A REVERSE PROXY FOR PROTECTING WEB BASED APPLICATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZGÜR ŞANLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

SEPTEMBER 2003

Approval of the Graduate School of Natural and Applied Sciences.

_____

Prof. Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Ayşe Kiper

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Dr. Attila Özgit

Supervisor

Examining Committee Members:

Assoc. Prof. Dr. Ali Doğru _____

Asst. Prof. Dr. Kürşat Çağıltay _____

Dr. Attila Özgit _____

Dr. Onur Tolga Şehitoğlu _____

Burak Dayıoğlu _____

ABSTRACT

A REVERSE PROXY FOR PROTECTING WEB BASED APPLICATIONS

Şanlı, Özgür

M. Sc., Department of Computer Engineering

Supervisor: Dr. Attila Özgit

September 2003, 73 pages

Web based applications started to be an integral part of ordinary people's life. They depend on them in many areas from business to education. Web based applications are open to the public; i.e. anybody on the Internet can access them without any limitations. While developing web applications, the main aim, most of the time, is to produce an application that meets the operational requirements. The security problems of the web applications are generally tried to be solved by the use of network level security products and access control mechanisms, but this is not enough. Security should be considered at all levels including the application level. In this thesis we propose a method to enforce the clients to use the application in a way that is consistent with the logic of the web application. A reverse proxy sitting in front of the web server provides protection against threats

towards web-based applications without any requirement for the configuration change in the web server.

Key Words: Web application security, web application proxy, web application firewall, reverse proxy

ÖZ

WEB UYGULAMALARINI KORUMAK İÇİN

TERSİNE ARA SUNUCU KULLANIMI

Şanlı, Özgür

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Dr. Attila Özgit

Eylül 2003, 73 sayfa

Web tabanlı uygulamalar normal insanların yaşamlarının önemli bir parçası haline gelmeye başlamıştır. İş yaşamından eğitime kadar birçok alanda bu uygulamalara bağımlı hale gelmişlerdir. Web tabanlı sistemler, İnternet üzerinde bulunan herkesin bir kısıtlama olmaksızın kullanabileceği sistemlerdir. Web uygulamaları geliştirilirken temel olarak, işletimsel gereksinimleri karşılayan bir uygulamanın üretilmesi hedeflenmektedir. Web uygulamalarının güvenlik problemleri ise, ağ katmanında çalışan güvenlik ürünleri ve erişim denetleme mekanizmaları kullanılarak çözülmeye çalışılmaktadır. Fakat bu yeterli değildir ve güvenlik, uygulama katmanını da kapsayacak şekilde tüm katmanlar için düşünülmelidir. Bu tez ile kullanıcıları, web uygulamasının gerektirdiği şekilde kullanmaya

zorlayan bir yöntem önerilmektedir. Web sunucularının önünde bulunan tersine çalışan bir ara sunucu, web sunucusu üzerinde herhangi bir değişiklik yapılmaksızın web uygulamalarını tehdit eden problemlere karşı koruma sağlamaktadır.

Anahtar Kelimeler: Web uygulama güvenliği, uygulama aracısı, web uygulama güvenlik duvarı, tersine ara sunucu

ACKNOWLEDGMENTS

I would like to thank to my supervisor Dr. Attila Özgit for his guidance in this study. I would also like to thank Burak Dayıoğlu for his innovative ideas while defining the thesis topic and for his support throughout the study. I also express my gratitude to my friend Hayrettin Genel on his feedback on the manuscript.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

One of the most important inventions in the twentieth century, possibly in the history of human kind, was the computer. It has enabled many other technological advancements possible. Genetic and astronomical research activity can be given to these advancements. But it is the Internet that has the most influence in ordinary people's life. It can be viewed as the collection of interconnected computers and networks.

Internet started as a research project and the Internet protocol suite, namely TCP/IP was developed by Defense Advanced Research Projects Agency (DARPA). Initial networks were academic research networks and the main aim was to enable the sharing of information between the agencies that were involved in the networks.

Until the beginning of 1990s, the Internet was mostly used in the academic environments. After then, however, it started to reach the home users and has

evolved into a huge internetwork of millions of computers and networks. Especially for the last few years it has gained so importance that it changed how people live, how they learn, how they do business etc.

One of the most important factors behind the popularity of the Internet is the World Wide Web also called web or www. It started as a research project at the European Center for Nuclear Research (CERN) in 1989. With the invention of World Wide Web, sharing of information has become much more easier. The documents contain links to the other related documents and user only have to follow this link to access the information without actually knowing where it resides. With the development of Mozilla in 1993, not only the textual information but also graphical data could be presented to the users. Today people use web to share anything they want including audio, video and graphics.

As it is the case in the Internet, the web has gone beyond its initial aims, which was sharing of information among dispersed particle physicists. Nowadays it is possible to see an "e" in front of the words of various subjects: e-business, e-banking, e-government, e-university, e-learning etc. As it is stated before, the Internet, web in particular, has changed people's life in many ways. Today many people prefer to buy from the online stores. Government and banking services started to be given from the Internet. Students can graduate from online universities.

Every major organization has a web site. According to a research the number of web sites has exceeded 42,5 million [28]. Companies see the web as a business

enabler. It will probably not be wrong to say that companies without a web site will not seriously be taken into account by the customers today.

TCP/IP protocol suite was not designed security in mind. At the beginning, the main aim was to establish a network where transferring files, sending e-mails, and sharing printers were enough. Since the protocols were implemented for the use of a limited set of people, it was assumed that the communicating peers could trust each other. However the TCP/IP protocol suite has inherent problems [6, 20]. It has not met the security requirements of contemporary networks anymore because it is unrealistic to assume such a trust relationship; nobody trusts anyone else. On the other hand, redesigning and re-implementing a much more secure network protocol stack is not an easy task especially if we consider the millions of connected computers all over the world.

Together with the incredible growth of the Internet, the number of security incidents has been increasing too. Figure 1.1 shows how the number of incidents reported to the CERT/CC has changed over years [10]. As it can be seen from the figure, the number of incidents in the first half of 2003 is almost the same as the number of incidents reported in 2002, which is quadruple of the number of incidents reported in 2000.

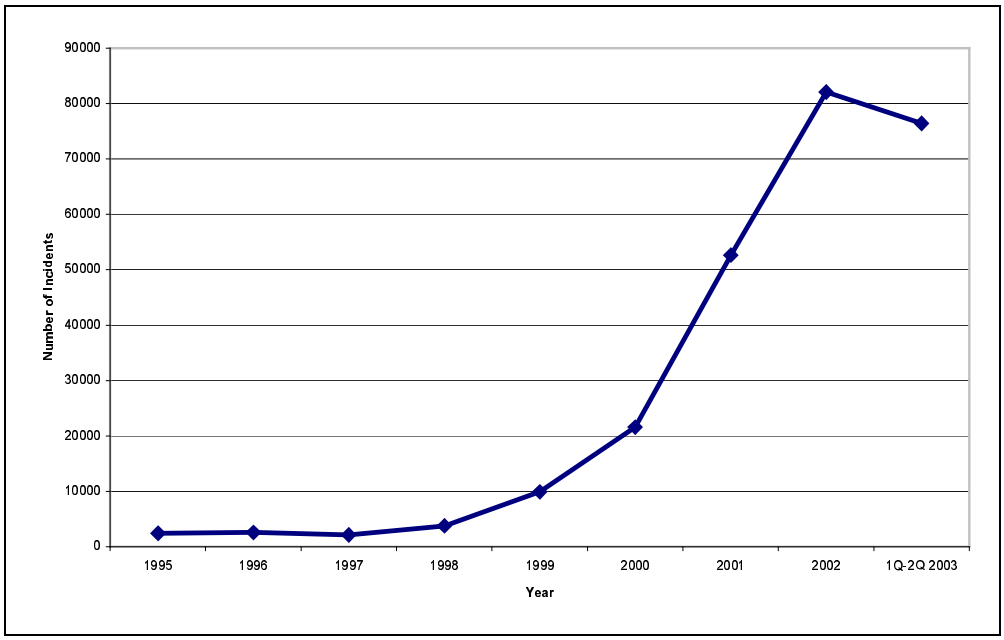

Figure 1.1: Number of Incidents Reported to CERT/CC

The increase in the number of incidents can be bound to the software toolkits developed by knowledgeable people. Today's average attackers do not have the knowledge or experience as the old attackers used to have. But this does not make them less harmful. They use the mentioned toolkits to launch complex attacks within short amounts of time. Figure 1.2 shows the relation between the expertise of the attacks and intelligence of the attackers.

A recent survey conducted by Computer Security Institute (CSI) and Federal Bureau of Investigation (FBI) reveals that 56 percent of the respondents have experienced unauthorized use of their computer systems within the last twelve months [12]. 15 percent of them do not know if they were faced with such an incident.

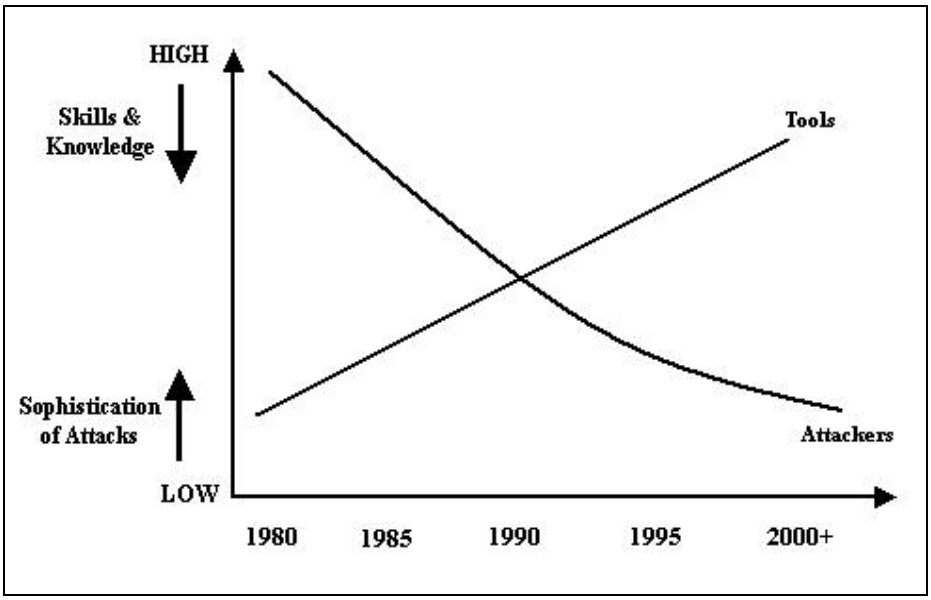

Figure 1.2: Intelligence of Hackers vs. Expertise of Attacks

For the last few years, attackers' interest has moved towards the web-based systems because the web services are open to the public which means people can access it without being hindered by other security technologies like firewalls. Most of the time, intruders do not need to have any tools other than a web browser and a text editor in order to launch attacks against web applications. SANS/FBI Top20 List includes the twenty most critical Internet security vulnerabilities for Windows and UNIX systems [36]. The web server vulnerabilities are at the top of this list. Table 1.1 gives the complete SANS/FBI Top 20 list. According to the analysis of SecurityFocus, five of the top ten threats in 2001 targeted the vulnerabilities in the web-based systems [16].

Attacks targeting web-based systems cause sensation. Even the most simple web defacements result in significant time and money loss [4].

Table 1.1: SANS/FBI Top 20 List

Top Vulnerabilities for Windows Systems	*Top Vulnerabilities for UNIX Systems*
Internet Information Services (IIS)	Remote Procedure Calls
MDAC – Remote Data Services	Apache Web Server
Microsoft SQL Server	Secure Shell (SSH)
NETBIOS – Unprotected Windows Networking Shares	Simple Network Management Protocol (SNMP)
Anonymous Logon – Null Sessions	File Transfer Protocol (FTP)
LAN Manager Authentication – Weak LM Hashing	R-services – Trust Relationships
General Windows Authentication – Accounts with weak/no passwords	Line Printer Daemon (LPD)
Internet Explorer	Sendmail
Remote Registry Access	BIND/DNS
Windows Scripting Host	General Unix Authentication – Accounts with weak/no passwords

In a web-based system, an attacker can attack the web application or the web server itself. Organizations generally do not have control over the web server software. What they ought to do is to configure it according to the requirements, follow the security advisories, and apply the necessary patches. In contrast, web applications are usually developed or adapted by the organizations that operate them. There might be several reasons behind an insecure web application. Time limitation is one of the factors. In order to fulfill deadline requirements,

applications are developed in hurry. Another factor is the developers' knowledge about secure programming practices. If developers do not follow secure programming practices, the resulting application might be vulnerable even to the simplest attacks.

Increasing number of threats and incidents has awakened the organizations about the importance of the security of information systems. Starting from the mid 1990's, organizations employed several technologies in order to protect their networking infrastructures. In figure 1.3, the percentages of respondents of the CSI/FBI Computer Crime Survey 2003 that use various security technologies are shown [12]. As it can be seen, almost every organization uses the firewall (98 percent) and the anti-virus software (99 percent). Physical security measures (91 percent) and access control mechanisms (92 percent) have commonly employed too. Around 75 percent of the organizations use Intrusion Detection Systems (IDS).

In order to protect web applications, the security technology being employed should have application level intelligence. Most of the currently employed security technologies lack this and they will not help in preventing or in some cases detecting the application level attacks.

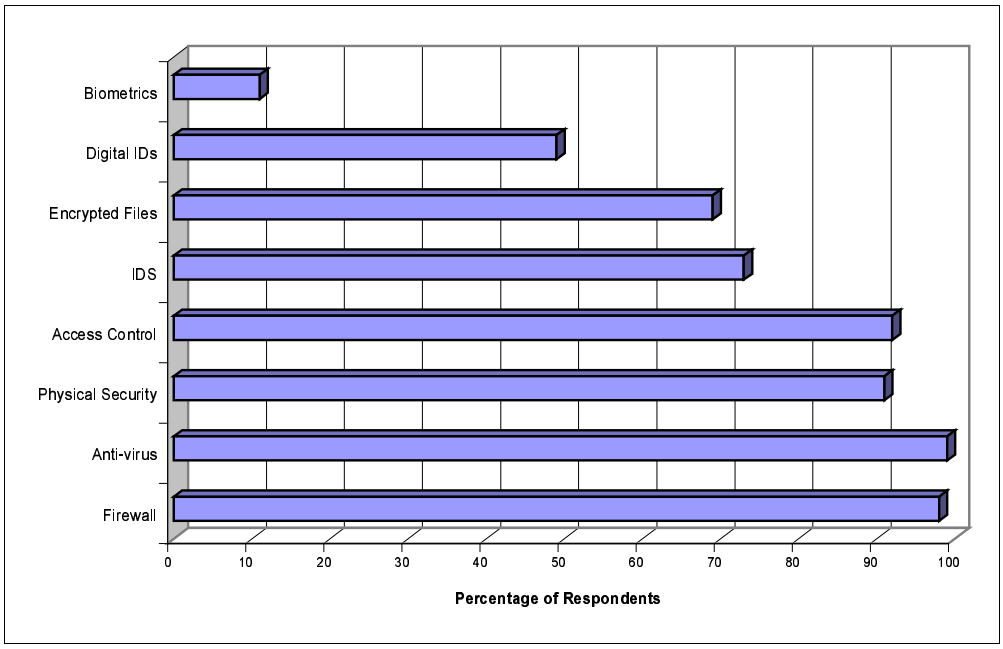

Figure 1.3: Security Technologies Used

1.1 Web Applications

The Open Web Application Security Project (OWASP) defines a web application as a "*software application that interacts with users and other systems using HTTP*" [41]. Not only the HTTP protocol but also other web protocols like HTTPS, HTML are used. The end users interact with the system by sending their choices. Both the sending input to and receiving output from the system are via web.

Figure 1.4 outlines three components of a typical web application [24]:

- Front-end Web Server
- Web Application Execution Environment

- Database Server

These components might be on the same server. Similarly a component can be on different servers.

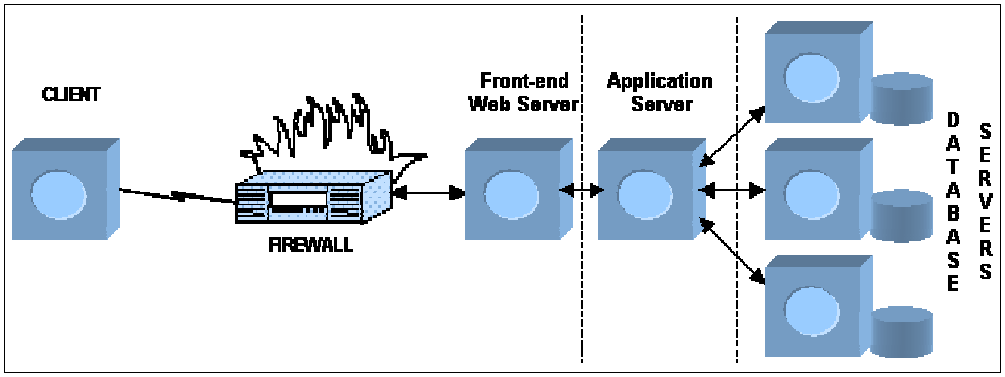

Figure 1.4: Components of a Web Application

Front-end web server is the component that is responsible for direct communication with the clients. It receives HTTP requests from the clients and sends the responses back to them. Two of the most popular front-end web servers are Apache Web Server and Microsoft IIS Web Server. They mainly serve static pages and forward client input to the application in order to be processed. Although some of today's web servers have dynamic scripting capabilities, they will be insufficient to serve big applications like banking applications by themselves alone.

Application Execution Environment is the platform used for writing custom applications that get their input mainly through the use of HTML forms. It has

capabilities to interface with the front-end web server and the database server. The result of the web application will be a dynamically generated HTML page. The application execution environment can be an extension to the front-end web server like PHP [32] or Active Server Pages [2] or it might be a specific web application platform.

Database servers are used to keep the application data. The application data is generally the most important asset and the web application should protect and process it in a secure manner. Most of the database servers are mature and they are specialized in this field. Therefore, although there is not any obligation to store application data inside databases, using them would be a wise choice.

1.2 Thesis Goals

There are many incidents regarding web-based systems and most of today's security technologies that are commonly employed are not enough to meet the security requirements of these systems because either they do not work at the application layer or they do not have application specific information.

The goal of this thesis is to identify the threats that target web-based systems, and design and implement a reverse proxy system that will protect the web-based systems against these threats. In their paper, David Scott and Richard Sharp state that security of applications is so important and the responsibility of it cannot be left to the individual coders [38]. Most of the attacks against web applications are possible because of application developers' insecure coding practices. If there

were a mechanism forcing the clients to follow the logic of the application, then there would not be much of today's problems.

Our approach to protecting web applications is through the development of an intelligent reverse proxy system. The reverse proxy stands in front of the web servers and after analyzing the requests, it either allows the requests to pass to the actual web servers or blocks them before reaching the actual servers depending on the evaluations of the request and previously stored state information of the requester. While analyzing, the proxy looks if the request is suitable to the normal logical flow of the application. In the rest of the thesis, the terms web application firewall and reverse proxy will be used interchangeably.

1.3 Thesis Structure

The rest of the thesis is structured as follows.

Chapter 2 – *Threats for Web-Based Systems.* This chapter identifies the major threats for the web-based systems. The causes of these threats are discussed in detail and the solutions that can be applied to get rid of those threats are given.

Chapter 3 – *Approaches and Design.* This chapter starts with the specifications of the web application firewall. Then it presents the details of issues related to the design of the system.

Chapter 4 – *Implementation of the Application Level Reverse-Proxy.* In this chapter, issues related to the implementation are given.

Chapter 5 – *Tests and Performance.* In this chapter, the tests applied to the implementation prototype are presented. The quality and the runtime performance of the implementation for the conducted tests are given in detail.

Chapter 6 – *Conclusions and Future Work.* This chapter gives a summary of the thesis study together with the further work that might be built upon this study and the final conclusions.

CHAPTER 2

THREATS FOR WEB BASED SYSTEMS

Because of its ubiquity, most of the attacks focus on the web based systems nowadays. These kinds of attacks are so popular that many weaknesses related to them are posted to security oriented mailing lists everyday. Most of the web-based applications employed today consist of various interconnected and dependent components that make the overall architecture too complex. In a complex environment, the control of the overall system can be lost easily which causes much more mistakes to be made. As a result, the probability of the exposure to the attacks is increased. Due to all of these, it is probable that more of those weaknesses will be seen in the future.

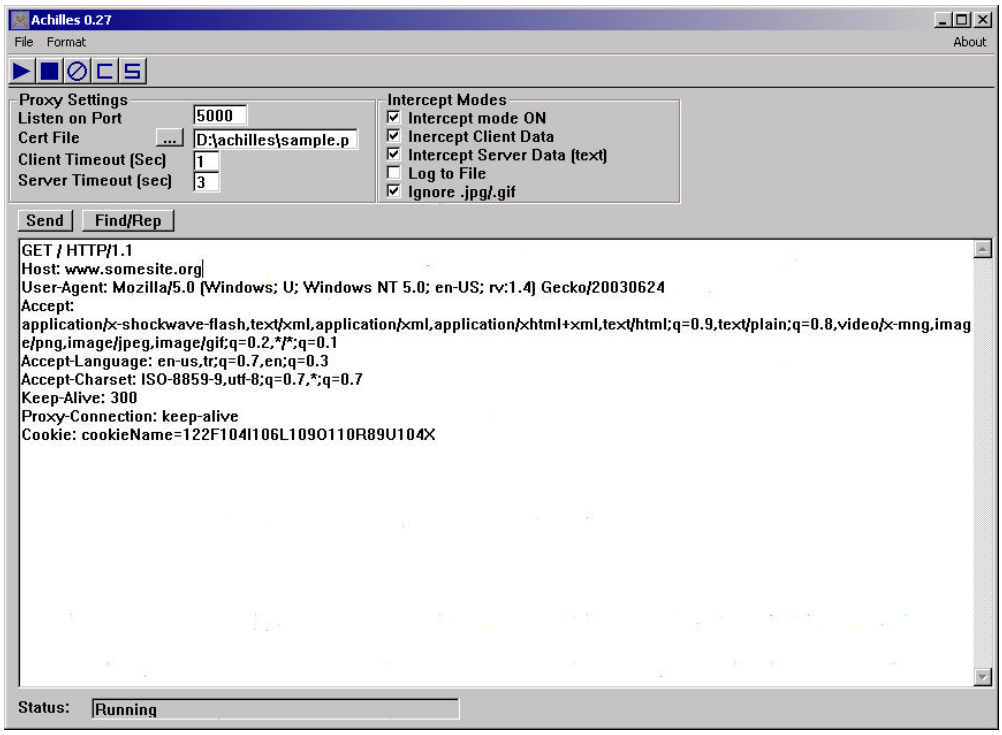

Figure 2.1: Achilles Proxy

There are various threats that web based systems face today. Invalidated parameters, improper session management and misconfigurations are the main causes of most of the weaknesses in web applications. In order to launch attacks against web-based applications, use of a web browser and a text editor is enough for most of the time but there are some tools that ease the job of the attackers as well. Achilles, a client-side proxy, can be given as an example to such a tool [1]. It gives the opportunity to the user to intercept any incoming or outgoing web traffic and modify the contents of the HTTP packets. Figure 2.1 gives a snapshot of the Achilles proxy.

In the following sections, some of the important threats for web-based applications will be discussed.

2.1. Cookie Poisoning

Once authenticated, clients generally interact with different parts of the web application and request multiple jobs to be done by that application. In order to do that, the client generally makes multiple requests to the web system. Separating the requests of a client from the requests of other clients cannot be done with the capabilities of the standard HTTP protocol itself. HTTP is a stateless protocol, which means the server handles every request independently. In order to keep the identity of the client, or session in more general terms, the web application should employ other mechanisms to keep track of the session information. There are a few popular methods that application developers use to keep track of sessions: Cookies, Hidden Form Fields and URLs [19].

Cookies are the most popular method used to keep session information. Specification of cookies is given in [23]. With an extension to the standard HTTP protocol, a cookie field is appended to the HTTP headers [17]. The logic behind it is simple: Web server sends a piece of information to the client and wants this information back when the client sends subsequent requests [17, 23, 29]. This idea is devised by Netscape Inc. Web browsers do the manipulation of the cookies at the client side. The browser simply appends the cookie to accompany the client's request.

There are two types of cookies [15]: persistent cookies and non-persistent (temporary) cookies. Non-persistent cookies are stored on memory and valid while the web browser runs. Certainly they can be removed or set to other values by the web application during that period. Persistent cookies on the other hand are stored in a file on the disk and available even after the web browser is stopped and restarted. They contain an expiration date and are valid until that date.

Cookies are not something that can be executed on the computer [15]. So regardless of what they contain, they cannot directly harm the computer. But they might contain sensitive information about the web client (personal information, account id or passwords, session identifiers etc.), which is the case in general. If somebody else captures it, this will definitely break the privacy of the client. The consequences of the disclosure of this information are proportional to the information contained in the cookie and the importance of the web application that set the cookie. For example, if a cookie is set by a banking application to keep track of identities of the clients, an attacker guessing the contents of the cookie will be able to start banking transactions with the rights of the actual user owning that particular cookie.

Although ordinary end users leave the manipulation of cookies to the web browsers, nothing forbids a curious or mal-intended client to interfere directly with them. By modifying the contents of a cookie, an attacker may assume the identity of another client, which is called session hijacking or impersonation attack [42], and can perform tasks on behalf of that client as in the banking example given above. There are various ways that an attacker can capture the

cookies. He may use sniffers to capture them on the wire, steal them from the shared workstations, use cross-site scripting attacks that will be discussed in section 2.3, or simply by guessing.

The values of the cookies should not be easily guessed or should not follow a specific pattern. They should be totally random. Persistent cookies should not be used and even if non-persistent cookies are used, they should be valid for only a small duration. The values of the cookies should not contain any meaningful data like account id, address, credit card number, etc. Furthermore, the contents should be encrypted if possible. The origin of the clients should be checked if they try to access with the right cookies [15].

2.2. Form Field Manipulation

Web applications need to interact with the clients. HTML forms are the most popular method used for this purpose. The client-supplied input is passed in the form input field parameters to the web application. After that, the web application processes the user data.

The specifications of HTML forms are given in [35]. The general format of them is as follows:

```
<form method=METHOD action=ACTION>
      <input type=TYPE name=NAME other_input_options>
      . . .
      <input type=TYPE name=NAME other_input_options>
</form>
```

Attackers may easily play with the form input field parameters and launch form tampering attacks [24, 37, 38]. They do not need any special tool to manipulate the HTML forms. They only need to save the page that contains the form fields onto the hard disk, modify them according to their needs using a standard text editor, reload the page into the web browser and finally send the request to the web application. Web application developers sometimes deploy client-side validation code into the web pages. JavaScript code that checks the size or the type of the supplied input can be given as an example to this. Most of the time an additional server-side check is not done, because of either the performance considerations or the lack of knowledge. But it is easy for somebody to disable the client-side checks.

One of the most interesting form fields is the “hidden form field”. Hidden form fields are not seen in the browser windows but they are present in the HTML source. So they are hidden from the screen display. They can be used to carry various things like session identifiers, preferences of the clients gathered previously, or any other information that web application wants to carry from one state to another. If the field contains session identifiers, then by changing it, the client may assume the identity of another client. A problem with hidden form fields in shopping chart applications is given in [18]. In web based shopping chart applications, hidden form fields might be used to carry the count and price of an online store’s item that a client wants to buy. If form field parameters are not checked properly on the server-side, an attacker can order the items for free by changing the price of the item in the hidden form field.

Form fields contain a number of parameters. One of them is the maximum length parameter that specifies the maximum length of the input that will be fed into the web application. If an attacker changes this parameter and supplies larger input than the web application expects, a buffer overflow condition may occur [21]. Furthermore, attackers may place additional form fields that will cause the application behave unexpectedly. Another possibility is that attackers may enter commands or malicious code as the form field values.

As described in the preceding paragraphs, attackers can manipulate form fields in various ways. In order to prevent these, there must be server-side validation and filtering of form input field parameters.

2.3. Cross-Site Scripting

One of the most popular attacks is the cross-site scripting (XSS) attack [26, 37, 38, 42]. XSS attacks differ from other attacks such that in a normal attack generally two parties are involved: an attacker and a victim. But in XSS attacks, there are three parties involved: an attacker, a vulnerable web site and the clients of this web site.

The attacker takes advantage of active content in the dynamically generated web pages. In order for the attack to be successful, the web server needs to echo back the user-supplied input without filtering its contents. Also the attacker should deceive a legitimate client of the web application into making a specially crafted request to the web application. If an attacker puts malicious content in a request and convince a client to send this request to the web server, then if the web

application echoes back the malicious code in the response, the malicious code will run on the web browser of whoever made the request. The malicious code can be in any scripting language that can be run on the web browser of the client but most popular one nowadays is JavaScript.

The XSS attacks are classified into two categories according to how the malicious code reaches the server: stored attacks and reflected attacks. In stored attacks, the malicious code is placed on the web server by the attacker himself. It can be put in a database or bulletin board of the web site, for example. In reflected attacks, however, the attacker tricks the user to send the malicious code. A link in an e-mail or newsgroup message or a link in the attacker's own web site can be used to achieve this.

The following example illustrates better how this attack works. The following link contains malicious code:

```
http://www.example.com/cgi-bin/vulnerable.cgi?param=
<SCRIPT>Malicious-Code</SCRIPT>
```

If the attacker convinces a client to follow this link, the following request to the vulnerable site will be generated:

```
GET /cgi-bin/vulnerable.cgi?param=<SCRIPT>Malicious-
Code</SCRIPT> HTTP/1.0
```

When the vulnerable web application receives the request, it constructs a dynamic web page that contains the parameter supplied and sends back the response. The response will look something like:

```
<HTTP>
<TITLE> Welcome Example.Com </TITLE>
<BODY>
...
<SCRIPT> Malicious-Code </SCRIPT>
...
</BODY> </HTML>
```

The malicious code need not only be in the request line. It may be placed in the header fields, form fields or cookies. Furthermore, for long scripts or scripts which contain special characters which might be filtered, it is possible to refer to a script on another site or use HTML tags other than *<SCRIPT>* :

```
<SCRIPT SRC="http://www.malicious-site.com/malicious.js">
<IMAGE SRC="javascript:Malicious-Code">
```

The impacts of XSS attacks are given in [37, 42]. Stealing session cookies, altering the contents of client's files, exposing SSL connections, redirecting client into other sites, modifying the presentation content are just a few of them.

In order to prevent XSS attacks, the HTTP requests should be filtered. Also the context of the request should be carefully inspected. Requests for the random URLs or requests containing parameters to the web application should not be allowed for the first requests. Additionally, dynamically generated error pages should not be allowed because the web server might echo back the user-supplied input in the error page if there is any. The drawbacks of this were discussed in the preceding paragraphs.

2.4. SQL and Command Injection

Today's web applications generally keep their data on the database servers. The database server can be on a dedicated, separate host. The user supplied input in the form fields are used to generate database queries and they are sent to the database servers. The result returned back from the database server is sent to the client in an appropriate format. The important thing here is the user supplied input. If this input is not checked properly, a mal-intended client can get unauthorized content from the database server, corrupt or even destroy it [15, 27, 38, 42]. Consider an application that uses student id as the search criteria in a student database and returns back the student's courses and associated scores using an SQL query like the one given below:

```
Select course,score from student_db where id='%Student_ID';
```

If the user enters the string " ' OR 1=1; " as the student id, then the query will look like:

```
Select course,score from student_db where id='' OR 1=1;';
```

This query will return the scores of all students from the database. Such SQL commands can be injected in the input data.

Web applications may also use shell commands or system calls to access the operating system features. Similar to the SQL injection, an attacker may inject scripts that contain commands to be executed in the web server. The result might be the complete takeover or crash of the server.

The consequences of command injection attacks might be devastating. As a rule of thumb, the web applications should never trust anything that is supplied by the client. The application developers should employ the necessary validation codes to filter out the supplied inputs.

2.5. Buffer Overflows

Buffer overflow attacks are a popular and important threat for all applications. The details of what is a buffer overflow and how it works are given in [13, 30]. By sending data to an application, in which the size of the data is larger than that of application can properly accept, the execution stack of the application is corrupted. As a result, the application might crash or if a specially generated input is sent which causes unwanted code to be executed by the application, the attacker can take the control of the application or the whole machine.

There are a number of input fields that are fed into the web servers and web applications: the URL, HTTP header fields, and form fields. Attackers take advantage of those fields to launch buffer overflow attacks against web-based systems.

Improper handling of inputs is the main cause of the buffer overflow attacks. If the size of the input data were properly checked, and the data having size larger than that of the local buffer were not accepted, there would not be buffer overflow attacks.

2.6. Misconfigurations and Default Configurations

Every site needs different operational and security requirements. So, web and application servers should be configured according to the needs of the organization that employ them. The servers must be configured with security in mind and it should be as precise as possible. Unfortunately, this is not the case for most of the web sites' administrators. They leave the default configurations of the servers intact or do little changes over them. If they do not have enough knowledge, they might also misconfigure the servers. A list of misconfiguration problems is given in [42].

Default server configurations are generally insecure because the main goal of the server software developers is the operation of their software not the security but ease of use. With a less specific default configuration, their server works in the first place for a larger amount of customers with the least effort. Shortly after, of course, the real problems, security problems, arise. Server software developers also pack their software with some default web pages and scripts in order for their customers to test the installed server. For example, there are a number of default CGI scripts installed with the Apache web server some of which give unnecessary information about the operating environment. These default pages poses an important threat to the security of the web site if not removed.

2.7. Information Disclosure

If not configured otherwise, web and application servers generally provide much more information than needed to the clients. The following example shows the HTTP header fields in response to a request:

```
HTTP/1.1 400 Bad request

Server: Apache/1.3.23 (Unix) (Red-Hat/Linux)
mod_ssl/2.8.12 OpenSSL/0.9.6b PHP/4.3.1

Date: Mon, 12 Jul 2003 12:20:50 GMT

Content-Length: 147

Content Type: text/html

Connection: close
```

The '*Server*' header field shows that Apache 1.3.23 web server with mod_ssl and PHP support runs on a Linux operating system. Versions of the software are unnecessarily disclosed. For an attacker, this is invaluable information. Error pages returned from the web server might similarly contain the unnecessary details like the directory structure of the web server, internal application details etc. The comments in the HTML source files are another source of disclosure. Attackers who want to discover the internal implementation details of the web applications might use these.

The web and application servers should be configured such that they provide only the necessary and the least amount of information to the clients. Additionally, short, simple and static error pages should be returned to the clients.

CHAPTER 3

APPROACH AND DESIGN

In the previous chapters web applications and common threats for them have been reviewed. This chapter is going to present our approach to provide protection against those web based threats, the design considerations of the reverse proxy system and the advantages of using such a system.

3.1 Design Issues

The obvious design goal of our system is to establish a shield in front of the web applications that will help in protecting against threats towards web based applications. This section details the issues that were taken into account during the design of the system in order to achieve this goal.

3.1.1 Logical Flow of Interactions in Web Applications

For the normal operation of a web application, the clients of this application should visit a set of web pages in the order that the application developer wants.

Each web page constitutes a state in the web application. According to the input supplied by the client, the web application processes this input and goes on to the next state. However, not all clients want to follow the normal flow of the web applications, and they try to trick the applications into acting in the way they want.

Our basic approach to eliminate the abnormal requests is that the information gathered from a state is used to determine the following acceptable states. Figure 3.1 gives the pictorial view of this idea.

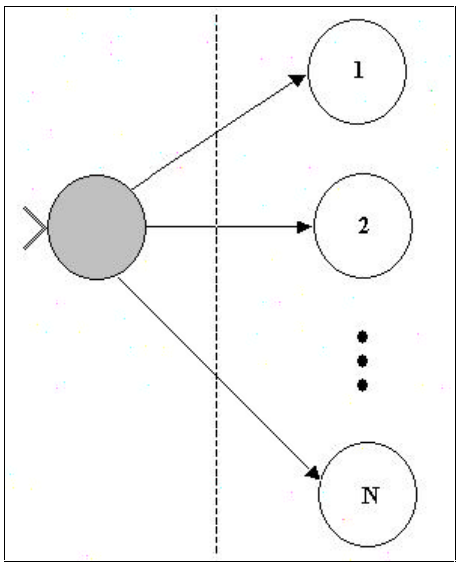

Figure 3.1: State Diagram

The data that can be obtained from a state in the web application can provide valuable information for the determination of what the client can demand on the subsequent request. The response of the web server contains all the links that a

client can follow through. Since the application developer places these links there, the application would probably handle the requests for these resources properly. Nevertheless, if a request comes from the client that the application did not expect (i.e. a request which was not proposed by the application as a result of the prior request), it might put the application in an unstable state depending on whether it handles these cases thoroughly or not.

3.1.2 Tracking Clients

In order for assessing the validity of a client's request, the client should first be identified unambiguously. It has already been discussed in section 2.1 that because of the statelessness of the HTTP protocol, alternative methods were devised to keep the session information in web applications and these methods might be abused by the mal-intended clients.

Web applications might keep track of the sessions in their own but in the reverse proxy, it would not be logical to rely on these session information kept by the application because the application's session tracking mechanism might not be robust enough. Therefore, the reverse proxy should manage its own sessions and should employ additional measures in order for a more reliable session identification is possible. Measures that we have taken will be mentioned in section 4.2.

3.1.3 Input Validation

Unusual inputs sent to the web applications are the source of most of the problems in web-based systems. Some of the web threats due to the malicious inputs were discussed in Chapter 2.

An attacker might place malicious content not only in the HTML form fields which are the standard means of passing client supplied input data to the web applications, but also in the HTTP headers. In fact, HTTP request headers can be viewed as a source of input to the web-based system.

Inspecting the contents of the client input for detection and prevention of possible attacks is a requirement. In addition to that, keeping the information about the form fields that were existed in the previously sent web page will make additional validity checks possible. These topics are going to be discussed in detail in chapter 4.

3.2 Implementation Decisions

In this section, some of the decisions that have been made before the implementation of the system are going to be discussed.

3.2.1 Programming Language

There are a few programming languages that can be used in the implementation of the reverse proxy. Proxies written in C, Java and Perl programming languages exist. The most important factor in the decision of the programming language was

the performance. When it comes to the performance, the C programming language seemed to be the best choice.

3.2.2 Reverse Proxy

A reverse proxy is a system that gets the service requests before the servers that is actually going to process these requests and passes them directly or after some pre-processing like validity checking to the real servers.

There are a few programs that can be used as a reverse proxy for web-based systems written in C programming language: Pound [34], Apache with mod_proxy [5], Squid [39], Stunnel [40], twHttpd [43].

Pound was selected as the program on top of which the actual implementation will be developed. It has three main capabilities:

- Acting as a Reverse Proxy: It stands in front of the web servers and passes the requests from clients to the back end web servers.
- Acting as a Load Balancer: It has the ability to share the traffic among a set of backend web servers. It inspects the backend web servers and do not pass the requests to dead hosts. It also checks if a previously dead host is alive.
- Acting as an SSL Wrapper: With this capability, if the backend web servers do not support SSL communication, clients can connect to the reverse proxy using HTTPS, and then the Pound will decrypt and pass the requests to the backend servers.

There are a few reasons behind the selection of Pound as the base for implementation. First of all, it has fairly small and simple code. Large and complex codes will eventually leave some vulnerabilities in the application no matter how well they be written. The next consideration is the performance of the reverse proxy. It is important because the reverse proxy is where all the requests and responses pass through. If the reverse proxy is not fast enough it would become a bottle-neck especially when the extra application level checks which is going to be done in the final implementation is considered. It is stated that Pound has been used in a production environment where 3.5 million requests/day and 300 requests/second could be handled. The state of being up-to-date for a reverse proxy is another factor that were taken into account. At the time of decision, Pound has an up-to-date version and supports HTTP/1.1.

3.2.3 Session Tracking Method

In section 2.1, it is stated that there are two basic schemes used to keep track of sessions: cookies and hidden form fields. As well as their advantages, there are some disadvantages of using both of these methods. When the session identifier is stored in the hidden form fields, every page that the web server sends should contain the HTML forms. In addition, nothing prevents the client from seeing and changing the hidden form field value. On the other hand, using cookies simplifies things. Unlike hidden form fields in which every page should contain the session identifier, it is only necessary to send the session identifier to the client at the beginning when the session is created for the first time. Additionally if it is not written to the disk, it will only be available in the memory. Nevertheless, web

browsers do not always support secure handling of cookies. As in the case in hidden form fields, they are not tamper proof; i.e., a mal-intended client can change their content.

In this implementation, cookies are used as the mechanism to keep track of sessions. With this, the reverse proxy would not have to modify the contents of every HTML page; instead, it is going to add a cookie header only for once. In terms of performance considerations, it seems that this would be better. However, cookies are alone will not be enough for reliable session tracking. The details of what have been done to improve the quality of session management scheme are given in the implementation section.

3.3 Advantages of Using Reverse Proxy

While it is possible to increase the security of the web servers by appropriately configuring them, placing our reverse proxy system in a standalone machine in front of the web server provides various benefits. First of all, the only system that is visible and directly accessible from the Internet will be the reverse proxy not the web server. Secondly, a web server can run additional applications/services like database server programs or telnet/ftp services. On the other hand, the reverse proxy will run on a system that does not run additional applications or provide additional services. Therefore, the vulnerabilities existing in those applications or services, will not affect the reverse proxy. Since the only system accessible from the Internet is the reverse proxy, these vulnerabilities will also not affect the web servers. Furthermore, in an organization, various people might need to access the

web server like application developers or people responsible for the modification of the web content. However, the only person that will access to the reverse proxy will be the administrator of the system. Therefore, the management and the monitoring of the system will definitely be easier.

CHAPTER 4

IMPLEMENTATION OF THE APPLICATION LEVEL REVERSE-PROXY

In this study, we have implemented a reverse proxy system that is placed in front of web applications and servers to reduce the security risks associated with the web applications. A diagrammatic view of the system can be seen in Figure 4.1.

The reverse proxy inspects the web content, specifically HTTP and HTML content, and based on its decisions, the details of which was given in Chapter 3 and will be given throughout this chapter, it either allows the requests to pass to the web server or does not allow those that might be malicious before reaching the actual web servers. As it can be seen from Figure 4.1, regardless of whether the web server supports SSL connections or not, the reverse proxy can accept the SSL connections from clients. The encrypted pages coming from the clients are decrypted if SSL is used, and the request is sent to the request-processing module. After that, the request-processing module checks the legitimacy of the request and decides whether it should forward it to the web server or reject it. If the request is

sent to the web server, the response from the server is received and sent to the response-processing module. In the response-processing module, only HTTP and HTML content is inspected and the response body is used in creating dynamic data structures that describe how the next request should look like. Finally, the response is encrypted if an SSL connection with the client has been established and sent to the client.

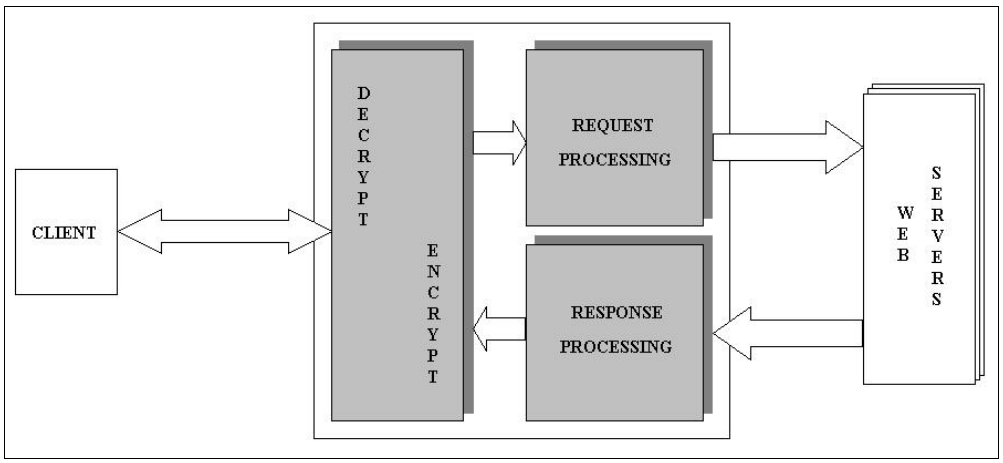

Figure 4.1: A diagrammatic view of the system

4.1 Data Structures

As it is stated before, the reverse proxy system looks for HTTP and HTML content that passes through it and saves necessary information that will help in the future inspections and inferences. This section gives the data structures used to store this information.

Figure 4.2 gives a general view of the data structures that have been used in the implementation. For each client, an entry is created and stored in the reverse proxy. Pointers to the entries are kept in the "clients" table. In an entry, the following items are stored:

- A list of URLs that can be granted to pass to the web server in the subsequent request,
- Number of URLs currently in the list,
- Form field element list if there is any,
- Number of form input elements in the form field element list,
- Session identifier used to identify clients at the reverse proxy,
- IP address of the client,
- Cookies that might be set by the backend web application,
- Error status,
- Creation time of the session at the reverse proxy,
- Last access time of the client.

In order to keep a list of URLs, an array of pointers to the "String" structure is used. This structure consists of only one-byte length character array. However, if a much larger space is allocated for the structure pointer, then the character array can use this extra space to store more elements. In some sense, this can be viewed as the dynamic arrays.

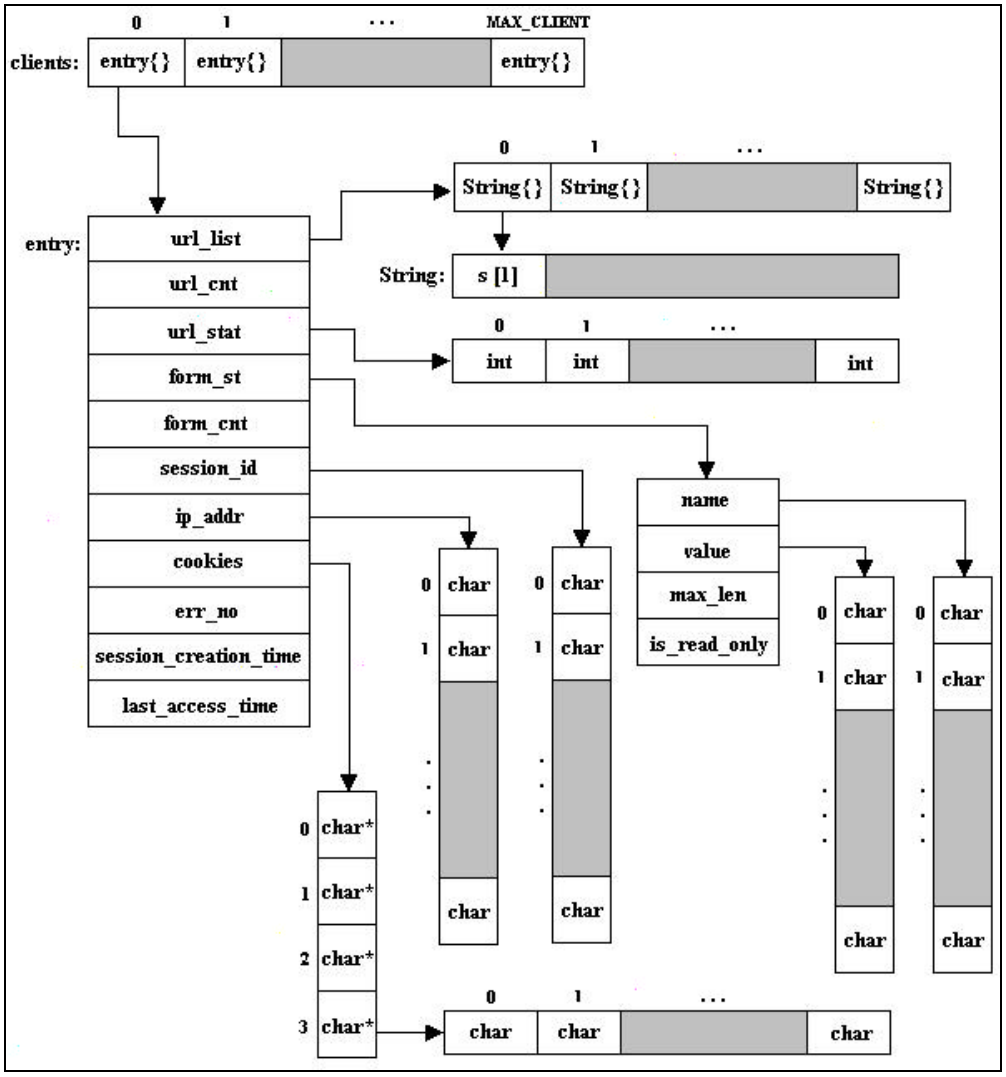

Figure 4.2: Data Structures

Requests for some URLs should not overwrite the previous URL list. Instead, the links in the response page should be appended to the existing URL list. In order to help to make this decision, an array of "short" integers is used. For each URL, there exists a corresponding entry in the URL status list.

If there are form fields in HTML pages, then information related to these form fields is stored in the form element list. This list contains the form field's name and its value if there exists any, the maximum length of the value that is acceptable, and the changeability of the form field's value. For some form field elements like hidden fields, the client should not change the default value of the form field.

The number of URLs and the number of form field entries are saved as integers. Session identifier and IP address are kept in the character arrays. In addition to these, a maximum of 4 cookies are stored in the two-dimensional character array. Error number, err_no, is used by the response processing functions to carry the status (success or failure) of the processing. Finally, session creation and last access times are the UNIX "time_t" variables.

4.2 Session Tracking

It is stated in section 3.2.3 that due to the stateless nature of the HTTP protocol, cookies are selected to keep the session information in the reverse proxy system. This session information is nothing to do with the backend web application. It is only used by the reverse proxy. However, cookies like other session tracking methods are prone to impersonation attacks. In order to prevent these kinds of

attacks, extra measures should be taken. For this reason, together with the session information, the IP address of the client and the creation time of the session are stored in the table entry for the client at the reverse proxy. When a request comes, session identifier value in the cookie is used to find the related table entry. After that, the IP address of the client is checked against the IP address of the requester. If they do not match, the request is not accepted. Nevertheless, IP addresses alone do not solve everything. After all, most of the clients are behind NAT capable devices, which means requests originating from the clients behind a NAT device will all have the same IP address. Therefore, it is decided that sessions should be active only for a small duration so that even if the cookies are stolen somehow, the intruders will have less chance to interfere with an active session. The administrator of the reverse proxy specifies this duration in the configuration file and after this period of time, new session identifiers are created and new cookies are set. In order to prevent eavesdropping of session identifiers in an unencrypted connection, SSL connections should be preferred.

Session identifiers should satisfy some properties. First of all, they should be unique so that the clients can be differentiated without any ambiguity. In addition, they must not be easily guessed. Easily guessed session identifiers lead to impersonation attacks. Considering these, hashing algorithms are chosen to be used to generate session identifiers. Either MD5 or SHA1 algorithms can be used. The main advantage of these algorithms is that they are one-way; i.e. using the resulting value, it is not possible to get the initial input. Another property is the very low probability of having the same message digest as a result of different

hash operations. Modifying only one bit in the input changes on the average half of the bits in the result. The selection of the hashing algorithms is done in the configuration file. A randomly generated string and the IP address of the client are fed into the hash function and the resulting digest is stored as the session identifier. The following lines show examples of a few session identifiers generated consecutively:

```
5ea56997e94999933881feca4c6138be (MD5)
344a7ae9a6cddd2b62fe767188c4f302 (MD5)
df952ece6fb2d383b33e6a17f961001b (MD5)
ee0873a488ed85fae53a7d34453dda3e (MD5)
42b01df6f568eb7c3db434f5f266ffa3 (MD5)
04e6f700f3c27a1a6d1cdc5d535b478b52065eb8 (SHA1)
ac1bd5c9869a9df41e35c9512b6e6903f802693a (SHA1)
ecfbfac170b0864538fc24457a8e59c87392f745 (SHA1)
8c174b3770617e636473528430d64396b0d15eea (SHA1)
f7a6261c782e243aabfadb3447d3809b3e37ce81 (SHA1)
```

The above examples illustrate that none of the generated session identifiers have any relationship, and they are fairly difficult if not impossible to guess some actual and active session identifier.

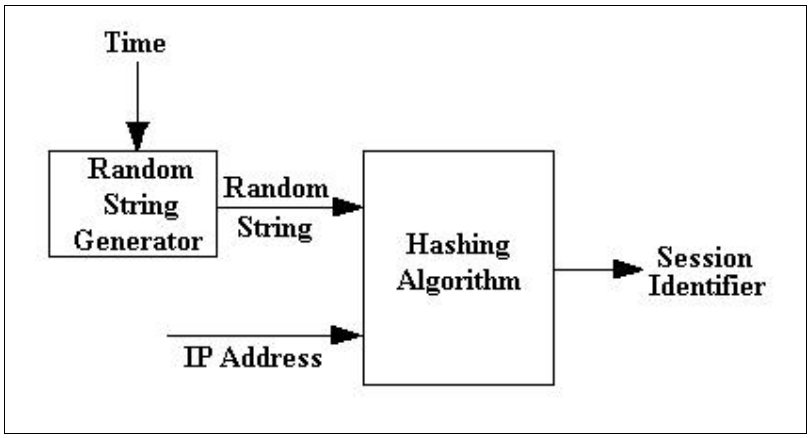

Figure 4.3: Session Identifier Generation

4.3 Request Processing

A HTTP request message consists of three parts [14]: request line, request headers and request body if there is any. The request line is made up of the HTTP method, the identifier for the resource on which the request will be performed and the protocol version in use. Request headers are used to pass additional information about the clients and the request. Finally, the request body is used to pass input from client to the web server.

The reverse proxy system that we have developed checks all three parts of the request message. Figure 4.3 is a flowchart that shows the steps to assess a request.

Some of the HTTP methods might be unnecessary and in some cases dangerous for a web site. For example, a site that runs an important web application will rarely need the PUT method to be used that will cause new resources to be created or existing ones to be updated at the server by the client. For this reason, only the

necessary methods should be accepted all of the rest should be denied. Some of the web servers provide the opportunity to deny or accept specified methods while others do not. Our system has this capability too since it is possible that the web server may not support this type of restriction or it might be difficult to configure such a properly.

In order for the clients to access to the web resources, they should have a valid session at the reverse proxy system. The session identifier for a client is set whenever the first request comes from this client. The session identifier information supplied by the client is checked with the one stored in the reverse proxy. Additional checks are done as discussed in section 4.2.

A set of URL checks is applied to the incoming requests. First, the requested URL should have an acceptable length. There is not any restriction on the maximum length of an URL [14]. So a default value of 1024 bytes is selected and the chance to set it to a different value is given to the administrator of the reverse proxy system. If the length of the URL exceeds the maximum allowed length, then it will not be passed to the web server.

Next step in URL processing is to filter the URLs for a set of patterns that are given in the configuration file. There are a number of attacks that have known patterns so if the administrator is knowledgeable enough and creates an extensive pattern list, those attacks can be avoided.

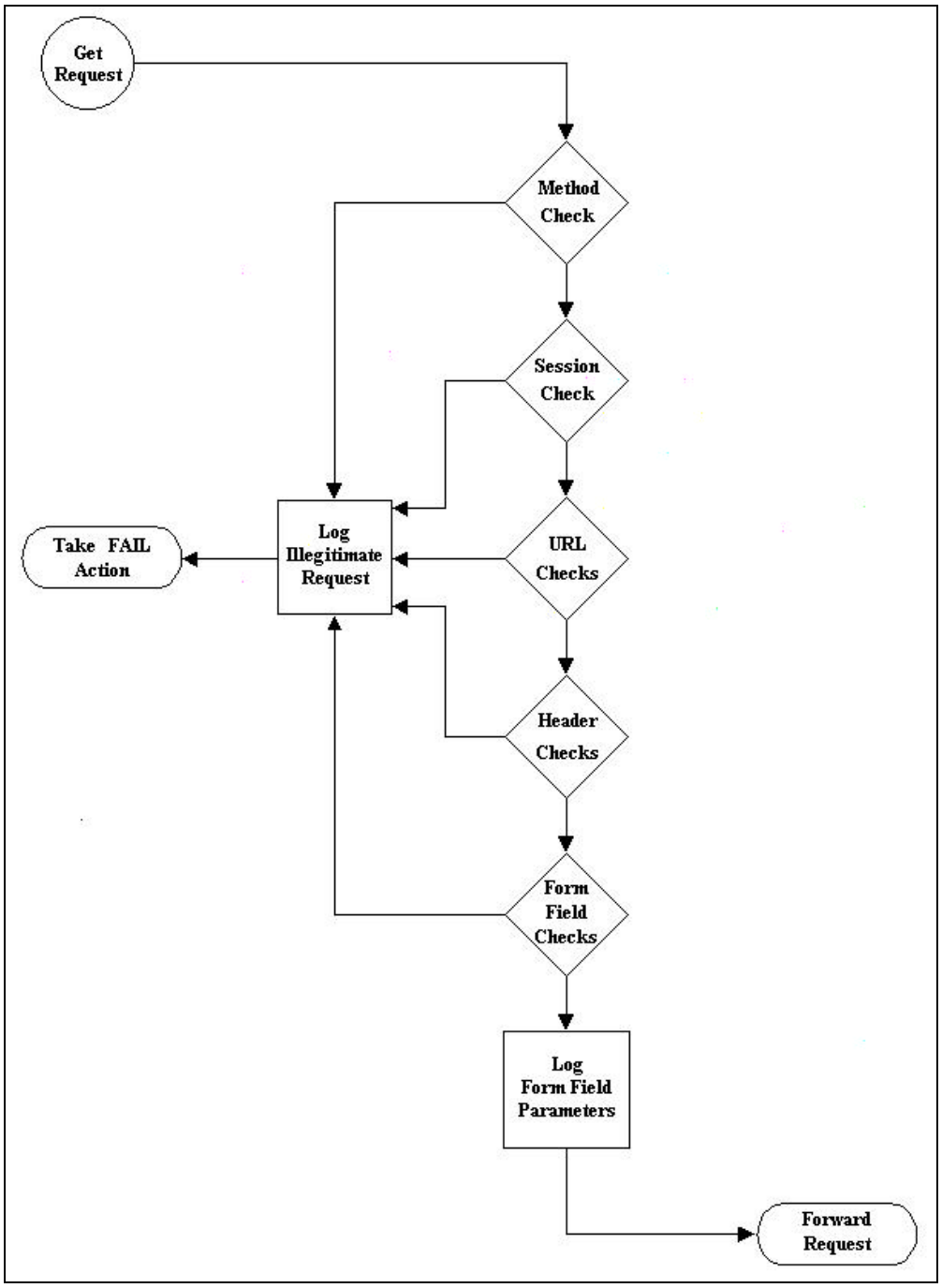

Figure 4.3: Request Processing

Final URL check depends on the operating mode of the reverse proxy system. It operates in two modes. The first one is based on a restrictive approach and the clients are only allowed to navigate through the links available on the web pages. Requests for arbitrary resources or resources outside the boundaries of web root directory are not allowed. In order to assure this, the URL list in the table entry for the client is used. This list is filled when the previous response page is sent. If the client connects for the first time, s/he must request an initial page that is defined in the configuration file. Furthermore, a set of URLs that are accessible without these restrictions can be defined. In the second mode, nevertheless, the negative security approach is adopted; i.e., accesses to the general resources are not restricted but restrictions might be applied for a set of specific URLs. The importance of the web-based system and usability will determine in which mode the reverse proxy will operate but certainly the first mode will be more resistant to the attacks and it should be used for most of the web applications.

If the administrator defines it in the configuration file, a set of patterns can be searched in the header fields. Requests with such headers are considered illegitimate requests. Moreover, some header fields may be declared necessary fields and in order for the operation to continue normally, they should be present within the request headers. Another important issue with headers is that some headers should not be passed to the web server. For example, for the correct operation of the reverse proxy in the restrictive mode, the 'If- …" type of headers ought not to be passed to the web server. These headers make the retrieval of the requested resource conditional. For instance, with the use of "If-Modified-Since:"

header, the requested resource will not be returned from the web server if the resource has not been modified since the time specified in the header field.

Another important check for the web applications is the form field parameter checks. These checks are done using the entry for the client at the reverse proxy. Names and default values of the form fields together with the maximum allowable lengths and the changeability of the fields are stored in the reverse proxy when the HTML page containing these form elements is sent to the client. The inputs supplied via the form fields are checked if their length is greater than the maximum allowable length. Next check is whether a read-only field has been changed or not. In Chapter 2, attacks to the shopping chart applications are mentioned which are possible because the read-only hidden fields are changed. Finally, the form field parameters are logged.

4.4 Response Processing

Similar to the HTTP request messages, an HTTP response message consists of three parts [14]: status line, response headers and response body. As its name suggests, the status line gives the status code for the requested operation together with the textual information about the code and the HTTP protocol version. Response headers are used to pass additional information from the server to the client and response body contains the requested resources if the operation was successful.

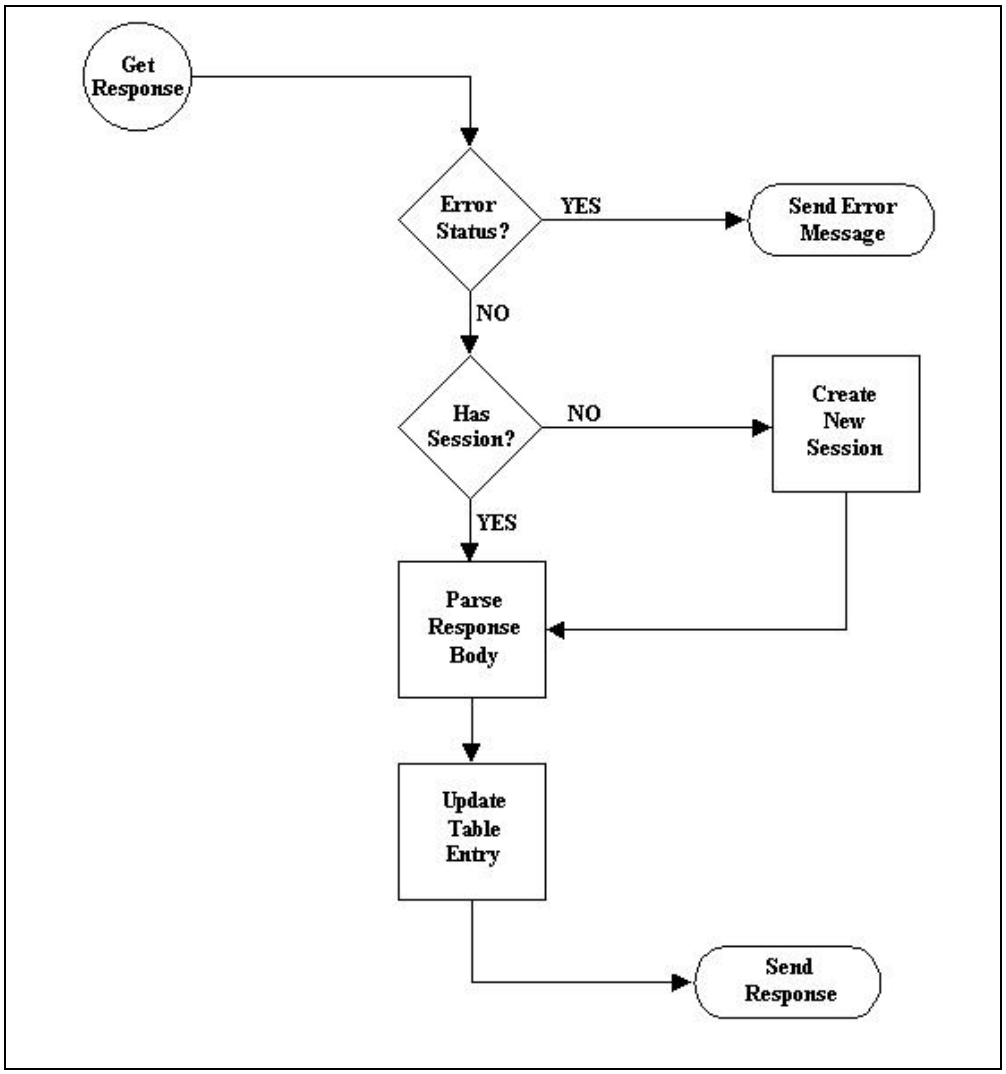

Figure 4.4: Response Processing

A status code starting with “4” (4xx) indicates that there is something wrong in the request and “5” (5xx) indicates that the server could not fulfill the request. In section 2.3 and 2.7, the danger of dynamically generated error pages and extra information disclosure in the error pages were discussed. Therefore, reverse proxy looks for the error codes that indicate an error, and then it replaces the error page sent by the server with a simple and static one.

The reverse proxy inspects whether there is any session defined for the client. If there is not any session, it creates a new table entry and a session identifier. After that, it sends the session identifier to the client by appending the cookie into the response headers.

After receiving the response from the web server, if the content is HTML, the reverse proxy parses the response body and updates the structures in the table entry for the client. What the reverse proxy does is as follows:

- It adds URLs that are found on the HTML page to the URL list. What normally expected from the client is that s/he use one of the links in the HTML page to request another resource or operation. Together with each URL, the effect of a request for that URL on the URL list is kept. Some HTML tags like IFRAME can be used to embed other HTML pages into the existing web page. In such a case, the links in the embedded page should not overwrite the previous entries in URL list. Instead, they should be appended to the existing list.

- If the outgoing web page contains HTML forms, it records the name and the default value of the form input element together with the maximum allowable length of the client supplied input and whether the value of the form field can be changed or not.

4.5 Configuration of the Reverse Proxy

The reverse proxy system is configured through a configuration file which is read when the reverse proxy initially starts. According to the options or values in the configuration file, the reverse proxy initializes its data structures.

Some of the parameters that are used to configure the reverse proxy are given below:

- The allowable HTTP methods
- The maximum URL length value
- Maximum number of clients reverse proxy can handle
- Illegal patterns in URLs
- Illegal patterns in HTTP header fields
- Illegal patterns in form fields
- Hashing algorithm (md5 or sha1)
- The extensions of the files that should not be processed (like gif, pdf, etc.)
- The session timeout value

An example configuration file is given in Appendix I.

4.6 Protection of the Reverse Proxy

Important threats for web-based applications were discussed in chapter 2. The following list summarizes what the reverse proxy does in order to defend against these threats:

- Cookie Poisoning: The reverse proxy keeps the copies of the application cookies and compares the user-supplied cookies with the stored ones.
- Form Tampering: In order to eliminate form-tampering attacks, the information about the form fields are stored at the reverse proxy and checked with the client-supplied values.
- Cross-Site Scripting: The reverse proxy does not allow access to the arbitrary URLs and does header and URL filtering.
- SQL and Command Injection: The reverse proxy filters the HTML form inputs. However the patterns used in filtering is global; i.e. a pattern is looked up in all form inputs.
- Buffer Overflows: The lengths of the URLs and form field values are checked at the reverse proxy.
- Information Disclosure: The error pages generated by the web server are replaced by the reverse proxy. A simple and static one is returned back. Also, the administrator of the reverse proxy has the ability to replace some header fields with a predefined string.

4.7 Similar Work and Comparisons

There are a few free and commercial products that take on the job of protection of the web-based systems. CodeSeeker of the OWASP [8] and AppShield of Sanctum Inc. [3] are two well-known examples of these products.

CodeSeeker is an open source product. Agents running on the web servers intercept the traffic before the actual web server receives it and checks the validity of the request. Currently Microsoft Windows agent and the alpha versions of the Linux and Solaris agents are available. There are a number of differences between the CodeSeeker and the reverse proxy that we have implemented. First of all, the agents run only on the machine that the actual web server resides whereas our reverse proxy can either share the machine with the web server or run on a standalone machine. Secondly, the main method for the detection of the attacks in CodeSeeker is through the use of signatures that should be defined previously. In our implementation, however, the validity of the requests is determined by the use of information gathered dynamically from the HTML pages.

AppShield is a commercial product developed by Sanctum Inc. It is the only product to our knowledge that, similar to our reverse proxy system, inspects the data available in the web pages to differentiate between legal and illegal requests; i.e. it generates its rules dynamically. While we came up with the idea, we did not have the knowledge about the details of AppShield. After learning that AppShield does similar work, we decided to go on the project because there was no open source implementation.

4.8 Limitations of the Reverse Proxy

The reverse proxy system that we have implemented has some limitations. Firstly, it supports only HTTP/1.0 protocol for the time being. The load balancing and the unicode support are not currently implemented. Furthermore, the reverse proxy does not have the ability of processing the JavaScript code. For example, the links in the JavaScript pop-up menus will not be processed and thus allowed by the reverse proxy, because it could not parse and add those links to the allowable URLs. Adding the JavaScript support to the reverse proxy is a challenging task. In order to do that, the code should understand the JavaScript execution environment.

CHAPTER 5

TESTS AND PERFORMANCE

The following sections present the tests that have been carried out to assess the performance and functionality of the developed system.

5.1 Test Environment

In the test environment, three different machines are employed to perform tests. One of them is used as the backend web server and web application server. It is a Pentium III box with 128 MB memory, running Linux. Next one is used to host the reverse proxy, which is a Pentium IV box with 256 MB memory, which also runs Linux. The final one is a Pentium III box with 256 MB memory, which runs Windows 2000; it acts as the web client and hosts the performance measurement tool.

As the backend web server, Apache Web Server is used. It is the mostly used web server [28]. The web application that is decided to be used is a web address-book application, phpAbook [33]. It is developed using PHP and has the characteristics

of a typical web application. It generates dynamic pages, connects to a backend database server, MySql in this case, and manipulates client input that is supplied through the form fields. This application also has some weaknesses that can be exploited. This is a preferable scenario, because the usefulness of the reverse proxy can be revealed much more easily.

In the following sections, the results of performance and functionality tests are presented.

5.2 Performance Tests

These tests were conducted to measure how much overhead the reverse proxy system places in the total HTTP response time. In order to see the overhead, the response time in the case of the requests being passed through the reverse proxy system is compared to the response time when requests made directly to the web server and made through the Pound proxy respectively. During the tests, logging was disabled in the reverse proxy.

In order to carry out the performance tests, the OpenSTA (Open Systems Testing Architecture) [31] tool is used. This performance measurement tool allows simulating an environment with multiple users. While running as an HTTP gateway, it records the HTTP data sent back and forth between the client and the web server. Using this data, it generates a script and this script is used to perform the tests. With the help of its scripting language, the tester can customize his tests. For example, it is possible to assign different user-id and password pairs to the virtual users in order for them to be authenticated into the web application.

Similarly it can handle cookies; for different virtual users, different cookies can be used. This is an important feature for the test of the reverse proxy because sessions are kept using cookies.

Total response times are measured in three different configurations:

- Single User, Multiple Iteration
- Multiple User, Single Iteration
- Multiple User, Multiple Iteration

The following subsections give the details.

5.2.1 Single User, Multiple Iteration

OpenSTA tool gives the opportunity to the tester to select the number of virtual users and the amount of time for which the test must continue or the number of times that the recorded script will be run. In this configuration, only one virtual user makes request and the test set to run for fifteen minutes. The test script is iterated over multiple times until the time is over. Same test script is used for all three tests: direct connection to the web server, connection through the Pound proxy and connection through the reverse proxy.

Figure 5.1 shows the graphical results for directly making requests for the web server whereas Figure 5.2 shows the results for the requests passing through Pound Proxy and Figure 5.3 through the Reverse Proxy.

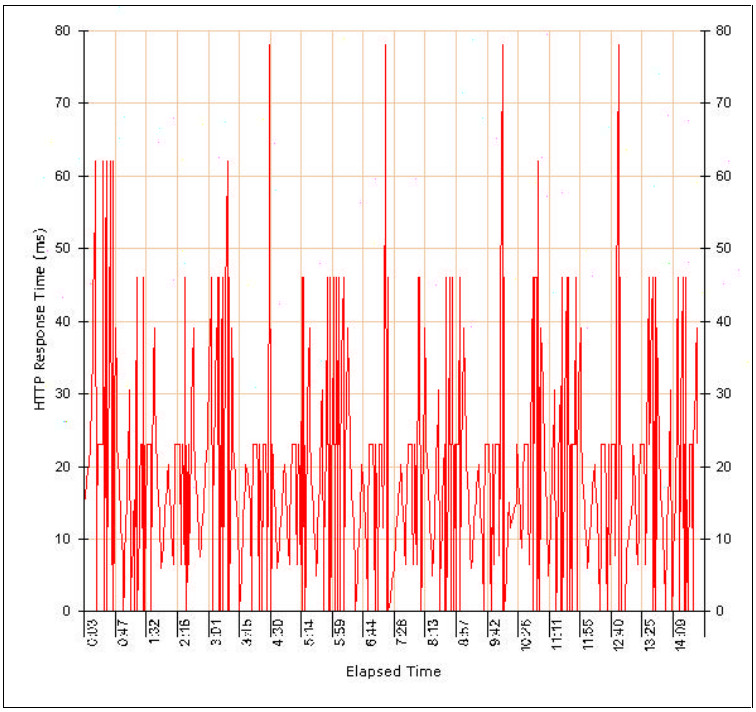

Figure 5.1 Direct Connection to the Web Server, Case I

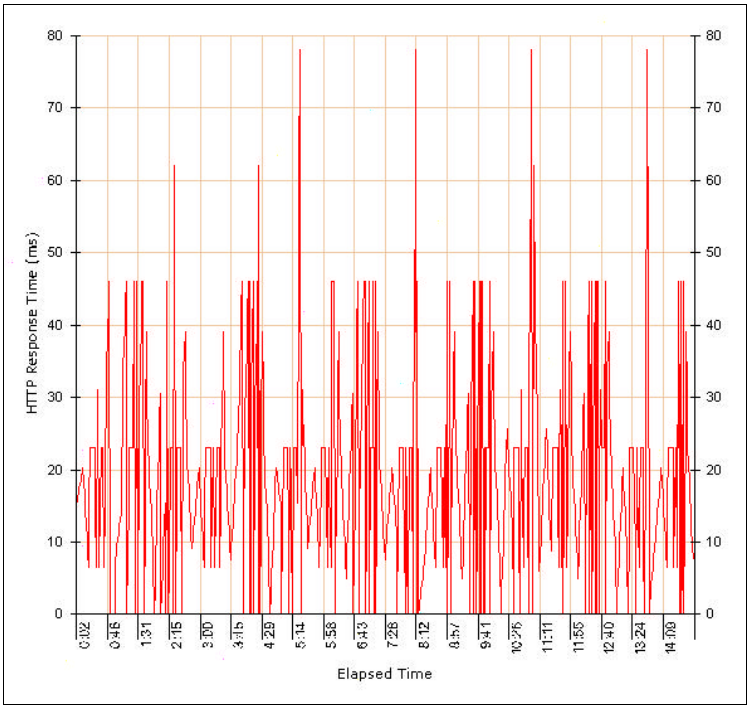

Figure 5.2: Connection through Pound Proxy, Case I

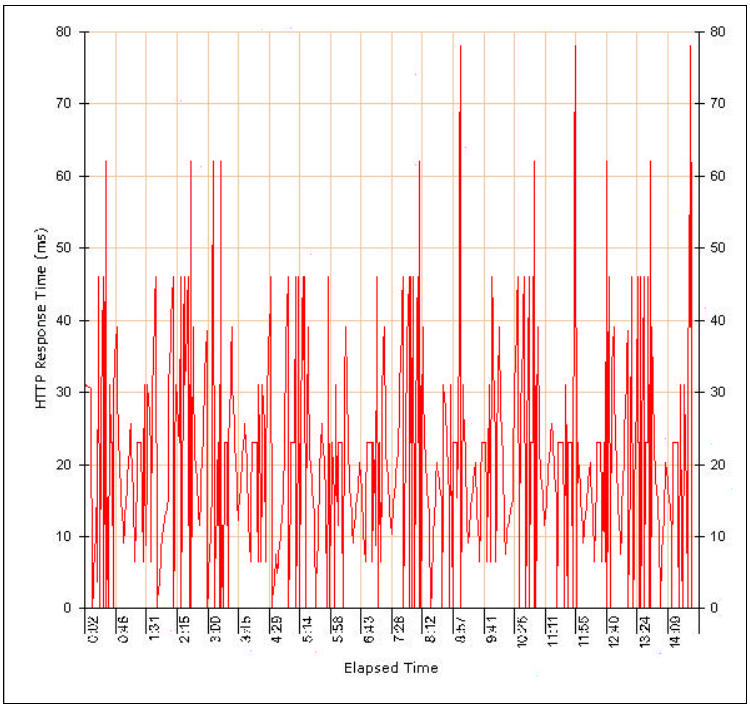

Figure 5.3: Connection through Reverse Proxy, Case I

As it can be seen from the graphics, there is not much significant difference between the reverse proxy and the others. The average HTTP response time is around 13.25 milliseconds for the direct connection, 14.15 milliseconds for the connections through pound proxy and 14.17 milliseconds for the connections through the reverse proxy. Revere proxy performs as well as the Pound proxy in this test. However, having only one user is not the case for most of the time so this test is too simple and unrealistic to base the decisions on it, but at least it shows that there is a price that should be paid when placing some kind of proxy in front of a web server.

5.2.2 Multiple User, Single Iteration

In this test, while the test script run only once, the number of virtual users set to 50. Each user connects to the web application using different user-id and password pairs. Same test script is used for all three tests: direct connection to the web server, connection through the Pound proxy and connection through the reverse proxy.

Figure 5.4 shows the graphical results for directly making requests to the web server whereas Figure 5.5 shows the results for the requests passing through Pound Proxy and Figure 5.6 through the Reverse Proxy.

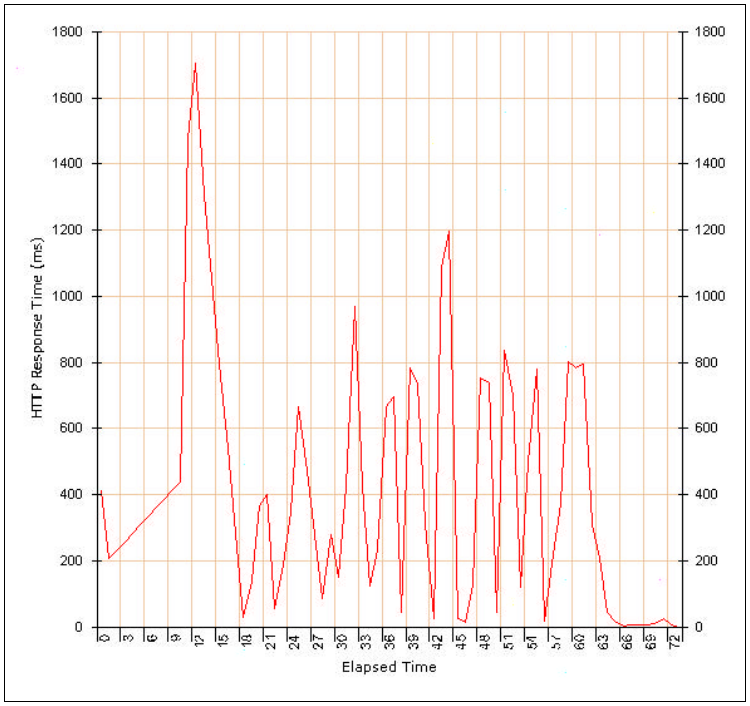

Figure 5.4: Direct Connection to the Web Server, Case II

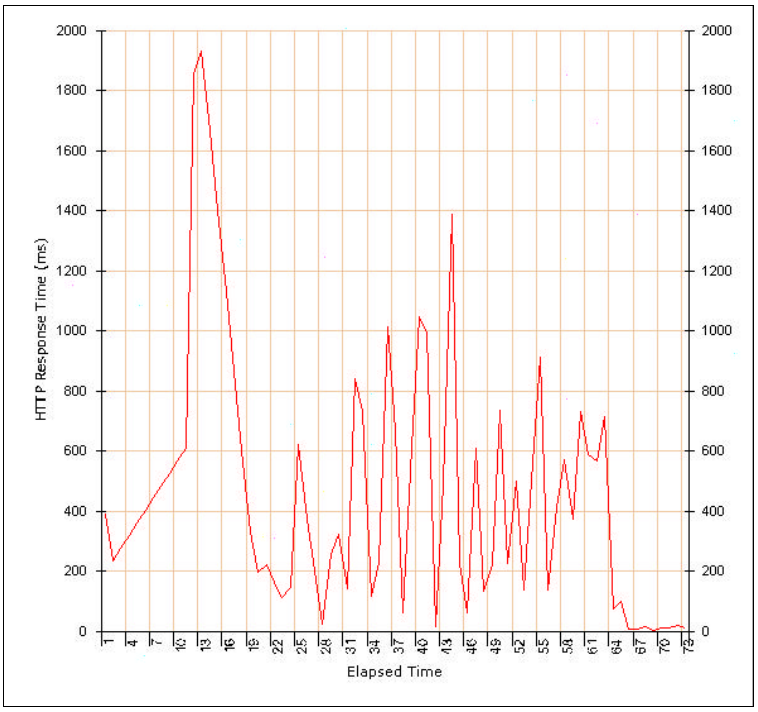

Figure 5.5: Connection through Pound Proxy, Case II

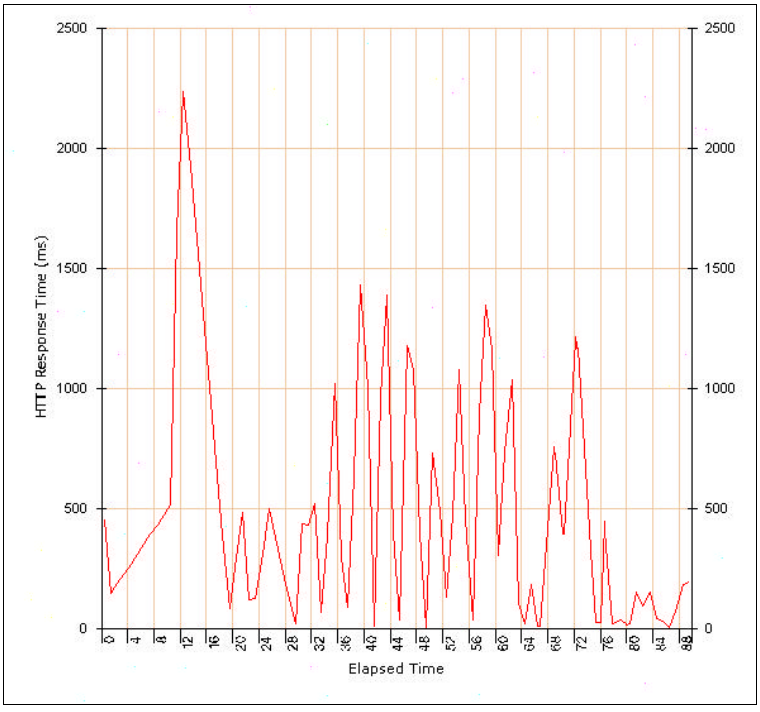

Figure 5.6: Connection through Reverse Proxy, Case II

Average response time for the direct connection is 429.86 milliseconds while this value is 451.36 milliseconds for the connections through the Pound proxy. For the reverse proxy, average HTTP response time is 502.96. The difference between the average response time of the reverse proxy and the pound proxy is 11 %. If the results have been compared with the results of previous test, as the load on the reverse proxy increases, the average response time increases too. Contrary to the previous test, in which the test duration is set to 15 minutes, the time that is taken to complete the tests vary in this scenario because each request is sent after receiving the response for the previous request and the response times for each test differ.

5.2.3 Multiple User, Multiple Iteration

Final test involves multiple users and the number of them is 50. Each user has unique user-id password pair and they continue their tests for fifteen minutes; i.e., the test script is iterated over multiple times. Similar to the previous tests, the same test script was used for all three tests.

Figure 5.7 shows the graphical results for directly making requests to the web server whereas Figure 5.8 shows the results for the requests passing through Pound Proxy and Figure 5.9 through the Reverse Proxy.

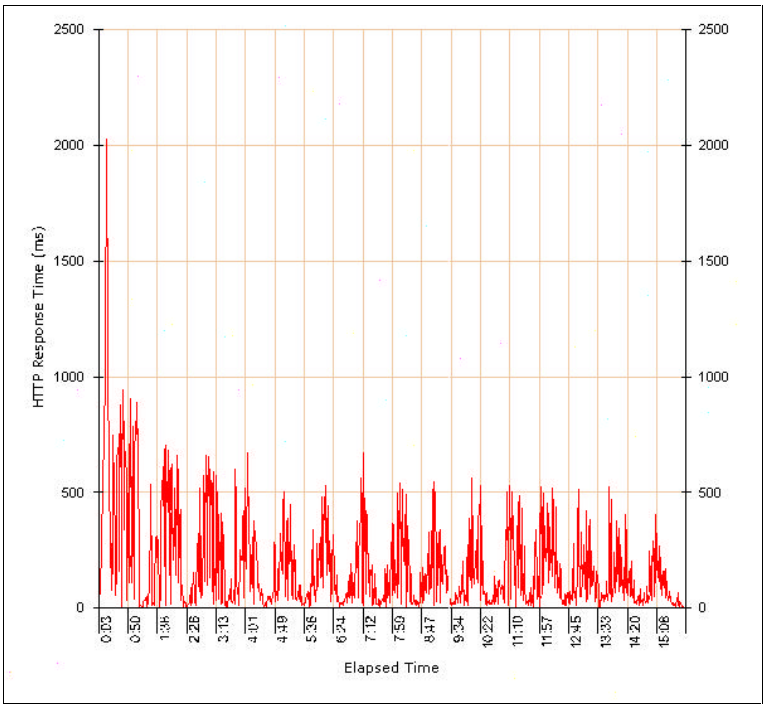

Figure 5.7: Direct Connection to the Web Server, Case III

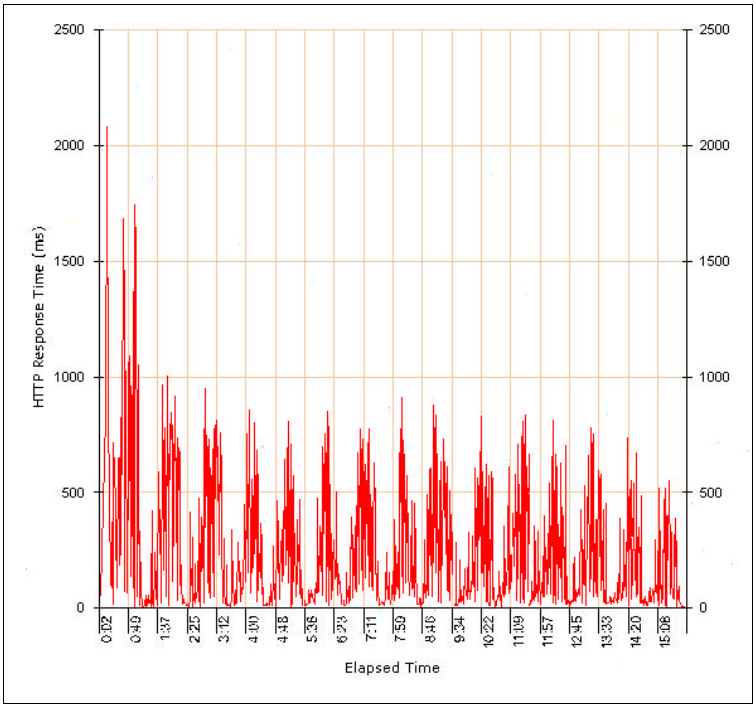

Figure 5.8: Connection through Pound Proxy, Case III

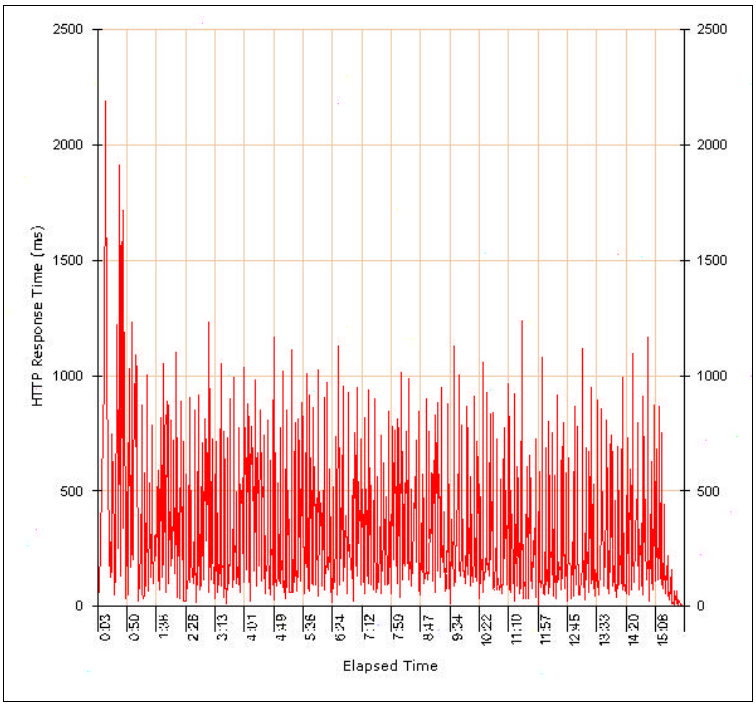

Figure 5.9: Connections through Reverse Proxy, Case III

The average response time for the direct connection is 211.38 milliseconds while this value is 352.47 milliseconds for the connections through the Pound proxy. For the reverse proxy, average HTTP response time is 604.96 milliseconds. The average response time for the requests directly made to the web server and made through the pound proxy decreased compared to the values in test case 2, in which the requests for the resources are generally made for the first time. This is because the unmodified resources are not returned to the clients. Instead, a response with status code 304 and no response body is returned and this reduces the overall average response time. The patterns in Figures 5.7 and 5.8 are also due to this. Whenever responses with 304 are returned the response time decreases. Otherwise the response time goes up and it takes much more time to return back the responses. In contrast to all of these, the reverse proxy does not allow headers like

"if-modified" or "if-unmodified" to be passed to the web server and web server always responses back with the requested resources. For this reason, the performance of the reverse proxy is worse compared to the performance in the previous tests cases. The difference in average response time between the Pound proxy and the reverse proxy also increases.

As it is expected, the connections through our reverse proxy system perform worse than the other connection types. This is because, the reverse proxy examines all of the content of an HTTP packet including the body, and it does not allow "if-modified" or "if-unmodified" types of headers which is required for the correct operation of the proxy. Test results show that as the load on the reverse proxy increases, the overhead in the HTTP response time increases as well. In order to improve the performance of the reverse proxy, it should be run on a machine with high configuration. Especially the memory should be large and fast.

5.3 Functionality Tests

These tests were conducted to assure that the reverse proxy system is functioning properly. In these tests phpAbook was used as the web application. To carry out the tests, either the web pages are directly modified with the help of a web browser and a text editor or the Achilles Proxy was used. Achilles is a powerful tool that can be used to modify any HTTP message, incoming or outgoing, as mentioned in chapter 2.

- Cookie manipulation tests were carried out to see if the reverse proxy could detect any changes in the application cookies.

- Form field manipulation tests were conducted to see whether the reverse proxy could grab modified hidden fields or the form inputs larger than the expected size.
- The URL tests were done to see if the reverse proxy allows access to the arbitrary URLs and if it detects illegal patterns in the URLs. Illegal patterns in URLs can cause cross-site scripting attacks or web servers and web applications to behave unexpectedly.
- Header fields and form input fields are tested if the reverse proxy can catch the illegal patterns in them. Illegal patterns in the header fields can cause cross-site scripting attacks or web servers and applications to behave unexpectedly. Illegal patterns in form input fields might be used to launch SQL or command injection attacks.

The following table summarizes the results of the conducted tests.

Table 5.1: Functionality of Reverse Proxy against Illegal Activities

Cookie Manipulation	Detected
Hidden Field Manipulation	Detected
Changing Maximum Length of Form Fields	Detected
Accessing Arbitrary URLs	Not Allowed
Illegal Patterns in URL	Detected
Illegal Patterns in Header	Detected
Illegal Patterns in Form Fields	Detected

Table 5.1 only lists some of the most popular illegal activities against web applications but it does not cover every illegal activity. The configuration of the reverse proxy plays an important role in the detection of these activities. So it should be configured properly.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The benefits of the World Wide Web have started to be realized by a growing number of people. But the interests of the mal-intended (malicious) people have also moved into the web area. Web applications are the most popular targets for the attackers today. In order to realize the real benefits of the web, organizations should be able to tackle with those threats. However, current security technologies and products that are popular amongst information system society are not enough to defend against most of the attacks targeting web servers or applications.

The main goal in this work is to design a reverse proxy system that decides on the validity of a request based on the data that it gathers during the previous response message. The gathered data is used to enforce the client to follow the normal flow of the application that is protected. Obviously, the reverse proxy understands the application level content.

The reverse proxy that we have developed has also some filtering capabilities. Filtering out the packets for predefined patterns is still an effective and commonly employed method in detecting or preventing intrusions. But the reverse proxy should be configured well in order to get the real advantage from this property.

Working in application level is advantageous when the security is concerned and in some cases the associated risk forces a security product with application level knowledge to be employed. Despite their value in establishing a much more secure environment, these products do badly when it comes to the performance. Performance is the price that should be paid for the increased level of security. As expected, our implemented prototype adds extra overhead to the response time of the requests due to the extra data that it processes. In order to reduce the associated overhead for the application level products, they should run on fast servers and heuristic should be added into the code wherever possible.

Further work may focus on improving the performance of the implemented prototype. Performance is probably one of the most important aspects of the system that will be taken into account by people. The performance of the reverse proxy might be improved by multi-threaded implementation of the checks that are performed sequentially in current implementation. Also the data structures for the static pages might be constructed before they are needed. The current implementation does not have support for web scripting languages like JavaScript. Adding this support is another possible future work. Adding load-balancing capability and the unicode support to the reverse proxy should also be considered. Moreover, keeping information not only about the previous state, but also about

other states might help in much more flexible rules to be generated. This can be considered as a future work. Finally, if some illegal activity occurs, the reverse proxy might use an external script to take the related action. With this capability it is possible to integrate the reverse proxy with other security products like firewalls through the use of these product's interfaces. Adding this external script invocation capability can be thought as a possible future work.

REFERENCES

[1] Achilles Proxy, http://achilles.mavensecurity.com

[2] Active Server Pages, http://www.microsoft.com

[3] AppShield, http://www.sanctuminc.com

[4] Anonymous, I only replaced index.html, http://packetstormsecurity.nl/docs/hack/i.only.replaced.index.html.txt, 1999.

[5] Apache Web Server, http://httpd.apache.org

[6] S. Bellovin, Security Problems in the TCP/IP Protocol Suite, Computer Communications Review, 19(2): pp. 32-48, Apr 1989

[7] R. J. Boncella, Web Security for E-Commerce, Communications of the Association for Information Systems. Volume 4, Article11, November 2000.

[8] CodeSeeker, http://www.owasp.org

[9] Computer Emergency Response Teams Coordination Center, CERT/CC Advisory CA-2000-02, Malicious HTML Tags Embedded in Client Web Requests, http://www.cert.org/advisories/CA-2000-02.html, Feb 2000.

[10] Computer Emergency Response Teams Coordination Center, CERT/CC Statistics, http://www.cert.org/stats/cert_stats.html, 2003.

[11] Computer Emergency Response Teams Coordination Center, Understanding Malicious Content Mitigation for Web Developers, http://www.cert.org/tech_tips/malicious_code_mitigation.html, Feb 2000.

[12] Computer Security Institute and Federal Bureau of Investigation, CSI/FBI Eight Annual Computer Crime Survey, 2003.

[13] C. Cowan, C. Pu, D. Maier, H. Hinton, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, Stackguard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks, Proceedings of the Seventh USENIX Security Symposium, pp. 63-77, Jan 1998

[14] R. Fielding et al., Hyper Text Transfer Protocol - HTTP/1.1, RFC 2616, Jun 1999.

[15] N. Gaur, Assessing the Security of Your Web Applications, Linux Journal, Apr 2000.

[16] D. Hanson, S. Hittel, ARIS Top Ten 2001 Threats, SecurityFocus, Jan 2002

[17] HTTP Cookies, http://www.mobydisk.com/nerdware/docs/cookies

[18] Internet Security Systems X-Force Team, Form Tampering Vulnerabilities in Several Web-Based Shopping Cart Applications, http://xforce.iss.net/xforce/alerts/id/advise42, Feb 2000.

[19] P. Jan, Validating User Input in Web Applications, M.Sc. Thesis Dissertation, Vrije Universiteit Brussels, 2000.

[20] L. Joncheray, A Simple Active Attack Against TCP, Proceedings of the Fifth USENIX Security Symposium, 1995

[21] A. Klein, Auditing and Securing Web-Enabled Applications, Sanctum Inc., 2002.

[22] A. Klein, Cross Site Scripting Explained, Sanctum Inc., Jun 2002.

[23] D. Kristol, L. Montulli, HTTP State Management Mechanism. RFC 2109**,** Network Working Group. Feb 1997.

[24] S. McClure, S. Shah, S. Shah, Web Hacking: Attacks and Defense, Addison-Wesley Pub Co., ISBN**:** 0201761769, Aug 2002.

[25] J. Melbourne and D. Jorm, Penetration Testing for Web Applications, Part One, http://www.securityfocus.org/infocus/1704, Jun 2003.

[26] J. Melbourne and D. Jorm, Penetration Testing for Web Applications, Part Two, http://www.securityfocus.org/infocus/1709, Jun 2003.

[27] P. Neff, Web Application Security, http://www.cgisecurity.net/lib/web_security.pdf.

[28] Netcraft, Web Server Survey, http://news.netcraft.com/archives/web_server_survey.html, Aug 2003.

[29] Netscape Communications Corporation, Persistent client state HTTP cookies, http://www.netscape.com/newsref/std/cookie_spec.html.

[30] Aleph One, Smashing The Stack For Fun And Profit, Phrack, 7(49), Nov 1996.

[31] OpenSTA, Open Systems Testing Architecture, http://portal.opensta.org

[32] PHP, http://www.php.net

[33] phpAbook, Personal Address Book, http://phpabook.sourceforge.net/

[34] Pound Reverse Proxy and Load Balancer, http://www.apsis.ch/pound.

[35] D. Raggett, A. Le Hors, I. Jacobs, HTML 4.0 Specification, http://www.w3.org/TR/REC-html40/, Dec 1997.

[36] SANS Institute / FBI, The Twenty Most Critical Internet Security Vulnerabilities, http://www.sans.org/top20, May 2003.

[37] D. Scott, R. Sharp, Abstracting Application Level Web Security, Proc. 11th International World Wide Web Conf., ACM Press, pp. 396-407, May 2002.

[38] D. Scott, R. Sharp, Developing Secure Web Applications, IEEE Internet Computing, Vol. 6, No. 6, pp. 38-45, Dec 2002.

[39] Squid Web Proxy Cache, http://www.squid-cache.org.

[40] Stunnel - Universal SSL Wrapper, http://www.stunnel.org.

[41] The Open Web Application Security Project, OWASP Frequently Asked Questions, http://www.owasp.org, 2003.

[42] The Open Web Application Security Project, OWASP Top Ten Web Application Vulnerabilities, Jan 2003.

[43] TrustWall HTTP Proxy – twhttpd, http://www.freshmeat.net/projects/twhttpd.

APPENDIX A

EXAMPLE CONFIGURATION FILE FOR THE REVERSE PROXY

```
# Initial URL that the clients could connect for the first time
InitialURL /

# HTTP methods that are allowed
AllowMethod GET
AllowMethod HEAD
AllowMethod POST

# SessionTimeout value gives the cookie recreation interval
SessionTimeout    300

# Max number of clients that the reverse proxy can handle
MaxClients  1024

# Max number of length of URL
UrlLength   1024

# Hash algorithm that will be used to create session identifiers
HashAlgorithm     md5

Server www.websrvr.com

# Enforcement Mode is either Restrictive or Free
# IF (Restrictive) THEN specify individually the pages that
# should be accessed without any restrictions
# Otherwise specify the pages that should be checked against
# the previous state of the application
EnforcementMode Restrictive
{
      Accept /static_dir/
}
```

```
# Log file that will be used to log form parameters and form
related incidents
FormLogFile /var/log/FormData
# Rewrite Header Field
HeaderRewrite     Server      OShttpd 1.0

# The files that have the following extensions won't be processed
# by the reverse proxy
{
      IgnoreExt   gif
      IgnoreExt   jpg
      IgnoreExt   jpeg
      IgnoreExt   jpe
      IgnoreExt   bmp
      IgnoreExt   png
      IgnoreExt   ief
      IgnoreExt   tif
      IgnoreExt   tiff
      IgnoreExt   xbm
      IgnoreExt   js
      IgnoreExt   mov
      IgnoreExt   avi
      IgnoreExt   mpg
      IgnoreExt   mpeg
      IgnoreExt   ico
      IgnoreExt   xls
      IgnoreExt   doc
      IgnoreExt   pdf
}

# Patterns that will cause the request to be denied
# in case they appear in the URL
{
      UrlDeny           .sh
      UrlDeny           .cgi
      UrlDeny           <script>
      UrlDeny           &lt;script&gt;
      UrlDeny           .exe
      UrlDeny           .cmd
}

# Patterns that will cause the request to be denied
# in case they appear in the specified header field
{
      HeaderDeny  Referer <script>
      HeaderDeny  Referer &lt;script&gt;
}
```