Web Application Vulnerability Scanner

By

Phenilkumar Buch

12MCEI36

CSE department

Institute Of Technology, Nirma University

Ahmedabad

May, 2014

Web Application Vulnerability Scanner

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

M.Tech in Information and Network Security

By

Phenilkumar Buch

12MCEI36

Guided By

Prof. Sharada Valiveti

CSE Department

Institue Of Technology, Nirma University

Ahmedabad

May, 2014

Declaration

I hereby declare that the Major Project entitled 'Web Application Vulnerability Scanner' comprises my original work towards the partial fulfillment of requirements for the degree of Master of Technology in Information and Network Security at Nirma University and has not been submitted elsewhere for a degree. Due acknowledgement has been made in the text to all other material used.

Phenilkumar Buch

Acknowledgements

I express my gratitude to my valuable guide Prof. Sharada Valiveti for her valuable guidance, motivation and encouragement during the completion of my project work. I would also like to thank her for the freedom she has given me in the way I carry out my work. I would like to thank Mr. Rahul Tyagi, Ethical Hacker, Techdefence Ltd., who through his wonderful practical insights inspired me to work in the area of web application security for my major project. And lastly I would like to thank my family and friends for their continuous support that kept my sprit up during the endeavor.

Certificate

This is to certify that the Major Project entitled 'Web Application Vulnerability Scanner' submitted by Phenilkumar Buch (12MCEI36), towards the partial fulfillment of the requirements for the degree of Master of Technology in Information and Network Security of Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this Project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Sharada Valiveti
Guide, INS PG Coordinator and Associate Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Ketan Kotecha
Director,
Institute of Technology,
Nirma University,
Ahmedabad

Abstract

Web applications have become very common and the role of web application security has garnered much attention as well. The number of sensitive online resources is increasing day by day, and so is the need to protect these resources. Many web applications are developed without taking care of security aspects and hence they are prone to attacks. Although it is easy to understand and avoid many web vulnerabilities, security awareness is what many web developers lack. Consequently, many web applications exist on the Internet that are vulnerable. For an organisation to identify if their applications are susceptible to attacks, it must perform regularly scheduled penetration testing, vulnerability assessment and updation of their applications.

The major security vulnerabilities that exist in today's web applications include SQL injection, Cross Site Scripting and a few others. These vulnerabilities can be detected automatically. Various methods are used for detection of these vulnerabilities and each of these methods have their own limitations. Black box web application vulnerability scanners are tools that are utilized to determine vulnerabilities in web applications. The vulnerability scanners interact with a web application in a way that is similar to the way in which regular users do. However, various sections of web applications must be accessed and tested by these tools, and as big a part of the application needs to be crawled as possible. An assessment of various black box web vulnerability scanners along with the scanner developed for current dissertation is also presented here. The evaluation process is composed of testing a scanning technique for different categories of web application vulnerabilities. The tests are performed on various realistic web applications with known vulnerabilities. The results of the evaluation show that the task of crawling is equally important to the overall effectiveness of a scanner as the various vulnerability specific detection algorithms. Many vulnerabilities are not detected by the scanners, and thus further research is warranted to better the automated detection of web application vulnerabilities.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Definition

These days a large number of web applications are custom-built. They are implemented using different web technologies. The highly contrasting nature of different web applications with their different implementation languages, browser compatibility, encoding standards and scripting languages used, makes it very difficult for developers to properly secure their applications and stay updated with new threats and zero day attacks.

There are two ways to test web applications for the presence of vulnerabilities [1]:

- White-box testing - the source code of the application is analyzed in order to detect vulnerable and buggy sections of the program. This process is more often than not, merged into the development life cycle itself.

- Black-box testing - the source code of the web application is not known and hence not directly tested. Instead, contorted input test cases are created and fed to the application entry points. After that, the results obtained as responses from the application are analyzed. Errors or vulnerabilities may also be found from this method of testing.

The Problem is to identify and analyze current web application vulnerability scanning methods that perform black-box testing and to find the optimum methods that can scan web applications comprehensively and quickly for specific vulnerabilities. A seconday goal for the dissertation is to develop an application which communicates with any user-specified web application through the web front-end and one which automatically identifies potential security vulnerabilities in the web application and its architectural weaknesses. The project is not just about vulnerability scanning techniques, it is also about studying the vulnerabilities themselves.

1.2 Solution Approach

Web Application Vulnerability Scanner (WAVS) is an automated web application security testing tool developed for this dissertation that performs pentesting on web applications by checking for vulnerabilities like SQL Injections, Cross site scripting and other vulnerabilities. This tool is capable of scanning any web application that is accessible via a web browser.

The basic working of any black box vulnerability scanner is as follows: Firstly, the crawler spiders through the entire web application whose url is provided by the user of the scanner. It then maps out the website structure and gathers detailed information about every form, page and file. After the crawling process is complete, each page is analyzed for entry points where the scanner can input data and interact with the web application. The scanner launches a series of attacks through each of the components that a user can interact with, often using several payloads for a single entry point. The scanner essentially emulates a human hacker looking to hack the web application in any possible way. This is the Automated Scan Stage.

1.3 Project Scope

The working of WAVS is similar to other scanners that can be found in the market like Acunetix Web Vulnerability Scanner [2][3] and yet is different in its implementation. There are several vulnerability scanners that are available commercially. Web application vulnerabilities have different variants and there is no single method for detecting all variants of a given vulnerability. Also, no vulnerability scanner is perfect. Therefore, different scanners need to be used together in order to better secure a web application. This way the weaknesses of one scanner can be overcome with the help of another.

1.4 Report Outline

In the next section, we take a brief look at the common security vulnerabilities that exist in web applications that are on the internet today. And in the section after that we enumerate the requirements of a standard black box vulnerability scanner. After that we look at how web application vulnerabilities can be detected in an automated fashion. In the results section, we see how automated vulnerabilitiy scanners can be tested. We also analyze the results and point out the factors which might reduce the effectiveness of automated scanners. In conclusion, we indicate the future scope of research in the field of automated vulnerability scanning.

Chapter 2

Literature Survey

Firgure 2.1 shows OWASP top 10 web application vulnerabilities 2013 [4].

2.1 SQL Injection

SQL injection attacks [5] are founded on inserting payloads into database queries such that some unauthorized/unintended action on the database is performed. There are many types of SQL injection attacks possible. Most SQL queries are dynamically generated based on the input from an end user. In order to defend against SQL injection vulnerabilities, it needs to be considered that a user may input malicious data and therefore the data needs to be properly cleaned before it is used to generate queries. User input needs to be properly validated and the input data should be encoded with the web application environment.

There are several ways of injecting malicious SQL statements. Normally, an attacker tampers with the input data such that the resulting query performs an unintended action like revealing sensitive data. The Attacker sends simple text data that exploits the syntax of the targeted database technology. In another case if the web application uses the contents of cookies to build SQL queries, an attacker could easily embed an

OWASP Top 10 – 2010 (Previous)	OWASP Top 10 – 2013 (New)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Merged with A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Broadened into →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<buried in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards
A9 – Insufficient Transport Layer Protection	Merged with 2010-A7 into new 2013-A6

Figure 2.1: OWASP top ten web vulnerabilities 2013

attack in a spoofed cookie. Attackers can also forge the values of server variables that are present in HTTP and network headers. If these variables are logged in to a database without any checks and modification, an SQL injection vulnerability is created. In second-order SQL injections, attackers plant data into a database that does nothing when inserted into the database but which is actually used to indirectly trigger an SQL Injection attack when that input is used for another query later. Second-order injections are highly complex attacks and are extremely difficult for automated scanners to detect.

The attacker probes a web application to discover which parameters and user-input fields are vulnerable to injection. The attacker discovers the type and version of database that the web application is using. Different types of databases have different syntax for queries and respond differently to different attacks. If the type of database used by the web application and its version is known, the attacker can formulate database specific attacks. In order to extract data from a database, the attacker needs to know about database schema information like table names, column names, and data types of those columns. Attackers extract, add and modify data and

cause denial of service by deleting tables or locking them. Some attacks are employed to avoid detection by intrusion detection systems. Bypassing authentication attacks allow the attacker to bypass database and application authentication mechanisms. These attacks allow the attacker to assume the rights and privileges associated with another application user.

SQL Injection flaws are very prevalent, particularly in old and re-used code. Preventing injection attacks like SQLI requires keeping untrusted user-supplied data separate from queries used in the web application. Many applications require special characters in their input and hence White-list input validation is difficult to achieve.

2.2 Cross Site Scripting

Cross Site Scripting (XSS) [6] refers to a type of attack in which a harmful script is injected into a web application by the attacker. XSS flaws occur when a web application includes user supplied data in a page sent to the browser as dynamic web pages without properly validating or escaping that content. When a victim of the attack views the vulnerable web page in which malicious script has been run, that user's cookies and session information that the web application has knowledge of can be stolen. This script is trustworthy to an end user because the script originates directly from the web application itself. Scripts have access only to cookies belonging to the web application from where the script originates. But this "Same Origin" rule is bypassed in an XSS attack.

Reflected XSS: Reflected XSS is the most frequent kind of XSS attack found on the internet. Reflected XSS attacks are also known as non-persistent XSS attacks and the attack payload is delivered and executed via a single request and response. When a web application is vulnerable to reflected XSS, it will pass unvalidated input sent through HTTP requests back to the client. The common phases of the attack

include a design step, in which the attacker tests a vulnerable URL and manipulates it, a social engineering step, in which the attacker convinces the victims to load the manipulated URL on their browsers, and the eventual execution of the malicious code using the victim's browser. The data provided by an attacker, most commonly in HTML form submissions or in HTTP query parameters, is used immediately by server-side interpreters to parse and display a page of results, without properly escaping the the characters in the malicious request.

Stored XSS: In stored XSS, the server saves the data provided by the attacker, and then it is permanently loaded on pages returned to other users when they are regularly browsing, without proper HTML escaping. In Stored XSS, a user does not need to click on a malicious link to be exploited. A successful attack occurs when a user visits a page with stored XSS. Attacker stores malicious code into the vulnerable page and the user visits vulnerable page. Malicious code is executed by the user's browser.

DOM XSS: The actual exploit happens entirely inside the browser of the victim. No communication to the webserver is required, and hence no evidence of the attack is recorded in the server's logs. The DOM, or Document Object Model, is the structural format used to represent documents in a browser. Dynamic scripts such as JavaScript are able to reference components of the document such as a form field or a session cookie with DOM. The DOM is also used by the browser for security purposes like limiting scripts on different domains to obtaining session cookies only for respective domains. A DOM-based XSS vulnerability occurs when a DOM element that can be controlled by an attacker with the use of specially crafted requests that modify active content like a javascript function. Automated scanning tools can find some XSS vulnerabilities automatically. Automated detection is made difficult because each web application generates output pages differently and uses different browser side interpreters like JavaScript, Flash, ActiveX and Silverlight.

2.3 Insecure Direct Object References

This vulnerability is exploited when an attacker, who is already an authorized system user, changes a parameter value that directly refers to a system object to some other value that refers to another object which the user isnt authorized access for but can still gain access [7]. When web applications generate web pages they often reference an object by using the actual name or key of that object. Sometimes, whether the user is authorized to access target object or not, is not verified by web applications. This results in an insecure direct object reference vulnerability. Typically, automated tools do not look for these vulnerabilities because it is difficult for them to recognize which object requires protection or what object reference is safe or unsafe. In its simplest form this vulnerability can be exploited by modifying a URL string. A more advanced way of attacking is to poison some hidden parameters and access to data the developer did not intend for an attacker to see is gained.

Once an attacker has discovered such a security hole in a web application, they will write a script that sends a bunch of GET or POST requests, incrementing the poisoned parameter through values as necessary. The responses are then logged and collected. A lot of sensitive data is exposed in this manner. Identifying this vulnerability is difficult using Automated tools because to exploit this vulnerability, the flawed interface which may include URL and Links, Hidden Variables, Unprotected View State (ASP.NET), JavaScript Array and client side objects like Java Applets, needs to be identified. Moreover the key pattern to identify an secure object needs to be predicted like Filename etc.

2.4 Cross Site Request Forgery

Cross Site Request Forgery (CSRF) is an attack that allows the attacker to trick a user into performing an action, using user's own authority and credentials. CSRF

can also be called "Session Hijacking", even though session hijacking can also be done through Cross Site Scripting attacks. The attack works by submitting HTTP requests from the victim's web browser to a vulnerable web application. The victim's browser is tricked and it automatically sends the authorization information stored in cookies with the HTTP request [8]. If the victim is logged into the vulnerable web application, the requested action is performed without any problems because the web application trusts the cookie sent and does not know that a forged request was sent. The attacker basically creates forged HTTP requests and fools a victim into submitting them via HTML image tags or using a combination of CSRF and XSS. It is important to note that for the attack to succeed the victim needs to be authenticated prior to the attack.

Authentication information like session cookies are automatically sent by the browser which allows attackers to create malicious web pages which craft forged HTTP requests that are indistinguishable from genuine ones. In order to defend against CSRF attacks, when critical HTTP requests are recieved by the server, the user must reauthenticate and prove that he/she is human and indeed sent the HTTP request via CAPTCHA or other mechanism [9]. It can be determined whether the web application is vulnerable to CSRF attacks if any forms or links contain tokens that can be easily forged because they are predictable.

2.5 Broken Authentication / Session management

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to jeopardize passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users identities. The method of attack is to exploit weaknesses in the authentication mechanism that the web application uses [10]. For example, Cookies are used to maintain session of a particular user and they should expire once the user logs out of their account. In secure web applications, Cookies immediately expire once the user logs out of their

account. When this does not happen same cookies can be used again and again to open the session of the victim.

There may be several forms of broken authentication and session management like user authentication credentials are weakly encrypted, credentials can be guessed or overwritten through weak account management functions like account creation, change password, recover password. Some other flaws may be like session Ids are exposed in the URL, session Ids don't expire and there are no timeouts, authentication tokens are valid even after logouts, session Ids arent changed after every login, logout cycles and passwords, session Ids, and other credentials are sent over unencrypted connections. This vulnerability can be exploited through SQL injection, and this is the only case in which it can be easily detected by automated tools. In other cases, this vulnerability is one which requires complex algorithms in order to even come close to the ease with which human hackers can detect it.

2.6 Information Leakage

Exposed system data or debugging information helps an attacker learn about the system and plan an attack. An information leak occurs when system data or debugging information leaves the program through an output stream or logging function [11]. In some cases the error message tells the attacker precisely what sort of an attack the system will be vulnerable to. For example, a database error message can reveal that there exists an SQL injection vulnerability in the web application. Other error messages can reveal more oblique clues about the system. Automated scanners like the one developed for this dissertation often use this method to detect vulnerabilities in web applications. HTTP Headers related to server information are a risk, which can cause an HTTP Banner Disclosure vulnerability. It is difficult to prevent someone from finding this information using methods other than the one mentioned above, disabling server headers reduces the likelihood of attacks on a site. 'Informa-

tion Gathering' is one of the first steps of a hacking attack on any system, not just web applications. Thus Information Leakage poses a threat to the security of any system.

2.7 Security Misconfiguration

Attackers may use default accounts, unused pages, unpatched bugs etc. to gain unauthorized access to the system or to knowledge of the system. Security misconfiguration can occur at any level of the application stack, including the platform, web server, application server, database, framework and custom code [12]. Automated scanners are useful for detecting missing patches, misconfigurations of servers, use of default accounts, unnecessary services available to users, etc. If error handling reveals overly informative error messages to users or default accounts and their passwords are enabled and unchanged or any unnecessary ports and services are enabled or installed, the web application can be attacked. Basically, a lot of information about the configuration of a web application is gathered prior to an attack.

A common Security Misconfiguration attack is the directory traversal attack. A directory traversal or path traversal consists in exploiting insufficient security validation of user-supplied input file names. It may be possible to access arbitrary files and directories stored on file system by manipulating variables that reference files which may include application source code, configuration and critical system files, limited by system operational access control. Directory traversal is also known as the dot-dot-slash attack, directory climbing, and backtracking.

2.8 Vulnerable components

If attackers identify that a component with known vulnerabilities has been used in the web application, they customize the exploit as needed and execute the attack.

Many developers do not focus on ensuring their components or libraries are updated. Sometimes, the developers don't even have the knowledge of which components are being used. Situations become more complicated because many components of the web application have inter dependencies among them. Not all vulnerabilities of out of date components are known and sometimes it is not even known that that component has vulnerabilities. Hence it is not easy to figure out if the web application is using a component with known vulnerabilities.

2.9 Cross Site Tracing

The HTTPOnly feature supported by browsers protects HTTP cookies from Cross Site Scripting attacks. However, this feature prohibits the leakage of cookie information via the 'document.cookie' object. The 'HTTP trace' method is utilized to echo input data for HTTP. The 'HTTP Trace' request containing headers and POST data generates a response which contains the information that was sent in the request, when it is sent to a web server that supports the trace method. The Trace method supplies a mechanism to verify that the server is receiving exactly what the http client is sending. Access to the 'document.cookie' object, which is normally an XSS attack target, for scripting languages is controlled by the 'HTTPOnly' cookie option. The idea of XST (Cross Site Tracing) is to obtain the cookie data that is usually within 'document.cookie' despite httpOnly being in use [13].

Trace has the capability to echo cookie and authentication information that is sent in the HTTP Request. A specially crafted HTTP request is sent to the target web server. The server will then echo, if it supports Trace, the information sent within the HTTP request via a resulting JavaScript alert for instance. If the browser has a cookie from the targeted domain or the victim is logged into the targeted web server using web authentication, the attacker will be able to capture the cookies and credentials of the victim through the alert. This method allows the attacker to access cookie

information without using 'document.cookie' object by bypassing 'HTTPOnly'. All the exposed data can be gained ever over HTTPS.

2.10 LDAP Injection

The Lightweight Directory Access Protocol (LDAP) is a protocol used for communicating with directory services over the TCP/IP protocol. LDAP directory services store and organize information whose structure is based on a tree of directory entries. LDAP is based on a client-server model and is object oriented. The security of many web applications rely on single sign-on capabilities which are provided through LDAP directories. An LDAP server provides powerful browsing and search capabilities.

Directory entries are searched using filters which are contained in queries sent by the clients to the server. LDAP services use the directory for the purposes of username password pair verification, user certificates management, privilege management and resource management. LDAP servers are placed behind a backend firewall along with database servers due to their importance.

LDAP injection works in a similar manner to that of SQL injection attacks [14]. A faulty LDAP query can be generated by the attacker using unsanitized parameters in the web application.

2.11 XPath Injection

An XML document's various sections are referred to by the XPath language. An XML document can be directly queried by an application using the XPath langage. The XPath query is quite similar to an SQL query and an XML document is analogous to a database. XPath Injection attack allows the hacker to obtain a complete XML document with the use of XPath querying [15].

XPath injection is a complete attack because the entire XML data is compromised. The attack is carried out using two methods viz. XPath crawling and XPath Query Booleanization. This attack can be very devastating for web applications that make use of XPath queries and XML databases for purposes like user authentication. If the return type of a query is a string or a numeric value, then such a query is called a scalar query. When a scalar query is substituted with a query whose return type is Boolean, and from which the result of the original scalar query can be recovered, the procedure is called the Booleanization process.

Any XPath based application that has the XPath injection vulnerability can be attacked in an automated fashion because XPath is a standard language. Unlike SQL, there are no access control restrictions for the XPath language and it can be used to refer the majority of an XML document. The exact structure of the XPath query need not be known by the attacker. With trial and error, the attacker can craft query data that can be utilized for Blind XPath Injection. After that, the database in the form of the XML document can be retrieved using an automated script. Similar to SQL injection testing, to determine if the application is vulnerable to XPath Injection can be checked by injecting a payload into the XPath query. The response is inspected and if an error is part of the response, then it can be known that an XPath injection vulnerability exists.

2.12 Cross Site Flashing

The attack takes advantage of the fact that Flash files can reference external URLs [16]. If variables that serve as URLs that the Flash application references can be controlled by through parameters, then by creating a link that includes values for those parameters, an attacker can cause arbitrary content to be referenced and possibly executed by the targeted Flash application. The attacker must convince the

victim to follow a crafted link to a vulnerable Flash application. Cross-Site Flashing occurs when user controlled data is not validated and used in functions or variables like loadVariables, loadMovie, getURL, loadMovie, loadMovieNum, FScrollPane.loadScrollContent, Sound.loadSound, NetStream.play and htmlText.

2.13 Open Redirects

Many web applications on the Internet use redirection. If there is no additional security, many redirection links can be manipulated and misused to camouflage phishing attacks. A redirect is open if the destination URL contained in its query string can be altered and the web server processing the redirect sends the client to the new destination without validation [17]. The best way to find out if a web application has any unvalidated redirects is to identify if the target URL for any redirects is included in any parameter values. The most comprehensive way of dealing with Open Redirects is to spider/crawl a web application and look for HTTP response codes ranging from 300 to 307 and then test each such redirect causing page for the vulnerability.

Chapter 3

Web Application Vulnerability Scanner Standards

In this chapter, web application security scanner evaluation criteria and scanner standards are discussed [18]. What a good vulnerability scanner should be functionally capable of is discussed.

3.1 Support for Protocols

A scanner should provide support for every protocol that is normally utilized by web applications and other network equipments as well for effectively testing web applications. Web applications utilize the HTTP protocol for communication between clients and servers.

- HTTP Keep-Alive: A web application vulnerability scanner must keep all sessions with the server open and utilize those sessions for many simultaneous HTTP requests unless explicitly told otherwise by the web server in order to increase performance.

- HTTP Compression: Application content is compressed often by web application servers before it is sent to the client to increase performance. HTTP

compression must be supported by web application scanners for better performance and proper parsing of compressed content that they come across during the scan.

- HTTP User Agent Configuration: It is very important that the web application scanner has the ability to behave like a web browser. Different content is presented by web applications according to the browser that the client uses. Configuration of particular user agent strings to be utilized during an ongoing scan should be possible for the penetration tester.

Normally, direct access to a web application is not available to the scanner. In such cases, the configuration of a proxy should be made available to the penetration tester. This should be avoided unless there is no other way because the use of a proxy while scanning the web application increases the number of false positives in the scan results. Proxy support must be available for proxies like Socks 4 and Socks 5. In many scenarios the scanner has to use separate rules to determine which URLs need to be requested via proxy and which ones are to be requested without a proxy. The use of proxy auto configuration (PAC) files must be supported by the scanner to appropriately handle such scenarios.

3.2 Support for Authentication Mechanisms

A web application security scanner should support the following authentication schemes: Basic, Digest, HTTP Negotiate (NTLM and Kerberos), HTML Form-based (Automated, Scripted and Non-Automated) and Single Sign On schemes like OpenID.

- Automated Authentication - A legitimate username and password are supplied to the scanner by the penetration tester during the configuration of the scan.

- Scripted Authentication - Before the scan is started the login process can be scripted or recorded by the tester.

- Non-Automated Authentication - The scanner supports user intervention while the scan is being performed - for example, to solve a CAPTCHA or enter a value from a hardware token for a web application that has implemented two-factor authentication.

3.3 Session Management

It is important that a live and valid session is maintained with the application by the scanner during the period of vulnerability scan. A valid session is absolutely necessary for properly crawling the web application. For achieving the optimum coverage of the application, a valid live session needs to be established by the scanner which would allow it to find all possible elements of the application like links, cookies, forms and parameters. A vulnerability scanner will most probably be redirected to the login page of the application under test, if, the HTTP request sent to the application has an invalid session token. The HTTP request would simply be ignored by the server. The scanner should be able to deal with all commonly used session management schemes for maintaining a valid session.

It should be understood by the scanner that it is being asked to start a new session by the web application under test with the use of a particular session token which uniquely identifies the current session. When an application instructs the scanner to refresh a session token, the scanner should be able to do so. The scanner should be capable of detecting that a session has expired.

Session Management Token Type Support: A web application vulnerability scanner should be able to support commonly used session management token types including HTTP cookies, HTTP parameters and HTTP URL path. Web application scanners must be capable of imitating the exact behaviour of web browsers like following instructions to set cookies or refresh cookie values.

A web browser can be instructed by the vulnerability scanner to use parameters as session tokens in many ways including embedding the tokens in consecutive HTML links. HTTP parameters should be used by web vulnerability scanners as session tokens. Session tokens are embedded inside the path of a URL by a few web aplpications, for example- http://site.com/opb/(Session Token)/final.asp. In scenarios like these, a web vulnerability scanner must be capable of dealing with specific token formats. It should be made sure of that, if an application structure analysis is being performed by the scanner during a scan, the scanner does not regard a session token like the one mentioned previously as a part of the application directory, because this may produce test cases that are not relevant.

3.4 Efficient Crawling

In order to thoroughly and efficiently crawl a web application being tested, the penetration tester should be allowed to configure the crawler so that a large number of criteria is covered. A penetration tester should be provided with the functionality to define a seed URL for the crawl. Normally, '/' is regarded as the root URL of a website, but that is not always true. A penetration tester should be able to provide additional hostnames or IP addresses in a list or range format for crawling. A website may be made up of any domains and subdomains like www.site.com, group.site.com and chat.site.com. It is efficient and time saving if the penetration tester can crawl all domains and subdomains of the web application together in a single crawl. The scanner should also provide the user with the option to define exclusions for specific hostnames (or IPs), specific URLs or URL patterns (regular expressions), specific file extensions(e.g: .jpeg, .css).

The tester should be able to limit the number of requests sent to a web application. Large applications that have photograph albums often have a large number of

extra pages that don't need to be crawled. A tester should be allowed to limit the number of requests sent to redundant web pages within the application. Concurrent sessions should be supported by the web application scanner. The tester should be provided with the option for using multiple login sessions or single session for crawling the application under test. The tester must be able to provide a delay time for requests. Networks with low bandwidths often require throttling of requests sent to the application. Network traffic can be controlled using this technique. The tester should be able to provide a maximum crawl depth. The 'depth' of an application is the number of links that need to be followed to arrive at a particular page from the home page. If the number of links clicked is four, then the tester is at a depth of four. Crawling up to a crawl depth of three to four should be enough for a regular scan.

Most web applications have a link to another application within themselves, and the scanner must be able to identify new hostnames. Identifying a new hostname from the current hostname is important if the area of the crawling session needs to be curtailed or expanded as per the tester's wishes. Most web applications these days contain some sort of forms within their pages and some pages can only be reached after filling of these forms. Hence, the scanner must be able to auto submit forms to the web application. The crawler should have the capability to follow HTTP redirects that have HTTP response codes in the range of 301 - 307, follow Meta Refresh redirects and follow JavaScript redirects. A crawler must support AJAX applications. At a minimum, it should be able to automatically submit XmlHTTPRequests that are found during the crawling process.

3.5 Scanner Parsing Ability

The first step for a web vulnerability scanner should be mapping of the structure and functionality of the web application under test. The web crawler component does this procedure by making use of varied content parsers to obtain data from web content.

This information might include URLs, HTML forms, HTML form parameters, HTML comments etc. A scanner must be able to parse the following content types to obtain data regarding the structure and functionalities of the application:

1. HTML
2. JavaScript
3. VBScript
4. XML
5. Plaintext
6. ActiveX Objects
7. Java Applets
8. Flash Objects
9. CSS

Oftentimes, web content may not conform to appropriate standards, either by mistake or due to various types of problems, ranging from bad developer habits to communication problems. In such cases, it is crucial that content parsers will be able to cope with partial or non well-formed content and still be able to extract the relevant information from the application responses. It is important that web applications be at least as robust in their HTML parsing as web browsers are. User interaction with client-side logic must be emulated by web application scanners for dynamically parsing scripting languages such as JavaScript, VBScript so that links and relevant application information can be detected.

3.6 Test Customization

Payload precision and test customization is necessary in order to find out application specific vulnerabilities although most web appliciations have the exact basic technology and use the same communication protocol. Modification of existing tests must be allowed by a web application vulnerability scanner. The scanner should allow modification of the algorithm used to carry out the attack i.e. the attack steps and the criteria to identify a vulnerability positively. A scanner must let the tester create new customized tests or simply let the tester to append new tests that use templates and regular expressions. Most web vulnerability scanners have a substantial number of implicit tests, however most of the time, just a subset of these tests is required. A web vulnerability scanner ought to permit the tester to make customized test arrangements that point out which tests to incorporate in a scan.

3.7 Scanning, Command and Control

Frequently applications are only permitted to be examined during specific time frames by the managers of web applications. Subsequently, it is vital that a web vulnerability scanner oblige the tester with the capacity to schedule a scan. This planning ought to incorporate a begin time and maximum span of the scan, after which the scan would be paused if it has not yet finished.

A scanner should provide the operator with the ability to pause a scan and then resume the scan later from the point at which it was paused. A scanner should provide the operator with a way of viewing the real-time status of running scans. This status could include information such as which tests currently run and scan completion percentage. Organizations that have numerous web applications to audit would approve the capability of running multiple scans together and find it an attractive characteristic.

A scanner should provide the user with the ability to save a scan's configuration so that it can be re-used for later scans. For organizations that want to provide a web application scanning solution for many clients, the scanner's capacity to help numerous clients ought to be considered. The capability to provide remote or distributed scanning is exceptionally imperative for decentralized configurations where web applications are found behind firewalls or must be gotten to with use of VPNs or over slow connections. To successfully scan web applications in these situations, a web vulnerability scanner ought to give the capability to send scanning "operators" inside an organization's system, which might be safely controlled by a remote administrator.

3.8 Report Types

In spite of the fact that particular reporting choices may fluctuate, scanners ought to give the following sorts of reports:

- Executive Summary: A brief idea about the scan results is provided through an executive summary.
- Technical Detail Report: Scanners must have the capacity to give all technical data needed by testers to duplicate the detected issues. The ability to include full request and response data and the ability to include a list of all hosts and URLs included in the scan is included.
- Delta Report: Scanners should provide the ability to compare results from two or more scans and show differences or trends over time.
- Scanners ought to have the ability to create reports in both human and machine-readable formats which include PDF, HTML and XML.

Chapter 4

Implementation Methodology

4.1 Implementation Details

The web application vulnerability scanner is made up of three parts [1]:

1. The Crawling Component

2. The Attack Component or The Fuzzing Component

3. The Analysis Component

4.1.1 The Crawling Component

The penetration tester seeds the scanner's crawling component with a root web address in order to start a crawling session. The most important part of the scanner is the crawling component. Classical black-box web vulnerability scanners crawl the web application to be tested, enlist all reachable pages of the application and then fuzz the data to be input, like URL parameters, form values and cookies, to trigger vulnerabilities.

The state of the web application is any data (database, system, time) that the web

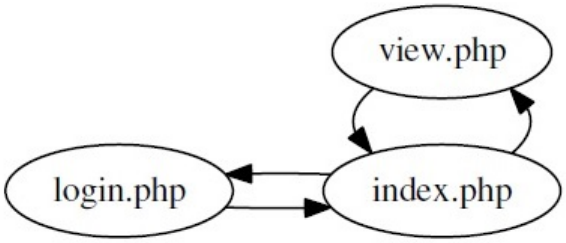

Figure 4.1: Navigation graph of a simple web application

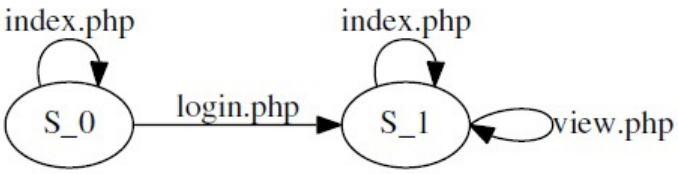

Figure 4.2: State machine of a simple web application

application uses to determine its output. The way a user interacts with the application determines the applications state. A black-box web vulnerability scanner will never detect a vulnerability on a page that it does not see, scanners that ignore a web applications state will only explore and test a (likely small) fraction of the web application [19]. As seen in figure 4.1 and figure 4.2, only state information can help a scanner understand that there are two instances of the web page "index.php", one when the user is logged in and one when the user is logged out.

4.1.2 The Fuzzing Component

A Link can be either an anchor or a form. An anchor always generates a GET request, but a form can generate either a POST or GET request. This component scans every page for existence of forms since they are the entry points for a user. Then, according to the attack that is to be performed, form field values are suitably chosen through Fuzzing.

The attack component creates values that would likely trigger an error due to an existing security hole for each entry point and every vulnerability for which the web application is to be tested. For example, JavaScript code would be injected by the attack component if the web application is being tested for XSS vulnerabilities, and when it is being tested for SQL injection vulnerabilities, special characters such as single quotes and double dash would be injected by the component. Either predefined input values are used or they are made up using fuzzing and heuristics. Predefined input values are available in XSS and SQL injection cheat sheets that real hackers use for attacking web applications.

Predefined Payloads

For automated detection of SQL injection flaws, following are the most common payloads that are injected by attackers: ' ") . (– ;

In order to automatically test for Reflected Cross Site Scripting, the payloads listed below may be used. Testing data can be generated by using a fuzzer, or a list of known attack strings may be predefined , or attack strings may be input manually by the tester.

- <xss>
- javascript:alert(xss found)
- <script>alert(xss found)</script>

There exist several ways in which the payload attack strings can be manipulated which are known as 'Filter Evasion Techniques'. These techniques are used by hackers when web application developers sanitize user input fields and web application firewalls are used. Some of these techniques are discussed in section 4.2 that follows this section.

For detection of Broken Authentication using SQL injection, the predefined payloads most commonly used are:

- 1'or'1'='1
- 1'or'1'='1';#

For Stored Cross Site Scripting, the payloads injected become persistent objects or permanent database entries. There is no erroneous response from the server after the storage of the payloads in the database. Hence there is no direct and immediate indication that the web application is vulnerable to Stored Cross Site Scripting attack. The crawler checks a second time for pages that contain input injected previously. This is the main reason why scanning for Stored XSS vulnerabilities takes longer than scanning for other vulnerabilities.

4.1.3 The Analysis Component

The HTTP protocol defines server responses for different HTTP requests sent by clients. The job of the analysis component is to parse/decipher the response page received from the server after the attack. An analysis module finds keywords and other criteria specific to different attacks in the HTTP to determine a confidence value that decides if the attack was successful. For example, if there exists a SQL injection vulnerability in the web application being tested then the server response page returned will contain a database error message.

Server Responses to Payloads

After the injection of predefined or fuzzed payloads if the web applications display database errors like the ones listed below then it can be assumed with high confidence that the web application is vulnerable to SQL injection:

- supplied argument is not a valid MySQL

- You have an error in your SQL syntax
- MySQL server version for the right syntax to use
- Column count doesn't match
- the used select statements have different number of columns
- System.Data.OleDb.OleDbException
- System.Data.SqlClient.SqlException
- Unclosed quotation mark after the character string
- Incorrect syntax near
- Syntax error in string in query expression
- DB2 SQL error
- PostgreSQL query failed

For Reflected Cross Site Scripting, if the response received from the server contains the same payloads that were used for the attack, then it can be inferred that there exists a Reflected Cross Site Scripting vulnerability in the web application. Similarly while analysing a web application for Broken authentication, the scanner sends requests to the same authentication page at regular intervals after injection of payloads. If the same page is displayed then there is no vulnerability, because a page would always look different after logging in. In order to check if any server information is being leaked, the HTTP Response headers are checked for phrases like: Apache, Win32, OpenSSL, PHP, IIS, Microsoft, Unix, Linux, Ubuntu, Perl, Python, ASP and NET.

4.2 Implementation Approach for Unimplemented Techniques

4.2.1 Recording User Input

Human penetration testers have a major advantage over automated scanning tools in the manner in which both can interact with the web application to be tested. Normally, a penetration tester has specific intentions and goals that are focused on when they interact with a web application. A human tester knows their targets, and the required steps to compromise those targets. However, a web application has use cases whose knowledge automated scanners lack. It can only try to obtain information about possible entry points and inject those with payloads. Most automated scanners fail to reach large sections of a web application by not generating enough legitimate requests. The part that limited payload variety and parsing abilities play in failure of automated scanners is minimal. Hence recording user input can lead to a deeper crawling of the web application.

User interactions with a web application are mapped to use cases. For each use case a user of the web application performs a series of actions to fulfill their agenda like visiting posting a comment on a forum or a blog [20]. Simple input data like log in credentials can be supplied by the penetration tester at the start of the scanning itself. Complex input data to be used for forms can be obtained by recording the requests sent to application's servers by utilizing a proxy as a section between the client and server. Of course the tester of the web application has to know that the inputs are being recorded, so that sensitive information is not recorded and stored. The recording of user inputs has to take place in a environment that can be controlled. The recorded use cases are then replayed by the fuzzing component in order to better crawl the web application.

4.2.2 Guided Fuzzing and Stateful Fuzzing

A previously recorded use-case is replayed by the vulnerability scanner, for each entry point iteratively. After every entry point has been passed through using recorded input values, the fuzzing/attack component is invoked. The legitimate inputs that were a part of the request sent before are replaced by the fuzzing component using a list of payloads and the response is analyzed. This is called Guided Fuzzing [20]. To extend this method, all entry points available after the use of correct values for one entry point are fuzzed and their responses are analysed. In essence, the crawler guides the fuzzing component through a use case of the web application in order to scan as much portion of the web application as possible.

However, the above method can lead to change in state of the web application. When replaying recorded use cases, this is a drawback, but it can be overcome by recording the state of the application after every step is replayed. After that, the fuzzing component is invoked. The applications state might change because of this. But, the application is rolled back to the previous state using the snapshot, after each round of fuzzing. After that, the replay component of the fuzzer can advance one step. This process is called Stateful Fuzzing.

It is difficult to record the state of a web application by taking its snapshot, but it is possible because most web applications use the Model-View-Controller Architecture for development. In order to record the state of the application and restore it later, the use of objects that were formed, changed or removed by the object manager according to HTTP requests can be made. The data about updation of objects may be retrieved at the boundary at the application and data layers of the MVC architecture.

4.2.3 Web Application Firewall Evasion

A Web Application Firewall (WAF) is used to block commonly known exploits by use of regular expressions and rule sets. Web application firewalls are part of the defense in depth model for web applications in which layered security mechanisms increase security of the system as a whole. WAFs support special features like cookie encryption, CSRF protection, etc. WAFs are typically deployed in 'blacklisting mode' that is more vulnerable to bypasses and targeted attacks. In this mode, WAFs filter out malformed inputs that are known to cause an attack. Application context is necessary to know for deploying of more secure 'whitelisting mode' in which only certain inputs are allowed and all other inputs are blocked. There are several ways in which security threats to web applications can be reduced as follows:

- Directive approach - Use of Secure Coding techniques during Software Development Life Cycle
- Detective approach - Black box/white box testing of functions, fuzzing as well as static/dynamic/manual analysis of program code.
- Preventive approach - Use of Intrusion Detection/Prevention Systems (IDS/IPS) and Web Application Firewalls(WAF)

If a WAF can be bypassed, it does not imply that the web application is vulnerable. But a mechanism to bypass the WAF must be incorporated in the automated scanner, because WAVS is a black box testing tool and on many of the occasions it will be scanning a web application after a WAF has been deployed. WAF filter rules directly reflects WAF effectiveness. For human penetration testers, the algorithm for bypassing a Web Application Firewall goes something like this [21]:

1. Find out which characters / sequences are allowed by the WAF.
2. Manipulate the previously injected payload so that it is obfuscated.

3. Test it and watch for the WAF/application response.

4. If it does not work, modify the payload again and repeat.

Blacklisted inputs used in the injected payload cause the 'Forbidden' Error 403-416 responses from the server. For an automated tool the testing would go something like this:

1. Scan for Entry points to the web application like usual.

2. Inject a 'Normal' predefined or fuzzed payload into the entry point according to the vulnerability being tested.

3. If the server/WAF responds with a 'Forbidden' Error Response, obfuscate the predefined/fuzzed payload with a filter evasion technique for that particular vulnerability.

4. Continue obfuscating the predefined/fuzzed payloads in different ways one after the another until the vulnerability is found through a server response besides the 'Forbidden' response.

Filter Evasion Techniques for SQLI and XSS

Filter evasion techniques for SQL injection are as follows:

- Using Comments - the scanner can use comments to enclose SQL query keywords so that the payload goes undetected. For example, http://www.site.com/index.php?page_id=-12 /*!UNION*/ /*!SELECT*/ 1,2,3.

- Changing the case of the letters in the query keywords - some words like union and select are blacklisted in WAFs. Changing the case of these keywords lets them through the WAF but they are still executed by the database server. For example, http://www.site.com/index.php?page_id=-12 uNiOn SeLeCt 1,2,3,4.

Blocked Attack	Undetected modification
'or 1=1--	' or 2=2--
alert(0)	%00alert(0)
<script>alert(0)</script>	<script type=vbscript>MsgBox(0)</script>
' or ""='r	'/**/OR/**/""='
<script>alert(0)</script>	<img src="x:x" onerror="alert(0)"></img>
</img>	</img>
1 or 1=1	(1)or(1)=(1)
eval(name)	x=this.name X(0?$:name+1)

Figure 4.3: Examples of WAF Evasion Techniques for SQLI and XSS

- Replaced keywords - Some Firewalls remove the 'UNION SELECT' Statement when it is found in the URL and replace them with an empty string. In order to bypass this, attackers enclose the full keywords within themselves, for example, http://www.site.com/index.php?page_id=-12 UNIunionON SELselectECT 1,2,3,4.

- Replace Characters with their HEX/ASCII Values - Character encoding can be done to make the keywords undetectable. Most database servers allow the execution of queries represented using the most common encoding techniques. For example,
  http://www.site.com/index.php?page_id=-12 /*!u%6eion*/ /*!se%6cect*/ 1,2,3,4.

Most of the techniques used like character encoding can be used similarly for XSS attacks. Figure 4.3 shows a few examples of how hackers modify their payloads in order to bypass WAFs [21].

It is necessary for the Vulnerability Scanner to be able to bypass a WAF because if the Scanner has the ability to bypass a WAF, then it most certainly is capable of

injecting a payload to a vulnerable entry point in the web application that will cause an attack. As mentioned earlier, being able to bypass a WAF does not imply that the web application itself is vulnerable. But many organizations make use of WAFs and they assume that a Web application firewall does enough input validation for the web application to be secure. In doing so they neglect the strictness of input validation for the web application itself. If the WAF is bypassed then there is not much defense left to stop an attack on the web application. With WAVS, it can be checked whether the web application is still vulnerable despite the use of a WAF.

Chapter 5

Scanner Evaluation Results

The primary step for evaluating WAVS is composed of selecting an appropriate web application that is to be scanned for vulnerabilities. The application to be tested must fulfill the following requirements: [22]

1. Vulnerabilities must be clearly defined in the application in order to evaluate the scanners detection capabilities.

2. Customization: It should be easy to add different types of vulnerabilities to the web application and make crawling more challenging.

3. The web application must be realistic and be representative of the application functionality and web technology found on the internet today.

Scanner Assessment Criteria:

1. Time Taken For the Scan

2. Success Percentage (No. of Vulnerabilities Found / Known Vulnerabilities)

3. No. of False Positives

Vulnerable Websites	Actual Reported Vulnerabilities
(IBM) http://demo.testfire.net	BlindSQLi-01, XSS-11, SQLi-13
(Acunetix) http://testphp.vulnweb.com	BlindSQLi-06, XSS-08, SQLi-05
(Acunetix) http://testasp.vulnweb.com	BlindSQLi-03, XSS-27, SQLi-01
(Acunetix) http://testaspnet.vulnweb.com	BlindSQLi-08, XSS-10, SQLi-21
http://crackme.cenzic.com	BlindSQLi-05, XSS-12, SQLi-02
(E-Eye) http://www.webscantest.com	BlindSQLi-15, XSS-19, SQLi-13
http://www.dvwa.co.uk	BlindSQLi-01, XSS-02, SQLi-01

Figure 5.1: Test Applications for Scanner Evaluation

Test Websites	demo. testfire.net		testphp. vulnweb.com		testasp. vulnweb.com		testaspnet. vulnweb.com		webscantest. com		dvwa.co.uk		crackme cenzic.com	
Tools	SQLI	XSS	SQLI	XSS	SQLI	XSS	SQLI	XSS	SQLI	XSS	SQLI	XSS	SQLI	XSS
Sqlmap	31%	0%	72%	0%	50%	0%	28%	0%	14%	0%	100%	0%	14%	0%
W3af	31%	10%	36%	25%	50%	7%	21%	0%	14%	5%	100%	0%	43%	0%
Darkmysqli	23%	0%	36%	0%	25%	0%	28%	0%	11%	0%	0%	0%	0%	0%
Bbqsql	0%	0%	55%	0%	50%	0%	31%	0%	14%	0%	50%	0%	57%	0%
Vega	0%	27%	55%	100%	50%	0%	14%	0%	0%	37%	0%	0%	0%	0%
Arachni	15%	36%	55%	100%	100%	7%	0%	20%	14%	37%	0%	0%	0%	0%
Web Securify	8%	36%	55%	50%	50%	0%	0%	0%	18%	79%	0%	0%	0%	0%
Acunetix WVS	15%	45%	55%	100%	100%	19%	80%	70%	14%	100%	0%	0%	57%	33%
Nikto	7%	18%	27%	0%	25%	0%	6%	0%	11%	0%	0%	0%	14%	0%

Figure 5.2: Success Percentage of Various Scanners

Applications in which vulnerabilities are intentionally added to be found easily, like "HacmeBank" and "WebGoat" [23], are usually developed for educational purposes instead of being realistic testing environments for scanners. Other researchers have used web applications like "Wackopicko" [24] for testing purposes, which can be downloaded on to a local machine and pentested with the scanner. It has been decided to use web applications "testphp.vulnweb.com" and others as shown in the figures 5.1 and 5.2 that have known vulnerabilities as the test applications [25].

5.1 Testing Conditions

- Automated tools basically work in two modes: point and shoot (PAS) and trained; semiautomated tools mostly perform trained scans. The effectiveness of the scan is affected accordingly [25].
- The scan results shown below are for tests carried out in PAS mode.
- The scanner developed for dissertation works better and finds more vulnerabilities in web applications for trained mode scanning than PAS mode.
- Standard SQL Injection and Reflected Cross Site Scripting are the only vulnerabilities that are scanned for some of the tests. Other vulnerabilities like Insecure Direct Object References, Stored XSS, Broken Authentication using SQL injection and Information Leakage are scanned for in only a few vulnerable web applications. This is because of insufficient data for result comparision and analysis.
- It is extremely difficult to find realistic applications that have known vulnerabilities other than SQL injection and Cross Site Scripting, on which the scanner developed for dissertation can be tested.

5.2 Test Results and Analysis

Following are the test results for various vulnerable web applications, when tested with the scanner developed for dissertation:

1. Website: http://demo.testfire.net
    - Scan Duration: 7 hours 15 minutes (User Abort)
    - Success Percentage: SQLI - 0% (0/14), Reflected XSS - 18% (2/11)
    - URLs found: 1914

- HTTP requests sent: 8270
- Remark: Scanning takes too long. It can be Improved with concurrent HTTP requests. Scanning slows down as it progresses, if the browser is being used for interacting with other web applications while scanning, because of low buffer memory of the web browser.

2. http://testphp.vulnweb.com

   - Scan Duration: 1 hours 32 minutes
   - Success Percentage: SQLI - 55% (6/11), Reflected XSS - 100% (8/8)
   - Other Vulnerabilities Found: Broken Authentication - 2, Insecure Direct Object References - 1, Information Leakage - 1
   - URLs found: 60
   - HTTP requests sent: 2765
   - Remark: Broken Authentication found using SQL injection and can be considered as SQLI with success percentage of 73%.

3. http://testasp.vulnweb.com

   - Scan Duration: 22 minutes
   - Success Percentage: SQLI - 100% (4/4), Reflected XSS - 26% (7/27)
   - Other Vulnerabilities Found: Broken Authentication - 2, Information Leakage - 1
   - URLs found: 23
   - HTTP requests sent: 785
   - Remark: Application had not been tested for Stored Cross Site Scripting because of long scan times. Application crawled several times, each time with improved crawling results.

4. http://webscantest.com

    - Scan Duration: (Scanner Abort)
    - Success Percentage: SQLI - 0% (0/28), Reflected XSS - 0% (0/19)
    - URLs found: 110
    - HTTP requests sent: 265
    - Remark: Scanning stops at a jpg URL because of a bug in the scanner.

5. http://dvwa.co.uk

    - Scan Duration: 2 minutes
    - Success Percentage: SQLI - 0% (0/3), Reflected XSS - 0% (0/2)
    - URLs found: 9
    - HTTP requests sent: 54
    - Remark: Application is not fully crawled by the crawler. Deep crawling yet to be implemented.

6. http://vulnweb.janusec.com

    - Scan Duration: 22 minutes
    - Success Percentage: SQLI - 100% (18/18), Reflected XSS - 100% (3/3)
    - URLs found: 19
    - HTTP requests sent: 275
    - Remark: All vulnerabilities present in the application were found. But several results were duplicated, with some vulnerabilities being reported more than once for the same URL.

7. http://php.testsparker.com

    - Scan Duration: 31 minutes

- Vulnerabilities Found: Reflected XSS - 2, Insecure Direct Object References - 3, Information Leakage - 2
- URLs found: 14
- HTTP requests sent: 1265
- Remark: Results of other scanners are not available for comparison. The number of known vulnerabilities is also not known to calculate success percentage.

Even though methods to find many types of vulnerabilities are well established and look like they work reliably, there exist some categories of vulnerabilities that are not researched and are not easily found by the best vulnerability scanners [26]. Applications that contain complicated forms and that aggressively check input values can completely prevent a PAS scanner because of wrong auto form fill data, blocking it from crawling pages that are hidden further inside the web application. However, in trained mode the tester possesses specific parameters, values, scripts, and possible injection points of the website and feeds the data into the scanner.

The time taken to perform scans and obtain results depends upon concurrency support and the search algorithm used. Concurrency support stresses the number of concurrent HTTP Request threads a scanner can have. Web application scanners basically use frequency-based analysis search, binary search, and linear search for parameter testing. Among these binary search is most effcient.

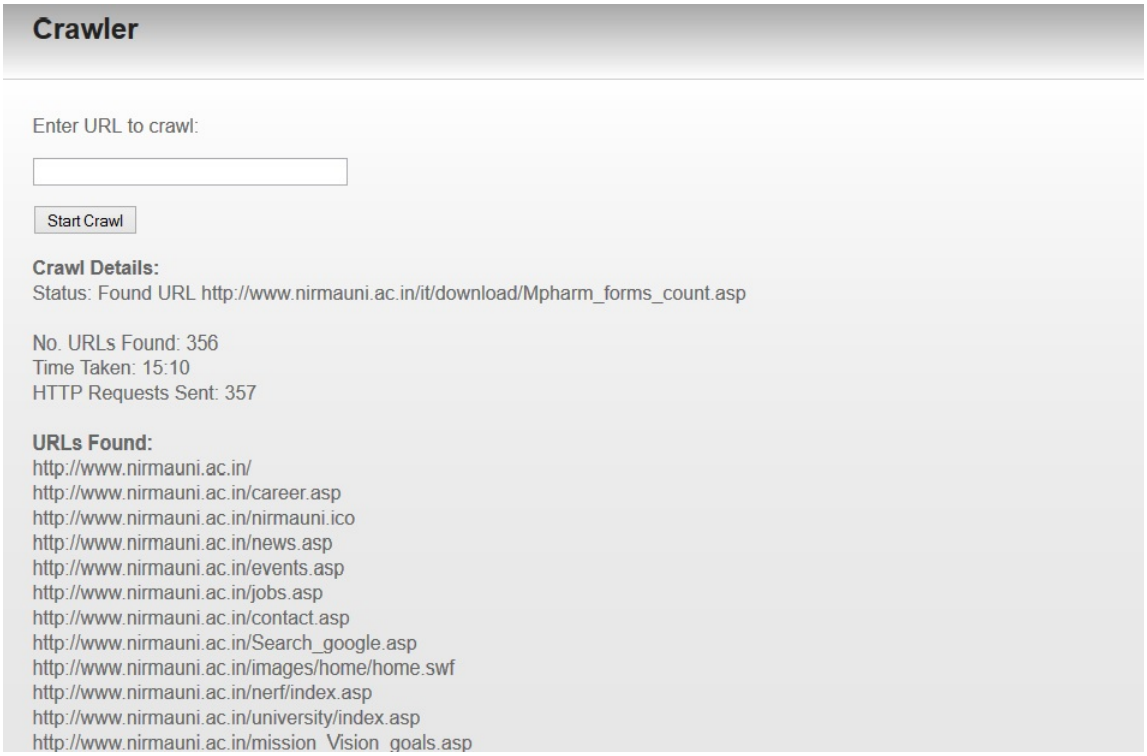

Figure 5.3: Web Application Vulnerability Scanner Screenshot 1

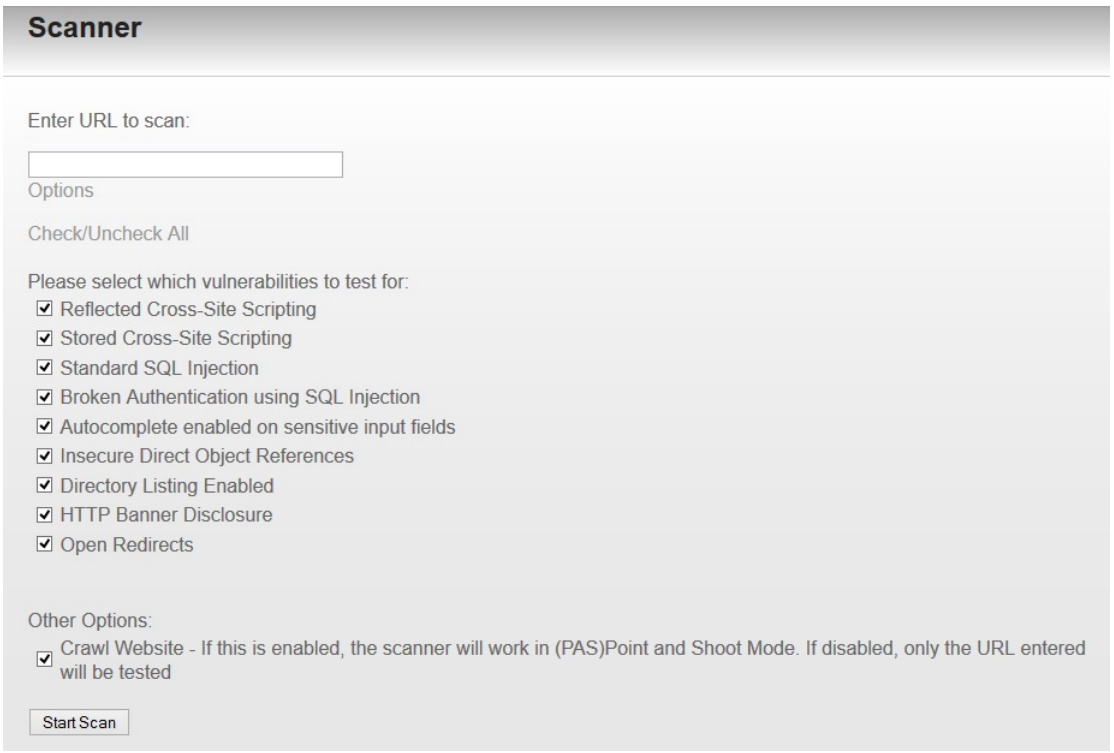

Figure 5.4: Web Application Vulnerability Scanner Screenshot 2

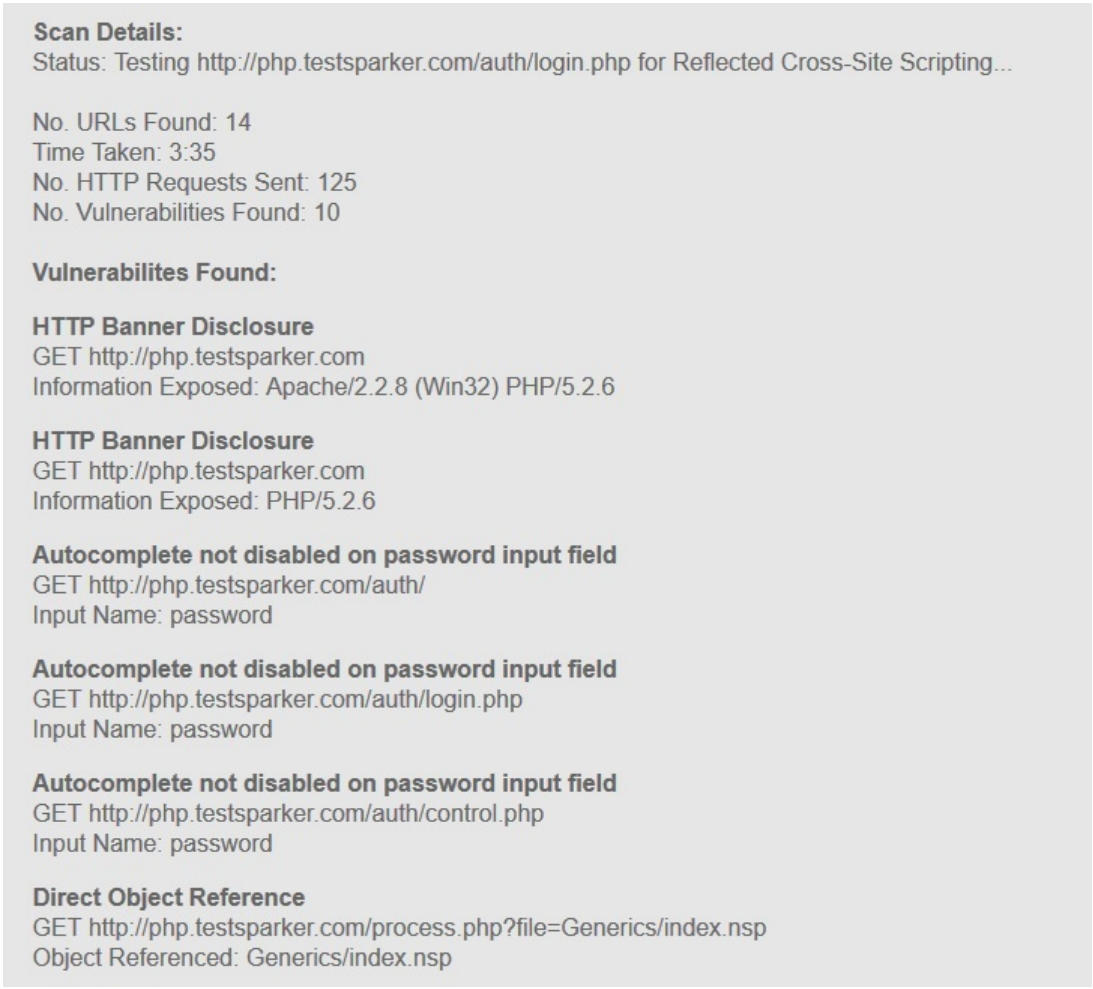

Figure 5.5: Web Application Vulnerability Scanner Screenshot 3

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Applications that contain forms that are highly complex and ones which aggressively check input values can completely block a PAS scanner because of wrong auto form fill data, disallowing it from crawling the pages that are hidden further in the web application. All relevant parts of the application cannot be crawled if the state of the web application is not exactly modelled and tracked by the scanner. More complex algorithms are required to perform deep crawling and maintain awareness of the state of the web application under test.

A scanner that does not provide most of the basic functional requirements of web application vulnerability scanners mentioned in this dissertation report, like protocol support, authentication, session management etc., are in reality useless when detecting vulnerabilities for real web applications that organizations host on the internet. In conclusion, in order for WAVS to be effetive, its user must have knowledge about the application that is begin scanned, and also the limitations of WAVS. Not just anyone can use WAVS only in Point and Shoot mode and hope to find all vulnerabilities within the application being tested.

6.2 Future Scope

The developed scanner is currently, not capable of authentication mechanisms like automated user creation and recording of log in procedure. This functionality needs to be added to WAVS in order for it even be categorized as an automated vulnerability scanning tool. The application developed for the dissertation is currently limited in its comprehension of the behaviour of applications with dynamic content such as JavaScript, Flash, etc. The tool does not implement many variants of attacks for a given vulnerability. More test cases can be added for predetermined attack vectors and functionality to allow user defined attack vectors to be used for testing. The predefined payload database of malformed values is currently severely lacking in numbers.

As web applications become more complex, application-specific logic vulnerabilities become more prevalent. More research is warranted to automate the detection of application logic vulnerabilities. Crawling can be made more effective and the methodology applied to scan for vulnerabilities can always evolve into something better, faster and more effiecient. Further research can be carried out by studying more web application vulnerabilities and developing ways to automate the detection of those vulnerabilities.

References

1. S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "SecuBat: A Web Vulnerability Scanner", Edinburgh, Scotland, WWW 2006, pages 247-256, May 23 - 26, 2006.

2. Acunetix Web Vulnerability Scanner, "http://www.acunetix.com/vulnerability-scanner", November, 2013

3. Subhash Chander, Ashwani Kush, "Vulnerabilities in web pages and web sites", International Journal of Advanced Research in IT and Engineering, Vol. 1, No. 2, August 2012

4. Owasp.org, OWASP Top 10 Project, available on the web at "http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project", May 10, 2014.

5. William G.J. Halfond, Jeremy Viegas, and Alessandro Orso, "A Classification of SQL Injection Attacks and Countermeasures", Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, USA, pages 13-15, 2006.

6. Jeremiah Grossman, Seth Fogie, "XSS Attacks: Cross-site scripting exploits and defense", Syngress, 2007.

7. OWASP A4 2013, Insecure Direct Object References, available on the web at "https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References", May 10, 2014.

8. Konstantin Kafer, "Cross Site Request Forgery", Hasso-Plattner-Institut, Potsdam, December 2008.

9. Nenad Jovanovic, Engin Kirda, and Christopher Kruegel, "Preventing Cross Site Request Forgery Attacks", Securecomm and Workshops, 2006, IEEE, 2006.

10. OWASP, Broken Authentication and Session Management, available on the web at "https://www.owasp.org/index.php/Broken_Authentication_and_Session_Management", May 10, 2014.

11. OWASP, Information Leakage, available on the web at "https://www.owasp.org/index.php/Information_Leakage", May 10, 2014.

12. OWASP A5 2013, Security Misconfiguration, available on the web at https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration", May 10, 2014.

13. Jeremiah Grossman, "Cross-Site Tracing (XST)", WhiteHat Security White Paper, 2003.

14. Alonso, Chema, Rodolfo Bordn, Marta Beltrn, and Antonio Guzmn, "LDAP Injection and Blind LDAP Injection.", 2008.

15. Amit Klein, "Blind XPath Injection", Whitepaper from Watchfire, 2005.

16. Adobe.com, Creating more secure SWF web applications, available on the web at, "http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html", May 10, 2014.

17. Craig A. Shue, Andrew J. Kalafut and Minaxi Gupta, "Exploitable Redirects on the Web: Identification, Prevalence, and Defense", WOOT ́08, USENIX, San Jose, July 28 - August 1, 2008.

18. WASC, Web Application Security Scanner Evaluation Criteria, v1.0, 2009, available on the web at "http://projects.webappsec.org/Web-Application-Security-Scanner-Evaluation-Criteria", May 10, 2014.

19. Adam Doupe, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna, "Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner", Securitý12, USENIX, August 2012.

20. Sean McAllister, Engin Kirda, and Christopher Kruegel, "Leveraging User Interactions for In-Depth Testing of Web Applications", Recent Advances in Intrusion Detection, pages 191-210. Springer Berlin Heidelberg, January 2008.

21. Ing. Pavol Luptak, CISSP, CEH, "Bypassing Web Application Firewalls", available on the web at "http://www.nethemba.com/bypassing-waf.pdf", May 10, 2014.

22. Adam Doupe, Marco Cova, and Giovanni Vigna, "Why Johnny Cant Pentest: An Analysis of Black-box Web Vulnerability Scanners", Detection of Intrusions and Malware, and Vulnerability Assessment, pages 111-131, 2010.

23. Open Web Application Security Project (OWASP): OWASP WebGoat Project available on the web at "http://www.owasp.org/index.php/Category:OWASP WebGoat Project",

    May 10, 2014.

24. WackoPicko scanner testing application, "https://github.com/adamdoupe/WackoPicko", May 10, 2014.

25. Bhanu Pratap Pradhan, Ayushman Tiwari, Dr. Anurika Vaish (Advisor), "Analyzing Performance of Open Source Web Application Security Scanners", ISSA Journal, Pages 33-39, May 2013.

26. Jason Bau, Elie Bursztein, Divij Gupta and John Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing", IEEE Symposium on Security and Privacy, 2010