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

MOHAMED BOUDIAF UNIVERSITY - M'SILA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

COMPUTER SCIENCE DEPARTMENT

Thesis submitted in partial fulfillment of the requirements for the Degree of MASTER

Domain: Mathematics and Computer Science

Branch : Computer Science

Specialty : Networks

By: BOULANOUAR SOUHIL LARBI

TOPIC

Optimized XSS Vulnerability Scanner Approach

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

Ms SAOUDI LALIA	**University of M'sila**	**Supervisor**
A.MOUSSAOUI	**University of M'sila**	**Examiner**
Y.ARIOUET	**University of M'sila**	**Examiner**

Academic Year : 2015 /2016

Contents

General introduction 1

1 Web applications and web security

1.1 Introduction 3

1.2 Definitions..3

1.3 Web application architecture 3

1.4 The Hypertext Transfer Protocol..4

1.4.1 Request methods..4

1.4.2 HTTP Sessions: 5

1.4.3 HTTP Requests 5

1.4.4 HTTP Responses..6

1.5 Web Security 6

1.5.1 What is web application security..6

1.5.2 Basic Security Concepts 7

1.6 Top 10 Web security vulnerabilities(2013) 8

1.7 Cross-Site Scripting(XSS) 10

1.7.1 Introduction to Cross-Site Scripting 10

1.7.2 The Problem..11

1.7.3 Example 11

1.7.4 XSS Classification..11

1.7.5 How to Determine If You Are Vulnerable..13

1.7.6 Dangers of XSS vulnerabilities..13

1.7.7 Prevention of Cross-Site Scripting Attacks 13

1.8 Conclusion 16

2 Scanners and web security testing

2.1 Introduction..17

2.1.1 Web application scanner 17

2.1.2 Security vulnerability 17

2.2 Introduction to Testing 17

2.3 Testing of web applications 18

2.3.1 Manual testing 18

2.3.2 Automated testing 18

2.4 Security testing of web applications 19
2.4.1 Static testing 19
2.4.2 Dynamic testing 20
2.5 Types of vulnerability scanner 20
2.5.1 White-box scanner 20
2.5.2 Black-box scanner 20
2.6 Techniques for Vulnerability Detection 21
2.7 Related work on XSS 21
2.7.1 Static Analysis Approach 21
2.7.2 Dynamic Analysis Approach....22
2.7.3 Static and Dynamic Analysis Approach....25
2.8 Vulnerability Scanning Tools 25
2.8.1 Wapiti 26
2.8.2 W3af 27
2.8.3 Zed Attack Proxy 28
2.8.4 Acunetix Vulnerability Scanner 28
2.8.5 Vega Vulnerability Scanner 29
2.9 Conclusion 29
3 Optimized XSS vulnerability scanner approach
3.1 Introduction 30
3.2 Contributions 20
3.3 Scanner architecture 20
3.3.1 Crawling phase 31
3.3.1.1 Definition Web crawler 31
3.3.1.2 Browser Simulator 32
3.3.1.3 The Crawling Process 35
3.3.1.4 Identifying Data Entry Points 35
3.3.1.5 Form Crawler 37
3.3.1.6 Database 42
3.3.1.7 Report Generator 42
3.3.2 Injection phase 43
3.3.2.1 Injection Code Generator 43
3.3.2 .2 XSS attack vector generator 44

3.3.2.3 The Injection Process ………………………………………………………46
3.3.2.4 The Difference between reflected and stored injection…………….....47
3.3.3 Detection phase ……………………………………………………………..…49
3.3.3.1 Detection of the vulnerability …………………………………………49
3.3.4 Re-injection Malicious code ……………………………………………….…51
3.3.4.1 Our approach …………………………………………………..…...…51
3.3.4.2 The Difference between reflected and stored detection 52
3.4 Conclusion ………………………………………………………………………54
4 Implementation and experimentations
4.1 Introduction…………………………………………………………………….....55
4.2 Language and tools used to develop ………………………………………………55
4.2.1 NetBeans …………………………………………………………………...55
4.2.2 jsoup: Java HTML Parser ……………………………………………..…55
4.2.3 HtmlUnit ……………………………………………………………………....56
4.2.4 MySQL …………………………………………………………………..56
4.3 The Physical Schema …………………………………………………………57
4.4 Optimized XSS vulnerability scanner interface ……………………………….....58
4.5 Experimentations ……………………………………………………...……59
4.5. 1Web application used in the test ..……………………………………………..59
4.6 Results discussion ……………………………………………………………..…61
4.6.1 First evaluation scenario …………………………………………………...61
4.6.2 Second evaluation scenario ………………………………………………..62
4.6.3 Discussion: …………………………………………………………………....64
4.7 Conclusion …………………………………………………………………………64
General conclusion...……………………………...…………...…………………... 65
Bibliography ..…………………...……………...…………...…………………... 66

List of Figures

1.1 Dynamic Web Applications ……………………………………………………………...4

1.2 Sample HTTP request message ……………………………………………………….5

1.3 Sample HTTP response message ……………………………………………………...6

1.4 Vulnerability stack ……………………………………………………………….…7

1.5 Alert message …………………………………………………………………..…11

1.6 Reflected XSS vulnerability attack …………………………………………………..12

1.7 Sample HTTP Response Page after Encoding ……………………………………….…15

1.8 Prevention of Cross-Site Scripting with Encoding ……………………………………..…15

2.1 HtmlUnit example simulate typing an address in the browser ……………..…………..19

3.1 Architecture of XSS vulnerabilities Scanner ……………………………………...…31

3.2 navigating www.google.dz ……………………………………………………..…32

3.3 click the "Home" link on the HTMLUnit homepage ………………………………..……32

3.4 Form Identification ………………………………………………………………..33

3.5 Get fields parameters of form ……………………………………………………..33

3.6 Submitting a Form by name ……………………………………………………….…34

3.7 Submitting a Form by XPath ………………………………………………...……34

3.8 The Crawling Process……………………………………………………………...…35

3.9 Extracting URL Parameters ……………………………………………………….…36

3.10 Extracting Form Parameters ………………………………………………………..…37

3.11 Form structure classification ……..…………………………………………………...38

3.12 Forms pair generation ……..…………………………………………………….39

3.13 pair validation step ……..………………………………………………………..40

3.14 The Final Crawling Process ……………………………………………………..…41

3.15 crawler class diagram ……………………………………………………………...42

3.16 Records ……………………………………………………………….............…43

3.17 XSS grammar tree ………………………….…………………………………………44

3.18 Injection of reflected XSS ……………………..…………………………………...…48

3.19 Injection of stored XSS ……………………..…………………………….......…49

3.20 Input script example ……………………..………………………………............…50

3.21 Catch prompt(123) ………………..……….………………………….............…50

3.22 inject <img src=javascript:alert(1)> ……………………..……………………………51
3.23 the reflected Attack Vector of low security page ……………………..…….…………51
3.24 the reflected Attack Vector of High security page ………………………………………51
3.25 input " <IFRAME SRC="javascript:alert(1);"> </IFRAME> ………………………..…52
3.26 DOM tree of the response page…………………………………………………...…52
3.27 Detection of XSS vulnerability "Reflected XSS"…………………………………...……53
3.28 Detection of XSS vulnerability "Stored XSS" …………………………………………...54

4.1 Our Scanner of XSS Vulnerability ………………………………………………….……58
4.2: Damn Vulnerable Web App (DVWA) …………………………………………………...60
4.3: testphp.vulnweb.com …………………………………………………………………..60
4.4: template ……………………………………………………………………………...61
4.5: XSS vulnerabilities detected ……………………………..……………………………..61
4.6: Average Scan Time(s) ………………………….……………….……………………..61

List of Tables

1.1 A List of Special Characters ……………………..……………………………………………..16

2.1 Vulnerability Scanners Tools ……………………………………………….…………..26

3.1 Setting Form Field Values ……………………..…………………………………….…34
3.2 type of XSS attack vectors ……………………...………………………………………44
3.3 Attack Vector Grammar HTML tag "script " ……………………………………..…45
3.4 Attack Vector Grammar "HTML event attribute"………………………………………45
3.5 Attack Vector Grammar "URL protocol" ……………………………………….....…46

4.1 Website.sql ………………………………………………………………………………...57
4.2 URL.sql……………………………………………..………………………………...57
4.3 FORM.sql……………………………………...…………………………………...57
4.4 INPUT.sql …………………………………..……………………………………………...57
4.5 RECORD.sql ……………………………………………………………………………58
4.6 Discovered Vulnerability (Reflected) of DVWA ……………………………….………...61
4.7 Discovered Vulnerability(Stored) of DVWA………………………...…….…………...62
4.8 Scanners evaluation metrics on TestPHP …………………………………….………...62
4.9 Scanners evaluation metrics on template ……………………………….……………...64

General introduction

1 Context of the study

Web applications are becoming more popular and widely being used in all aspects of work and social activities.

Now web applications are the dominant method for implementing and providing access to on-line services and becoming truly pervasive in all kinds of business models and organizations.

Today, most systems such as Social Networks, health care, banking, or even emergency response are relying on these applications.

However, the exponential development of web technologies 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.

Cross-site scripting (XSS) attack considered as one of the top 10 web application vulnerabilities of 2013 by the Open Web Application Security Project (OWASP) [13]. According Cenzic Application Vulnerability Trends Report (2013) Cross Site Scripting represents 26% of the total population respectively [42] and considers as top most first attack.

Cross Site Scripting attack carried out using HTML, JavaScript, VBScript, ActiveX, Flash, and other client-side languages. A weak input validation on the web application leads Cross Site Scripting attacks to gather data from account hijacking, changing of user settings, cookie theft.

2 Statement of the Problem

The Detection of XSS is a topic of active research in the industry and academia. To achieve those purposes, automatic scanners have been implemented.

XSS Vulnerability scanner is a tool that detects the XSS vulnerabilities in web applications, generally, there are two types of vulnerability scanners:

White Box scanner: provides the penetration testers with the knowledge of implementation details (e.g. the internal structure of the program).From this information; test cases are created according to the coverage criteria [27].

Tools taking white-box approaches suffer from the following shortcomings:

- Sometimes source code is not available.

- Different programming languages are used for building Web applications.

Black-box scanner*:* Black-box means that the implementation details are not examined by the tester. According to inputs, outputs are verified with expected (predefined) behavior. Since code details are unknown, black-box testers should identify as many inputs as possible [28], to help him to detect as many vulnerabilities as possible, for this reason the XSS attack vector of input may have a large number.

By using a large-number of attack vector , XSS vulnerability detection may cost too much time, and the result may have a high false positive.

3 Objectives

In order to solve the above problem of black box scanner with large number of attack vector, we propose an approach to implement an efficient XSS scanner which aims to optimize its XSS attack vector in order to reduce the scan time and maximize the detected vulnerabilities point. Our work makes the following contributions:

- A method for generating and optimizing XSS vector attack: we propose a grammar to dynamic generation of all possible XSS scripts but we select the most promising ones.
- A method for detecting stored XSS; we search for each not searchable form (entry point for stored XSS injection) its related searchable form (entry point for stored XSS detection) , this link help us to go directly to the page in which the stored XSS code will be executed.

4 Report Outline

This report divided in four chapters:

The first chapter provides the basic concepts of Web Applications and Web Security , we present the top 10 OWASP web vulnerabilities, we focus on the Cross-Site Scripting(XSS)

The second chapter provides a general overview of Web Security Testing and different approaches of web scanners implementation, then we present the most popular web scanners.

The third chapter presents the conceptual design of our scanner with its different modules and methods for generating and optimizing XSS vector attack **,**and for detecting stored XSS**.**

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

Finally, we conclude this project by a general conclusion and perspectives.

CHAPTER 1

WEB APPLICATIONS AND WEB SECURITY

1.1 Introduction

Web applications are becoming more popular and widely being used in all aspects of work and social activities.. However, the exponential development of web technologies 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, then we give an overview of web Security and the common security problems addressed in Web applications , we continue by briefly explaining the XSS problem, then we present the classification of XSS, we describe which risks XSS may cause and some solutions for users and developers to defend against XSS attacks.

1.2 Definitions

Definition (1):

Web application or web app is a client–server software application which the client (or user interface) runs in a web browser [1].

Definition (2):

According to the definition of OWASP, "a Web application is a client/server software application that interacts with users or other systems using the HyperText Transfer Protocol (HTTP) [2].

1.3 Web application architecture

Web applications are generally structured as three-tiered, which consist of :

- The first tier is a Web browser(client) such as Google chrome, Mozilla Firefox and Opera, etc... .
- The middle tier is an engine, which generates pages dynamically using technologies such as PHP, ASP and JSP.
- The third tier is a database; it enables Web applications to store data and other content elements. By using SQL.

As illustrated in figure 1.1 , a client (Web browser) sends requests to the middle tier, which handles these requests, searches information required by making SQL queries against the

database and generates response pages using this information, and shows them to the user in the browser.

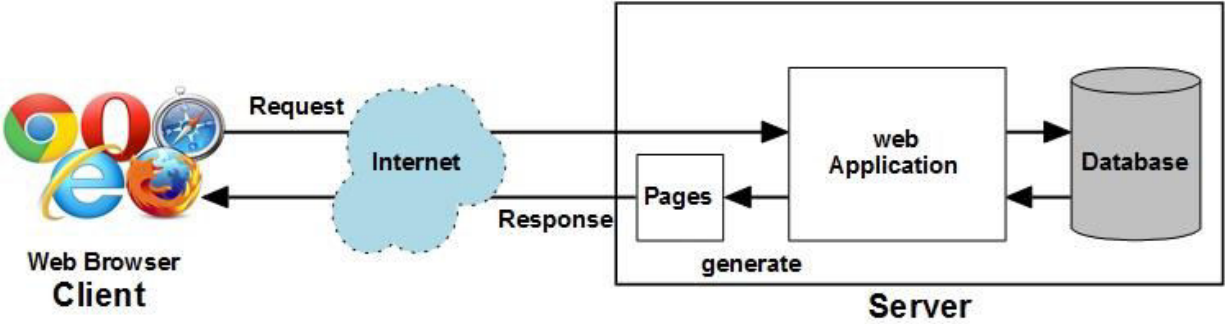

Figure 1.1 Dynamic Web Applications

1.4 The Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems .HTTP is the foundation of data communication for the World Wide Web [3].

1.4.1 Request methods:

HTTP defines methods (sometimes referred to as verbs) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server. The HTTP/1.0 specification [4] defined the GET, POST and HEAD methods and the HTTP/1.1 specification [5] added 5 new methods: OPTIONS, PUT, DELETE, TRACE and CONNECT.

GET: The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.) [3]

POST: The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database. [6]

1.4.2 HTTP Sessions:

An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (typically port 80, occasionally port 8080; see List of TCP and UDP port numbers). An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of this message is typically the requested resource, although an error message or other information may also be returned [3].

1.4.3 HTTP Requests

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format: [7]

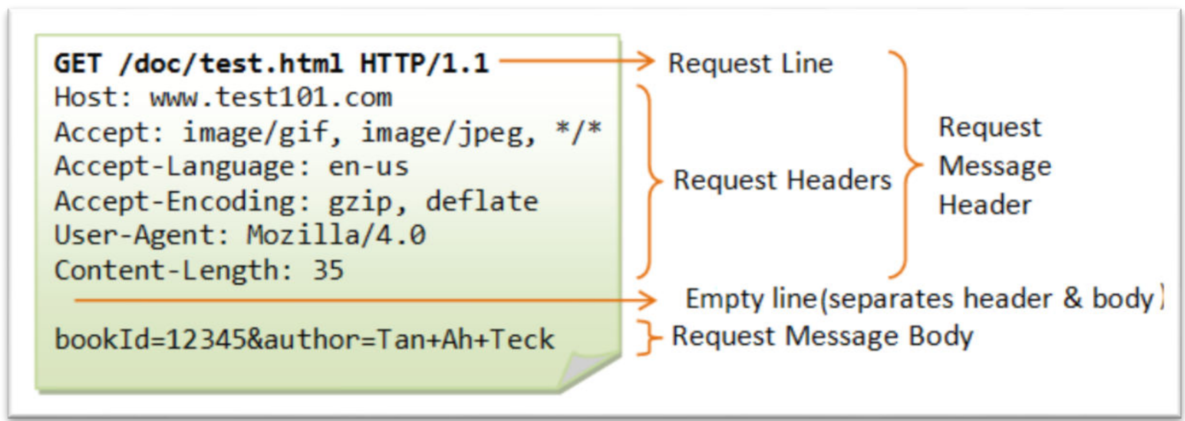

Figure 1.2 Sample HTTP request message [8]

1.4.3.1 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 [7].

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

1.4.3.2 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 [8].

1.4.4 HTTP Responses

After receiving and interpreting a request message, a server responds with an HTTP response message [8].

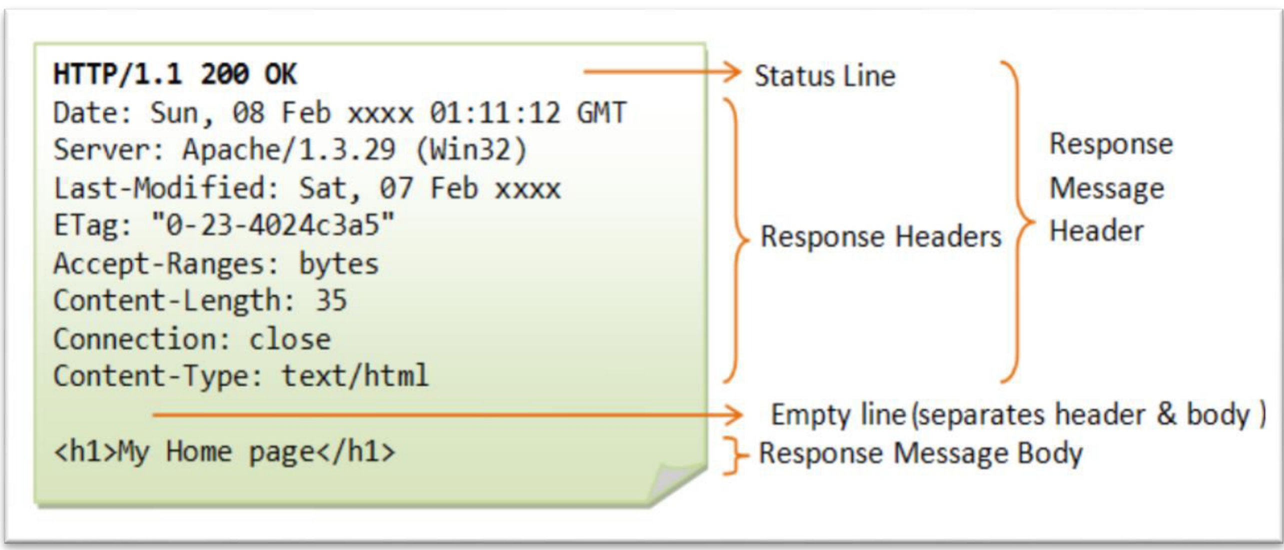

Figure 1.3 Sample HTTP response message [8]

1.5 Web Security

Security is a critical part of the Web applications. Web applications by definition allow users access to a central resource — the Web server — and through it, to others such as database servers. By understanding and implementing proper security measures, you guard your own resources as well as provide a secure environment in which your users are comfortable working with your application.

1.5.1 What is web application security:

Web application security is a large topic encompassing many disciplines, technologies, and design concepts. Normally, the areas we're interested in are the software layers from the Web server on up the vulnerability stack as illustrated in Figure 1.4 .This includes application servers such as JBoss, IBM WebSphere, BEA WebLogic, and a thousand others. Then we progress in the commercial and open source Web applications like PHP Nuke, Microsoft Outlook Web Access, and SAP. And after all that, there are the internal custom Web applications that organizations develop for themselves. This is the lay of the land when it comes to Web application security [9].

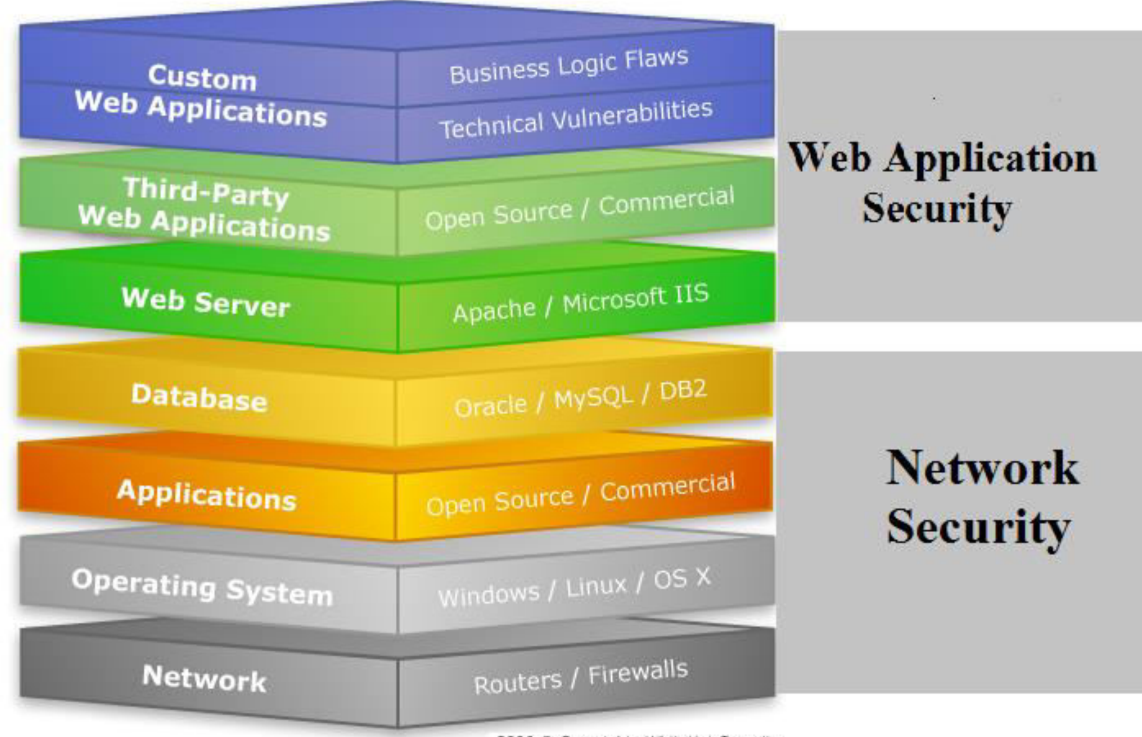

Figure 1.4 Vulnerability stack

.1.5.2 Basic Security Concepts

The three generally accepted aims of information systems security are confidentiality, integrity and availability (CIA) [10]. In a word, that is "the right information to the right people at the right time in the right context" [11]. Supplemented to CIA, authenticity and non-repudiation are also considered as important components of information security.

1.5.2.1 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 [12].

1.5.2.2 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

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 [12].

1.5.2.3 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 . [12]

1.5.2.4 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.

1.5.2.5 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.

1.6 Top 10 Web Security Vulnerabilities(2013)

The Open Web Application Security Project (OWASP) is an international organization dedicated to enhancing the security of web applications. As OWASP on 2013 publishes a list

of what it considers the current top 10 web application security risks worldwide [13]. Which are :

1.6.1:Injection flaws:

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

1.6.2: Broken Authentication:

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.

1.6.3: Cross Site Scripting (XSS):

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

1.6.4: Insecure Direct Object References:

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

1.6.5: Security misconfiguration :

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

1.6.6: Sensitive data exposure:

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

1.6.7: Missing function level access control:

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

1.6.8: Cross Site Request Forgery (CSRF):

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

1.6.9: Using components with known vulnerabilities:

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

1.6.10: Unvalidated redirects and forwards:

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

1.7 Cross-Site Scripting(XSS)

1.7.1 Introduction to Cross-Site Scripting

Cross Site Scripting (CSS for short, but sometimes abbreviated as XSS) is one of the most common application level attacks that hackers use to sneak into web applications today. Cross site scripting is an attack on the privacy of clients of a particular web site which can lead to a total breach of security when customer details are stolen or manipulated.

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.

1.7.2 The Problem:

The technological of Web applications consists of HTTP and HTML. Web applications generate dynamic Web pages containing both text and HTML markup (HTML uses special characters such as the less than sign "<" to identify the beginning of HTML tags like <body>,<img>....etc, in order to distinguish HTML markup from text).

When the input data in the request contains such special characters and Web applications generate response pages using this data, Web browsers will treat it as HTML tags, which can take effect on the formatting of the page or execute code written in scripting languages without any security consideration. Therefore, it is essential for Web applications to check the input data of users, otherwise Cross-Site Scripting vulnerabilities will result.

1.7.3 Example

index.php:

```
<?php
$name = $_GET['name'];
echo "Welcome $name<br>";
?>
```

Now the attacker will craft an URL as follows and send it to the victim:

```
index.php?name=<script>alert(xss)</script>
```

When the victim load the above URL into the browser, he will see an alert box

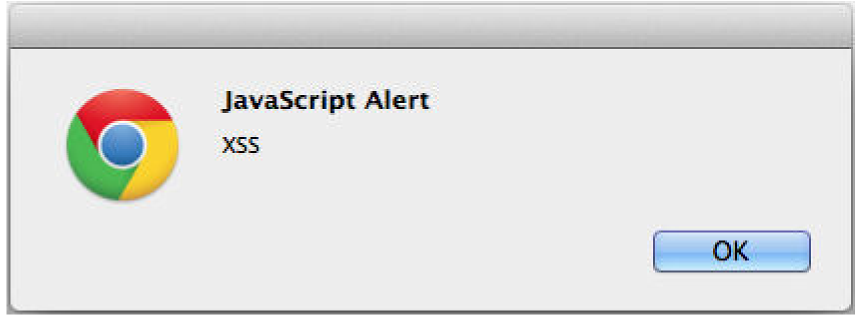

Figure 1.5 Alert message

1.7.4 XSS Classification :

Generally, Cross-Site Scripting attacks can be classified into three categories: non persistent (reflected), persistent (stored) and DOM-based[14].

1.7.4.1 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

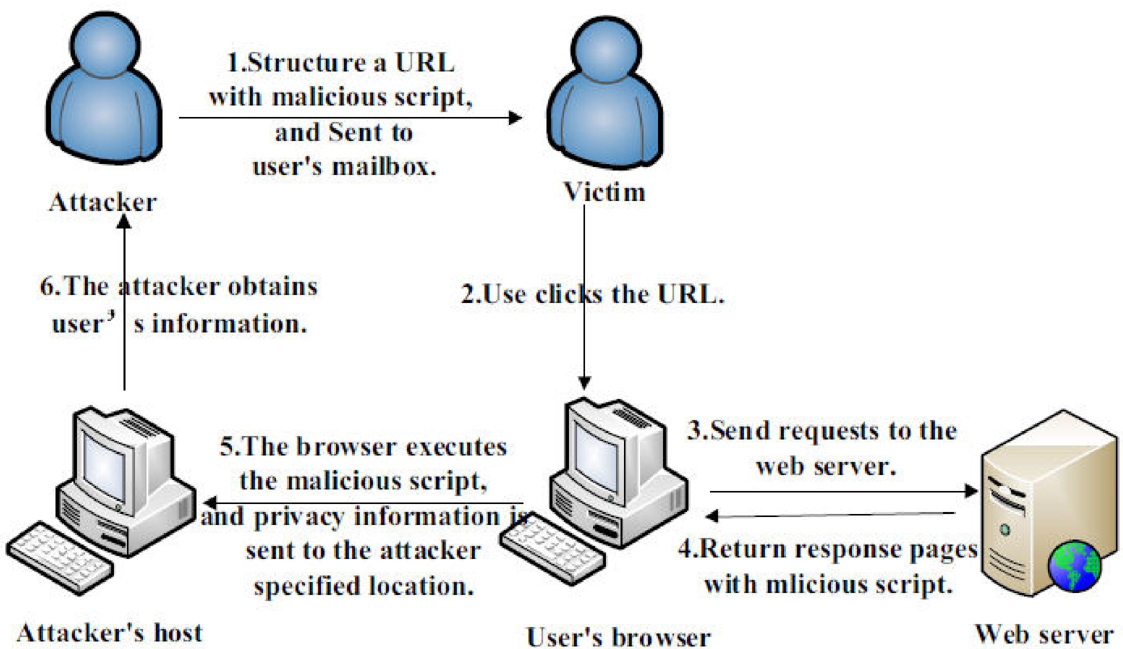

Figure 1.6 Reflected XSS vulnerability attack

1.7.4.2 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**.

1.7.4.3 DOM Based XSS (“type-0 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.

1.7.5 How to Determine If You Are Vulnerable:

XSS flaws can be difficult to identify and remove from a web application. The best way to find flaws is to perform a security review of the code and search for all places where input from an HTTP request could possibly make its way into the HTML output. Note that a variety of different HTML tags can be used to transmit a malicious JavaScript. Nessus, Nikto, and some other available tools can help scan a website for these flaws, but can only scratch the surface. If one part of a website is vulnerable, there is a high likelihood that there are other problems as well [14].

1.7.6 Dangers of XSS vulnerabilities

The XSS vulnerability can be exploited to do any or all of the following:

- Stealing a user's cookie.
- Modifying a web page.
- Collecting statistics.
- Exploit a browser vulnerability.
- Capturing Clipboard Contents.
- Stealing History and Search Queries.
- Port Scanning and other advanced attacks.

1.7.7 Prevention of Cross-Site Scripting Attacks:

In order to protect our Web Applications we need to check data received from user input, which can be in any part of the client request such as

- URL query string,
- HTML form fields, etc.. .

Therefore, we should perform input validation on the whole client request.

1.7.7.1 Client-side Solutions:

To protect users against XSS attacks, they should :

- Increase their general security awareness. For example, users should entering the link of the site directly in the address bar or using well-known search engines.
- Turn off all scripting languages in the browser settings. But many Web applications may become unusable when users disable scripting languages such as JavaScript.

Therefore, client-side solutions can only reduce the XSS problems, but not solve the problems completely .

The prevention of XSS attacks should depend on building secure Web applications.

1.7.7.2 Server-side Solutions:

Developers should build their Web applications with security in mind, never trust user input and always encode or filter the special characters. Table 1.1 lists some characters, which are considered as special based upon the HTML specifications.

A. Filtering:

Filtering based on the elimination of the special characters during input or output. There are two different methods of the implementation of filtering,

- **whitelists** :only accepting legal characters
- **blacklists** :excluding the special characters.

Blacklists are usually non-preferred, since they are never complete and thus may not defend against other unknown XSS vulnerabilities.

An example of a wise whitelist is to limit the values of a form field expecting a person's gender to "male" and "female" [15].

B. Encoding:

Encoding methods make such that the special characters lose their special meaning. For example, if user type the "<" character for mathematic formulas, and we removing the user input may result in data loss.

By using encoding, user input is encoded to the appropriate HTML characters before redisplayed to the user. As illustrated in figure 1.7 and 1.8, the "<" character is encoded as "**<**;" and the ">" character is encoded as "**>**" which are displayed as the less-than character, more than character, and Web applications treat it as normal text instead of HTML markup, so that there is no JavaScript execution any more.[16]

Example of Encoding character **:**

- & to &
- < to <
- > to >
- " to "
- ' to '
- / to /

```
<html>
......
Hello &lt;script&gt;alert(123)&lt;/script&gt;
.....
<html>
```

Figure 1.7 Sample HTTP Response Page after Encoding

What's your name? <script>alert(123)</script> Submit

Hello <script>alert(123)</script>

Figure 1.8 Prevention of Cross-Site Scripting with Encoding.

Character	Description
In the middle of text	
<	The less-than character introduces a HTML tag.
>	Some browsers treat the greater-than character as special, on the assumption that the author of the page really meant to put in an opening < but omitted it in error.
&	The ampersand introduces a character entity.
/	HTML tags are enclosed within the forward-slash.
Attribute values	
”	The double quote marks the end of the attribute value.
’	The single quote marks the end of the attribute value.
SPACE, TAB	They mark the end of the attribute value, if the attribute value does not use any quotes.
In URLs	
&	The ampersand character separates CGI parameters.
%	The percentage character is used for encoding characters.
+	The plus character can be used as a space within a URL.
SPACE, TAB	They mark the end of the URL.
Non-ASCII	Non-ASCII characters are not allowed within a URL.
Others	
:	The colon character is often used as script markers as javascript:alert(“XSS”) or in the HTML style attribute like background-image:url(bg.jpg).
()	Brackets are often used as script expressions. Examples are similar to “:”.
+	The plus character is often used to join strings in scripting languages.
’ and ”	The single and double quote are often used to enclose a variable in scripting languages.
;	The semicolon character is often used to mark the end of a statement in scripting languages.

Table 1.1 A List of Special Characters [17] [18]

1.8 Conclusion:

The core security problem faced by web applications arises in any situation where an application must accept untrusted data that may be malicious, in each entry point .

Most common security vulnerabilities in Web applications occur because of programming errors and lack of testing of entry point. If programmers focus on making intelligent design and careful implementation, and testing their Web applications purposefully and sufficiently, many security problems are avoidable.

CHAPTER 2

SCANNERS AND WEB SECURITY TESTING

2.1 Introduction

In the first chapter, we have introduced some server-side solutions to prevent the XSS problem, which are based on building secure Web applications. But how can we ensure that our Web applications are secure ?

In this chapter, we will talk about several different methods and tools for testing of Web application security issues. Before it, we give first an overview of testing and then discuss how to perform testing on Web applications.

2.1.1 Web application scanner

Web application scanners are computer programs which communicate with web applications through the web front-end in order to identify potential security vulnerabilities in the web application. Web application scanners are automated tools checking a website's applications for common security problems such as cross site scripting [19].

2.1.2 Security vulnerability

Security vulnerability is a weakness in a product that could allow an attacker to compromise the integrity, availability, or confidentiality of that product [20].

2.2 Introduction to Testing

when we ask Programmers What they think about testing , a lot of them think that testing is a demonstration of the correctness of their programs. Unfortunately, this is not correct. As Dijkstra said in [21], “Testing can show the presence of bugs but never their absence.” Therefore, the purpose of testing is to find errors and mistakes made by programmers in a program’s source code or its design.

Definition 2.1 (Testing) :“Testing is the process of executing programs with the intention of finding errors” [22].

Testing is targeted on finding errors, but what is considered as an error? When is an error present?

Definition 2.2 (Error): “A software error is present when the software does not do what the user reasonably expects it to do” [23].

For finding errors, a program should be verified with proper input data and its outputs should be compared with the expected results .

Including the pre- and postcondition, such pair of input and expected output data is also referred to as a test case.

Definition 2.3 (Test Case) "Test case is a specific set of test data along with expected results for a particular test objective" [24].

Definition 2.4 (Test Data) "Test data, input data and file conditions associated with a particular test case" [24].

Design of test cases is the most important step in the program testing. Successful test cases can increase the probability of finding most errors in a program. Test cases can be classified into the following two categories:

• **Positive** test cases verify the expected results with "normal" input data.

• **Negative** test cases verify the expected failures with faulty input data1.

Since testing of Web applications is quite different from testing of common programs, we give next an introduction to this special kind of testing.

2.3 Testing of web applications

Web applications can be verified by manual or automated testing.

2.3.1 Manual testing

By using Web browser, tester can click on links, submit forms with appropriate information, and verify if the response page is expected or not.

Manual testing of Web applications has the best advantage when the number of pages web are not many and not complex.

But, for large and complex Web applications manual testing is overextended. It becomes expensive, and need large time and effort.

2.3.2 Automated testing

Automated Testing is the ability to navigate Web pages automatically without using a Web browser ,but without using a Web browser ,

- How can we navigate in Web pages ?
- How can we enter data in a form ?
- How can we click on the submit button ?

The answer is to use tools that are able to emulate a **browser's behavior** like **HtmlUnit**. HtmlUnit is a testing framework written in Java. It can access a Web application from a Java

program and emulate a browser's behavior such as navigation, form submission, etc. and allow the Java program to verify the content of the returned pages. An example of HtmlUnit tests is illustrated in figure 2.1.

```
1 @Test
2 public void testGoogle(){
3     WebClient webClient = new WebClient();
4     HtmlPage currentPage =
  webClient.getPage("http://www.google.com/");
5     assertEquals("Google", currentPage.getTitleText());
6 }
```

Figure 2.1 HtmlUnit example simulate typing an address in the browser

2.4 Security testing of web applications

As more and more data is stored in web applications and the number of transactions on the web increases, proper security testing of web applications is becoming very important. Security testing is process to detect diverse vulnerabilities and determines if confidential data stays confidential or not.

In this section, we will give an introduction to static and dynamic approaches to conduct security testing. Some tools for testing of Web application security issues (particularly of XSS vulnerabilities) are also presented for both approaches.

2.4.1 Static testing

Static Application Security Testing (SAST) is a technique that statically analyzes program source code to detect problems within the source code. That is, SAST performs such analysis without actually executing (running) the source code.

In some examples, problems within the source code can compromise the security of a computer program. Such problems can be caused by unchecked (un-validated) data-flows from a sink, e.g., input from a user, to a source, e.g., access to a database [25], and we can inspect the source code by manual reviews or static analyses.

2.4.1.1 Review:

Manual reviews require reviewers to have a close look at the code. The reviewers should be familiar with the programming languages or frameworks used for development.

2.4.1.2 Static Analysis

Static analysis do not require the test objects being executed., static analysis are applied by means of automated or semi-automated tools , like compilers often apply static analysis for testing the syntax of program languages automatically.

2.4.2 Dynamic testing

Dynamic Application Security Testing (DAST) tests the application from the “outside” when the application is running in test or production environment. [26].

Testers usually use penetration testing for assessing Web application security. As the name implies, penetration testing is conducted by simulating the potential attacks with a predefined goal.

2.5 Types of vulnerability scanner

Generally, there are two vulnerability scanners:

2.5.1 White-box scanner :

Provides the penetration testers with the knowledge of implementation details (e.g. the internal structure of the program).From this information; test cases are created according to the coverage criteria [27].

Tools taking white-box approaches suffer from the following shortcomings:

- Sometimes source code is not available.
- Different programming languages are used for building Web applications.

Example: tools like:

- FORTIFY .
- Ounce .
- Pixy .

2.5.2 Black-box scanner

In contrast to white-box testing, black-box testing (also referred to as functional testing) is only concerned with functionality. Black-box means that the implementation details are not examined by the tester. According to inputs, outputs are verified with expected (predefined) behavior. Since code details are unknown, black-box testers should identify as many inputs as possible. [28].

Example:

- W3af.
- Wapiti .

- Acunetix Vulnerability Scanner.
- Zed Attack Proxy.

2.6 Techniques for Vulnerability Detection:

Many techniques are available for identifying web application vulnerabilities [28]:

2.6.1 Fault Injection

Fault injection is a testing technique that introduces faults in order to test the behavior of the application, some knowledge about the application is required to generate the possible faults. With fault injection is possible to find security flaws in the system.

2.6.2 Fuzzing Testing

The idea of this test is to provide random data as input to the application in order to determine if the application can handle it correctly. Fuzzing testing is easier to implement than fault injection because the test design is simpler and previous knowledge about the application to test is not always required, additionally it is limited to the entry points of the program.

2.6.3 Syntax Testing

Syntax testing refers to testing that is based on the syntactic specification of an application's input values. The idea is to determine what happens when inputs deviate from this syntax .

Syntax testing helps the tester confirm that input values are being checked correctly, which is important when developing secure software.

2.6.4 monitoring program behavior:

When a large number of tests are automatically applied, it is useful to also have automatic techniques for monitoring how the program responds. This saves testers from having to check for anomalous behavior manually. Of course, a human is better at seeing anomalous behavior, but the anomalies that signal the presence of a security vulnerability are often quite obvious.

2.7 Related work on XSS

There are a different approaches for Detection of Cross-site Scripting (XSS) vulnerabilities

2.7.1 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.

Y.W Huang, F. Yu, C. Hang, C. H. Tsai (2004) [43] 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 [44**].** 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 **[45].** 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 [46].

Gary and Zhendong [47] presented a static analysis for finding XSS vulnerabilities that directly addresses weak or absent input validation. Proper input validation is difficult largely because of the many ways to invoke the JavaScript interpreter; they faced the same obstacle checking for vulnerabilities statically, and they address it by formalizing a policy based on the W3C recommendation, the Firefox source code, and online tutorials about closed-source browsers. They provide effective checking algorithms based on their policy. And they implemented their approach and provided an extensive evaluation that finds both known and unknown vulnerabilities in real-world web applications.

2.7. 2 Dynamic Analysis Approach:

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

2.7.2.1 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 [48]. 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 [49]. Every client needs to have this modified version of browser on their system.

2.7.2.2 Syntactical Structure Approach: Su and Wassermann in [50] 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 [51].

2.7.2.3 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 [52]. 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 [53].

And there is other approaches and methods of XSS vulnerability detection like :

Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic developed SecuBat in this work [54], a generic and modular web vulnerability scanner that, similar to a port scanner, automatically analyzes web sites with the aim of finding XSS vulnerabilities.
they were able to find many potentially vulnerable web sites. To verify the accuracy of SecuBat, they picked one hundred interesting web sites from the potential victim list for further analysis and confirmed exploitable flaws in the identified web pages. Among their victims were well-known global companies and a finance ministry. More than fifty responded to request additional information or to report that the security hole was closed.

Xiaobing Guo, Shuyuan Jin, and Yaxing Zhang proposes [55] in this work a method of XSS vulnerability detection using optimal attack vector repertory.
This method generates an attack vector repertory automatically, optimizes the attack vector repertory using an optimization model, and detects XSS vulnerabilities in web applications dynamically. an optimization model is built in this work with a machine learning algorithm, reducing the size of the attack vector repertory and improving the efficiency of XSS vulnerability detection. Based on this method, an XSS vulnerability detector is implemented, which is tested on 50 real-world websites. The testing results show that the detector can detect a total of 848 XSS vulnerabilities effectively in 24 websites.

Fabien Duchene and Sanjay Rawat propose in this work [56] KameleonFuzz, a blackbox Cross Site Scripting (XSS) fuzzer for web applications.
KameleonFuzz can not only generate malicious inputs to exploit XSS, but also detect how close it is revealing a vulnerability.
The malicious inputs generation and evolution is achieved with a genetic algorithm, guided by an attack grammar.
A double taint inference, up to the browser parse tree, permits to detect precisely whether an exploitation attempt succeeded.
This evaluation demonstrates no false positives and high XSS revealing capabilities,KameleonFuzz detects several vulnerabilities missed by other black-box scanners.

Thiago S. Rocha and Eduardo developed ETSSDetector in this work [57], a generic and modular web vulnerability scanner that automatically analyzes web applications to find XSS vulnerabilities.
ETSSDetector is able to identify and analyze all data entry points of the application and generate specific code injection tests for each one.
The results shows that the correct filling of the input fields with only valid information ensures a better effectiveness of the tests, increasing the detection rate of XSS attacks.

Fabien Duchene, Roland Groz, Sanjay Rawat, Jean-Luc Richier in This work [58] present an approach to detect web injection vulnerabilities by generating test inputs using a combination of model inference and evolutionary fuzzing.

Model inference is used to obtain a knowledge about the application behavior. Based on this understanding, inputs are generated using genetic algorithm (GA).
GA uses the learned formal model to automatically generate inputs with better fitness values towards triggering an instance of the given vulnerability.

2.7.3 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.
There is a tool called WebSSARI [59] 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 [59]. 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 [60]. 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 [61] 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 Vulnerability Scanning Tools

Web Application Vulnerability Scanners are the automated tools that scan web applications to look for known security vulnerabilities such as cross-site scripting, a large number of both commercial and open source tools are available and all these tools have their own strengths and weaknesses [29].
In the table 2.1 we will provide a listing of vulnerability scanning tools currently available in the market.   The tools listing in the table has been presented in an alphabetical order

Name	Owner	Licence	Platforms
Acunetix WVS	Acunetix	Commercial / Free (Limited Capability)	Windows
AppScan	IBM	Commercial	Windows
App Scanner	Trustwave	Commercial	Windows
AVDS	Beyond Security	Commercial / Free (Limited Capability)	N/A
BugBlast	Buguroo Offensive Security	Commercial	SaaS or On-Premises
Burp Suite	PortSwiger	Commercial / Free (Limited Capability)	Most platforms supported
Contrast	Contrast Security	Commercial / Free (Limited Capability)	SaaS or On-Premises
GamaScan	GamaSec	Commercial	Windows
Grabber	Romain Gaucher	Open Source	Python 2.4, BeautifulSoup and PyXML
Grendel-Scan	David Byrne	Open Source	Windows, Linux and Macintosh
GoLismero	GoLismero Team	GPLv2.0	Windows, Linux and Macintosh
IKare	ITrust	Commercial	N/A
IndusGuard Web	Indusface	Commercial	SaaS
N-Stealth	N-Stalker	Commercial	Windows
Netsparker	MavitunaSecurity	Commercial	Windows
Nexpose	Rapid7	Commercial / Free (Limited Capability)	Windows/Linux
Nikto	CIRT	Open Source	Unix/Linux
AppSpider	Rapid7	Commercial	Windows
ParosPro	MileSCAN	Commercial	Windows
Proxy.app	Websecurify	Commercial	Macintosh
QualysGuard	Qualys	Commercial	N/A
Retina	BeyondTrust	Commercial	Windows
Securus	Orvant, Inc	Commercial	N/A
Sentinel	WhiteHat Security	Commercial	N/A
Vega	Subgraph	Open Source	Windows, Linux and Macintosh
Wapiti	Informática Gesfor	Open Source	Windows, Unix/Linux and Macintosh
WebApp360	TripWire	Commercial	Windows
WebInspect	HP	Commercial	Windows
SOATest	Parasoft	Commercial	Windows / Linux / Solaris
Trustkeeper Scanner	Trustwave SpiderLabs	Commercial	SaaS
WebReaver	Websecurify	Commercial	Macintosh
WebScanService	German Web Security	Commercial	N/A
Websecurify Suite	Websecurify	Commercial / Free (Limited Capability)	Windows, Linux, Macintosh
Wikto	Sensepost	Open Source	Windows
w3af	w3af.org	GPLv2.0	Linux and Mac
Xenotix XSS Exploit Framework	OWASP	Open Source	Windows
Zed Attack Proxy	OWASP	Open Source	Windows, Unix/Linux and Macintosh

Table 2.1 Vulnerability Scanners Tools [29]

2.8.1 Wapiti

Wapiti allows you to audit the security of your web applications .It performs "black-box" scans, i.e. it does not study the source code of the application but will scans the web pages of the deployed web application, looking for scripts and forms where it can inject data [30]. Once it gets this list, Wapiti acts like a fuzzer, injecting payloads to see if a script is vulnerable. Wapiti can detect the following vulnerabilities :

- File disclosure (Local and remote include/require, fopen, readfile...)
- Database Injection (PHP/JSP/ASP SQL Injections and XPath Injections)
- XSS (Cross Site Scripting) injection (reflected and permanent)

- Command Execution detection (eval(), system(), passtru()...)
- CRLF Injection (HTTP Response Splitting, session fixation...)
- XXE (XmleXternal Entity) injection
- Use of know potentially dangerous files (thanks to the Nikto database)
- Weak .htaccess configurations that can be bypassed
- Presence of backup files giving sensitive information (source code disclosure)

2.8.2 W3af

w3af is a Web Application Attack and Audit Framework**.** The project's goal is to create a framework to help you secure your web applications by finding and exploiting all web application vulnerabilities

2.7.2.1 w3af's architecture

The w3af framework is divided into three main sections:

1. The core, which coordinates the whole process and provides libraries for using in plugins.
2. The user interfaces, which allow the user to configure and start scans
3. The plugins, which find links and vulnerabilities

2.7.2.2 w3af's phases

w3af follows the steps you would perfom in a web application penetration test. In order to do so it defines different types of plugins which are going to be called by the core in a specific order.

Starting with a target URL provided by the user, w3af will first try to identify all URLs, forms and query string parameters in the application by the means of crawl_plugins. A very good example of this type of plugin is the web_spider which will extract URLs from a page, follow those links and once again extract URLs from it. Following that process it will create a complete application link and form map.

Once the application has been mapped, audit plugins will send specially crafted strings to each parameter in order to trigger bugs in the application's code. When a bug is found it will be reported to the user.

Identified vulnerabilities, debug and error messages, all are reported to the user with output_plugins**.** These plugins will write the messages in different formats to suit your needs. In most cases a text file is what users need, but for integration into other tools XML file format is also available. [31].

2.8.3 Zed Attack Proxy

Zed Attack Proxy is also known as ZAP. This tool is open source and is developed by AWASP. It is available for Windows, Unix/Linux and Macintosh platforms. I personally like this tool. It can be used to find a wide range of vulnerabilities in web applications. The tool is very simple and easy to use. Even if you are new to penetration testing, you can easily use this tool to start learning penetration testing of web applications [32].

These are the key functionalities of ZAP:

- Intercepting Proxy
- Automatic Scanner
- Traditional but powerful spiders
- Fuzzer
- Web Socket Support
- Plug-n-hack support
- Authentication support
- REST based API
- Dynamic SSL certificates
- Smartcard and Client Digital Certificates support

2.8.4 Acunetix Vulnerability Scanner

Securing the web applications of today's businesses is perhaps the most overlooked aspect of securing the enterprise. Web application hacking is on the rise with as many as 75% of cyber attacks done at web application level or via the web.

Most corporations have secured their data at the network level, but have overlooked the crucial step of checking whether their web applications are vulnerable to attack. Web applications which often have a direct line into the company's most valuable data assets are online 24/7, completely unprotected by a firewall and therefore easy prey for attackers.

Acunetix was founded with this threat in mind. It was understood that the only way to combat website hacking was to develop an automated tool that could help companies scan their web applications to identify and resolve exploitable vulnerabilities. In July 2005, Acunetix Web Vulnerability Scanner was released: a heuristic tool designed to replicate a hacker's methodology to find dangerous vulnerabilities like SQL injection and cross site scripting before hackers do. A decade later and Acunetix Vulnerability Scanner has become the tool of

choice for many customers in the Government, Military, Educational, Telecommunications, Banking, Finance, and E-Commerce sectors, including many Fortune 500 companies.
Acunetix Vulnerability Scanner is available both as an online and on premise solution. It detects and reports a wide array of vulnerabilities in applications built on architectures such as WordPress, PHP, ASP.NET, Java Frameworks, Ruby on Rails and many others. Acunetix Vulnerability Scanner brings an extensive feature-set of both automated and manual penetration testing tools, enabling security analysts to perform a complete vulnerability assessment, and repair detected threats, with just the one product. Results can be used to generate reports aimed towards developers and management alike.
The Acunetix development team consists of highly experienced security developers, all with extensive development experience in network security scanning software prior to working on Acunetix WVS. The management team is backed by years of experience in marketing and selling security software. [33].

2.8.5 Vega Vulnerability Scanner :

Vega is a free and open source scanner and testing platform to test the security of web applications. Vega can help you find and validate SQL Injection, Cross-Site Scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities. It is written in Java, GUI based, and runs on Linux, OS X, and Windows.
Vega includes an automated scanner for quick tests and an intercepting proxy for tactical inspection. The Vega scanner finds XSS (cross-site scripting), SQL injection, and other vulnerabilities. Vega can be extended using a powerful API in the language of the web: Javascript.
Vega was developed by Subgraph in Montreal.[34]

2.9 Conclusion

In this chapter we presented several concepts and automated tools for Web application security scanners.

Some of them are commercial and closed source, , some of scanners are limited by the technologies used in Web applications (eg PHP) . While others can't detect non-persistent XSS.

In the next chapter we will present our proposed approach to build an scanner of XSS vulnerability, an automated testing tool using the black box approach for the detection of cross-site scripting vulnerabilities in web applications.

CHAPTER 3

OPTIMIZED XSS VULNERABILITY SCANNER APPROACH

3.1 Introduction

We have seen in the previous chapters that XSS vulnerabilities which commonly present in most Web applications can create serious security problems.

Due to the complexity and code size of such web applications, automatic detection of XSS is a non-trivial problem .

In our work, in order to improve the efficiency of XSS vulnerability detection, we propose a black box detection approach using optimal attack vector repertory. This method generates an attack vector repertory automatically, optimizes the attack vector repertory using an optimization model, and detects XSS vulnerabilities in web applications dynamically.

3. 2 Contributions:

This work makes the following contributions.

- A method for detecting stored XSS; we search for each not searchable form (entry point for stored XSS injection) its related searchable form (entry point for stored XSS detection.)
- A method for generating and optimizing XSS vector attack: we propose a grammar to dynamic generation of all possible XSS scripts but we select the most promising ones.

3.3 Scanner architecture

Our scanner architecture should to respond to the following problems:

- How we can identify all injection entry points in the web application?
  - For this problem we create a specified page crawler to identify all types of injection point: URL, Form input: stored injection point, reflected injection point.
- Which injection code should we use?
  - For this problem we create a dynamic XSS generator based on XSS grammar, and use mutation rules to select the most promising ones.
- How can we detect that the injected code is executed?
  - For this problem we have two cases: in the case of reflected XSS detection, we compare the response page with the injected XSS, but in stored XSS detection, we

first search the linked pages of this injection point, then we submit search key values to extract the stored injected script.

Our XSS scanner have three basic phases, as illustrated in figure 3.1 :

- Crawling phase
- Injection phase
- Detection phase

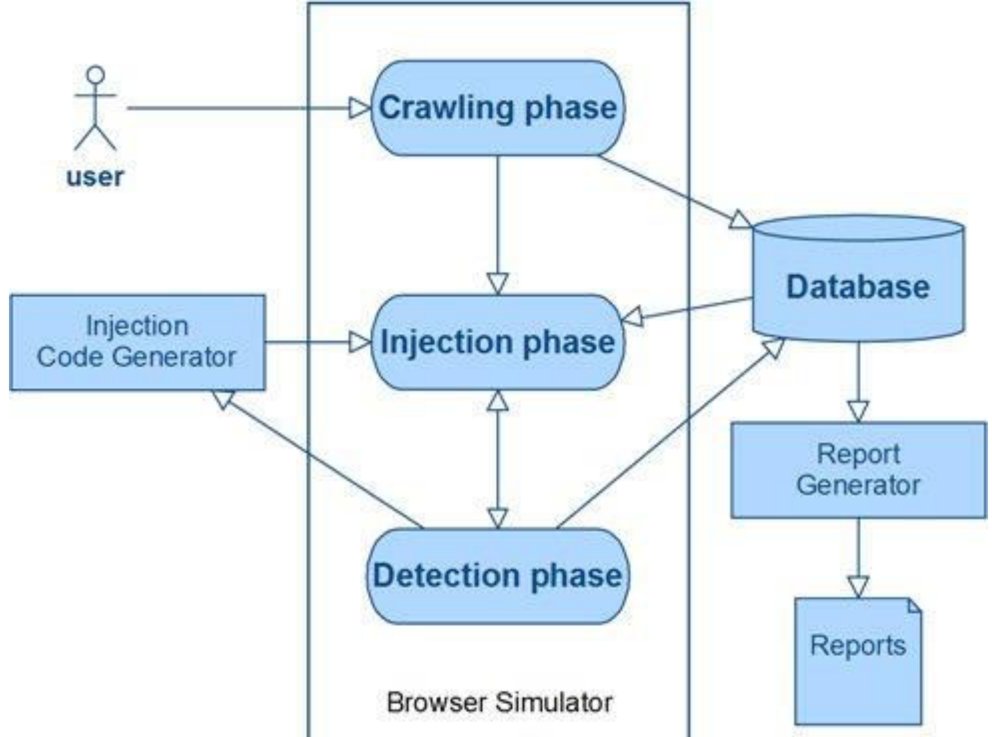

Figure 3.1 Architecture of XSS vulnerabilities Scanner

3.3.1 Crawling phase

As described previously, for testing the Web application we need to collect the information from the Web application.

This information, which will be required later in the injection and detection phase, includes all data entry points and all pages in the application.

3.3.1.1 Definition Web crawler:

(also known as a web spider or web robot) is the process of finding web links starting from a list of seed URL's. Based on the Browser Simulator.

3.3.1.2 Browser Simulator

In order to detect XSS vulnerabilities in Web application automatically, Our Scanner should be able to simulate a browser's behavior, which are:

- Navigating Web applications,
- Form submission,
- Analyzing the returned HTTP responses.

For this reason our crawler uses two libraries:

- ***HtmlUnit***, which provide the capabilities of navigating Web applications, working with forms, and analyzing the returned HTTP responses, and support javascript .
- ***Jsoup*** :java html parser. which is a java library that is utilized to parse HTML record. **Jsoup gives programming** interface to concentrate and control information from URL or HTML document.[35]

A. *Navigating Web applications :*

Using a "normal" Web browser, we can navigate a Web application by entering a URL in the address bar or following links within a page. Our Browser Simulator should be able to do the same thing:

- Navigating to a particular resource: the Browser Simulator should provide the possibility of navigating to a page by a given URL address.

```
final WebClient webClient = new WebClient()) {
final HtmlPage page1 = webClient.getPage("https://www.google.dz"); }
```

Figure 3.2 navigating www.google.dz

- Navigating through links: In certain cases, we need to navigate through the links specified in a page.

```
final WebClient webClient = new WebClient() {
  final URL url = new URL("http://htmlunit.sourceforge.net");
    final HtmlPage page = (HtmlPage)webClient.getPage(url);
    HtmlAnchor anchor = page.getAnchorByName("Home");
    anchor.click();
};
```

Figure 3.3 click the "Home" link on the HTMLUnit homepage

B. *Handling Forms :*

HTML Forms are essential for Web interactions between users and Web servers. An HTML form consists of several form fields such as text input fields or checkboxes, etc.

Our Browser Simulator should have the ability to :

- Identify a form,

- Get input fields from,
- Access to form fields and Modify their values by entering text
- Submit the form.

- *Form Identification:*

A page can contain several forms. because that , such forms must be identifiable by ID, But Problems will occur if a form created without a name (or an ID) specified .or a forms on a page with the same name .So, we have to use another approach to identify forms, that is, assigning forms with IDs according to the order of the forms on a page. As illustrated in figure 3.4**.**

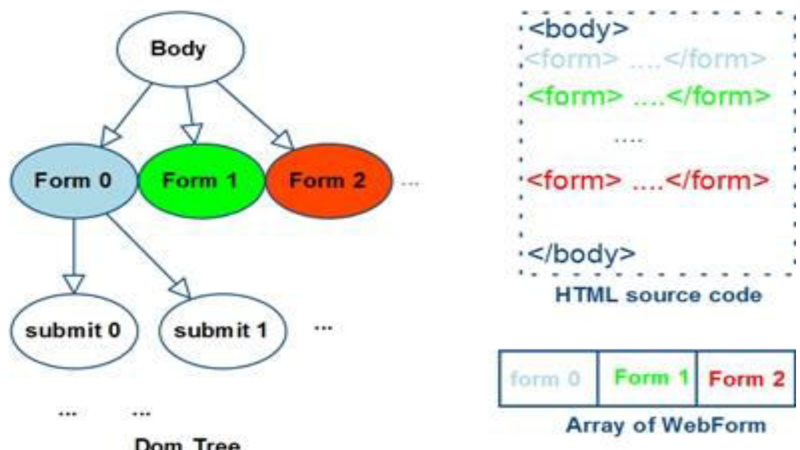

Figure 3.4 Form Identification

by using :HtmlForm form = page.getForms().get(0);

- *Get Form input fields*:

When a form has been identified we can access it with the ID specified and get all its fields parameters ,

```
for (Element input : form.select("input,select,textarea"))
{
System.out.printf("name=%s,type=%s",input.attr("name"),input.attr("type"));
}
```

Figure 3.5 Get fields parameters of form

- *Filling a Form:*

We use the parameters of form fields, to access and fill them by setting values , there are a different types of form fields should be treated differently :

Type	Method
Text field	HtmlTextInput textField = form.getInputByName("pseudo");

	textField.setValueAttribute("XSS");
Password field	HtmlPasswordInput passwordField = form.getInputByName("psw"); passwordField.setValueAttribute("XSS");
Time field	HtmlTimeInput TimeField = form.getInputByName("time"); TimeField.setValueAttribute("XSS");
·······	······

Table 3.1 Setting Form Field Values

- *Submitting a Form*

Our Browser Simulator can submits a form, by using the name of the submit buttons to submit the form with a particular button through its name.

```
HtmlSubmitInput button = form.getInputByName("submit");
final HtmlPage page2 = button.click();
```

Figure 3.6 Submitting a Form by name

But if the form has multiple submit buttons. We search the type submit with an XPATH query:

```
HtmlSubmitInput button =  form.getByXPath("//input[@type='submit']").get(0);
final HtmlPage page2 = button.click();
```

Figure 3.7 Submitting a Form by XPath

C. *Analyzing HTML Pages*

The Browser Simulator can analyses html pages in two different ways as following:

- Analyzing the HTTP response text. When the Browser Simulator has navigated to a page, the response text is returned as string data, so that we can analyses .by using function **asText()**
- Analyzing the DOM tree. The structure of a page can be represented in a DOM tree. HtmlUnit offer the possibility of building up the DOM tree from an HTML page. by using function **asXml()**

3.3.1.3 The Crawling Process

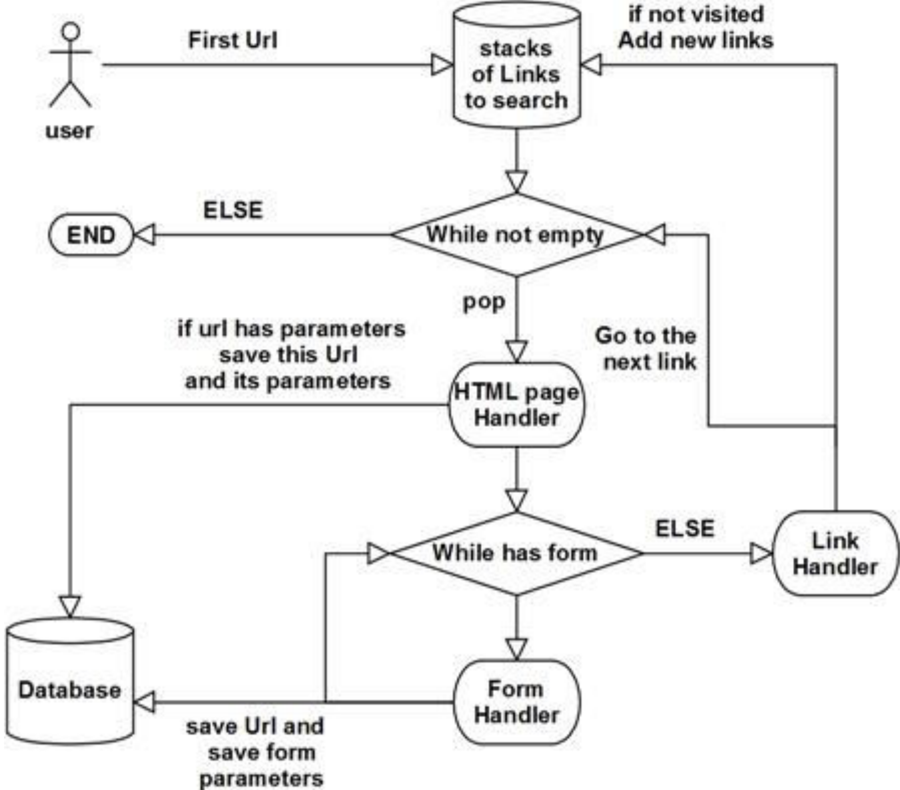

Figure 3.8 The Crawling Process

- the HTMLHandler is offer basic functions to deal with an URL or an HTML page.
- the LinkHandler is responsible for handling all links included in a page.
- the FormHandler is responsible for dealing with the forms on a page .

3.3.1.4 Identifying Data Entry Points

The HTMLHandler provides the capability of identifying data entry points from a given Web page. Currently, we focus on the following sources of user inputs :

- URL query string
- HTML form fields.

A. *URL Parameters :*

URL query string is one of the possible sources of malicious code (especially for non persistent XSS). URL query string consists of name/value pairs, which are separated by the "&" character, while the parameter name and value are delimited by the "=" character.

Given a URL, we can first cut out its query string. To extract all parameter names from the query string, the following steps should be followed as illustrated in Figure 3.9:

1. Split the given URL query via the "?" character in two parts .
2. Split the second part(the part after "?") via "&" in name/value pairs.
3. Split the name/value pair via the "=" character.
4. Repeat step 3, until all pairs are processed.

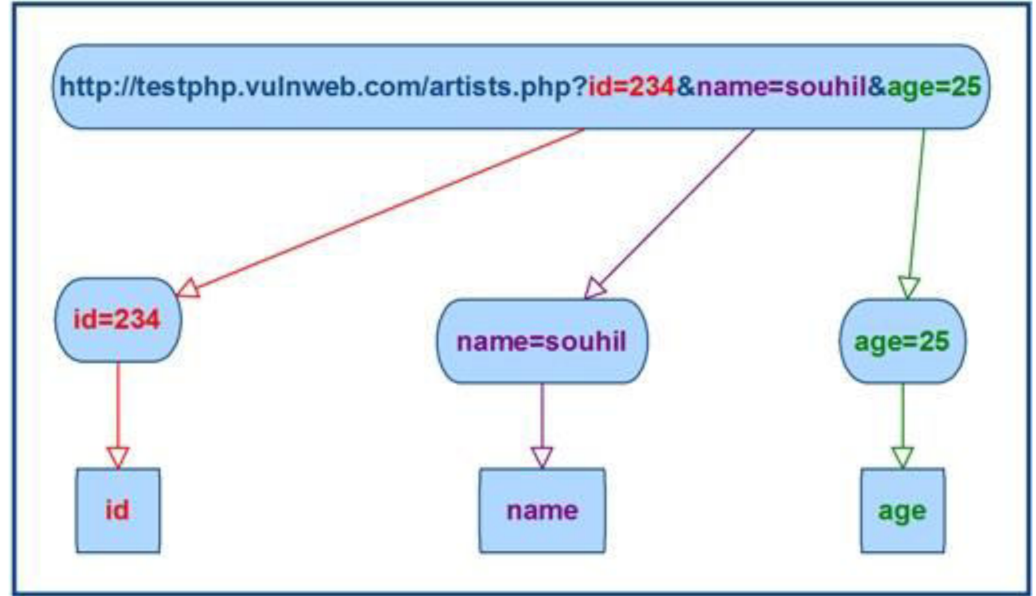

Figure 3.9 Extracting URL Parameters

B. *Form Parameters*

The HTMLHandler offers also basic functions for retrieving all form fields from a page, which are used in the FormHandler, as we have seen in **3.3.1.2 B**

For each form of a DOM tree, we can traverse all children of its form node, as illustrated in figure 3.10. We can also access a group of nodes directly,

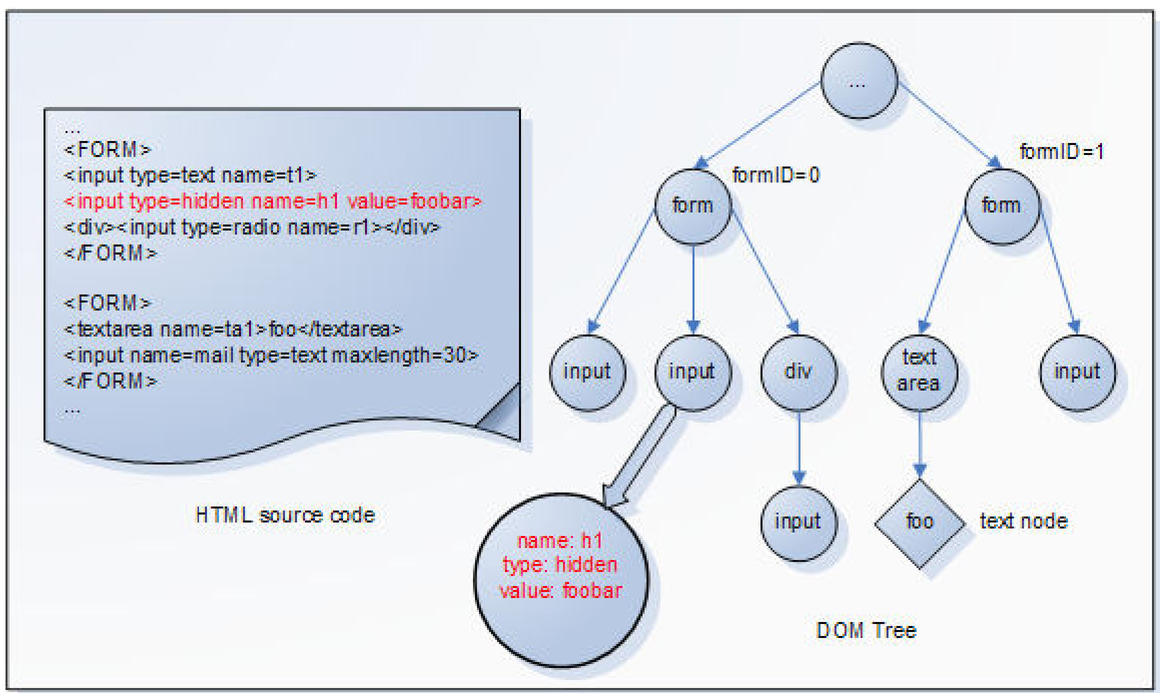

Figure 3.10 Extracting Form Parameters

3.3.1.5 Form Crawler

As described in the First Chapter, XSS attacks can be classified into three categories, Reflected XSS Attacks ,Stored XSS Attacks, and DOM Based XSS .

Stored XSS is the most dangerous one, because the attacks stored in the database and could be showed anywhere in the site web and to any user .

The form of page web could be :

- searchable (search forms.).
- non-searchable (other forms)

Our goal is to develop a system (Form Crawler) able to detect e automatically whether a particular web form is searchable or non-searchable .

Steps of Form Crawler:

Our form crawler works in three steps :

A) Recognizing searchable and non-searchable forms:

we Recognize that the biggest difference between searchable and non-searchable forms is number of the inputs, generally if number of inputs<2 this mean that the form is searchable , else the form is non-searchable.

During the crawling, we add this test to Form Handler to generate two form Lists

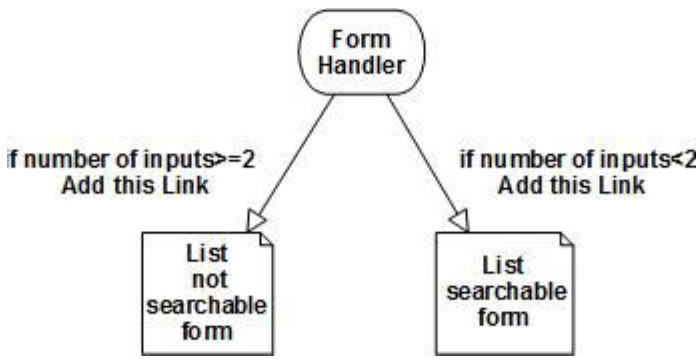

Figure 3.11 Form structure classification

B) Get the Relation between searchable and non-searchable forms:

As we see in example when attacker register with Malicious Code on non-searchable form , The problem and the risk appears in many searchable pages ,because every non-searchable form have relation with one or more searchable page, so we need to get all the pair of (non-searchable / searchable) forms, to do this relation we propose the following algorithm:

```
for each (n_f IN List_non-searchable)
    Label_N =Get_Label(n_f)
    for each (s_f IN List_searchable )
        Label_S =Get_Label(s_f)
        for each (L  IN List Label_S)
            if(Contain_Label (L, Label_N))
                CREAT_PAIR ()
            end if
        end for each
    end for each
end for each
```

Algorithm 1.1: searchable-non_searchable pair extraction

N_f : is a form in the list of non searchable forms .

Label_N: is the list of labels on non searchable form.

s_f : is a form in the list of searchable forms .

Label_S: is the list of labels on non searchable form.

Contain_Label: is a Boolean function which return true if a searchable label exist in a non searchable form.

Create_Pair: is a function which return a pair of forms: non searchable and its corresponding searchable form.

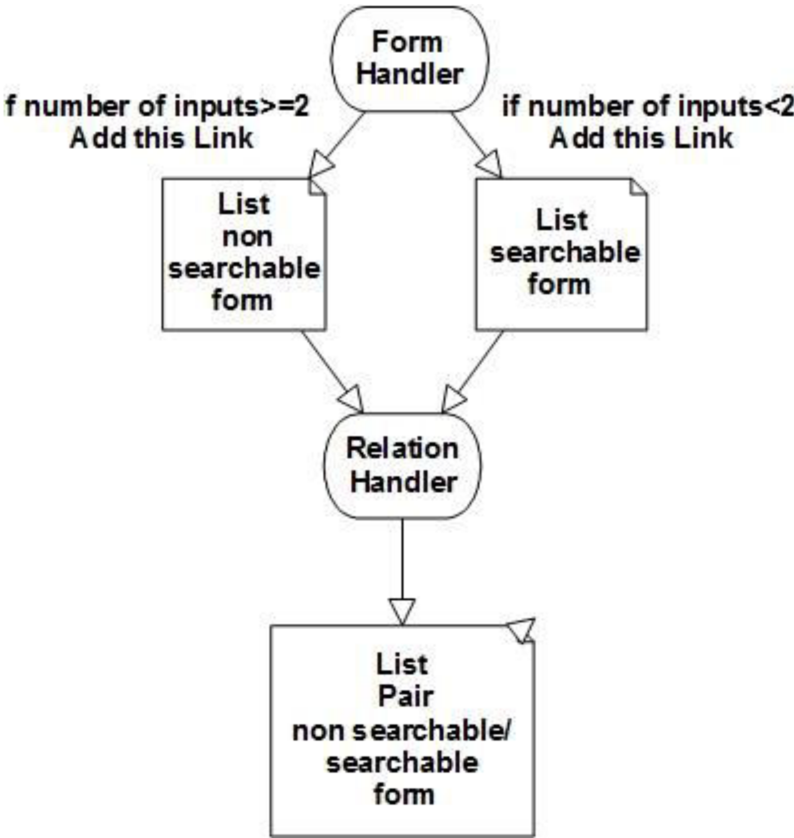

Figure 3.12 Forms pair generation

C) Test of the relation pair (non-searchable / searchable):

To validate the form pair, we fill the fields of every non-searchable form with predefined values, then we call it's Corresponding searchable forms for testing. if the response page of the searchable form contains one of the predefined values, we validate the pair , else we remove it from the list of pair.

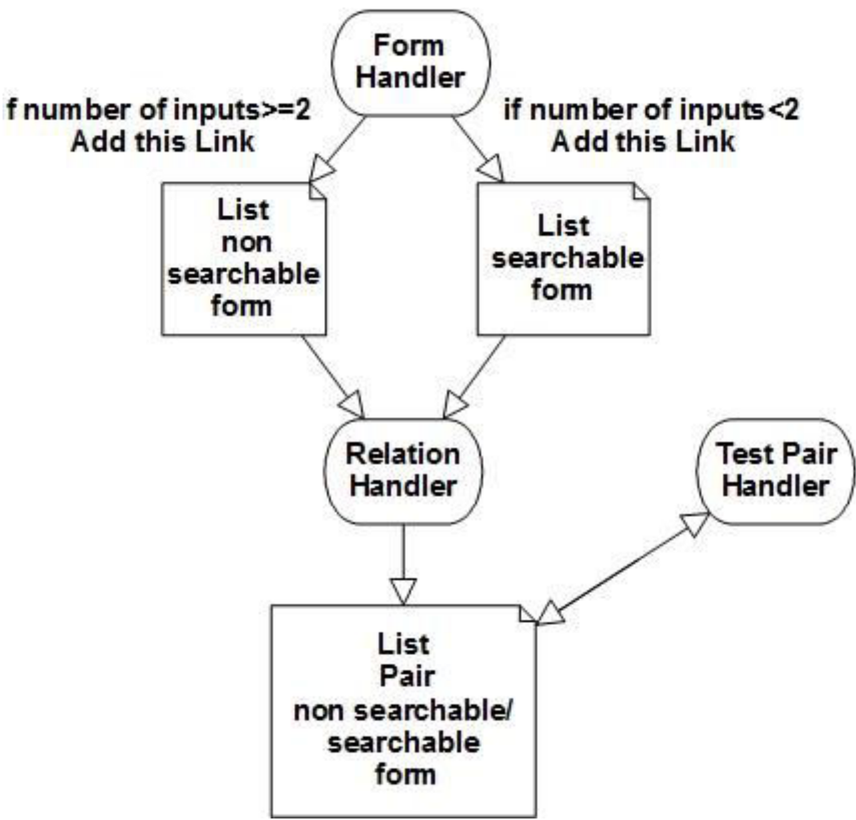

Figure 3.13 pair validation step

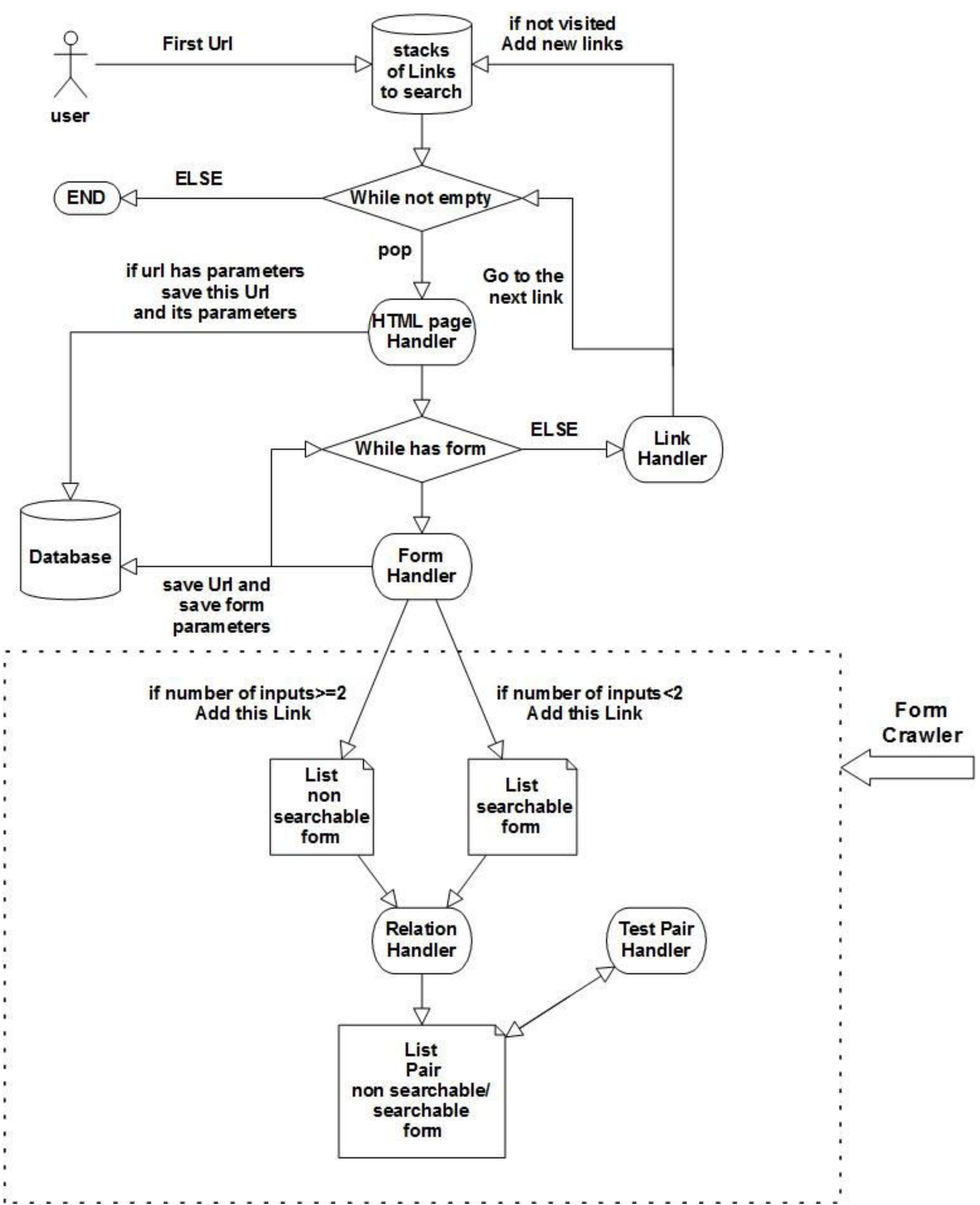

Figure 3.14 The Final Crawling Process

3.3.1.6 Database

As mentioned earlier, to detect XSS vulnerabilities we must collect a lot of information on Web pages, as well as their forms and store them in a database, in order to save this information effectively and for simplicity.

- Class Diagram:

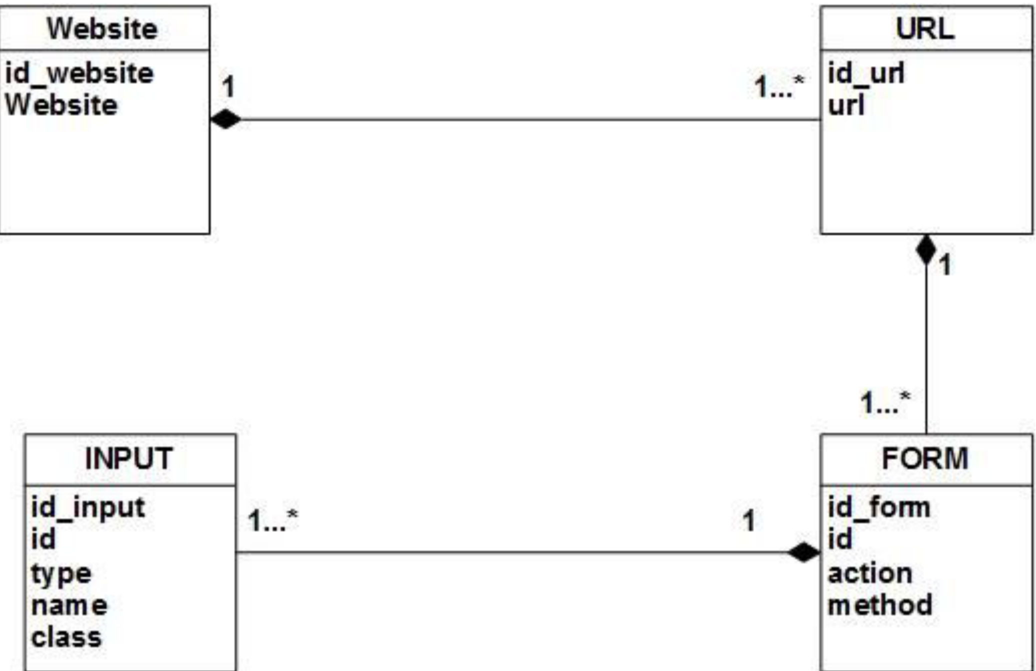

Figure 3.15 crawler class diagram

- Relational Model

- Website (**<u>id_website</u>,** Website)

- URL (**<u>id_url</u>,** url, **id_website**)

- FORM (**<u>id_form</u>,** id,type,name,class, **id_url**)

- INPUT (**<u>id_input</u>,** id,type,name,class, **id_form**)

3.3.1.7 Report Generator

Since all Web page information and testing results are stored in a database, we can generate reports easily with the help of library " itextpdf. "
The reports are saved as PDF files,
we create a table "Records" on the database to stored all the testing results

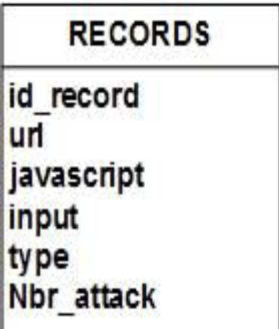

Figure 3.16 Records table

Records table represent all the vulnerability founded on a site web

url : url of the vulnerable page

javascript : the attack vector which used to found the vulnerability

input : input caused the vulnerability

Type : type of the vulnerability (Reflected XSS/stored XSS)

Nbr_attack : number of generated attack codes needed to find a vulnerability

3.3.2 Injection phase:

The goal of injection phase is simulating XSS attacks based on the injection codes generated by the Code Generator and the user data entry points collected in the Crawling Phase.

3.3.2.1 Injection Code Generator :

Our Injection Code Generator is responsible for creating “malicious code” (XSS attack vector) for fault injection. *by using*

- Initial population of *XSS Attack Vector*
- *XSS Attack Vector Grammar*

A. Definition :

Malicious code is the kind of harmful computer code or web script designed to create system vulnerabilities leading to back doors, security breaches, information and data theft, and other potential damages to files and computing systems [36].

B. Population of XSS Attack Vector

At the first time ,We select and save the most popular malicious code of each type OF XSS attack vectors in a list (initial population list) to start our injection and after that if we don't detect any vulnerability, we continue our injection by generating automatically other vectors , and for that we use XSS Attack Vector Grammar.

C. XSS Attack Vector Grammar

We use an attack grammar for generating fuzzed values, to send such fuzzed values to the application , Figure 3.17 illustrates its structure.

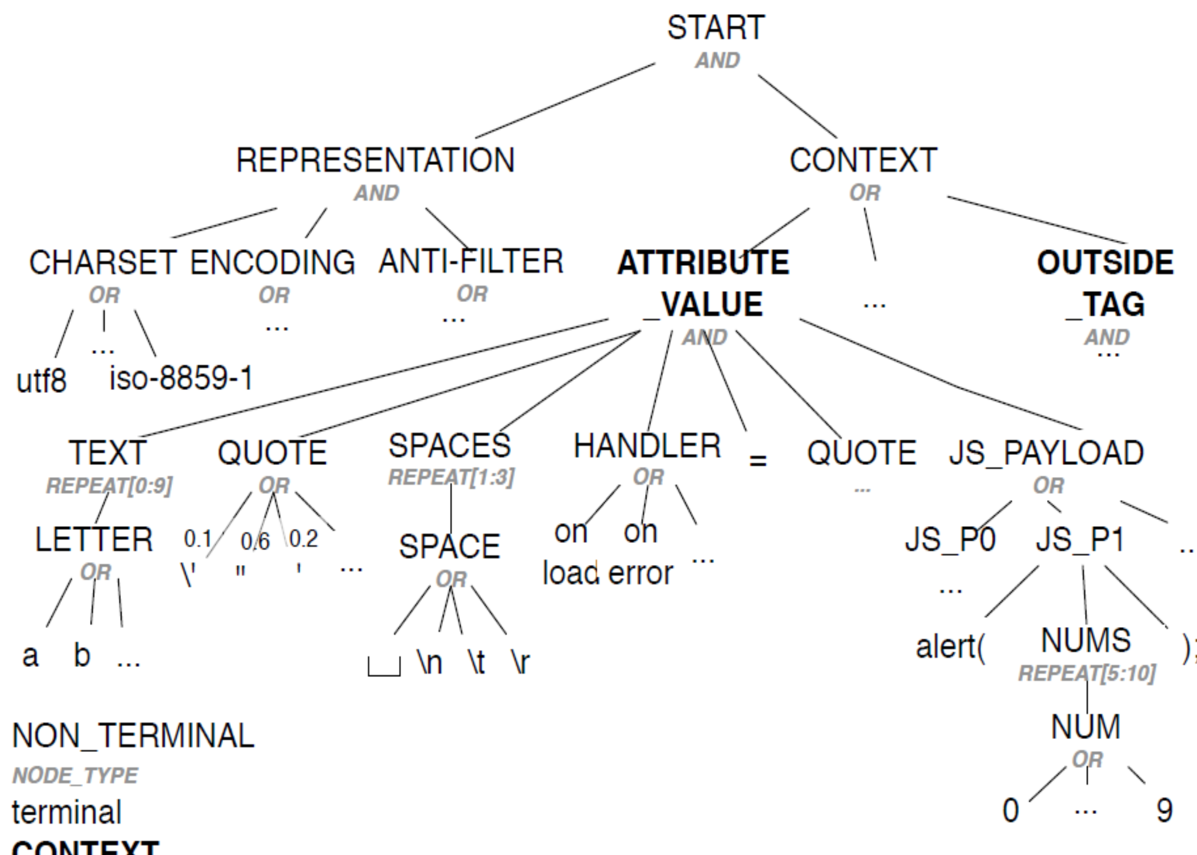

Figure 3.17 XSS grammar tree

3.3.2.2 XSS attack vector generator

To facilitate and speed up the process of injection we Choose to use three types of XSS attack vectors :

	Type	Example
1	HTML tag "script"	<script> alert(1) </script>
2	HTML event attribute	<body onload=alert(1)> </body>
3	URL protocol	 click_me

Table 3.2 type of XSS attack vectors

For each type we create an Attack Vector pattern

A)Attack Vector Grammar HTML tag "script "

<table>
<tr><td>TAG</td><td><script> | <script x> | |<ScRiPt> | <SCript> | <scRIPT> |<SCRIPT>|<script /*%00*/>/*%00*/ | <ScRipT 5-0*3+9/3=>|< <script ~~~>| ...</td></tr>
<tr><td>QUOTE1</td><td>NULL | ; | ‘; | “; | ...</td></tr>
<tr><td>SPACE</td><td>\n | \t | \r | \ r\n | \ a | \b | \ c | NULL ...</td></tr>
<tr><td>QUOTE2</td><td>NULL | ; | ‘; | “; | ...</td></tr>
<tr><td>PAYLOAD</td><td>alert(1) | alert(document.cookie) | +-+-1-+-+ alert(1) |alert(0%0)| prompt(3)| | ...</td></tr>
<tr><td>CLOSINGTAG</td><td></script> </script 1=2| </ScRiPt> |</ SCript> | </scRIPT >|</SCRIPT>| /*%00*/</script /*%00*/ |</ScRipT giveanswerhere=? |</script ~~~>| ..</td></tr>
<tr><td>Conditions</td><td>TAG(i)= CLOSINGTAG(i)</td></tr>
</table>

Table 3.3 Attack Vector Grammar HTML tag "script "

B)Attack Vector Grammar "HTML event attribute"

<table>
<tr><td>TAG</td><td><img | <iframe | <input | < image |<body|<HEAD|< video <object|<map|<audio|<source|<header|<section ...</td></tr>
<tr><td>QUOTE1</td><td>NULL | ; | ‘; | “; | ...</td></tr>
<tr><td>SPACE</td><td>\n | \t | \r | \ r\n | \ a | \b | \ c | NULL ...</td></tr>
<tr><td>EVENT</td><td>NULL| onload=| onclick= | onmouseover= | onerror= | FSCommand= | onAbort= | onActivate= | onAfterPrint= | onDrag= | onInput= | onKeyUp= | onFinish =|...</td></tr>
<tr><td>SPACE2</td><td>\n | \t | \r | \ r\n | \ a | \b | \ c | NULL ...</td></tr>
<tr><td>QUOTE2</td><td>NULL | ; | ‘; | “; | ...</td></tr>
<tr><td>PAYLOAD</td><td>src=java:alert(1)> | SRC=javascript:alert('XSS')> | SRC=JaVaScRiPt:alert('XSS') |alert(1) > | alert(document.cookie)> | prompt(3)>| SRC=http://xss.rocks/xss.js> |......</td></tr>
<tr><td>CLOSINGTAG</td><td></ img > | </ iframe > |</ <input > | </ image >| </ body >|</ HEAD >|</ video >|</ object >|</ map >|</ audio >|</ source >|</ header >|</ section >|...</td></tr>
<tr><td>Conditions</td><td>TAG(i)= CLOSINGTAG(i)</td></tr>
</table>

Table 3.4 Attack Vector Grammar "HTML event attribute"

C) Attack Vector Grammar "URL protocol"

<table>
<tr><td>TAG</td><td><a | <A|...</td></tr>
<tr><td>QUOTE1</td><td>NULL | ; | ‘; | “; | ...</td></tr>
<tr><td>SPACE</td><td>\n | \t | \r | \ r\n | \ a | \b | \ c | NULL ...</td></tr>
<tr><td>EVENT</td><td>href= |HreF=|HREF=......</td></tr>
<tr><td>SPACE2</td><td>\n | \t | \r | \ r\n | \ a | \b | \ c | NULL ...</td></tr>
<tr><td>QUOTE2</td><td>NULL | ; | ‘; | “; | ...</td></tr>
<tr><td>PAYLOAD</td><td>"//google">| "//www.google.com/">| "http://0x42.0x0000066.0x7.0x93/">| "javascript:document.location='http://www.google.com/'">| http://xss.rocks/xss.js >| javascript:alert('XSS')> | JaVaScRiPt:alert('XSS')> |......</td></tr>
<tr><td>TEXT</td><td>XSS|Click|...</td></tr>
<tr><td>CLOSINGTAG</td><td></ a > | </ A >...</td></tr>
<tr><td>Conditions</td><td>TAG(i)= CLOSINGTAG(i)</td></tr>
</table>

Table 3.5 Attack Vector Grammar "URL protocol"

3.3.2.3 The Injection Process

As we know ,Data entry points are stored as URL and form parameters. Next, we will describe how codes are injected through such parameters.

A. *Injection Through URL Parameters*

We can modify the values of url parameters for fault injection by:

1. Extract the name/value pair from the given URL.(as we have seen in **3.3.1.4 A**)
2. Replace the value with an injection code and build a new name/value pair.
3. Recombine the modified name/value pair into the original URL.
4. Navigate to the page specified by the new URL.

Example E1

Given url = http://testphp.vulnweb.com/artists.php?artist=1

name/value **=** artist/1 **==>** artist =<script>alert("XSS !")</script>

New url = http://testphp.vulnweb.com/artists.php? artist =<script>alert("XSS !")</script>

B. *Injection Through Form Parameters*

We can modify the values of a form for fault injection by:

1. Get the form parameters from the database .
2. fill the form fields by the malicious code (as we have see in **3.3.1.2 B**)

3. Submit the form (as we have see in **3.3.1.2 B**).

3.3.2.4 The Difference between reflected and stored injection

As we had explained in the crawling phase, all the parameters of injection in a web page are saved, at this phase we use this information to do the injection.

But there is a difference between reflected and stored XSS.

A) Injection on reflected XSS:

For this injection type, we use the list of searchable form to inject the malicious code

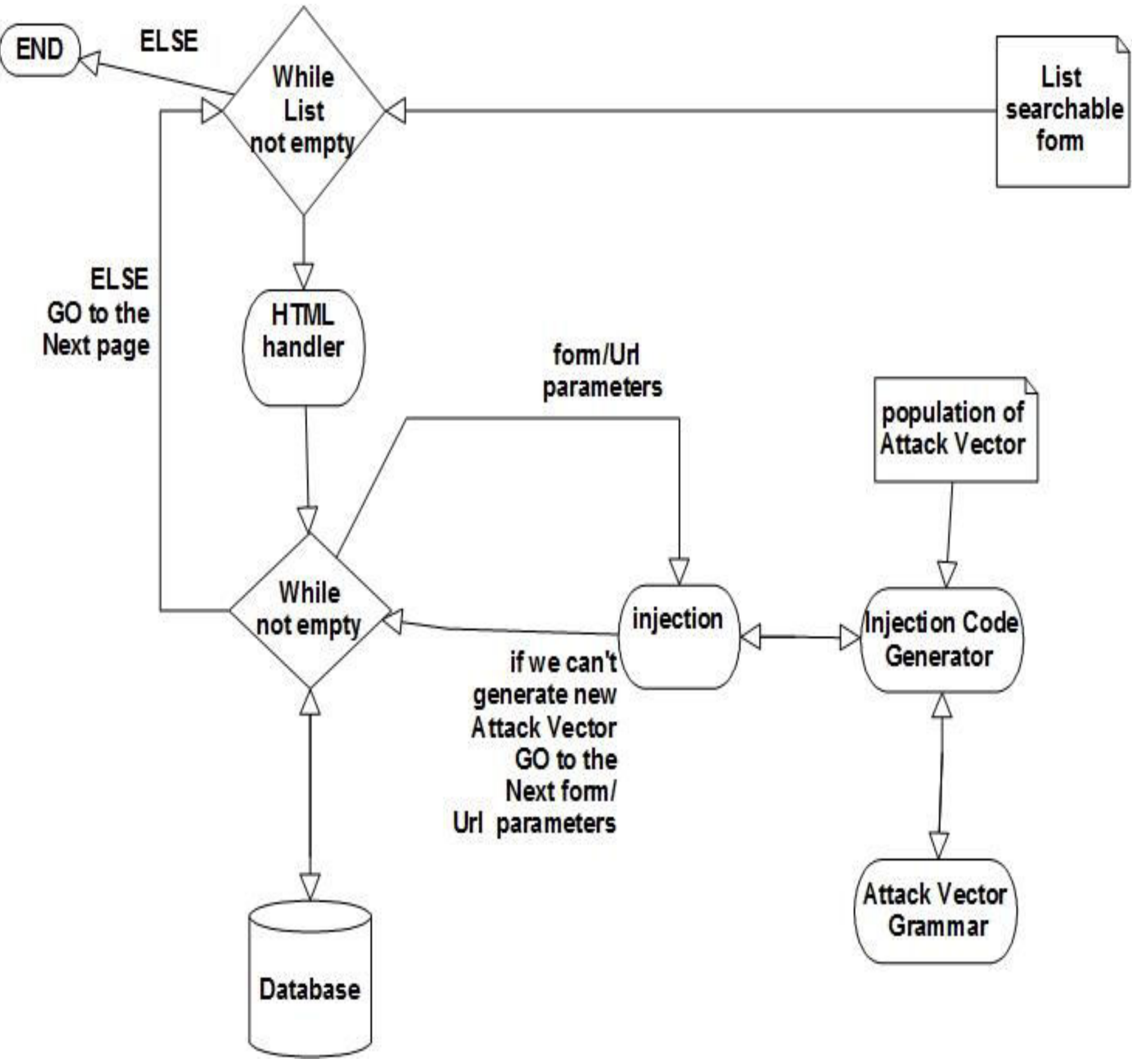

Figure 3.18 Injection of reflected XSS

B) Injection on Stored XSS:

The injection module uses the list of non searchable form to inject the malicious code and save the value of " key of pair " which help us to search our injected code .

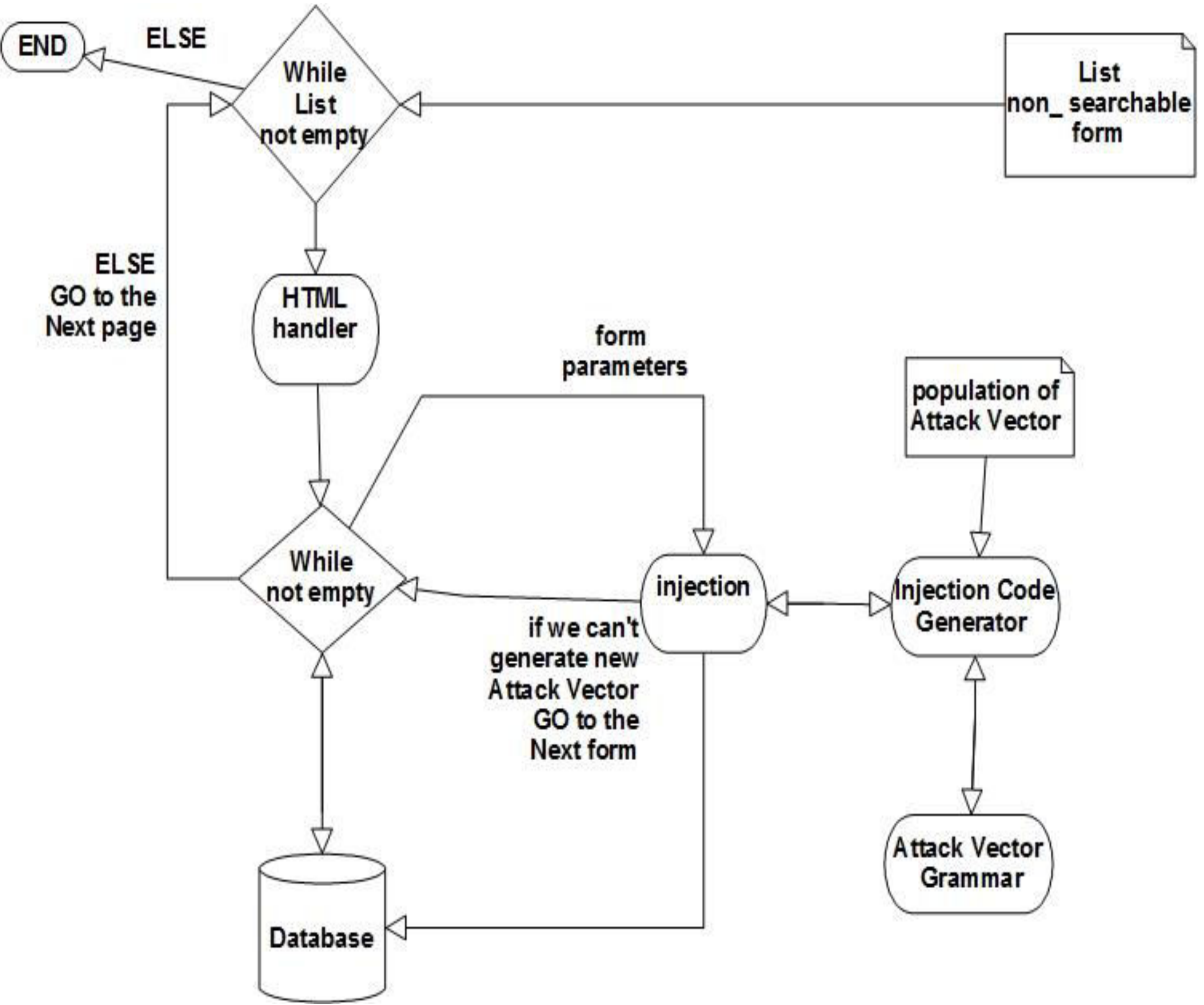

Figure 3.19 Injection of stored XSS

In the case of persistent XSS attacks, an injected code is stored in a database, and it appears in a particular page each time when we request this page. But how can we know which entry point caused the vulnerabilities?

We solve this problem by combining the generated injection codes with the address of the entry point on the injected code (url+form name+ entry point name).

Example : <script>alert("https://www.test.com/+form2+name")</script>

3.3.3 Detection phase:

The Detection phase is the responsible for detecting failures resulting from the Injection phase (vulnerability), a failure occurs when the injected script is executed.

3.3.3.1 Detection of the vulnerability:

After the crawling and the injection process, we analyzing the HTTP response text (or the corresponding DOM tree) instead, and determining, whether the injected script is present and unmodified in the response text. We use two methods:

A) *Events handlers :*

HtmlUnit Offers some ways to detect the vulnerability like the Events handlers (alert, prompt ....). Handlers is the responsible for catching all the events appear in the response pages and saves all messages into a list.

The detection become easy, we just have to analyzing the list and if we found String that we inject (" 123 " in Example E2) that means we found a vulnerability .

But if we don't found the same String , We re-try the injection with another malicious code

Example E2

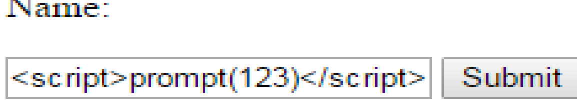

Figure 3.20 Input script example

After Submit we get

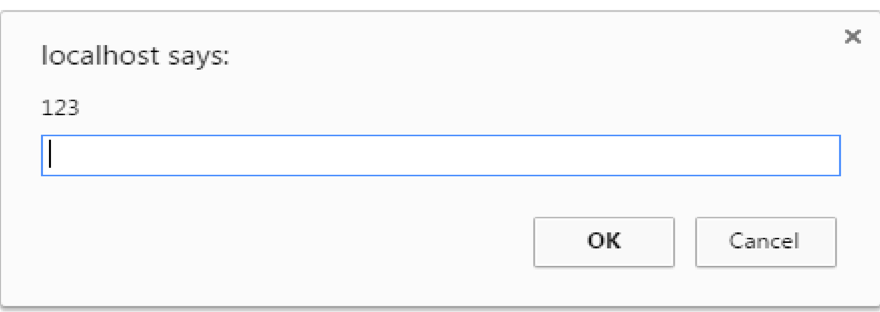

Figure 3.21 Catch prompt(123)

B) Calculate the distance between injected and obtained code :

for other script which we can't use the handlers , we analyze the DOM tree of the response page , and search for the code that we inject in the injection phase ,if we found the same code that means we found a vulnerability

But if we don't found the same code, We re-try the injection with another malicious code

Example E3 : we inject the code <img src=javascript:alert(1)>

Name:

<img src=javascript:alert(1)> Submit

Figure 3.22 inject <img src=javascript:alert(1)>

The DOM tree of the response page :

- Case of XSS vulnerability detection **:**

```
<html>
<head>
    <title>SIT Generator!</title>
</head>
<body>
    <h1>Your name: <img  src=javascript:alert(1)></h1>
</html>
```

Figure 3.23 the reflected Attack Vector of low security page

- Case of no XSS vulnerability detection

```
<html>
<head>
    <title>SIT Generator!</title>
</head>
<body>
    <h1>Your name: <  src=java:alert(1)></h1>
</html>
```

Figure 3.24 the reflected Attack Vector of High security page

3.3.4 Re-injection Malicious code:

We must focus on the Re-injection because it's basic factor to determining the quality of the scanner ,

3.3.4.1 Our approach:

We noticed that when we inject a attack vector (malicious code) ,and after analyzing the DOM tree of the response page ,that we can Splitting the attack vector and the reflected Attack Vector on tokens

Example E4

Name:

<IFRAME SRC="javascript:alert(1);"> </IFRAME> Submit

Figure 3.25 input " <IFRAME SRC="javascript:alert(1);"> </IFRAME>

The DOM tree of the response page has three parts

```
<html>
<head>
    <title>SIT Generator!</title>
</head>
<body>
    <h1>Your name: <IFRAME SRC="javascript:alert(1);">  </IFRAME></h1>
</html>
```

Figure 3.26 DOM tree of the response page

we can use this notice to propose a new method of the detection and Re-injection.

As we said, if we didn't found a vulnerability in a web page, we return to the second phase (injection phase) , and The Injection Code Generator creates automatically a new Attack code by using Attack Vector Grammar.

Our objective is to optimize the number of the XSS Attack codes by using the **mutation operator** on the tokens of the generated code.

- **Mutation method**

If we injected an attack vector and didn't detect any vulnerability, we work on the mutation operator.

We should divide the Attack Vector and the reflected Attack Vector (of the generated response page) on tokens and compare the tokens one by one , if The first token of the Attack Vector is identical to the first token of the reflected Attack Vector , we compare the second tokens of the two vectors , we repeat this comparison until we find a token in the attack vector which is not identical to its Corresponding token into the reflected Attack Vector ,in this case we create a new Attack Vector by mutate only this part by the next token in the grammar .

Example E5

Grammar "HTML event attribute" generate first attack vector :<img src=java:alert(1)> </img>

	First token	Second token	Third token
first Attack Vector is :	<img	src=java:alert(1)>	</img>
first reflected Attack Vector is:	<	src=java:alert(1)>	</ >

<img NOT identical to **<** we use Attack Vector Grammar "HTML event attribute" to generate second Attack Vector ,the method mutate **<img** by **<iframe**

	First token	Second token	Third token
second Attack Vector is :	<iframe	src=java:alert(1)>	</ iframe>
second reflected Attack Vector is:	<	src=java:alert(1)>	</ >

< iframe NOT identical to **<** we use Attack Vector Grammar "HTML event attribute" to generate second Attack Vector ,the method change **<iframe** by **<input**

	First token	Second token	Third token
Third Attack Vector is :	<input	src=java:alert(1)>	</ input>
Third reflected Attack Vector is	<input	src=java:alert(1)>	</ input>

We have to stop now ,because the attack vector is identical to the reflected Attack Vector ,that means we find a vulnerability by using just **Three** attack vector .

3.3.4.2 The Difference between reflected and stored detection

There is a difference between the Detection on reflected and stored XSS

A) Reflected XSS :

After the Injection on searchable page , the Detection module analyzes the DOM tree of the response page

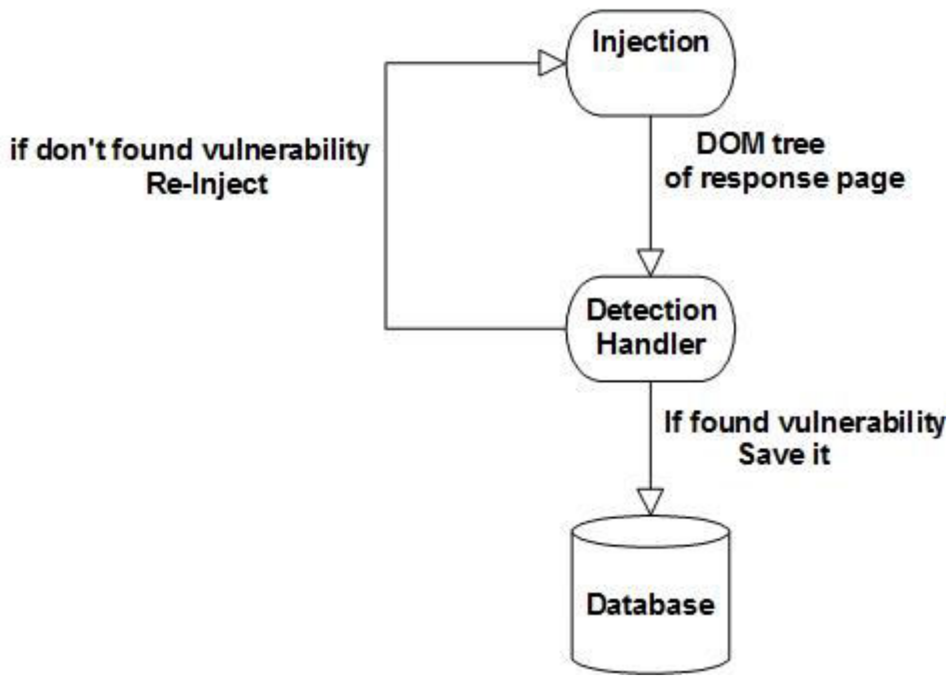

Figure 3.27 Detection of XSS vulnerability "Reflected XSS"

B) Stored XSS :

After the Injection on non_searchable form, the Detection module go to its corresponding searchable form and set the value of the" key of pair " and submit the form ,then analyzing the DOM tree of response page

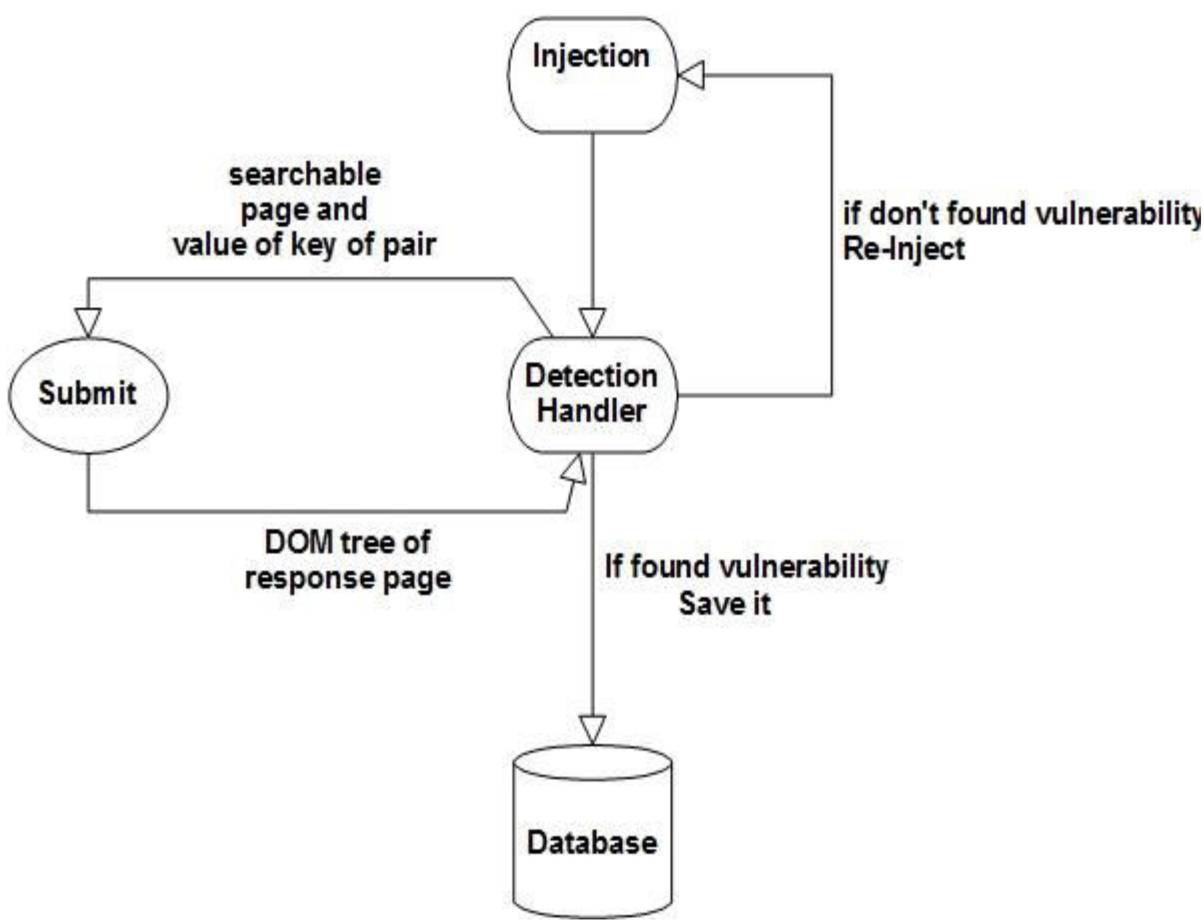

Figure 3.28 Detection of XSS vulnerability "Stored XSS"

3.4 Conclusion

In order to detect XSS vulnerabilities, we proposed a method using optimized attack vector. This method generates XSS attack vectors automatically, and uses a mutation operator to reduce the size of attack vector. After that, we detect XSS vulnerabilities with XSS attack vectors dynamically .

In this chapter we presented the architecture of our scanner and the detailed implementation of each phase (crawling, injection, detection).

In the next chapter we will give our implementation of the proposed scanner and its experimentation results.

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTATIONS

4.1 Introduction

This chapter describes the experimental study of different proposed techniques in our implemented scanner, which are described in the previous chapter. In which our goal is implementing an XSS vulnerability scanner which detect vulnerabilities in web pages with an optimized XSS attack vector.

We conducted a number of experiments to study and measure the performance of our scanner. A series of attacks has been specified to achieve this objective. Three of the most popular scanners are used to conduct a comparative study with our approach. The OWASP ZAP scanner is an academic scanner , Acunetix Web Vulnerability (IBM) representing commercial scanners, and Vega Vulnerability Scanner.

4.2 Language and tools used to develop

4.2.1 NetBeans :

Most developers recognize the NetBeans IDE as the original free Java IDE. It is that, and much more! The NetBeans IDE provides support for several languages (PHP, JavaFX, C/C++, JavaScript, etc.) and frameworks.

NetBeans is an open-source project dedicated to providing rock solid software development products (the NetBeans IDE and the NetBeans Platform) that address the needs of developers, users and the businesses who rely on NetBeans as a basis for their products; particularly, to enable them to develop these products quickly, efficiently and easily by leveraging the strengths of the Java platform and other relevant industry standards. [37]

4.2.2 jsoup: Java HTML Parser

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

- scrape and parse HTML from a URL, file, or string
- find and extract data, using DOM traversal or CSS selectors
- manipulate the HTML elements, attributes, and text
- clean user-submitted content against a safe white-list, to prevent XSS attacks

- output tidy HTML

jsoup is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; jsoup will create a sensible parse tree [38] .

4.2.3 HtmlUnit

HtmlUnit is a "GUI-Less browser for Java programs". It models HTML documents and provides an API that allows you to invoke pages, fill out forms, click links, etc... just like you do in your "normal" browser.

It has fairly good JavaScript support (which is constantly improving) and is able to work even with quite complex AJAX libraries, simulating Chrome, Firefox or Internet Explorer depending on the configuration used.

It is typically used for testing purposes or to retrieve information from web sites.

HtmlUnit is not a generic unit testing framework. It is specifically a way to simulate a browser for testing purposes and is intended to be used within another testing framework such as JUnit or TestNG. Refer to the document "Getting Started with HtmlUnit" for an introduction.

HtmlUnit is used as the underlying "browser" by different Open Source tools like Canoo WebTest, JWebUnit, WebDriver, JSFUnit, WETATOR, Celerity, Spring MVC Test HtmlUnit, ...

HtmlUnit was originally written by Mike Bowler of Gargoyle Software and is released under the Apache 2 license. Since then, it has received many contributions from other developers, and would not be where it is today without their assistance [39].

4.2.4 MySQL :

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

Oracle drives MySQL innovation, delivering new capabilities to power next generation web, cloud, mobile and embedded applications [40] .

4.3 The Physical Schema

Field	Type	sizes
id_website	INT	100
Website	VARCHAR	1000

Table 4.1 Website.sql

Field	Type	sizes
id_url	INT	100
url	VARCHAR	1000
id_website	INT	100

Table 4.2 URL.sql

Field	Type	sizes
id_form	INT	100
id	VARCHAR	100
type	VARCHAR	100
name	VARCHAR	100
class	VARCHAR	100
id_url	INT	100

Table 4.3 FORM.sql

Field	Type	sizes
id_input	INT	100
id	VARCHAR	100
type	VARCHAR	100
name	VARCHAR	100
class	VARCHAR	100
id_form	INT	100

Table 4.4 INPUT.sql

Field	Type	sizes
id_record	INT	100
url	VARCHAR	1000
Javascript	VARCHAR	100
Input	VARCHAR	100
Type	VARCHAR	50
Nbr_attack	INT	100

Table 4.5 RECORD.sql

4.4 Optimized XSS vulnerability scanner interface

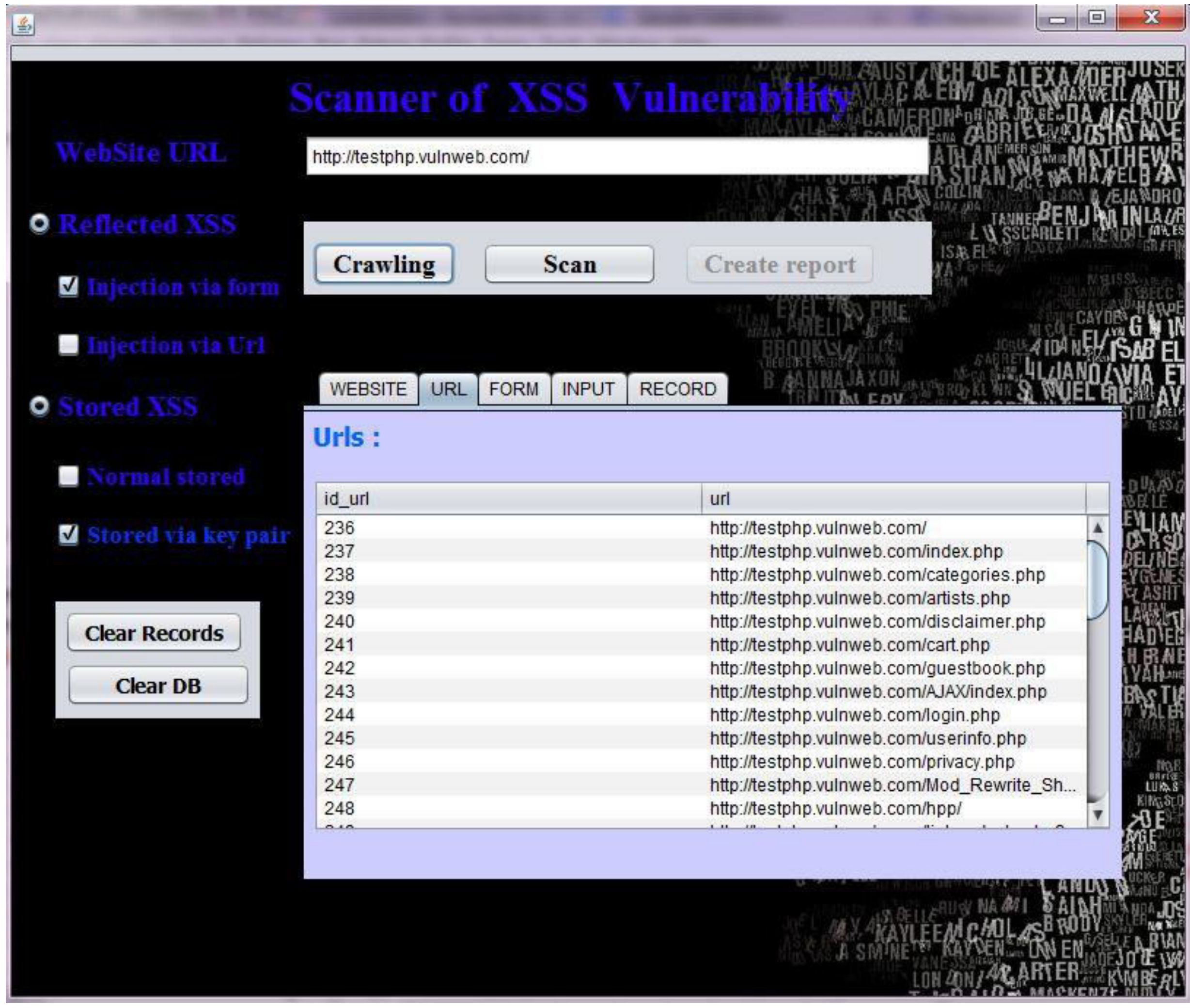

Figure 4.1 Our Scanner of XSS Vulnerability

- Website URL: in this text field the user enter url of website to scan for XSS vulnerabilities.
- Crawling Button : start crawling the site web and save all the information in database .
- Create report : Create report of the vulnerabilities detected on PDF Format .
- Tabbed Pane : Display all the information (sitewebs , urls , forms , inputs, records) .
- Radio Buttons : to choose type of XSS Vulnerability(Reflected or Stored).
- Checkbox Button: to choose type of injection :
  - Injection via Form : injected all the searchable page by using their forms to detect the Reflected XSS Vulnerability .
  - Injection via Url : injected the searchable page by using their Url parameter to detect the Reflected XSS Vulnerability .
  - Normal Stored : injected all the non_searchable page by using their forms to detect the Stored XSS Vulnerability .
  - Stored Key pair : injected all the non_searchable page and go to the correspondent searchable page to detect the Stored XSS Vulnerability .
- Scan Button : start the injection for searching of XSS Vulnerabilities.
- Clear DB Button : Delete all the information from the database .
- Clear Records Button : Delete the list of vulnerabilities detected.

4.5 Experimentations

The experiment will have more credibility if it involves scanners adopted by the scientific and professional community as a kind of reference to prove the power of our approach, for that we used:

- The OWASP ZAP : scanner academic
- Acunetix Web Vulnerability : commercial scanners.
- Vega Vulnerability Scanner

4.5.1 Web application used in the test :

To demonstrate the effectiveness of our scanner on vulnerabilities detection and time of scan, we test it on three different web applications:

4.5.1.1 Damn Vulnerable Web App (DVWA):

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment [41].

Figure 4.2 Damn Vulnerable Web App (DVWA)

4.5.1.2 testphp.vulnweb.com:

Testphp is demonstration site for Acunetix Web Vulnerability Scanner

Figure 4.3 testphp.vulnweb.com

4.5.1.3 template:

Template is our developed vulnerable site to test our scanner.

Figure 4.4 template

4.6 Results discussion

4.6.1 First evaluation scenario

The DVWA application has three level of security , we try to find vulnerabilities for each level ,the obtained results are shown in the table below :

Security level	Vulnerable page	input	Attack Vector	Type	Number of generation
Low	http://localhost/DVWA-1.9/vulnerabilities/xss_r/	name	<script> alert(1) </script>	Reflected	1
Medium	http://localhost/DVWA-1.9/vulnerabilities/xss_r/	name	<SCripT> alert(1) </SCripT>	Reflected	2
High	http://localhost/DVWA-1.9/vulnerabilities/xss_r/	name		Reflected	5
Impossible	NULL	NULL	NULL	NULL	15

Table 4.6 Discovered Vulnerability (Reflected) of DVWA

Security	URL	input	Attack Vector	Type	Number of generator
Low	http://localhost/DVWA-1.9/vulnerabilities/xss_s/	txtName	<script> alert(1) </script>	Stored	1
	http://localhost/DVWA-1.9/vulnerabilities/xss_s/	mtxMessage	<script> alert(1) </script>	Stored	1
Medium	http://localhost/DVWA-1.9/vulnerabilities/xss_s/	txtName	<SCripT> alert(1) </SCripT>	Stored	2
	http://localhost/DVWA-1.9/vulnerabilities/xss_s/	mtxMessage	<SCripT> alert(1) </SCripT>	Stored	2
High	http://localhost/DVWA-1.9/vulnerabilities/xss_s/	txtName		Stored	5
	http://localhost/DVWA-1.9/vulnerabilities/xss_s/	mtxMessage		Stored	5
Impossible	NULL	NULL	NULL	NULL	15

Table 4.7 Discovered Vulnerability(Stored) of DVWA

Discussion:

The obtained results confirm that our scanner can detect vulnerabilities on three levels of security with a reduced number of generated XSS codes .

4.6.2 Second evaluation scenario

In this scenario we compare our scanner results with three scanners, using two web applications: TestPHP and template.

The metrics used to evaluate the tools are: number of detected XSS vulnerabilities and average scan time.

4.6.2.1 testphp.vulnweb.com:

The obtained results are shown in the table below:

Tool	***XSS vulnerabilities found***	***Average Scan Time***
The OWASP ZAP	12	1 minute and 54 seconds
Acunetix Web Vulnerability	27	55 seconds
Vega Vulnerability Scanner	14	1 minute and 27 seconds
Our scanner	31	50 seconds

Table 4.8 Scanners evaluation metrics on TestPHP

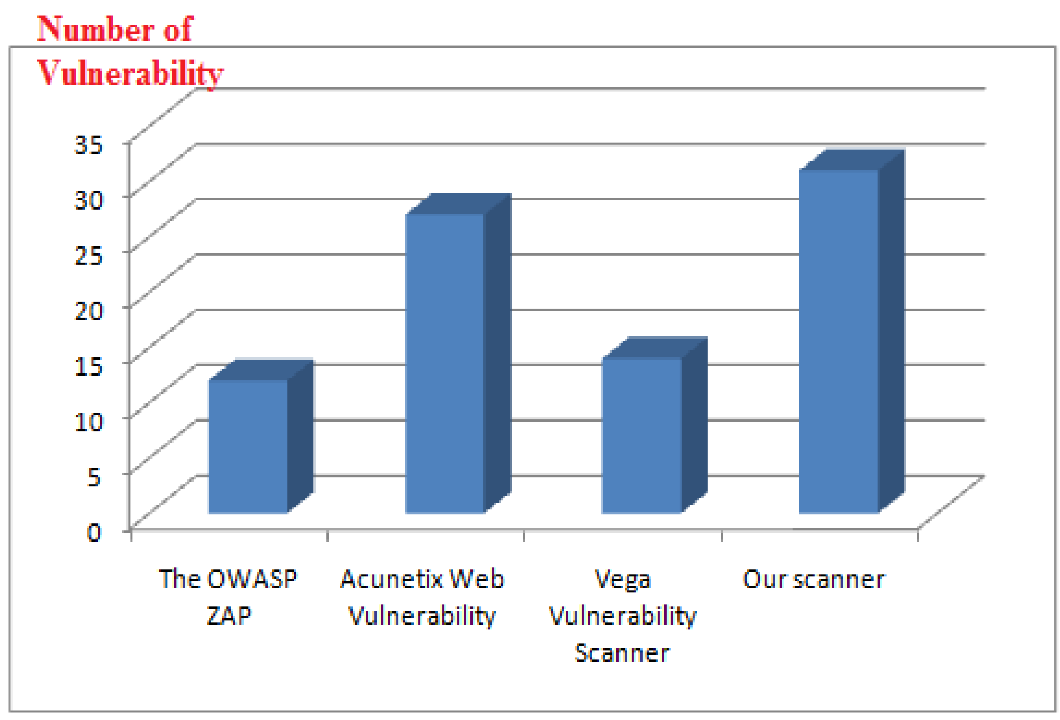

Figure 4.5 XSS vulnerabilities detected

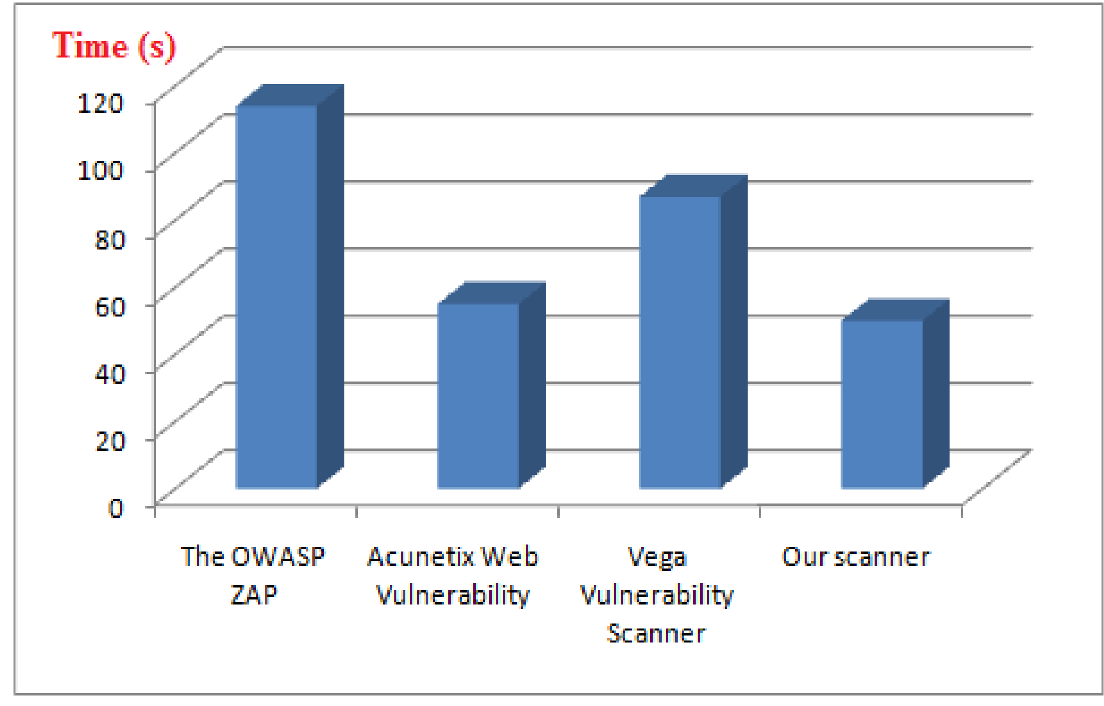

Figure 4.6 Average Scan Time(s)

4.6.2.2 template:

We test the four scanners on our developed web site; the obtained results are shown in the table below:

Tool	*XSS vulnerabilities found*	*Average Scan Time*
The OWASP ZAP	3	2 minutes and 22 seconds
Acunetix Web Vulnerability	2	24 seconds
Vega Vulnerability Scanner	2	1 minute and 40 seconds
Our scanner	4	11 seconds

Table 4.9 Scanners evaluation metrics on template

4.6.3 Discussion:

From the Table 4.8 and table 4.9 ,

- The first point analyzed on this scenario is that most of the scanners, got an execution time between 1 and 2 minutes, except Acunetix tool that got an execution time of 55 seconds in the first site and 24 seconds in the second site , in all cases our scanner got the best execution time , between 50 (in the first site)and 11seconds(in the second site).
- The second point ,Regarding XSS vulnerabilities,

in the first site(testphp) ,Acunetix and our scanner detected the largest number of vulnerabilities, but our scanner found more vulnerabilities,

in the second site(template) also our scanner found more vulnerabilities than the others .

this prove that our optimization approach is more efficient.

4.7 Conclusion

In this chapter we described the implementation and the experimentation of the proposed XSS scanner approach.

The experimental study clearly shows the performance of our scanner in detection of XSS vulnerabilities, especially since it was compared with three of the best known scanners OWASP ,IBM Acunetix and Vega.

The experimental study proves that our approach minimizes the time of the scan and maximizes the number of detected vulnerabilities, this prove that our optimization approach is more efficient.

General conclusion

Current-day scanner of XSS vulnerability are used to detect most of the vulnerabilities (Reflected or Stored XSS). However, none of them are complete or accurate enough to guarantee an absolute level of security on web application.

In this project, we developed our scanner of XSS vulnerability, which have three basic phases :

- Crawling phase : for collect and save the information from the Web application.
- Injection phase : for simulating XSS attacks based on the injection codes generated by the Code Generator and the user data entry points collected in the Crawling Phase
- Detection phase: for detecting failures resulting from the Injection phase (detection of vulnerabilities)

Time and number of vulnerabilities detected are the most factors to evaluate the Success of scanners, for this reason ,We proposed a method to optimize the detection of stored XSS vulnerabilities, and other method for generating and optimizing XSS vector attack.

In the experimentation phase we choose three of the most popular scanners used to conduct a comparative study with our approach. The OWASP ZAP scanner , Acunetix Web Vulnerability (IBM), and Vega Vulnerability Scanner .
we choose three websites for testing: Damn Vulnerable Web App (DVWA), testphp.vulnweb.com ,and template site.

The results prove the effectiveness of our scanner in identification of vulnerabilities and reduce the time of scan.

Perspective: detection of DOM vulnerabilities

Bibliography

[1] *Nations, Daniel.* "Web Applications". *About.com. Retrieved 20 January 2014.*

[2] OWASP. A Guide to Building Secure Web Applications. The Open Web Application Security Project, 2005.

[3] Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim (June 1999). Hypertext Transfer Protocol -- HTTP/1.1. IETF. RFC 2616.

[4] Berners-Lee, Tim; Fielding, Roy T.; Nielsen, Henrik Frystyk. "Method Definitions".Hypertext Transfer Protocol -- HTTP/1.0. IETF. pp. 30-32. sec. 8. RFC 1945.

[5] Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim (June 1999). *Hypertext Transfer Protocol -- HTTP/1.1*. IETF. RFC 2616 "Method Definitions". pp. 51-57. sec. 9.

[6] Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim (June 1999). *Hypertext Transfer Protocol -- HTTP/1.1*. IETF. RFC 2616."POST". p. 54. sec. 9.5.

[7]. "http tutorial",[Online]. Available: http://www.tutorialspoint.com/http/http_pdf_version.htm Visited : 07/04/2016.

[8] Programming notes , HTTP (HyperText Transfer Protocol), https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html ., Visited : 07/04/2016 .

[9] XSS Attacks: Cross Site Scripting Exploits and Defense 1st Edition by Seth Fogie (Author), Jeremiah Grossman (Author), Robert Hansen (Author), Anton Rager (Author), Petko D. Petkov (Author) .

[10] Marcel Dekker. Security of the Internet, 1997.

[11] Chris Joscelyne. Information Management, 2005.

[12] Claudia Eckert. IT-Sicherheit. Oldenbourg–Verlag, second edition, 2003.

[13] https://www.owasp.org/index.php/Top_10_2013-Top_10. , Visited : 15/04/2016 .

[14] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) , Visited : 12/04/2016 .

[15] http://www.acunetix.com/blog/articles/preventing-xss-attacks , Visited : 14/04/2016 .

[16] https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_ Cheat_Sheet ,Visited : 14/04/2016 .

[17] Gunter Ollmann. HTML Code Injection and Cross-Site Scripting. http: //www.technicalinfo.net/papers/CSS.html, Visited : 10/042016 .

[18] CERT Coordination Center. Understanding Malicious Content Mitigation for Web Developers. http://www.cert.org/tech tips/malicious code ,Visited : 10/042016 .
[19] Web application vulnerability scanner US 8365290 B2 ,Frederick Young ,18 11 2010.
[20] https://msdn.microsoft.com/en-us/library/cc751383.aspx ,Visited 20/04/2016.
[21] Edsger W. Dijkstra. Structured Programming. Software Engineering Techniques,1990.
[22] Glenford J. Myers. The Art of Software Testing. Wiley, 1979.
[23] Glenford J. Myers. Software Reliability. Wiley, 1976.
[24] Bill Hetzel. The Complete Guide to Software Testing. QED Information Sciences,second edition, 1988.
[25] AD Brucker, T Deuster - US Patent 8,881,293, 2014.
[26] https://cmsreport.com/articles/web-application-security-testing-sast-dast-or-iast--13728 ,Visited : 21/04/2016
[27] Bill Hetzel.. QED Information Sciences, second edition, 1988.
[28] Black Box Security Testing Tools C.C. Michael, Ken van Wyk, and Will Radosevich July 31, 2013 .
[29] https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools 2016 , Visited: 23/04/2016
[30] http://wapiti.sourceforge.net/ , Visited: 23/04/2016
[31] http://w3af.org/howtos/understanding-the-basics , Visited: 23/04/2016
[32] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project ,Visited: 23/04/2016 .
[33] http://www.acunetix.com/company/ ,Visited : 23/04/2016.
[34] https://subgraph.com/vega/ Visited : 23/04/2016 .
[35] https://jsoup.org 2016 , Visited : 06/05/2016.
[36] https://usa.kaspersky.com/internet-security-center/definitions/malicious-code#. VynIWYQrLIU ,Visited : 07/05/2016.
[37] https://netbeans.org/about/ ,Visited : 10/05/2016
[38] https://jsoup.org/ ,Visited : 10/05/2016
[39] http://htmlunit.sourceforge.net/ ,Visited : 10/05/2016
[40] http://www.mysql.com/about/ ,Visited : 10/05/2016
[41] http://www.dvwa.co.uk/ ,Visited : 27/04/2016.
[42] H. Chen and M. V. Gundy, “Using randomization to enforce information flow tracking

and thwart cross site scripting attacks," In Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS), (2009).

[43] 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).

[44]: 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.

[45]: 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.

[46]: 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.

[47] G. Wassermann, Z. Su. Static Detection of Cross-Site Scripting Vulnerabilities. ICSE'08, May 10–18, 2008, Leipzig, Germany.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[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).

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

[55] Xiaobing Guo, Shuyuan Jin, and Yaxing Zhang Institute of Computing Technology, Chinese Academy of Sciences Beijing, China XSS Vulnerability Detection Using Optimized Attack Vector Repertory 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery .

[56] Fabien Duchene **,**Sanjay Rawat_IIIT Hyderabad, India**,** Jean-Luc Richier, Roland Groz LIG Lab Grenoble F-38402, France KameleonFuzz:Evolutionary Fuzzing for Black-Box XSS Detection March 2014

[57] Fabien Duchene ,Sanjay Rawat_IIIT Hyderabad, India, Jean-Luc Richier, Roland Groz LIG Lab Grenoble F-38402, France KameleonFuzz:Evolutionary Fuzzing for Black-Box XSS Detection March 2014

[58] Fabien Duchene, Roland Groz, Sanjay Rawat, Jean-Luc Richier. XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing. SECTEST 2012 - 3rd International Workshop on Security Testing (a_liated with ICST), Apr 2012, Montreal, Canada. IEEE Computer Society, pp.815-817, 2012, <10.1109/ICST.2012.181>. <hal-00857294>

[59] 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.

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

[61]: 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 .

الملخص

أصبحت تطبيقات الويب أكثر شعبية مع تقدم التكنولوجيا ، ومع ذلك، أمن الواب أصبح واحد من القضايا الأمنية الأكثر شيوعا.

تركز هذا المذكرة على ثغرة XSS الموجودة عادة في معظم تطبيقات الويب و يمكن أن تخلق مشاكل أمنية خطيرة.

في عملنا ، نقترح نهج للكشف عن الثغرات باستعمال الصندوق الأسود و جدول الهجوم الأمثل ،هذه الطريقة تولد جدول الهجوم تلقائيا ، و تحسنه باستخدام نموذج التبديل ،لتكشف عن ثغرات XSS الموجودة في تطبيقات الويب ديناميكيا .

- **الكلمات المفتاحية :** كشف ثغرات XSS ، تحسين جدول الهجوم ,ماسح الصندوق الاسود ، ، ماسح ثغرة XSS.

Summary

The Web applications are becoming more popular with the advancement of technology. However, the web security is becoming one of the most common security issues.

This report focuses on the XSS vulnerabilities which commonly present in most Web applications and can create serious security problems.

In our work, we propose a black box detection approach using optimal attack vector. This method generates an attack vector automatically, optimizes the attack vector repertory using a mutation operator model, and detects XSS vulnerabilities in web applications dynamically.

- **Keywords**: XSS vulnerability detection, attack vector optimization, black box scanner, XSS vulnerability scanner.

Résumé

L’avancement de la technologie web prépare la voie à la popularité des applications Web, Cependant, la sécurité de Web est devenu l'un des problèmes de sécurité les plus courants.

Ce mémoire porte sur les vulnérabilités XSS qui sont généralement présentent dans la plupart des applications Web et peuvent créer des problèmes de sécurité graves.

Dans notre travail, nous proposons une approche de détection de boîte noire utilisant le vecteur d'attaque optimale. Cette méthode génère automatiquement un vecteur d'attaque, puis l’optimise à l'aide d'un modèle basé sur la mutation, puis détecte dynamiquement les vulnérabilités XSS dans les applications web.

- **Mots-clés :** détection de vulnérabilité XSS, optimisation du vecteur d'attaque, scanner boîte noire, scanner de vulnérabilité XSS.