

AN ABSTRACT OF THE THESIS OF

Rajagopal Gaarudapuram Sriraghavan for the degree of Master of Science in Computer Science presented on March 13, 2008.

Title: Data Processing for Anomaly Detection in Web-Based Applications

Abstract approved:

Luca Lucchese

Web applications are popular attack targets. Misuse detection systems use signature databases to detect known attacks. However, it is difficult to keep the database up to date with the rate of discovery of vulnerabilities. They also cannot detect zero-day attacks. By contrast, anomaly detection systems learn the normal behavior of the system and monitor its activity to detect any deviations from the normal. Any such deviations are flagged as anomalous. This thesis presents an anomaly detection system for web-based applications. The anomaly detection system monitors the attribute value pairs of successful HTTP requests received by webserver applications and automatically creates parameter profiles. It then uses these profiles to detect anomalies in the HTTP requests. Customized profiles help reduce the number of false positives. Automatic learning ensures that the system can be used with different kinds of web application environments, without the necessity for manual configuration. The results of the detection are also visualized, which enable the system administrator to quickly understand the state of the system and respond accordingly.

©Copyright by Rajagopal Gaarudapuram Sriraghavan
March 13, 2008
All Rights Reserved

Data Processing for Anomaly Detection in Web-Based Applications

by
Rajagopal Gaarudapuram Sriraghavan

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March 13, 2008
Commencement June 2008

Master of Science thesis of Rajagopal Gaarudapuram Sriraghavan presented on March 13, 2008

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Rajagopal Gaarudapuram Sriraghavan, Author

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude to my major Professor, Dr. Luca Lucchese. He encouraged and motivated me at my every step, and kept me working towards my goal. Even during times when I was slow-paced or making no progress, he kept nudging me gently, and giving me advice to keep me going. Thanks a lot for all that and for having faith in me till the very end Professor!

I would like to thank Dr. Giovanni Vigna, Associate Professor, Department of Computer Science, University of California, Santa Barbara, whose work has been a major inspiration for my research. His guidance from the start of this research work has been very important in setting the pace and the direction for my thesis.

I am grateful to Dr. Thinh Nguyen for having introduced me to this research work in the beginning and having helped me and given me guidance, whenever I was in need of any. My sincerest thanks also go to Dr. Raviv Raich, for having played a major part in my research work by giving his valuable inputs and advice. Thanks Professor for having taken the time out whenever I had some doubts to be clarified or ideas to be discussed.

Dear Madhu, Sriraam and Kshitij, thanks a lot for having provided all the guidance, advice, motivation and encouragement throughout my research work. I am also very grateful to all my roommates, past and present: Madhu, Sriraam, Chandra, Sandeep and Rajesh, for having encouraged me, for being very understanding and also for having been my roommates and friends! Without your co-operation, I wouldn't have completed my work! Innumerable thanks go to you guys and also to all my other friends, who have all played their part in my progress, academically and personally. I spent a very wonderful time with all of you guys!

Dearest Amma, Appa and sister Priya, I don't have enough words to thank you! Right from the beginning of the days here at the Graduate School, all the way through to this day, your support, and encouraging, motivating, loving and caring words over the phone and via emails have been my greatest source of inspiration. Your innumerable prayers and invaluable wishes have helped me reach my goal. Even though we are thousands of miles apart, you have always been thinking about my well-being and my happiness. As ever, I am deeply indebted to you!

TABLE OF CONTENTS

	<u>Page</u>
1. CHAPTER ONE	1
1.1 Introduction.....	1
1.2 Intrusion Detection Systems	3
1.3 Classification of Intrusion Detection Systems	3
1.3.1 Classification based on audit data location.....	3
1.3.2 Classification based on detection strategy.....	4
1.3.3 Classification based on analysis mode	5
1.4 Approaches to Intrusion Detection	6
1.5 Thesis Overview	7
2. CHAPTER TWO	9
2.1 Related Work	9
2.2 HTTP.....	11
2.3 HTTP query based attacks	13
2.3.1 Nimda worm	14
2.3.2 Cross-site scripting attacks	15
2.3.3 Apache Information Disclosure vulnerability	16
2.3.4 Powerscripts PlusMail Poor Authentication vulnerability	16
2.4 HTTP as a universal transport protocol	17
2.5 Motivation.....	18
3. CHAPTER THREE	19
3.1 Unknown Program Identification	22

TABLE OF CONTENTS (continued)

	<u>Page</u>
3.1.1 Learning.....	23
3.1.2 Detection.....	23
3.2 Unknown Parameter Identification.....	24
3.2.1 Learning.....	25
3.2.2 Detection.....	25
3.3 Attribute Length.....	26
3.3.1 Learning.....	27
3.3.2 Detection.....	28
3.4 Attribute Character Distribution.....	28
3.4.1 Learning.....	29
3.4.2 Detection.....	30
3.5 Implementation of 6-Bin Character Distribution algorithm.....	31
3.5.1 Learning.....	32
3.5.2 Detection.....	32
4. CHAPTER FOUR.....	34
4.1 Experimental Setup.....	34
4.1.1 The MIT Lincoln Laboratory IDS dataset.....	34
4.1.2 The ECML/PKDD 2007 Discovery Challenge dataset.....	34
4.1.3 Dr. Kenneth Ingham's HTTP attack dataset.....	36
4.1.4 Synthesis of the evaluation dataset.....	37
4.2 ROC and ROC Convex Hull Plots.....	38

TABLE OF CONTENTS (continued)

	<u>Page</u>
4.3 Evaluation and Results.....	39
4.3.1 ROC and ROC Convex Hull plots of the results	40
4.3.2 Discussion of the results	48
4.3.3 Visualization of the identified alerts.....	49
5. CHAPTER FIVE	51
5.1 Conclusions.....	51
5.2 Future Work	52
BIBLIOGRAPHY.....	53

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. SANS @RISK Data from November 2006 to October 2007 – 4396 total vulnerabilities reported.	2
2. Example HTTP Request	12
3. Example HTTP Response	13
4. Nimda worm example.....	14
5. Cross-Site Scripting attack example	15
6. Apache M=D Information Disclosure attack.....	16
7. PowerScripts PlusMail Web Control Panel vulnerability.....	17
8. Sample HTTP Request from the ECML dataset. The request is stored in XML format. The portion of the query containing the attack is marked with a rectangle. The header portion has been truncated to reduce size	35
9. Sample attack request. The attack portion of the request has been marked with a rectangle.....	37
10. ROC plot and Convex Hull obtained by applying the length test on parameter values	40
11. ROC plot and Convex Hull obtained by applying the 3-Bin Character distribution tests on parameter values	41
12. ROC plot and Convex Hull obtained by applying the 6-Bin Character distribution tests on parameter values	42
13. ROC Convex Hull plots obtained by applying the 3-Bin and 6-Bin Character distribution tests on parameter values.....	43
14. ROC plot and Convex Hull obtained by applying the length test on queries	44
15. ROC plot and Convex Hull obtained by applying the 3-Bin Character distribution tests on queries	45
16. ROC plot and Convex Hull obtained by applying the 6-Bin Character distribution tests on queries	46

LIST OF FIGURES (continued)

<u>Figure</u>	<u>Page</u>
17. ROC Convex Hull plots obtained by applying the 3-Bin and 6-Bin Character distribution tests on queries	47
18. Visualized anomaly counts for Week 4 Friday data. “Unknown Parm” refers to the “Unknown Parameter” test	49
19. Visualized anomaly counts for a single program in Week 4 Friday data	50

LIST OF TABLES

<u>Table</u>	<u>Page</u>
Table 1. Bin distribution for calculating the ICD	32

DEDICATION

To my dearest Amma, Appa and sister Priya:

Without you,

I wouldn't have made it through!

DATA PROCESSING FOR ANOMALY DETECTION IN WEB-BASED APPLICATIONS

1. CHAPTER ONE

1.1 Introduction

The following adage has been taken from the frequently asked questions section of the Alt.security newsgroup [1]:

“The only system that is truly secure is one that is switched off and unplugged, locked in a titanium lined safe, buried in a concrete bunker, and is surrounded by nerve gas and very highly paid armed guards. Even then, I wouldn't stake my life on it.”

As the World Wide Web grows rapidly on a daily basis, the number of new computer attacks is also growing at a matching rate. Securing computer systems and networks is not only important but it is also difficult. The importance aspect is due to the fact that there would be heavy losses both in monetary terms and also in terms of privacy, as a result of systems that are compromised. There could also be liabilities, both civil and criminal, for having failed to secure computer systems, as specified by the laws such as the Sarbanes-Oxley [2] and the Health Insurance Portability and Accountability Act of 1996 [3,4].

The difficulty in providing security to the computer systems arises from the fact that the lists of computer vulnerabilities that become known, such as the list of Common Vulnerabilities and Exposures (CVE) [5] or the bugs discussed on forums such as Bugtraq [6] is ever-expanding. According to the National Institute of Standards and Technology [7], American companies have suffered losses, as much as 59.6 billion dollars following IT attacks, during the year 2004 alone [8]. Also, the nature of threats is very dynamic. Just during the last year, the kinds of vulnerabilities that were being exploited are very different from the ones that were exploited in the past [9].

The World Wide Web has become the ubiquitous application delivery medium. With the advent of Web 2.0, online communities, blogs and other such web-based applications, the influence of the web on the people has increased. It is easy to develop, deploy and

access these applications. Web-based applications such as Content Management Systems (CMS), Wikis, Portals, Bulletin Boards, and Discussion Forums are used by small and large organizations. A large number of organizations also develop and maintain custom-built web applications for their businesses.

For some of large-scale networks of computers that host web-applications, the number of attempted attacks can range from hundreds of thousands to even millions on a daily basis [9]. Of the 4396 total vulnerabilities reported by SANS (SysAdmin, Audit, Network, Security) Institute in their @RISK Database [10] from November 2006 to October 2007, nearly 50% are web application vulnerabilities, as indicated in Figure 1 [9].

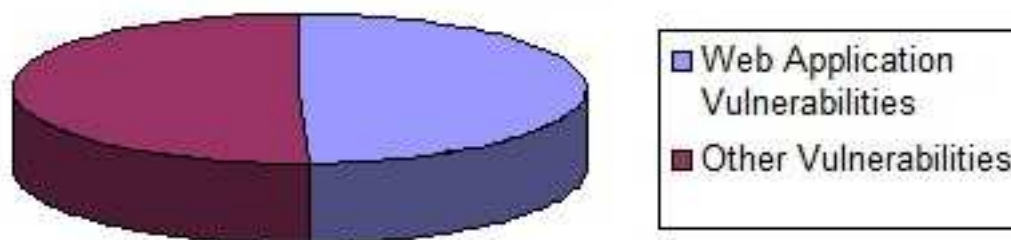


Figure 1. SANS @RISK Data from November 2006 to October 2007 – 4396 total vulnerabilities reported.

Web frameworks and software used to create these frameworks such as PHP, .NET, J2EE, Ruby on Rails, ColdFusion, etc. and web applications of all types are at risk from web application security vulnerabilities, such as insufficient validation, application logic errors, etc [9]. Apart from web-applications every other part of the web infrastructure, including the host operating systems, the backend databases, the routers too are subject to attacks [11,12]. Even anti-virus software have been exploited for vulnerabilities. Since many of these applications require administrator privileges, the vulnerabilities present in these can be used to take total control of the target systems with little or no user interaction required [9].

A *zero-day vulnerability* is a defect in software code that has been discovered and exploits of this defect appear even before any fix or patch program for that defect could be made available. Once the piece of code exploiting this vulnerability is unleashed on to the internet, until the time a software patch or some other form of fix to remove this vulnerability becomes available, users of the affected software will either have already been compromised or be at the risk of being compromised [9]. These attacks pose a major problem to businesses as they are not detected by the security systems/firewalls installed and as a result, cause heavy losses to these businesses.

1.2 Intrusion Detection Systems

Intrusion detection is defined in [13] as

“the art of detecting inappropriate, incorrect, or anomalous activity”

Intrusion detection systems (IDSs) are ideally intended to detect all kinds of attacks launched against either stand-alone computers or entire computer networks [14]. Also, a computer system or a network that is well-protected against attacks from the outside world can still be vulnerable to attacks from within the network itself. This is possible because either the authorized users themselves, or others posing as authorized users, abuse the existing privileges provided on these systems. These attacks should also be detected by the IDSs.

1.3 Classification of Intrusion Detection Systems

Intrusion detection systems are classified in 3 ways: based on the audit source location, the general detection strategy and the analysis mode [14,1].

1.3.1 Classification based on audit data location

Based on the location of the audit data, IDSs are classified as host-based and network-based IDSs. Host-based detection systems operate on a single computer and perform their analysis on user activity or behavior of the programs, observed either at the

operating system level or at the application level to detect malicious activity on that host. So, the auditing process is done and the audit data is stored on the host computer itself.

Network-based detection systems monitor the traffic in a network and analyze the headers and/or the payloads of the packets flowing through the network [13,15]. Here, the auditing can be done at different locations. A portion of the auditing can be performed at the network servers, while some of it can be done in the routers used in the network. Hence, in the case of a network-based intrusion detection system, the audit data is located in more than one place and comes from a variety of sources, like multiple operating systems, and different kinds of network servers.

1.3.2 Classification based on detection strategy

Based on the detection strategy used, IDSs are also classified as misuse-based and anomaly-based intrusion detection systems. *Misuse-based systems* use attack descriptions or signatures to identify the attacks. They analyze the stream of incoming data, and look for patterns or sequences of events that resemble any of the attack signatures. If a known attack pattern is detected, the data is flagged anomalous and an alarm is raised.

Because the misuse-based systems detect known attack patterns, they have very low false positive rates and are also efficient. Anti-virus software is one such type of misuse-based systems that use signatures of known viruses, worms and other malicious programs, to detect and if possible remove those harmful data. The main disadvantage of the misuse-based systems is that they can only detect the attacks for which they have the models. So, any novel attacks/intrusions for which there are no signatures will go undetected.

Anomaly based intrusion detection systems follow an approach that is complementary to that of the misuse-based systems. These systems analyze the data and create models, called ‘profiles’, to characterize the “normal” or attack-free data. Then, when they are deployed for detection purposes, they monitor the incoming data and compare it with the models of normal behavior, which they had created earlier. Any data that deviates from these models is considered as being associated with an attack. So the data is flagged as anomalous and an alarm is raised.

By defining the expected behavior of the normal, benign data, these systems can ideally detect any kind of abnormal data. Therefore, anomaly-based intrusion detection systems are capable of detecting even previously unknown attacks. But these systems have much higher false positive rates than the misuse-based systems. This increase in the false-positive rate can be attributed to a small extent to the fact that the data flagged as anomalous need not necessarily be malicious data. They can also be benign data that have abnormal behavior.

1.3.3 Classification based on analysis mode

Intrusion detection systems are classified as batch or real-time, based on the mode of analysis. *Batch-mode detection systems* have two separate stages of operation. The first stage involves the collection of the audit data. This is done at the source, which could either be a single host computer or it could be a network of computers and other systems. The second stage is the actual intrusion detection analysis, which is performed on the offline audit data that had been collected earlier.

Batch-mode IDSs have the advantage that the analysis can be performed when the CPU usage is low and/or at a computing facility that is different from the audit source. Hence, a thorough analysis is possible with these systems. However, as the analysis is performed offline, they suffer from the disadvantage that they cannot prevent/preempt any attacks. The damage will be detected only after it has been done [1].

Real-time intrusion detection systems, as the name suggests, monitor the data stream for anomalies and raise alarms in real-time. These systems analyze the data at the time that they are being made available to the host computer or the network. Hence, these systems can detect the attacks as and when they are happening. This helps in the preemption of the attacks or at least the minimization of the impact due to the attacks.

Because the real-time IDSs systems have to operate at the rate at which the data traffic flows in the system or the network, they have to be fast enough. To increase the processing speed, these systems compromise on the thoroughness of the analysis. This in turn would both affect the anomaly detection capability of the system and also increase the false positive rate of the system. Also, if a real-time analysis needs to be done at a

different location than the audit data source location, then a reliable, high-speed data channel is required between the target system and the analysis system [1].

Any system or a computer network that requires high security and cannot afford to have any kind of security breach, leading to loss or leakage of information, cannot use the batch detection systems. It would have to compulsorily use the real-time detection systems, so that such breaches can be avoided or preempted. They could also use batch mode systems to analyze the logs collected offline, to make sure that no breaches took place in the past. For large networks or organizations, a thorough post-analysis is also essential to ensure the integrity of the businesses and of the customers.

1.4 Approaches to Intrusion Detection

Intrusion detection tools use different approaches for their purposes. These approaches are:

- *Statistical anomaly detection tools.* They perform statistical analyses to detect the variations in the data. These are further classified into 2 types:
 - *Threshold-based detection tools.* They record the features from each occurrence of a specific event. They then analyze these features to create threshold values for each of the features. The idea is that malicious data would have values that exceed these pre-determined threshold values. The detection capability of this system relies heavily on the threshold value used.
 - *Profile-based anomaly detection tools.* These tools analyze the audit data to create the expected normal behavior or profiles of the users or applications. The system then monitors the data stream to identify an anomaly by detecting any activity that deviates from the established profiles. These types of systems have the advantage that they don't need prior information about the security flaws of the target system. But these systems suffer from factors like false positive rates, false negative rates and gradual misbehaviors [1].

- *Rule-based anomaly detection tools.* They analyze the incoming data just as in the case of the profile based systems, but instead of creating statistical profiles, they create sets of rules to represent and store the patterns of normal behavior. The advantages/disadvantages of this technique are similar to those of the statistical anomaly detection tools [1].
- *Rule-based Penetration identification tools.* These are based on the concept of expert systems which use properties to fire rules when they detect abnormal activities in the audit data [1].

All these approaches and their corresponding tools are discussed in greater detail in [16,1].

As the number of web-based systems developed and deployed with serious security vulnerabilities keeps increasing on a daily basis, security has become a basic requirement of these systems and additional steps have to be taken to protect these systems against attacks. Ideally, the security systems should be able to rapidly adapt to the changing web environments and be able to detect the various kinds of attacks/exploits. Hence, the anomaly detection system should ideally be able to self-train, operate in real-time mode and have low false positive rates.

1.5 Thesis Overview

This thesis is organized as follows:

- Chapter 1 presents an overview of the importance of computer security. It also gives a brief introduction to the concept of intrusion detection and its classification.
- Chapter 2 reviews the current literature and other related work on the topic of intrusion detection and anomaly detection systems. It also gives a brief introduction to HTTP, its structure and an overview of the attacks based on HTTP queries.
- Chapter 3 describes the intrusion detection system proposed in this thesis.

- Chapter 4 describes the experimental setup, the evaluations performed on the system and the results of the evaluation.
- Chapter 5 presents the conclusions and discusses some possible directions for the future work in this area.

2. CHAPTER TWO

A system administrator has to be alerted whenever his or her webserver is under attack and has either been compromised already or is in the process of being compromised - this is known as intrusion detection [4]. Anomaly detection systems create models of the observed behavior, which is considered as 'normal' behavior and classify any deviation from this 'normal' behavior as anomalous [12,26]. These systems function under the assumption that the attack behavior patterns differ from the normal characteristics of the target system.

The motivation behind anomaly detection is that this 'difference' can be identified by the detection systems when an anomalous occurrence is compared with the normal behavior [26]. This chapter surveys the existing literature on this topic. Some of the existing works related to anomaly detection for HTTP data are also discussed. This chapter also provides some background for the thesis, including some background about the HTTP protocol and its structure. Then a brief discussion of some HTTP query based attacks is also presented.

2.1 Related Work

The concept of computer based intrusion detection was introduced in 1972 by Anderson in his paper [11]. In his work, he emphasized the need for what are currently being termed as intrusion detection systems [4]. Initially, the anomaly detection systems were based on statistical analysis of the 'normal' behavior found in the audit records of the host system and also using other kinds of network data [13,14]. Porras [49] and Ilgun [1] proposed the State Transition Analysis concept for intrusion detection on computer systems. They monitor the states and change of states of the system. A penetration, as the term used by them to refer to the intrusions, is identified by the series of state changes which would lead the system from some initial state to a target state in which the system is compromised [49,1].

As the usage of computer networks grew, techniques such as system call evaluation gained importance. In Warrender et al. [15] and Forrest [16], the sequence analysis is performed on system calls generated by applications to determine the ‘normal’ sequence of system calls for each application. These profiles are then used to detect anomalies, with the assumption that the anomalous behavior would generate previously unseen system call sequences.

Lee [17] and Lee et al. [18] proposed a framework for constructing features and models for intrusion detection based on data mining approach. When the concept of webservers grew in popularity, the system call analysis technique was also extended to webservers, by Kruegel et al. [19]. However, these systems generated high false positives, in spite of the fact that they used much more sophisticated strategies for modeling the system call sequences than what Forrest et al. [15] had originally proposed [4].

Kruegel et al. [20] explored the concept of anomaly detection for specific services, namely DNS and HTTP. They extend beyond examining the header information and analyze the payload distribution as part of their detection process. Wang and Stolfo [21] also explored the concept of payload-based intrusion detection system. These work on the assumption that a malicious user would not be aware of the normal flow of the distribution of the application payload.

Then a similar work analyzing the characteristics related to HTTP requests for webservers/web-based applications was developed by Kruegel and Vigna [26]. Subsequently, several papers have been published that discuss models for webservers and web-based applications [56,57,58,59]. Kruegel, Vigna et al. have researched this topic extensively and have made significant contributions in this area [26,27,28]. Recently, Kenneth Ingham’s dissertation [26] provided an exhaustive overview of the existing algorithms for HTTP Intrusion detection and a comparison of these algorithms. As part of the work, Ingham also created a large database of publicly available HTTP query related attacks [60]. Publications [22,23] based on the ECML/PKDD Conference's [24] Web-Analyzing Traffic Challenge contest [25], also discuss algorithms for web-based

intrusion detection. But these algorithms are tailored specifically to the ECML/PKDD dataset.

The idea of using Visualization as a tool for aiding intrusion detection has not gained much popularity even though it had been proposed quite sometime ago. A majority of the research work has been done on visualizing the network traffic to identify attacks such as Denial of Service, Port scans [29,30,31,32,33]. In addition to visualizing network traffic to detect anomalies, Kim and Reddy [34], used “motion prediction” techniques to understand the patterns of some of the attacks and also to predict any impending attacks on the network system.

Visualization of webserver log based HTTP data has been analyzed in great detail by Stefan Axelsson [35,36,37,38,39], as part of his PhD dissertation work [40]. His research work mainly focuses on visualizing the requests as a whole. By contrast, the focus of our research work is on the query portion of the HTTP request. We use visualization as a tool to display the results of our anomaly detection system, in order to help the network administrator gain a quick understanding of the state of the system.

2.2 HTTP

The Hypertext Transfer Protocol (HTTP) has been described in RFC 2616 [41] as

“a generic, stateless, application-level protocol”

It works based on a request-response model. A client sends an HTTP request to a server. The request would contain a request method, a “Uniform Resource Identifier” (URI), and protocol version, followed by a request message header. The server sends back an HTTP response that contains a status line, including the protocol version of the message and a status code, indicating the success of the request or error, followed by a response message header [41].

The client sends the HTTP request for a specific resource on the server. These clients are typically web browsers, like Internet Explorer or Firefox, for example, or, they could also be automated programs such as bots, used for such purposes as indexing the web

pages on the internet. The request can be for a specific file hosted by the webserver or for a program running on the webserver. The URI identifies this resource. The request method indicates the desired action to be performed on that resource [42]. The most common request methods are GET, POST and HEAD.

A web application program accepts inputs from a client through HTTP requests. This is done using the optional query portion of the HTTP request. The query is the component of the request following a '?'. It contains parameter-value pairs, of the form $p1=v1\&p2=v2\&\dots\&pn=vn$, where $p1, p2, \dots, pn$ are the parameters and $v1, v2, \dots, vn$ are the corresponding values. Figure 2 shows an example of an HTTP request containing the query portion, taken from the 1999 MIT Lincoln Labs dataset [43].

```
GET /cgi-bin/chat/chat_entry?area=helpchat%2Esomesite%2Ecom HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en
Host: www.somesite.com
Referer: http://www.somesite.com/main/help/
User-Agent: Mozilla/2.0 (compatible; MSIE/3.01; Windows 95)
UA-Color: color32
UA-CPU: i686
UA-OS: Windows 95
UA-Pixels: 1024x768
```

Figure 2. Example HTTP Request

Upon receiving the request, a webserver performs some processing if necessary, and sends back an HTTP response. As mentioned earlier, the response consists of a status line that indicates the success or failure of the request, using status codes. These status codes are of 5 classes [44]:

1. *1xx - Information,*
2. *2xx - Success,*
3. *3xx - Redirection,*

4. *4xx - Client Error,*
5. *5xx - Server Error.*

Specifically, the status codes of the form *2xx* indicate a success. If the HTTP request was for a specific file on the server, this response code means that the requested file was found on the server and that it is being sent as part of the response (usually as part of the response message body). If the request was for a program on the server, then the success code means that the requested program was executed successfully, and the response message body might contain anything that the webserver program returned at the end of its execution. Figure 3 shows an example of a successful HTTP response taken from the 1999 MIT Lincoln Labs dataset [43].

```
200 OK
Cache-Control: private
Connection: Keep-Alive
Date: Mon Mar  1 13:37:54 UTC 1999
Server: Microsoft-IIS/4.0
Content-Length: 28826
Content-Type: text/html
Expires: Thu, 04 Feb 1999 23:20:03 GMT
```

Figure 3. Example HTTP Response

2.3 HTTP query based attacks

Our research work focuses on successful HTTP requests that use the GET method and contain the query component. Using the parameter-value portion of the query, malicious inputs can be sent to a web application. In such a case, an unsuccessful request cannot do much harm to the web application as the returned error code indicates that the requested operation was not performed successfully. But a success response means that the malicious request had been processed successfully. Because of the success response

generated by the server, these attacks might most likely remain unnoticed, though, the webserver program might have been compromised. In this section, we present a few attacks that make use of the query portion of the request, to supply malicious input to the webserver application program.

2.3.1 Nimda worm

One of the most famous attacks in the Microsoft platform, that spreads using the query portion of a request, is the Nimda worm and its variants [45]. It is capable of affecting both the clients that use any version of Windows as the host operating system, like Windows 95, 98, ME, NT, or 2000 and also the servers running Windows NT or 2000 [45].

The default configuration of the Windows operating system has a vulnerability, which is exploited by this worm. Using this vulnerability, the worm enables any user with a malicious intent to execute arbitrary system commands on the operating system of the host machine running the webserver. Figure 4 shows an example of a Nimda worm attack [46].

```
GET /scripts/root.exe?/c+dir HTTP/1.0
Host: www
Connection: close
```

Figure 4. Nimda worm example

This attack has several known variants, all of them exploiting the vulnerability in the Windows operating system. The popularity of this worm, along with the presence of several variants for it indicates the successful nature with which it has infected several systems. Despite the fact that the signature for this worm and its variants have been part of the signature based intrusion detection systems whenever they become known, the worm, with the help of its variants has successfully evaded the detection by these signature based systems.

2.3.2 Cross-site scripting attacks

Another popular attack type is the Cross-site scripting attack [47]. This vulnerability enables the malicious user to use a web application program to inject code, most likely as client (browser) side scripts, into the web pages viewed by a lot of other users, who then become the victims of this attack. The malicious user causes a legitimate webserver to send a response page to a client's browser that contains malicious script or HTML that the attacker chooses.

The client cannot distinguish between the legitimate portion of the response that was actually sent by the webserver and the malicious piece of code that the attacker sent as part of this page. Hence, the client would execute the entire page, including all the harmless and harmful scripts that it may contain. The malicious script runs with the same permissions as that of the legitimate script sent by the webserver. This gives the malicious code access to sensitive information on the page such as personalized cookies, any passwords or other sensitive information such as credit card numbers that maybe present as part of that web page. Figure 5 shows an example of a cross-site scripting attack:

```
GET /comment.cgi?mycomment=<SCRIPT>rm+-rf</SCRIPT> HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Host: http://example.com
Referer: http://example.com/
User-Agent: Mozilla/2.01 (Win3.1; I;)
```

Figure 5. Cross-Site Scripting attack example

The malicious code is present in the parameter value for the *mycomment* parameter, embedded between the `<SCRIPT>` and `</SCRIPT>` tags. This attack example shows how a malicious code injected as part of the parameter value of a HTTP request can cause harm to the target system.

2.3.3 Apache Information Disclosure vulnerability

The Apache webserver was vulnerable to an information disclosure attack. Apache webserver version 1.3.20 with the Multiviews option enabled, allows remote attackers to list the contents of its directories and bypass the index page by using a URL containing the “M=D” query string [48]. Google used this vulnerability to index pages that had no externally accessible links to them [48,4]. Figure 6 shows an example of this vulnerability [4]. Though at first sight, this seems like a harmless attack, the severity of this vulnerability depends on the kind of sensitive information stored as part of the files that get listed. In case these files contain private information or information that should not be revealed to anyone except authorized users, this vulnerability poses a serious problem. This is an example of an attack, where the default parameter options have not been properly taken care of. Figure 5 shows an example query for this vulnerability.

```
GET /~immsec/data/SM/CERT/?M=D HTTP/1.0
Host: www.cs.unm.edu
User-Agent: msnbot/0.3
Accept: text/html, text/plain, application/*
Accept-Encoding: identity;q=1.0
From: bot(at)somesite.com
```

Figure 6. Apache M=D Information Disclosure attack

2.3.4 Powerscripts PlusMail Poor Authentication vulnerability

The Powerscripts PlusMail Web Control Panel is a web based administration suite that can be used to manage mailing lists, websites, etc [61]. As part of the administration features, it provides the facility to change the administration password from remote clients, using HTTP requests. But because of the password file has public permission by default (which is a grave mistake/bug), it is possible for an intruder to craft a HTTP request to the PlusMail CGI script and create a new username and password on the system. The intruder could in fact create the new user account with administrator privileges and take complete control of the system! This vulnerability highlights the risks

introduced by software whose default installation configurations are not modified properly. A sample attack string for this vulnerability is shown in Figure 7 [61].

```
GET /cgi-bin/plusmail?login=pluz&password=pluz&password1=pluz&new_login>Login HTTP/1.0
```

Figure 7. PowerScripts PlusMail Web Control Panel vulnerability

2.4 HTTP as a universal transport protocol

HTTP has become the universal transport protocol for almost all kinds of webserver applications [4]. Earlier, for any new service provided by a webserver, a custom protocol for communicating with this service was developed and some port was assigned to be used for communications with this service. But due to issues related to security associated with many new protocols developed in this manner, network administrators blocked access to these services/ports with the help of firewalls [50,51]. But HTTP passes through most of these firewalls, with little or no trouble at all. Hence web application developers started using HTTP as a transport protocol for their new software [4]. Some of the examples for this include SOAP [52], tunneling secure shell (SSH) connections [53], Apple's QuickTime multimedia [54], and Microsoft RPC for accessing Exchange (email) servers [55]. The creation of all these new services over the HTTP protocol creates additional opportunities for the intruders [4]. They also create a lot of variations in the characteristics of the HTTP requests.

2.5 Motivation

The motivation for our proposed algorithms is based on the nature of the attack requests. A review of the HTTP query based attacks reveals abnormalities such as unusual length, usage of system commands and default webserver variables, usage of special characters as part of the attack string. We use this knowledge of the attacks to create models for our anomaly detection system, which is discussed in the next chapter.

3. CHAPTER THREE

The attacks related to web application programs manifest themselves in many different forms as seen in the previous chapter. These attacks can vary greatly in terms of their characteristics, such as the length of the attack portion of the request, the grammar (or the lack of it) in the attack pattern, etc. It is difficult to capture such variations in the attack patterns using any single model. The term model, as used for the purposes of anomaly detection, is defined by Kruegel et al. [27] as:

“a set of procedures used to evaluate a certain feature of a query”

Any single feature of a query or its parameter value would only reveal a certain amount of information about the query or the parameter. So a detection model based on that feature would only be helpful in identifying any anomalies related to that particular feature of the parameter value or the query. Moreover, any intrusion detection system based on a single model would fail to identify several potential attacks in the HTTP request. Therefore, it is necessary to use a number of different models, in order to enable the system to effectively detect as many kinds of attacks as possible.

The anomaly detection system proposed as part of this thesis is one such multi-model system that attempts to capture as much information as possible from the query and its parameters. The system captures this with the help of a variety of features extracted from the query and its parameter values. It also attempts to detect many different kinds of attacks by applying multiple models to classify the HTTP request based on the extracted features. Most of the processes involved in this system are performed automatically, involving minimal amount of human effort. This chapter discusses the set of models/heuristics used as part of our anomaly detection system, namely:

1. Unknown program identification
2. Unknown parameter identification
3. Attribute Length and
4. Attribute Character Distribution

These models can be based on either the whole query or on the individual parameters of the query. The unknown program identification and unknown parameter identification models listed above classify the query as a whole while the attribute length and character distribution models are based on the individual parameters of the queries and hence, classify each of the parameter values as benign or anomalous. The cumulative effect of the set of these detection models is that the entire HTTP request gets classified based on the results for the individual parameter values.

Each anomaly detection model, whether based on the whole query or on the individual query parameters, operates on the URL portion extracted from successful HTTP requests that contain query parameter values. As mentioned in Chapter 2, successful HTTP requests are those that invoke a response from the webserver containing the status code of the form $2xx$. An example of such a response is shown in Figure 3 in that chapter. These HTTP request URLs are further partitioned into subsets based on the directory path and the web application program to which they are addressed. The system then processes each subset of queries independently.

The anomaly detection system operates in two modes, namely, *learning* and *detection*, in that order. During the learning phase, each model builds a profile of the “normal” characteristics of the corresponding feature of the parameter values, like the length of the attribute value, for example. The profile is then used by the corresponding model to determine a detection threshold for that parameter. The model performs this process for all the programs that it encounters during the learning phase.

During the detection phase, the model extracts the necessary features from the query parameter values of incoming HTTP requests, using the same procedures that it used in the learning phase. But this time, instead of creating profiles with the extracted features, it compares the features with the corresponding profile, created during the training phase, using either a simple threshold value comparison or by using some statistical methods to compare the two quantities and returns an “anomaly score” for each observed parameter value. The anomaly score is nothing but a probability value in the interval $[0,1]$ indicating how normal or abnormal the observed value is when compared with the established

profile for that attribute. A score close to one indicates a normal instance, or in other words, an instance drawn from the expected distribution for that parameter, while a score close to zero indicates a highly anomalous value.

Each detection model in the system returns one such anomaly score, if applicable, for each of the parameter values in the incoming HTTP request. Based on these values, the request is reported either as a potential attack or as a normal event. Even if one of these scores lies outside the corresponding detection threshold value, the associated HTTP request is reported as anomalous. Although classifying a request based on the outcome of any single model may seem to portray the system as being over-cautious, this is essential as the attacker might try to embed a single malicious parameter value in the query and mask the value in such a way that it looks like a normal query as much as possible.

The models of the anomaly detection system are built independently and can also be tuned, if applicable, independently of the other models. Tuning is necessary to adjust the sensitivity of the model, in terms of the number of true positive and false positive rates. These rates are explained further in the next chapter. This tuning can depend on the policies of the organization of which the web application server is a part.

Some organizations might require a high rate of success in detecting the anomalies, but they maybe lenient with the number of false alarms generated. Some other organizations might be fine with a system that misses some of the anomalies, but would ideally not want any false alarms to be generated. All these factors need to be considered thoroughly when tuning the sensitivity of the anomaly detection system. The tuning of the system can be performed by adjusting the threshold values associated with the models of that system.

The following sections describe the models used as part of our anomaly detection system.

3.1 Unknown Program Identification

A web application server hosts a set of applications or programs as part of its system. These programs provide a set of services to the client machines over a network. An HTTP request that contains a query sent to a web application server is actually a request for service addressed to one such program or application that is hosted by the server. The webserver itself will have a set of programs which would enable the system administrator to configure the server properties and maintain its operations.

The webserver operations could include creating server logs, providing an administration page to change the settings for the webserver and even to change the system administration password or other security features. All these functionalities would be provided as part of the default installation options for the webserver. They would be available in standard directory locations, determined by the developers of the webserver software, and would usually be accessible over the network for ease of use.

If the server has not been configured properly and/or if the default settings have not been properly taken care of, then any user with a malicious intent who has some knowledge about the kind of server software being used, can attempt to exploit these vulnerabilities, using properly crafted HTTP requests. Therefore, any request that targets a new or an unknown program can most likely be targeting something not related to the applications that are being provided as services by the webserver. Such requests could either be exploiting the vulnerabilities present in the webserver.

In most of the cases, these web servers also provide functionalities to run host operating system commands, with the privileges of the webserver administrator, who would most likely also have administrator privileges in the host operating system. Hence, a webserver that is vulnerable can also be used to exploit the operating system on which the server is hosted. In certain cases, these HTTP requests could also be targeted at some

malicious code that has somehow been placed in the server previously. Hence, such requests addressed to unknown applications or programs should be flagged as anomalous. This technique is motivated by the unknown parameter heuristic that Kruegel and Vigna proposed in their system [26].

3.1.1 Learning

During the training phase, the model extracts the web application program name, from each of the HTTP requests that it encounters. It then creates a unique list of all these programs. The assumption is that when sufficient amount of data has been used during the training phase, all services considered a part of the normal day to day activities of the web application server would have been covered completely. This in turn would imply that all the programs and applications hosted by the server to provide these services would have at least one HTTP request addressed to each of them (note that a single request, though enough for this particular model, is not at all considered as sufficient data for those models that build profiles of the query parameter values). Ideally, by the end of the training phase, this process would have created the list of all the “known programs”, that is the list of all the programs or applications that the server is hosting.

3.1.2 Detection

During this phase, the system identifies the program to which the request is directed to, by extracting the program name. The system then checks this name against the list of all known programs that it created during the training phase. If this program name is found on the known list of programs, the incoming request is considered normal. If not, the algorithm flags the request as anomalous. It is possible that, in reality, a new program has been added to the webserver recently. The HTTP request could have been addressed to that new program and hence, is in fact a benign request. But, in this case, it is the responsibility of the system administrator to train the detection system on this new program, before making the program available to the clients of the webserver. This ensures that web application security system is properly updated.

Proper updating and maintenance of the anomaly detection system is essential for the effectiveness of the system in ensuring the security of the web application server.

3.2 Unknown Parameter Identification

As mentioned in Chapter 2, the web application programs accept user inputs through the HTTP requests. They do this by using the query portion of the request, which contains the parameters of the programs and the corresponding values. Since these parameters are nothing but the inputs that the programs hosted on the server accept, they cannot vary drastically across the requests. A few parameters maybe optional for a program, so they may or may not be present in all the queries addressed to that program. But a query cannot have any parameter that the target program cannot accept or is not expected to accept. This is the motivation behind this model of the anomaly detection system.

Kruegel and Vigna [26] state that the unusual presence of one or more parameters in a query might indicate some malicious content in the query. As mentioned in the previous section, some of the programs exist as part of the installation of the webserver software. These programs may have certain default parameter values that can be used to change the configuration of the webserver. Or they can be used by the webserver to obtain inputs from the system administrator for other functionalities such as creating the server logs. A previously unseen parameter present in the HTTP request might in fact be exploiting the presence of these default parameters that come as part of the web applications.

Just like the case of the unknown program identification model, sometimes default parameters come as part of the webserver software. Depending on the popularity of the webserver software used, the presence of these parameters would be well-known not only to the system administrator, but also to the people who have some knowledge about the server software. These parameters could be as harmful as resetting the password of the server application, as seen in the PowerScripts PlusMail vulnerability mentioned in the previous chapter [61].

At the least, the default parameters could be exploited to list the contents of the webserver directories, as shown in the Apache M=D vulnerability mentioned in Chapter 2 [48]. These directories might actually be private, containing important information. The contents of such directories would never have been meant to get listed. So, even though this directory listing seems to be a harmless vulnerability, in reality it is not so. Hence, all requests containing unknown parameters should be flagged as anomalous by the model. This heuristic was proposed as part of the anomaly detection work by Kruegel and Vigna [26].

3.2.1 Learning

In the training phase, the model creates a list of all the unique parameters that it notices as part of the requests sent to each of the programs or applications on the webserver. It records the program to which the request was made along with the parameters that were present in the query portion of that request. As with the previous model, here too we make the assumption that when a sufficient number of requests per web application program exist in the training datasets, all the parameters that are associated with that program get covered in the set of requests.

3.2.2 Detection

In the detection mode, the algorithm extracts the name of the program to which the incoming request was addressed and the set of parameters that were part of that request. It then verifies these parameter names with the parameter list that it maintains for that particular application. If it encounters a parameter that is not present in the list, it flags the parameter as anomalous.

It is possible that the “unknown” parameter could actually be a genuine one. This means that the parameter was either never present in the training dataset, or was added to the program recently, after the system switched to the detection phase. The first case in turn implies that the training dataset was insufficient. If so, we need to ensure that the system is retrained on a set of requests that contain this parameter. In the second case, in the event of any modification to the webserver software, the system administrator should

ensure that the anomaly detection system is updated properly, as this is necessary for ensuring the security of the server.

3.3 Attribute Length

This heuristic was proposed as part of the anomaly detection system work done by Kruegel, Vigna et al. [26,28,27]. In practice, the length of parameter values sent as part of the HTTP requests does not vary greatly. In some cases, the web application itself would have length restrictions to be adhered to. For example, a login application might require that the login id have a minimum length of 8 and a maximum of 15 alphanumeric characters. Fields accepting credit card numbers have an exact length requirement of 16 digits while US phone numbers must have 10 digits, when they expect the numbers including the area codes.

For some of the malicious inputs, the length of the parameter value deviates greatly from the normal. For example, in order to overflow a buffer in a target web application program, the entire shell code and extra padding based on the length of the target buffer, needs to be sent as part of the request. Therefore, that parameter value maybe several hundred bytes long [27]. As another example, in the case of some cross-site scripting attacks, in which malicious scripts are included in pages whose content is determined by the data supplied by a user, the amount of data that is sent as part of the parameter value of a request can significantly exceed the length of normal parameter values [27]. Such anomalies can be detected by learning the unknown distribution of the attribute lengths [26].

In many of these cases, the situation can actually be avoided if the web application developer had proper length checks in place, whenever such application-based restrictions are relevant. But as is the case with many existing applications, sometimes, code necessary to perform such checks get missed out, when developing the application. Also, for many of these parameters, there may not be any length restrictions imposed by the web application system. In such cases, length checks will not be performed.

As mentioned in the previous section, the parameters are actually inputs that the web application program accepts. The programming or scripting language that was used to create this program would have data type related length restrictions, which would apply to these parameters. The size restriction can also depend on the host operating system used by the server. In any case, extremely long inputs might violate these length restrictions and if proper checks are not in place, this could crash the program or even the operating system itself. This could also overwrite or reveal the memory contents in the server/host operating system, as with the case of the buffer overflow attacks [63].

Minor deviations from the normal can be expected from genuine request parameter values too. For certain parameters, the benign data can also be expected to have large variance. The intention of this model is only to capture significant deviations from the normal length of parameter values. This reduces the number of false positives generated by the model, while still retaining the ability to detect obvious length related abnormalities.

3.3.1 Learning

The system records the lengths of the values for each of the parameter present in the incoming HTTP requests. It stores this information separately for every program that it encounters during the training phase. Suppose the system observed n requests containing a certain parameter for a particular web application program during the training phase. Let the lengths of the observed values be l_1, l_2, \dots, l_n . The anomaly detection system then approximates the mean μ and variance σ^2 of the real attribute length distribution by calculating the sample mean and the sample variance of the observed lengths.

3.3.2 Detection

Once the system has calculated the sample mean and variance of the observed parameter values, during the detection phase, the heuristic monitors the parameter value of an incoming request and determines its length l . It has to then check if the observed length is “normal”. The probability that the observed length l is from the distribution of the parameter’s length values, is calculated using a modified version of the Chebyshev inequality, proposed in [26] as shown below:

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

In case the observed length l equals the mean length μ , the denominator is replaced by a small, insignificant value, in order to avoid divide-by-zero error. The closer the value of p is to one, the higher the probability that the observed length is normal. A threshold value is determined and used to classify the length as normal or anomalous. The bound computed by the Chebyshev inequality is, in general, very weak and is independent of the underlying distribution [26]. As a result of this weak bound, our detection model has a high degree of tolerance to variations in attribute length, and it flags only obvious outliers as anomalous [28], as desired.

3.4 Attribute Character Distribution

The character distribution model is motivated by the observation that attribute values are usually regular in their structure, in general, contain only printable characters and can usually be understood by humans [28]. To quote the login application again, many of these programs require that people create login ids and passwords only with alphanumeric characters. They may also include a very restricted set of special symbols (usually just 2 or 3 different special symbols). Hence, the values of this login parameter tend to have similar character distributions.

By character distribution, we mean the relative frequencies of the alphabetical characters, numbers and special symbols. Unlike the normal data, buffer overflow attacks, for example, often have a distinct character distribution [4]. These attacks sometimes use special characters (mostly whitespace characters) to pad the string in order to overflow the buffer. Some other attacks, like cross-site scripting, use script related tags and other symbols as part of the scripting code that are embedded into the parameter values. Thus all these values tend to have more than the usual number of special characters as part of the string.

The attribute character distribution model creates profiles to detect these kinds of deviations in the parameter values. The assumption behind this model is that there is a marked distinction between the character distributions of the parameter values that are benign and those of parameter values that are anomalous and that this distinction can be captured by the detection model.

3.4.1 Learning

During the learning phase, the anomaly detection system counts the number of alphabetical characters, numerical characters and special symbols found in each of the values associated with a specific parameter of a web application program. It then normalizes these values to get the relative counts or frequencies. These counts are binned into 3 bins, one each for alphabetical characters, numbers and special symbols. At the end of the learning phase, the system calculates the average of all the observed values for each of these 3 bins. Because the individual character distributions sum up to unity, the sum of the averages of these will also be equal to one. Therefore, the final character distribution is well-defined [26].

As an example, consider the word *pas123!@#*, as a sample value for the parameter 'password' of a login application. The bin counts for the alphabetical characters, numbers and special character bins would be 3, 3, 3. The relative character distribution would be 0.33, 0.33, 0.33. These relative bin counts are stored for the 'password' parameter and in the end, the average bin count is calculated for these three bins over all the observed values.

3.4.2 Detection

During the detection phase, the algorithm counts the number of alphabetical characters, numbers and special symbols for each of the parameter values in the incoming request and normalizes these values to get the relative counts. Then the system has to determine the probability that the obtained distribution is a sample drawn from the expected character distribution of the associated parameter. In order to achieve this, the detection algorithm uses a variant of the Pearson χ^2 -test as a ‘goodness-of-fit’ test [62]. The variant to this test was proposed by Kruegel and Vigna as part of their anomaly detection system work [26].

The counts obtained in the detection phase can be considered as the set of observed frequencies O_i for the corresponding parameter. The expected frequencies E_i for the three bins of the corresponding parameter have already been calculated during the learning phase. The χ^2 value is then calculated as

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

The degrees of freedom of this χ^2 test are equal to the number of bins used for the calculation minus one, which yields the value two [26]. The probability p that the observed parameter value comes from the expected character distribution is obtained by looking up a predefined χ^2 table using the calculated χ^2 value for two degrees of freedom. The higher the value of p , the greater the probability that the parameter value belongs to the expected character distribution. Thus, the p value obtained from the χ^2 table is used to classify the parameter as normal or anomalous.

3.5 Implementation of 6-Bin Character Distribution algorithm

For comparing the performance of our proposed character distribution algorithm, we implemented the 6-Bin Character Distribution algorithm (from now on, we shall call our algorithm, described in the previous section as the 3-Bin Character Distribution algorithm) proposed by Kruegel, Vigna et al. [26,28,27].

In this model, a character distribution is considered to be the relative frequencies of the 256 ASCII character values sorted in descending order [28]. The motivation for this algorithm is that for normal inputs, the relative character frequencies are expected to have a slow decrease in the values, while malicious inputs, can have either an extreme drop-off due to a lot of repetition of a single character, or little drop-off because random characters were used [28].

The frequency counts for each of the 256 ASCII characters, for the given parameter value is determined. These counts are then sorted in descending order. The example string used in their paper [26] is *passwd*. The absolute frequency counts in descending order of frequencies are 2 for *s*, 1 each for *a*, *d*, *p* and *w*, and 0 for all the other characters. The relative frequencies obtained with these counts are 0.33, 0.17, 0.17, 0.17, 0.17, followed by 0 occurring 251 times.

They called the character distribution of an attribute that is perfectly normal the *idealized character distribution (ICD)*, induced from the training data and noted that

$$\sum_{i=1}^{256} ICD(i) = 1.0$$

As mentioned earlier, the *ICD* is sorted in the descending order of the relative frequencies. So the largest frequency is denoted as *ICD(1)* and the least is denoted as *ICD(256)*.

3.5.1 Learning

The *ICD* is calculated during the training phase. For each of the observed query parameter values, the character distribution is determined and stored. The idealized character distribution is then approximated by calculating the average of all these stored character distributions.

Similar to the case of the 3-bin Character Distribution, in this model too, the individual character distributions sum up to unity, and in turn, their averages would do so as well. Hence, the idealized character distribution is also well-defined [26].

3.5.2 Detection

For testing, they binned the *ICD* (the expected distribution) and the distribution of the incoming request (observed distribution) into six bins as shown in the table 1 below.

Table 1. Bin distribution for calculating the ICD

Bin	1	2	3	4	5	6
<i>I</i>	1	2-4	5-7	8-12	13-16	17-256

where $i \in [1, 256]$. For example, bin 4 contains

$$\sum_{i=8}^{12} ICD(i)$$

Once these values have been binned, they used a χ^2 test to check if the character distribution of the parameter values is similar to that of the expected values. The modified χ^2 test as proposed by them is

$$\chi^2 = \sum_{i=1}^6 \frac{(O_i - E_i)^2}{E_i}$$

where E_i is the value in bin i for the *ICD*, and O_i is the value in bin i for the observed distribution. The degrees of freedom for this test is five (one minus the number of bins used in the test). The χ^2 value so obtained is used to lookup the values from a table for five degrees of freedom and the corresponding probability value p is obtained. As in the case of the 3-Bin Character Distribution test, the higher the value of p , the greater the probability that the observed parameter value is a sample drawn from the idealized character distribution.

4. CHAPTER FOUR

4.1 Experimental Setup

In order to perform rigorous testing and evaluation in a simulated environment, the data used for the testing purposes should be as similar as possible to the actual traffic and actual attacks found in real-world computer networks. Usually, such a kind of data is difficult to obtain because the real-world data would contain personal and/or private information. The real-world data would also reveal the potential vulnerabilities present in the networks which were used to gather this data, thereby putting those networks under the risk of attacks.

4.1.1 The MIT Lincoln Laboratory IDS dataset

Because of these factors, the MIT Lincoln Laboratory's 1998 intrusion detection evaluation was performed with sponsorship from DARPA, to create the first standard corpus of labeled datasets that could be used to evaluate intrusion detection systems [64,43]. Subsequent evaluations were also conducted in 1999 and 2000 creating more datasets of this kind.

These datasets contain host-based and network-based labeled attacks. The datasets are available in *tcpdump* format [68]. But none of the HTTP requests in the *tcpdump* datasets contain the attacks in the parameter values of the queries. As our anomaly detection system is based on the concept of creating profiles of the parameter values and then using these to detect anomalies, the attacks present in the MIT Lincoln labs dataset were not relevant to our research work.

4.1.2 The ECML/PKDD 2007 Discovery Challenge dataset

One of the most recent publicly available labeled HTTP dataset is the web traffic dataset created for the ECML/PKDD 2007 Discovery Challenge, held in September 2007 [25]. This dataset contains around 50000 requests of which nearly 20000 are attack requests. The data is available in XML format [69]. Figure 8 shows a sample HTTP Request from the ECML dataset.


```

<sample id="43552">
  <reqContext>
    <os>UNIX</os>
    <webserver>APACHE</webserver>
    <runningLdap>UNKNOWN</runningLdap>
    <runningSqlDb>TRUE</runningSqlDb>
    <runningXPath>UNKNOWN</runningXPath>
  </reqContext>
  <class>
    <type>OsCommanding</type>
    <inContext>TRUE</inContext>
    <attackIntervall>query:21-39</attackIntervall>
  </class>
  <request>
    <method>GET</method>
    <protocol>HTTP/1.1</protocol>
    <uri><![CDATA[/pQXQWq/dnatfysVJEj/g4Yrcerhhoim/wcdDUF0s-6Yo0mficXtc/HN/6wdkDItF0xp_2@Z/dSay2Z1Py9-D/dBMj-EaPVDBsHlsvh/nQlhP.gif]]></uri>
    <query><![CDATA[lehr=wG.6s6Hw2Zn55a+%60+rm+-rf+%2F+%60
&gdmyaA=oejTu&AenlLoaMneeRi=4257&si.no2nenb=3658&fsnbenhnUoEo1f=eis&g112freplaceGqc7Rq=394231125
&wnMsyxf=4096574936&manatAead=aTrtlloS+bue%28ts%3D3+&stylewhAGN3N_=_zr%29s-6zckCrKBD=ccn&ehr4t+=nph-adminiframe]]></query>
    <headers><![CDATA[Host: www.tEincjsen.st:9182
Accept: audio/*, application/*;q=0.9, application/zip;q=0.7
Client-ip: 214.70.1.175
Date: Sun, 25 Mar 07 08:48:12 UTC
User-Agent: Mozilla/0.8 (Machintosh; U; Mac OS X 8.6; 7d-yQ; rv:9.9.8) Gecko/51561068
-----
~~~~~: ~~~~~]]></headers>
  </request>
</sample>

```

Figure 8. Sample HTTP Request from the ECML dataset. The request is stored in XML format. The portion of the query containing the attack is marked with a rectangle. The header portion has been truncated to reduce size

Each portion of the HTTP request is stored using appropriately named XML tags, as shown in Figure 8. The tags also indicate whether the particular request has an attack or not. If the request contains an attack, then the XML tags also indicate the portion of the request that contains the attack and another set of tags indicate the exact byte offsets within that portion of the request in which the attack is found. In Figure 8, the attack portion of the query has been marked with a rectangle. The attack requests belong to one of the following 7 categories:

1. Cross-Site Scripting
2. SQL Injection
3. LDAP Injection
4. XPATH Injection

5. Path traversal
6. Command execution
7. Server-Side Include (SSI) attacks

The main objectives of the challenge were classification of the attacks based on the context and isolation of the attack patterns from the requests. Except for the attack portion of the request, the remaining portions have been randomly generated, to the extent that no two requests in the dataset are addressed to the same host/same application and the requests themselves don't convey any meaning. As our research work is based on learning the characteristics of each application of a server, we need a lot of requests per application program in order to build the profiles for that program. But since this dataset does not contain more than one request per web application program, this dataset too could not be used for our evaluation.

4.1.3 Dr. Kenneth Ingham's HTTP attack dataset

An attack dataset containing the attacks that are delivered as part of HTTP requests was recently published on the internet by Dr. Kenneth Ingham [60]. This is a collection of 63 attack requests that Dr. Ingham collected as part of his recent PhD dissertation work [4]. He had collected these data from various online security forums and other online sources.

The site details the type of the attack, gives a description of how the attack is performed, what vulnerabilities the attack exploits and includes a sample HTTP request containing the attack. Quite a few of these attacks are delivered as part of the query portion of the HTTP requests. Hence, these attacks are relevant to our research work. But this dataset only contains HTTP requests containing attacks. There is no data for the normal or genuine HTTP requests. Hence, we cannot use this dataset to build the profiles during the training phase, which is crucial for our anomaly detection system.

4.1.4 Synthesis of the evaluation dataset

We did not have access to any relevant labeled datasets that we could use as-is for our evaluation. Though the MIT dataset was not useful as-is for our purposes, the dataset was organized into different weeks of data. Also, since none of the attacks in that dataset were present in the query portion of the request, those attacks are not relevant to our evaluation. Hence, these attack requests can also be considered as benign requests for the purposes of our evaluation. Since the dataset was organized as three weeks of training data and two weeks of testing data, this dataset would have been ideal for our purposes, if only the test datasets contained attacks relevant to our work.

We solved this issue by synthesizing our own testing dataset containing labeled attacks by merging the attacks from the ECML dataset and the attacks from Dr. Ingham's database into the 1999 MIT Lincoln labs' fourth week dataset (one of the two test datasets). We randomly picked ten attack requests each from the seven categories of the ECML dataset. The attack portion of the query parameter values were then isolated from these requests. We also isolated the attack portion of the query parameter values from the relevant attack queries in Dr. Ingham's database.

Then we replaced the normal query parameter values from some of the queries in the fourth week of the 1999 MIT dataset with the extracted attacks. Figure 9 shows a sample attack request created in this manner. The query parameter *idx* in the sample request shown in Figure 9 had originally contained some alphanumeric character string as its value. That value was replaced with the attack pattern, which marked with a rectangular box in the figure.

```
GET /adverts/imp/c04379/103x62po129109.gif?g=r408&c=a19804&idx=%60+rm++-rf+%2F+%60 HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Host: ads2.xyz.com
Referer: http://www.xyz.com/
User-Agent: Mozilla/2.01 (Win3.1; I;)
```

Figure 9. Sample attack request. The attack portion of the request has been marked with a rectangle.

We modified queries that were relevant to the attacks being performed. For example, if an attack was exploiting a vulnerability in Microsoft’s ASP server, we used this attack to modify a request in the MIT dataset that was addressed to an ASP page (which implies that the server running the ASP page is actually an ASP server). Similarly, if the attack was exploiting a vulnerability in the Apache server, we chose a request from the MIT dataset that was addressed to an application running on an Apache server.

This way we ensured that the attacks we injected into the HTTP requests were actually relevant to the environment of the requests. We also took care to replace normal values with attacks of comparable lengths in most of the cases. This ensured that a majority of the attacks did not get flagged as abnormal simply because of length mismatches. The synthesized dataset contained 1753 normal queries and 149 attack queries in total.

4.2 ROC and ROC Convex Hull Plots

Traditionally, IDS results have been reported using a *receiver operating characteristic* (ROC) curve which indicates the trade-off between identifying genuine attacks (true positives) and incorrectly classifying non-attack requests as attacks (false positives) [4,65]. ROC graphs are suitable for visualizing the performance of the classifiers [66]. The graph is obtained by plotting the false positive rates against the true positive rates. False positive rate is calculated as the fraction of the normal dataset that is incorrectly classified as attack data. True positive rate is calculated as the fraction of the attack dataset that is correctly classified as attack data.

$$\text{False Positive Rate} = \frac{\text{Number of normal requests incorrectly identified as attacks}}{\text{Total number of normal requests}}$$

$$\text{True Positive Rate} = \frac{\text{Number of attacks correctly identified}}{\text{Total number of attack requests}}$$

The ROC Convex Hull (ROCCH) method uses a combination of the methodologies from ROC analysis, decision analysis and computational geometry [67]. The ROC convex hull separates the classifier performance from distributions specific to a class or to the associated costs. It also provides a common ground for the comparison of several classifiers at the same time [67]. We used the ROCCH plot to compare the performance of our 3-Bin character distribution algorithm with that of the 6-Bin character distribution technique.

4.3 Evaluation and Results

The evaluation consisted of two phases: *training* and *testing*. For the training phase, we used the first three weeks of data from the 1999 MIT datasets. Each week in turn, has five datasets, one each for the five weekdays from Monday to Friday. During this phase, the system learnt the characteristics of all the applications that it encountered.

In the testing phase, we switched the system to the detection mode. First, we ran our detection system on the normal data, which did not contain any attacks. All the requests that were classified as anomalous by our system, during this step, were counted as false positives. Next, we ran our detection system on the attack data and all requests classified as anomalous were counted as true positives. Thus, we obtained the true and false positive rates.

4.3.1 ROC and ROC Convex Hull plots of the results

Once we obtained the true and false positive rates, we created the ROC plots. As mentioned in section 4.2, this graph is obtained by plotting the true positive rates vs. the false positive rates. Each point on the plot indicates a different p value (*similarity threshold value*). For the purpose of the evaluation, we created two sets of plots, each set containing three ROC Curves, one each for the length, 3-Bin and 6-Bin Character distribution algorithms.

For the first set of plots, we created the graphs for the true and false positive rates based on the total number of parameter values in the entire dataset. For each parameter value present in all the queries in the dataset, we obtained the p value for the three algorithms. We then used these p values to obtain the true and false positive rates and plotted the curves. Figure 10 shows the ROC plot and the convex hull for the length test.

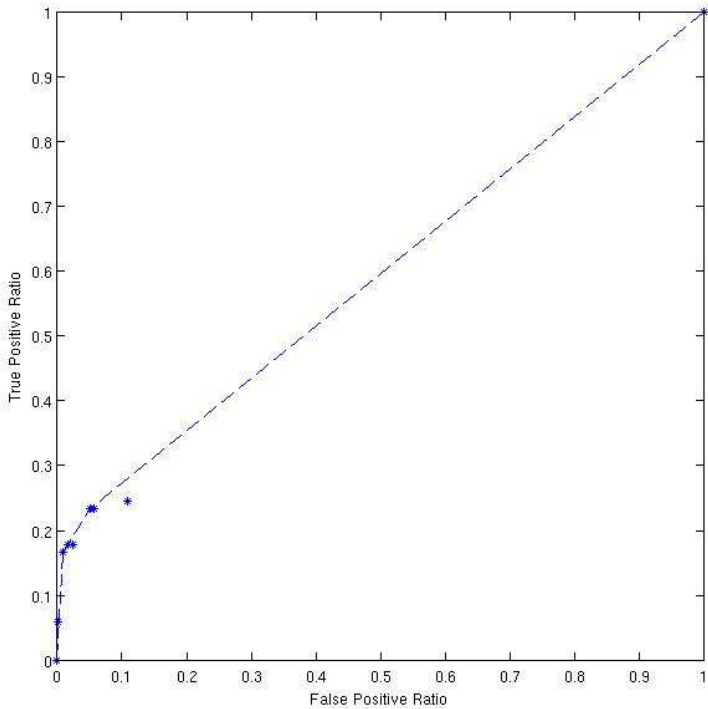


Figure 10. ROC plot and Convex Hull obtained by applying the length test on parameter values

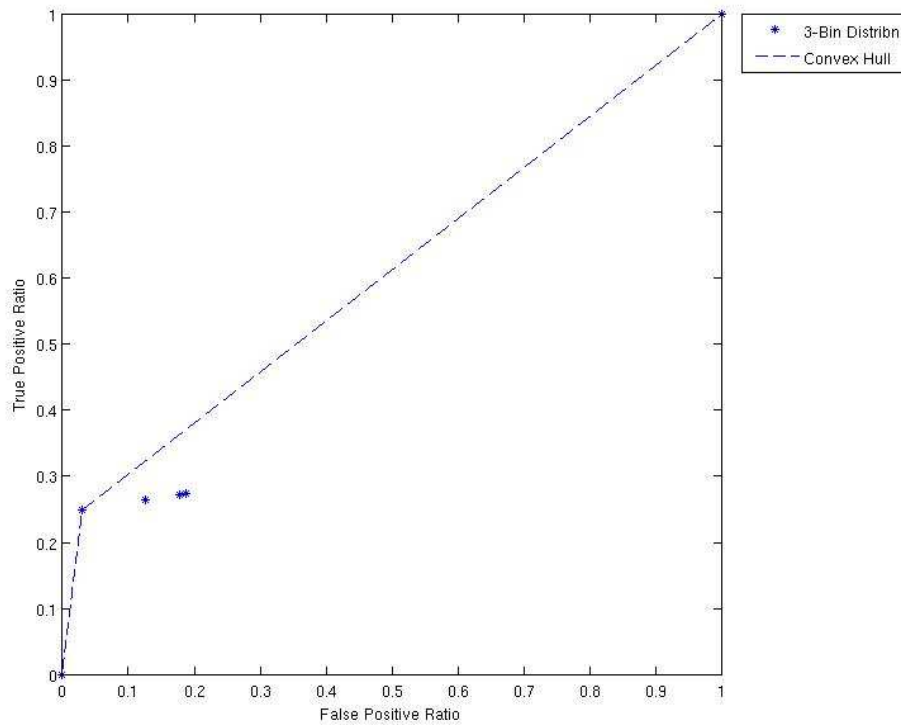


Figure 11. ROC plot and Convex Hull obtained by applying the 3-Bin Character distribution tests on parameter values

Figure 11 shows the ROC plot and its corresponding convex hull for the 3-Bin Character distribution test, while Figure 12 does the same the 6-Bin Character distribution test.

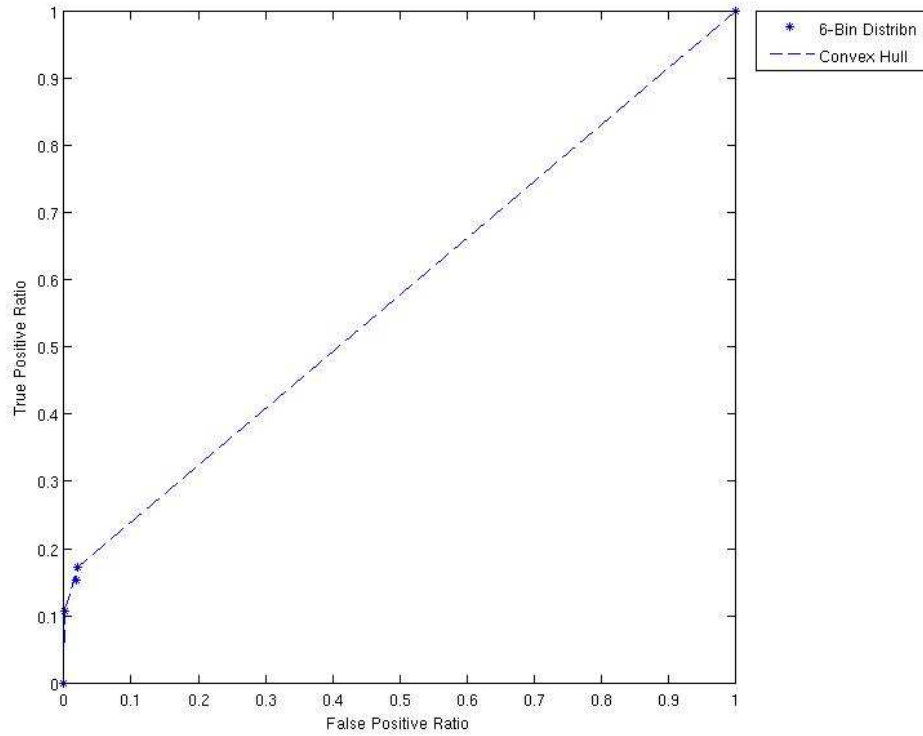


Figure 12. ROC plot and Convex Hull obtained by applying the 6-Bin Character distribution tests on parameter values

In order to compare these two character distributions, we plotted the corresponding convex hulls in a single plot, which is shown in Figure 13.

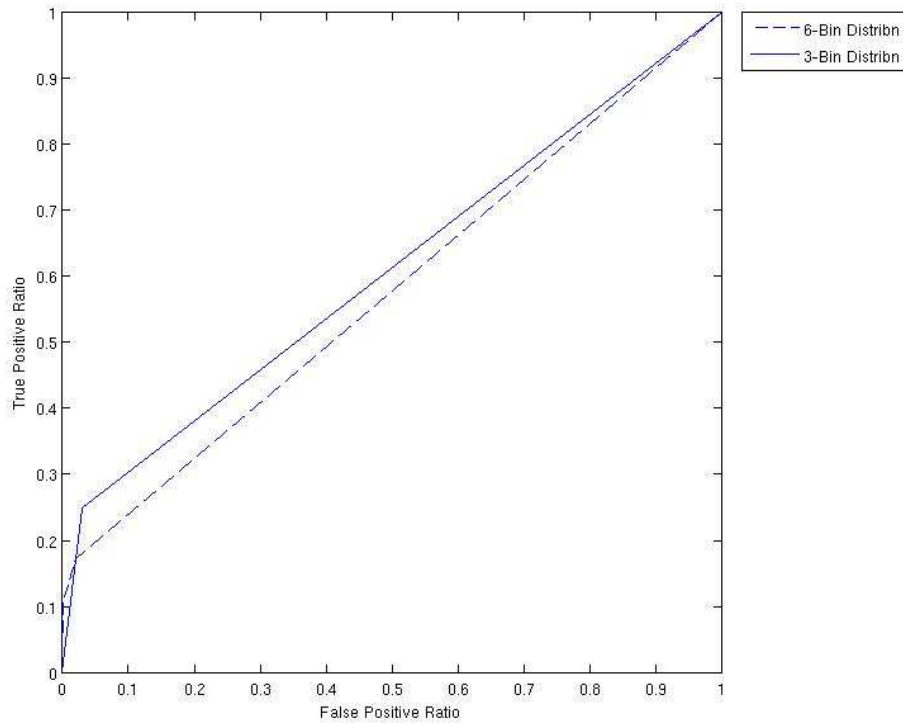


Figure 13. ROC Convex Hull plots obtained by applying the 3-Bin and 6-Bin Character distribution tests on parameter values

Though the anomaly detection system classifies each parameter value as being normal or anomalous using its constituent models, the end result is that the entire request gets classified based on the classification of each of its parameter values. For example, consider a request containing three parameter values and the p values obtained using the length test for these three values are: 1, 0.95 and 0.33. Since 0.33 is the least amongst the three values, the classification of the query as normal or abnormal would depend on whether this value is above or below the threshold value respectively.

Therefore, it is enough if we consider this value alone when classifying the query as a whole. Using this observation, we created a second set of ROC plots, in which the true and false positive rates were based on the number of queries in the dataset. Figure 14 shows this plot and corresponding convex hull for the length test and Figure 15 and 16 for the 3-Bin and 6-Bin character distribution tests respectively.

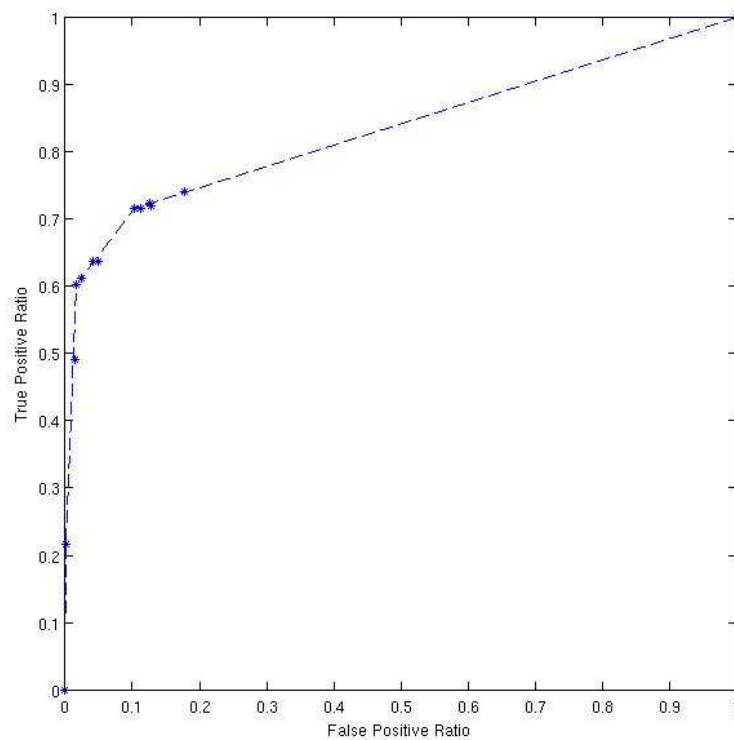


Figure 14. ROC plot and Convex Hull obtained by applying the length test on queries

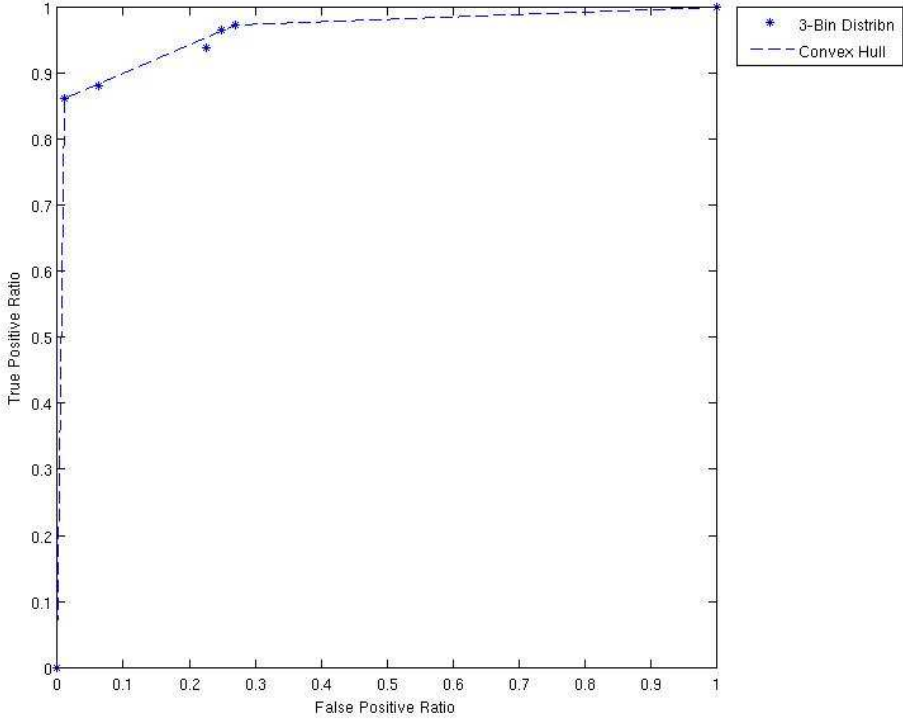


Figure 15. ROC plot and Convex Hull obtained by applying the 3-Bin Character distribution tests on queries

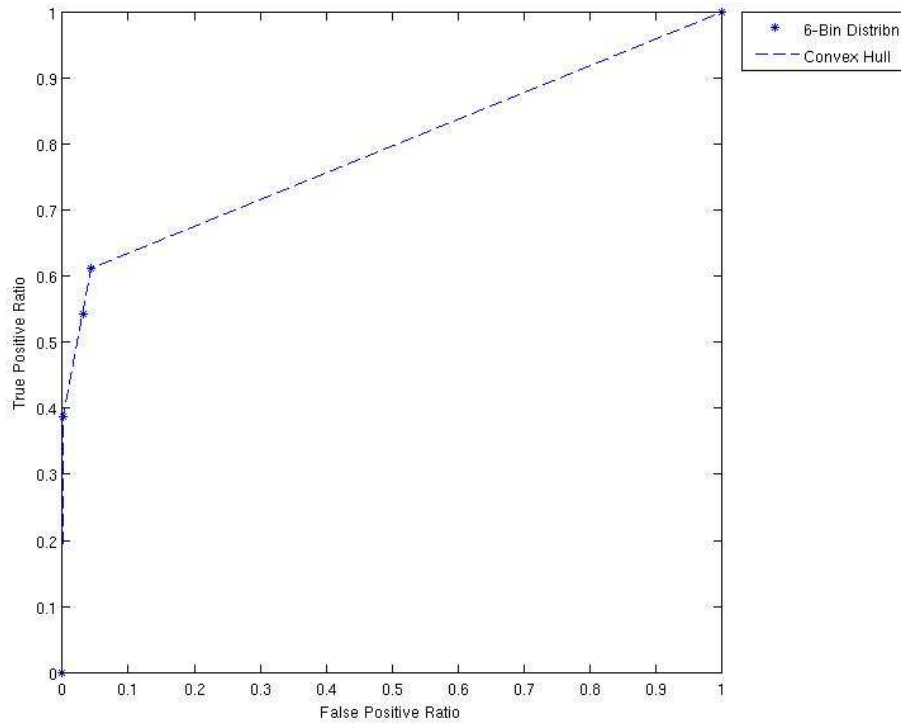


Figure 16. ROC plot and Convex Hull obtained by applying the 6-Bin Character distribution tests on queries

Again, for comparison of the two character distribution tests, we plotted the corresponding convex hulls in a single plot, which is shown in Figure 17.

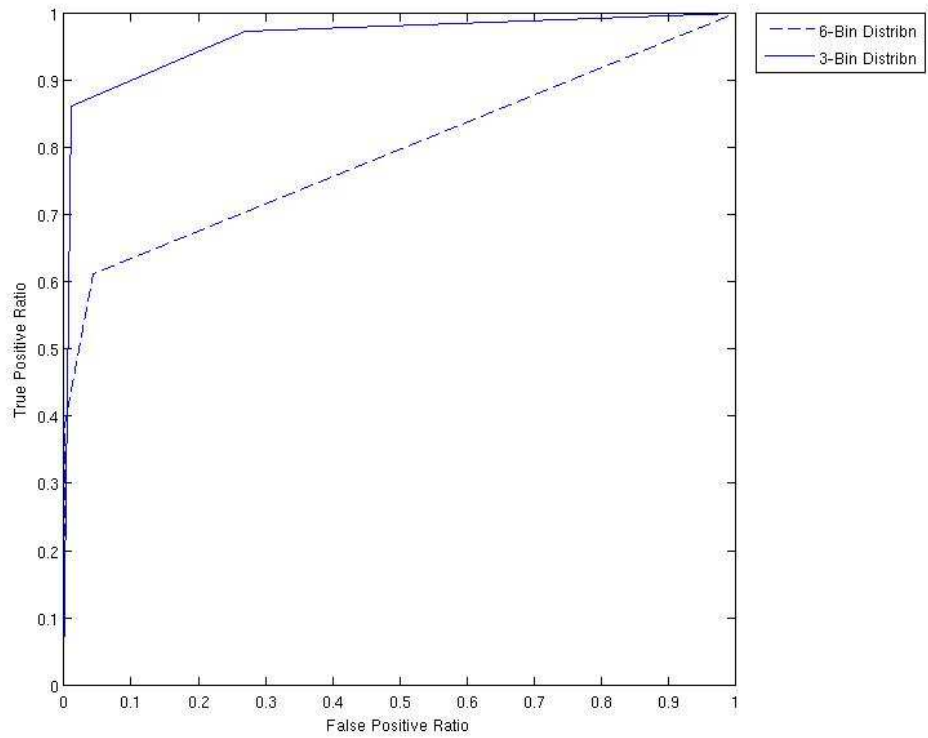


Figure 17. ROC Convex Hull plots obtained by applying the 3-Bin and 6-Bin Character distribution tests on queries

4.3.2 Discussion of the results

From the two ROC plots for the length test (Figures 10 and 14), we see that the true positive rate, at its best, is still less than 0.8, when the (1,1) point is not considered. This implies that the length test has low accuracy. But we use the length test to detect buffer overflow type of attacks, which have huge deviations from the normal values. So, we need not maintain a tight bound on the threshold value used for the length test. By relaxing this threshold, we can bring down false positive rate to an acceptable level, while still retaining the algorithm's ability to detect the major deviations.

From the two ROC Convex Hull plots in Figures 13 and 17, we can notice that except for a very minor region near the (0,0) position, the true positive rate for the 3-Bin character distribution test increases faster than its false positive rate, when compared to the 6-Bin test rates. Also, the 3-Bin test has more area under the ROC Curve when compared to the 6-Bin test. Hence, the 3-Bin test performs better than the 6-Bin test.

4.3.3 Visualization of the identified alerts

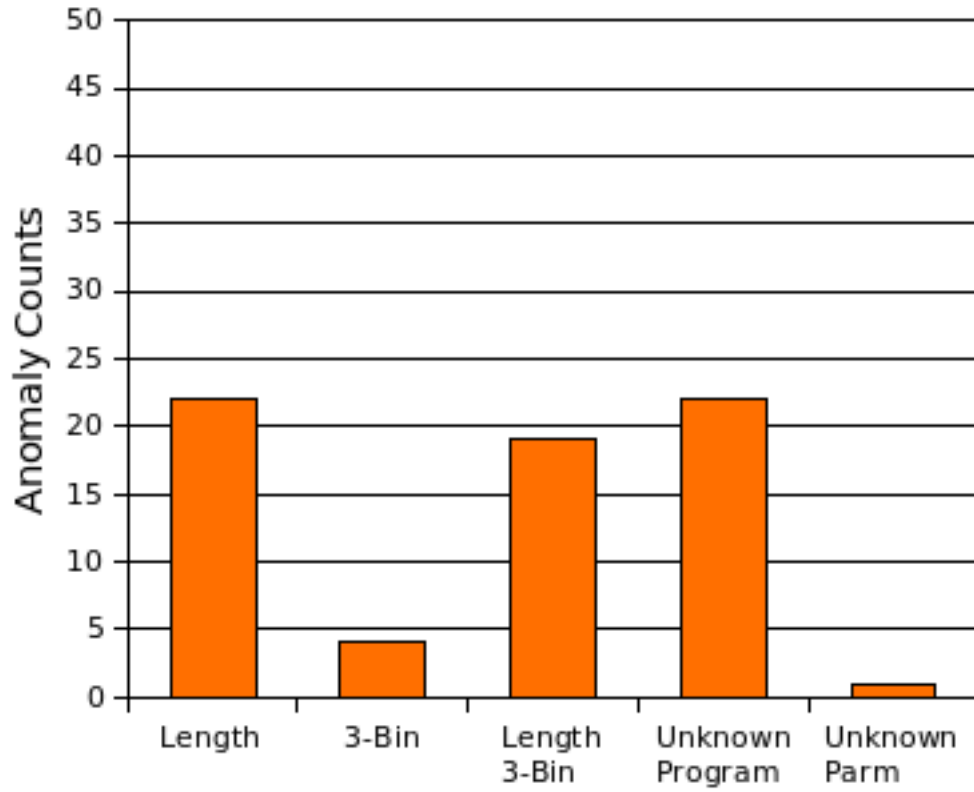


Figure 18. Visualized anomaly counts for Week 4 Friday data. “Unknown Parm” refers to the “Unknown Parameter” test

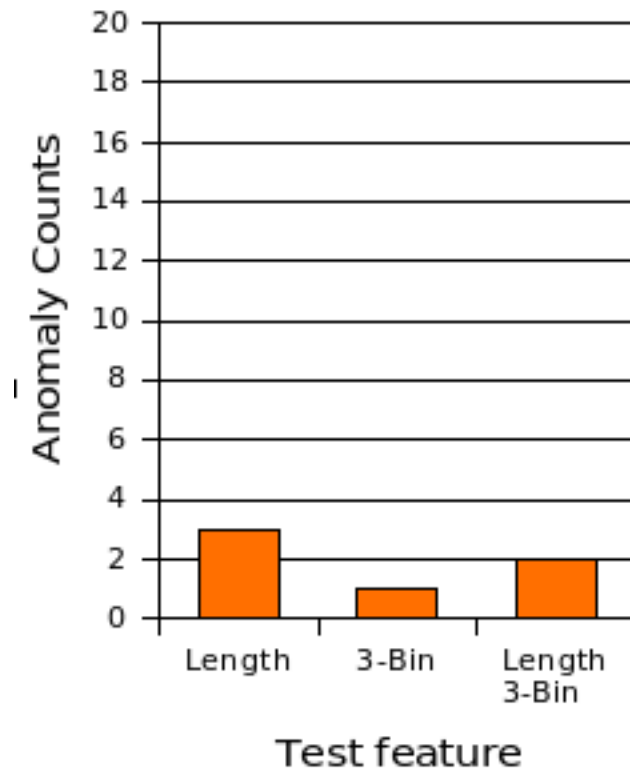


Figure 19. Visualized anomaly counts for a single program in Week 4 Friday data

Figure 16 shows the visualized anomaly counts that we obtained using the anomaly detection system for the Friday dataset of Week 4 (the test week), as bar graphs. The bars represent the absolute counts of the alerts for each of the algorithms. The “Length 3-Bin” bar in the figures represents the count of all parameter values that were flagged as anomalous by both the length and the 3-Bin character distribution tests. Figure 17 shows the visualized anomaly counts for a single program. This visualized representation provides a quick understanding of the current state of the system to the administrator or whoever is monitoring the system.

5. CHAPTER FIVE

This chapter presents the conclusions that were arrived at, as part of this thesis work. It also discusses the future research opportunities that exist in this area, some of which are extensions to the current work.

5.1 Conclusions

Web-application related vulnerabilities/attacks have become a major concern in recent times. This has led to the development of several web-application based intrusion detection systems. The concept of intrusion detection system and its different types were discussed in this thesis. A brief introduction to HTTP, its structure and some HTTP query related attacks were also described.

A web-application based anomaly detection system was proposed and the approach was discussed in detail. The system learns the characteristics of web applications by creating parameter profiles of the associated HTTP queries. It then uses these profiles to monitor the incoming requests and classifies them as benign or anomalous. The system makes use of features that help distinguish anomalous queries from genuine ones, like requests to previously unknown programs, or requests with previously unseen parameters, abnormal length of the parameter values and unusual distribution of symbols/characters in these values. The system learns these characteristics with minimal human intervention. The detection capability and false positive rate can be adjusted to suit the specific needs of the business, by tuning the threshold values used for the detection process.

To evaluate the algorithms, a labeled database of HTTP queries containing training data and attack data was needed. But a good database that contained all of these did not exist. Therefore as part of this thesis work, an evaluation dataset was created. The MIT Lincoln labs 1999 tcpdump dataset [43] was used. The first 3 weeks of data was used as is for training the system. Attack data, obtained from Kenneth Ingham's database of HTTP attacks [60] and the ECML/PKDD 2007 discovery challenge dataset [25], and was

merged with the fourth week's data from the MIT dataset, to get the test data for our system. The results of the experiments have been presented in the thesis. For comparing the performance of the proposed technique, an existing algorithm that performs character distribution based analysis was implemented. With the given dataset, our approach performed better than the existing one. The results of the comparison have been presented. The alerts from the anomaly detection system have been visualized as bar graphs. This helps the system administrator quickly understand the state of the system and enables him/her to easily identify the attacks.

5.2 Future Work

There exists a lot of scope in terms of future research in this area. The algorithms proposed in this work can further be investigated to reduce the number of false positives to the minimum possible. Also, the current number of false positives and success rates indicate that no single technique provides a good rate of detection. Therefore, better detection techniques need to be identified. The current work focuses on HTTP query parameters alone. Future work can examine all portions of the request and response to provide a more complete HTTP based anomaly detection system. Moreover, systems that work on protocols other than HTTP can also be developed. Apart from the detection systems, work can be done to create a large database of attack and non-attack labeled data. This needs to be created with a network setup that reflects the current network configurations as much as possible. This would greatly aid the researchers in understanding the attacks, identifying any patterns and to come up with better detection techniques.

BIBLIOGRAPHY

- [1] Ilgun, K., “USTAT: A Real-time Intrusion Detection System for UNIX”, *Masters Thesis*, Department of Computer Science, University of California Santa Barbara, California, USA, November 1992.
- [2] Securities and Exchange Commission, “Sarbanes-Oxley act of 2002”, *17 CFR* Parts 240, 249, 270 and 274.
- [3] Department of Health and Human Services, “Health insurance portability and accountability act of 1996”, *45 CFR* parts 160 and 164, subparts A and E, <http://www.hhs.gov/ocr/hipaa>
- [4] Ingham, K., “Anomaly Detection for HTTP Intrusion Detection: Algorithm Comparisons and the Effect of Generalization on Accuracy”, *PhD Dissertation*, University of New Mexico, May, 2007.
- [5] Common vulnerabilities and exposures, <http://cve.mitre.org/>
- [6] SecurityFocus, “What is bugtraq?”, <http://online.securityfocus.com/popups/forums/bugtraq/intro.shtml>
- [7] National Institute of Standards and Technology, <http://www.nist.gov>
- [8] Raissi, C., Brissaud, J., Dray, G., Poncelet, P., Roche1, M., Teisseire, M., “Web Analyzing Traffic Challenge: Description and Results”, *Proceedings of the ECML/PKDD 2007 Discovery Challenge*, Warsaw, Poland, September 17-21, 2007.
- [9] SANS Top-20 2007 Security Risks (2007 Annual Update), <http://www.sans.org/top20/>
- [10] @Risk: The Consensus Security Alert, <https://www.sans.org/newsletters/risk/>
- [11] Anderson, J. P. “Computer security technology planning study”, *Tech. Rep. ESD-TR-73-51*, United States Air Force, Electronic Systems Division, Oct. 1972.
- [12] Denning, D.E., “An Intrusion Detection Model”, *IEEE Transactions on Software Engineering*, 13(2), pp. 222–232, February 1987.
- [13] Javitz, H., AND Valdes, A., “The SRI IDES statistical anomaly detector”, *1991 IEEE Computer Society Symposium on Research in Security and Privacy*, 20–22 May 1991, Oakland, CA, USA (Los Alamitos, CA, USA, 1991), IEEE Computer Society Press, pp. 316–326.
- [14] Lunt, T. F., “Detecting Intruders in Computer Systems”, *1993 Conference on Auditing and Computer Technology*, <http://www.sdl.sri.com/papers/c/a/canada93/canada93.ps.gz>
- [15] Warrender, C., Forrest, S., Pearlmutter, B.A., “Detecting intrusions using system calls: alternative data models”, *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, 1999, pp. 133–145.

- [16] Forrest, S., “A sense of self for UNIX processes”, *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1996, pp. 120–128.
- [17] Lee, W., “A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems”, *PhD thesis*, Columbia University, 1999.
- [18] Lee, W., AND Stolfo, S., “A framework for constructing features and models for intrusion detection systems”, *ACM Transactions on Information and Systems Security* 3, 4 (November 2000), pp. 227–61.
- [19] Kruegel, C., Mutz, D., Valeur, F., Vigna, G., “On the detection of anomalous system call arguments”, *ESORICS 2003: 8th European Symposium on Research in Computer Security*, vol. 2808 of Lecture Notes in Computer Science, Springer, 2003, pp. 326–343.
- [20] Kruegel, C., Toth, T., and Kirda, E., “Service specific anomaly detection for network intrusion detection”, *Proceedings of the 2002 ACM Symposium on Applied Computing* (Madrid, Spain, March 11 - 14, 2002). SAC '02. ACM, New York, NY, 201-208.
- [21] Wang, K., AND Stolfo, S. J., “Anomalous payload-based network intrusion detection”, *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004*, Sophia Antipolis, France, September 15-17, 2004. Proceedings (2004), vol. 3224 of Lecture Notes in Computer Science, Springer, pp. 203–222.
- [22] M. Exbrayat., “Analyzing Web Traffic: A Boundaries Signature Approach”, *ECML/PKDD Discovery Challenge 2007*, pages 53–64, Warsaw, Poland, September 2007.
- [23] Pachopoulos, K., Valsamou, D., Mavroeidis, D., Vazirgiannis, M., “Feature Extraction from Web Traffic data for the Application of Data Mining Algorithms in Attack Identification”, *ECML/PKDD Discovery Challenge 2007*, pages 65-70, Warsaw, Poland, September 2007.
- [24] *The 18th European Conference on Machine Learning (ECML) and the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, Warsaw, Poland, September 2007. <http://www.ecmlpkdd2007.org/>
- [25] “Analyzing Web Traffic”, *ECML/PKDD Discovery Challenge 2007*, Warsaw, Poland, September 2007. <http://www.lirmm.fr/pkdd2007-challenge/>
- [26] Kruegel, C., Vigna, G., “Anomaly detection of web-based attacks”, *Proceedings of the 10th ACM conference on Computer and communications security (2003)*, ACM Press, pp. 251–261.
- [27] Kruegel, C., Vigna, G., and Robertson, W., “A multi-model approach to the detection of web-based attacks”, *Comput. Networks* 48, 5 (2005), 717–738.
- [28] Robertson, W., Vigna, G., Kruegel, C., and Kemmerer, R., “Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks”, *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, San Diego, CA, February 2006.

- [29] Foresti, S., Agutter, J., Livnat, Y., Moon, S., Erbacher, R., “Visual correlation of network alerts”, *IEEE Computer Graphics and Applications*, 26(2):48-59, March-April 2006.
- [30] Livnat, Y., Agutter, J., Moon, S., Erbacher, R.F., Foresti, S., “A Visualization Paradigm for Network Intrusion Detection”, *Proceedings of the IEEE Workshop on Information Assurance and Security (IAW)*, pp. 92-99, 15-17 June 2005.
- [31] Kenneth Cox, S. E., and He, T., “3d geographic network displays”, *ACM Sigmod Record*, vol. 25(50), December 1996.
- [32] D. Estrin, M. Handley, J. Heidemann, S. McCanne, Y. Xu, and H. Yu, “Network visualization with nam, the vint network animator”, *IEEE Computer*, vol. 33, pp. 63–68, November 2000.
- [33] Polla, D., McConnell, J., Johnson, T., Marconi, J., Tobin, D., Frincke, D., “A framework for cooperative intrusion detection”, *21st National Information Systems Security Conference*, pp. 361–373, October 1998.
- [34] Kim S.S., Reddy, A.L.N., “Image-based Anomaly Detection Technique: Algorithm, Implementation and Effectiveness”, *IEEE Journal on Selected Areas in Communications-High Speed Network Security (J-SAC)*, vol. 24(10), pp 1942-1954, October 2006.
- [35] Axelsson, S., “The Base-Rate Fallacy and the Difficulty of Intrusion Detection”, *ACM Transactions on Information and System Security (TISSEC)*, 3(3), pp. 186–205, ACM Press, 2000.
- [36] Axelsson, S., “Visualising Intrusions: Watching the Webserver”, *Proceedings of the 19th IFIP International Information Security Conference (SEC2004)*, Toulouse France, 22–27 Aug, 2004.
- [37] Axelsson, S., “Combining A Bayesian Classifier with Visualisation: Understanding the IDS”, *Proceedings of the ACM CCS Workshop on Visualization and Data Mining for Computer Security*, (Held in conjunction with the *Eleventh ACM Conference on Computer and Communications Security*), Washington DC, USA, pp. 99-108, Oct 29, 2004.
- [38] Axelsson, S., “Visualising the Inner Workings of a Self Learning Classifier: Improving the Usability of Intrusion Detection Systems”, *Technical report 2004:12, Department of Computing Science, Chalmers University of Technology, Goteborg, Sweden*, 2004.
- [39] Axelsson, S., “Visualization for Intrusion Detection: Hooking the worm”, *Proceedings of the 8th European Symposium on Research in Computer Security (ESORICS 2003)*, Springer Verlag: LNCS 2808, 13–15 Oct, Gjovik Norway, 2003.
- [40] Axelsson, S., “Understanding Intrusion Detection Through Visualisation”, *PhD Dissertation*, Department of Computer Science, Chalmers University of Technology, Goteborg, Sweden, 2005.

- [41] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., “Hypertext transfer protocol HTTP/1.1”, *RFC 2616*, June 1999, <ftp://ftp.isi.edu/in-notes/rfc2616.txt>
- [42] Hypertext Transfer Protocol, Wikipedia, <http://en.wikipedia.org/wiki/HTTP>
- [43] 1999 DARPA Intrusion Detection Evaluation Dataset, MIT Lincoln Laboratory. http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html
- [44] List of HTTP status codes, Wikipedia, http://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- [45] Danyliw, R., Dougherty, C., Householder, A., Ruefle, R., *CERT advisory CA-2001-26 Nimda worm*, September 2001, <http://www.cert.org/advisories/CA-2001-26.html>
- [46] Nimda, variant 10, <http://www.i-pi.com/HTTP-attacks-JoCN-2006/10/>
- [47] *CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests*, <http://www.cert.org/advisories/CA-2000-02.html>
- [48] CVE.MITRE.ORG., *CVE-2001-0731*, June 2002, <http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0731>
- [49] Porras, P.A., “STAT: A State Transition Analysis Tool for Intrusion Detection”, *Masters Thesis*, Department of Computer Science, University of California Santa Barbara, California, USA, July 1992.
- [50] Ingham, K., Forrest, S., “A history and survey of network firewalls”, *Technical Report TR-CS-2002-37*, University of New Mexico Computer Science Department, 2002. http://www.cs.unm.edu/colloq-bin/tech_reports.cgi?ID=TR-CS-2002-37
- [51] Ingham, K., Forrest, S., “Network firewalls”, *Enhancing Computer Security with Smart Technology (2006)*, Vemuri, V.R., Ed., Auerbach Publications, pp. 9-40. Chapter 2.
- [52] SOAP, Wikipedia, 2006. <http://en.wikipedia.org/w/index.php?title=SOAP>
- [53] Wiers, D., “Tunneling SSH over HTTP(S)”, 2006. <http://dag.wieers.com/howto/ssh-http-tunneling/>
- [54] Apple Computer, “Tunneling RTSP and RTP over HTTP”, 2006. http://developer.apple.com/documentation/QuickTime/QTSS/Concepts/chapter_2_section_14.html
- [55] Microsoft Corporation, “Exchange server 2003 RPC over HTTP deployment scenarios”, 2006. <http://www.microsoft.com/technet/prodtechnol/exchange/2003/library/ex2k3rpc.mspx>
- [56] Tombini, E., Debar, H., L. Me, L., Ducasse, M., “A serial combination of anomaly and misuse IDSes applied to HTTP traffic”, *20th Annual Computer Security Applications Conference*, 2004.

- [57] Estevez-Tapiador, J. M., Garcia-Teodoro, P., Diaz-Verdejo, J. E., “Measuring normality in HTTP traffic for anomaly-based intrusion detection”, *Comput. Networks* 45 (2004), 175-193.
- [58] Ingham, K., Somayaji, A., Burge, J., Forrest, S., “Learning DFA representations of HTTP for protecting web applications”, *Comput. Networks* 51, 5 (2007), 1239-1255.
- [59] Ingham, K., Inoue, H., “Comparing anomaly detection techniques for HTTP”, *Proceedings of the 2007 Conference on Recent Advances in Intrusion Detection*, 2007, pp. 42-62.
- [60] HTTP-delivered attacks against web servers. <http://www.i-pi.com/HTTP-attacks-JoCN-2006/>
- [61] PowerScripts PlusMail WebConsole Poor Authentication. <http://www.i-pi.com/HTTP-attacks-JoCN-2006/54/>
- [62] Billingsley, P., “Probability and Measure”, Wiley-Interscience, 3rd edition, April 1995.
- [63] Open Web Application Security Project, “Buffer Overflow”, http://www.owasp.org/index.php/Buffer_Overflow
- [64] Kendall, K., “A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems”, *Masters Thesis*, Massachusetts Institute of Technology, 1998.
- [65] Hancock, J., and Wintz, P., “Signal Detection Theory”, McGraw-Hill, 1966.
- [66] Fawcett, T. 2006. “An introduction to ROC analysis”, *Pattern Recogn. Lett.* 27, 8 (Jun. 2006), 861-874.
- [67] Provost, F., and Fawcett, T., “Robust Classification for Imprecise Environments,” *Machine Learning*, vol. 42/3, pp. 203-231, 2001.
- [68] Tcpcdump. <http://www.tcpcdump.org/>
- [69] World Wide Web Consortium (W3C), "Extensible Markup Language (XML)", <http://www.w3.org/XML/>