Kalibera, T., Lehotsky, J., Majda, D., Repcek, B., Tomcanyi, M., Tomecek, A., Tuma, P. and Urban, J. 2006. Automated benchmarking and analysis tool. *Proceedings of the 1st International Conference on Performance Evaluation Methodologies and Tools* (Valuetools '06). ACM, Pisa, Italy, pp. 30-39. Doi: 10.1145/1190095.1190101

[98] Vijayalakshmi, M., Shalinie, S. and Pragash, A. 2012. IP Traceback System for Network and Application Layer Attacks. *Proceedings of International Conference on Recent Trends In Information Technology (ICRTIT),* Chennai, Tamil Nadu, pp. 439-444. Doi: 10.1109/ICRTIT.2012.6206778

[99] Kendall, K. MIT Lincoln Laboratory offline component of DARPA 1998 intrusion detection evaluation. Retrieved from https://www.ll.mit.edu/ideval/data/1998data.html

[100] KDD Cup Dataset. 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[101] Bronte, R., Shahriar, H. and Haddad, H. Benchmark for Empirical Evaluation of Web Application Anomaly Detectors. *Empirical Research for Software Security: Foundations and Experience (under review).*