International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 1, ISSN No. 2455-2143, Pages 132-136
Published Online May 2019 in IJEAST (http://www.ijeast.com)

AUTOMATED WEB VULNERABILITY SCANNER

Rahul Maini
Department of Computer Engineering
BVDUCOEP, Pune,
Maharashtra, INDIA

Rahul Pandey
Department of Computer Engineering
BVDUCOEP, Pune,
Maharashtra, INDIA

Rajeev Kumar
Department of Computer Engineering
BVDUCOEP, Pune,
Maharashtra, INDIA

Rajat Gupta
Department of Computer Engineering
BVDUCOEP, Pune,
Maharashtra, INDIA

***Abstract*— In this era, when the time and internet has evolved, the web application threats have increased by ten folds. The cause of the web vulnerabilities are still due to the lack of input validation. This causes the CIA (Confidentiality Integrity and Availability) Triad Model to break. To solve this, we develop a scanner for finding common vulnerabilities in web applications including SQL Injection, Cross-Site-Scripting (XSS), CRLF Injection, and Open Redirect. It also include a simple port scanner along with a web crawler module which helps to identify other services which may be running on the web server. In this paper, we introduce a simple black-box security test technique for finding these issues. At the end of the paper, we demonstrate how easy it is to scan a complex enterprise-grade web application with our scanner. The main goal of the scanner is to uncover the vulnerabilities and produce a better result/report of each web application in effective manner.**

Keywords— **SQL Injection, XSS, CRLF Injection, Open Redirect, Web application vulnerability, Port Scanner, Web Crawler, Web Scanner tool**

I. INTRODUCTION

Web Applications are continuously emerging and largely prevalent critical piece of our daily lives. The Web technology stack, the languages, frameworks etc. have improved a lot. However, the security of web application is still basic proposed by Marcus Pinto et al [1]. Security is never thought of while developing an application. The term vulnerability is a weakness or a shortcoming in a piece of software that allows a threat actor to harm or destroy the respectability of a system. The Most Common Vulnerabilities present in Web Applications as of 2019 are: Cross Site Scripting (XSS), SQL (Structured Query Language) Injection, Carriage Return Line Feed (CRLF) Injection, Open Redirects and others. Even after a noteworthy period of existence, these vulnerabilities still do not cease to exist. Exploiting these vulnerabilities are also very simple and easy for a threat actor. Identifying vulnerabilities for the most part is not an easy task, and not many basic vulnerabilities can be effectively identified via automated scanners proposed by V. Suhina et al [7].

Most of the software bugs in web application are a result from an invalid input sanitization proposed by David Shelly et al [2][4]. These vulnerabilities may be SQL injection, Cross-Site Scripting, Carriage Return Line Feed Injection, Open Redirect. Although the dominant part of web vulnerabilities are straight-forward and to maintain a calculated distance from, numerous web designers are, shockingly, not security-mindful. Consequently, there exist an expansive number of vulnerable applications as well as websites on World Wide Web. Most important ways to deal with testing programming applications for the Web Applications are static (white-box) and dynamic (black box) as well as gray-box approach.

In Black Box based Security testing, only high-level of information is made available to testers such as URL or address of the organization to perform penetration testing. Here, tester may see himself as a hacker who is unaware of the system/network. Black box testing is a time consuming approach as the tester is not cognizable of system/network's attributes and he/she will need considerable amount of time to explore system's properties and details. Further, this approach of testing may result into missing out of some areas, keeping in view limited time period and information. proposed by University of Zagreb et al [8]

At the end, we present that how the port scanner finds the services running on the server and open ports, how the web crawler module work into the identification of endpoints and paths or how the scanner implements to sidestep the verification of web application and recognize web application helplessness existed in them by re-enacting web assaulting and investigating the information of reaction. In the scanner module, we send specially crafted payloads for identifying vulnerable web

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 1, ISSN No. 2455-2143, Pages 132-136
Published Online May 2019 in IJEAST (http://www.ijeast.com)

application for each supported vulnerability class and based on the response given by the web server, we identify if the web application is vulnerable to a specific attack or not i.e. SQL Injection, XSS, CRLF Injection or Open Redirect proposed by Antunes et al.[9]

II. VARIOUS WEB VULNERABILITIES

2.1 SQL Injection Attack

SQL-injection (SQLI) attacks works by injecting strings into SQL queries that change their planned use. This can happen if different website doesn't use proper user input proposed by Acunetix et al [3]. The attacker is able to manipulate the query if the application is vulnerable to SQLI, the attacker could extract information from the database, Insert, Update, Delete or modify the information stored across the database, tables and columns.

SELECT * from tbl_login WHERE email='xyz@xzy.com' AND password= '12345secret';

This kind of SQL query is mostly used to verify user authentication. So attacker mostly targets this type of SQL query. If we consider query, it will check if the email and password matches and returns the matched rows, if the rows are returned then user is considered to be authenticated. Now suppose client enter email and password in to input field, query look like following.

SQL-Query = "SELECT * from tbl_login WHERE email=''+email+'' AND password= ''+password+''"

In web application code if developer does filter the user input then attacker can inject some SQL queries which might alter its meaning in case of executing SQL query. For example: anyone can insert email and password like following.

Email: **' OR 1=1 --**
Password : <empty>

Using the provided form data, the vulnerable web application constructs a dynamic SQL query for authenticating the user as shown in

SELECT * from tbl_login WHERE email='' OR 1=1 -- ' AND password= '';

In SQL-query, the single quote(') is used to break the query and OR 1=1 will make the running query TRUE and '-- ' characters at end are SQL comments which causes the rest of the query to be ignored. So whenever database engine executing this type of query it returns all user data, means its valid login for that email.

2.2 Cross-Site-Scripting (XSS) Attack

XSS or Cross-Site-Scripting is a Client-Side Security Vulnerability which affects a victim's browser. An attacker can inject client-side script in input fields of web application. If the attacker is able to input HTML tags and they are reflected as is in the output then attacker could inject JavaScript in the victim's browser by sending the vulnerable URL to the Victim.

Cross-Site-Scripting is of three types:

1. Reflected XSS
2. Stored XSS
3. DOM based XSS

The most widely recognized attack in web application is Reflected XSS.

Reflected XSS: The web server reflects the user input directly into the web server response without sanitizing the user input. eg. In a search field of a web application

Stored XSS: In this case, the Invalidated User Input is permanently stored into the Database of the web application and it's reflected un-sanitized from the database in the response. The attack surface of this type of XSS is more than reflected XSS. e.g. XSS in posts or comments sections of a web application.

DOM Based XSS: This happens when the user input (source) is extracted via JavaScript and put into the DOM of the web page via some dangerous HTML sinks. e.g. XSS via URI fragment (after '#' in the URL)

For example if any client enter string "<B>TEST</B>".So whenever client search with these HTML tags, result might come from server that "no matches found for **TEST**" (here string '**TEST**' is shown in bold letters). This means since we are able to inject HTML tags. We could inject **<script>** tags and execute arbitrary JavaScript in the Victim's browsers which allows us to steal the Victim user's Cookies or make requests on his behalf without him knowing about it and hence steal his session.

Example:
<script>location.href='http://attackerserver.com/?'+document.cookie;</script> Injecting this string would send the victim's Cookies to attacker controlled server which attacker could reuse again to authenticate to victim's account without his username or password

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 1, ISSN No. 2455-2143, Pages 132-136
Published Online May 2019 in IJEAST (http://www.ijeast.com)

2.3 CRLF Injection Attack

When a browser sends a request to a web server, the web server answers back with a response containing both the HTTP headers and the actual website content. The HTTP headers and the HTML response (the website content) are separated by a specific combination of special characters, namely a carriage return and a line feed. For short they are also known as CRLF.
The server knows when a new header begins and another one ends with CRLF, which can also tell a web application or user that a new line begins in a file or in a text block.

Since the header of a HTTP response and its body are separated by CRLF characters an attacker can try to inject those. A combination of **<CRLF><CRLF>** will tell the browser that the header ends and the body begins. That means that he is now able to write data inside the response body where the html code is stored. This can lead to a Cross-site Scripting vulnerability.

An example of HTTP Response Splitting leading to XSS:
Imagine an application that sets a custom header, for example:

X-Language: **en_US**

The value of the header is set via a get parameter called "lang". If no URL encoding is in place and the value is directly reflected inside the header it might be possible for an attacker to insert the above mentioned combination of <CRLF><CRLF> to tell the browser that the request body begins.

That way he is able to insert data such as XSS payload, for example:.

?lang=en_US%0d%0a%0d%0a<script>alert(document.domain)</script>

The above will display an alert window in the context of the attacked domain. Moving on, JavaScript could send the victim's Cookies to attacker controlled server which attacker could reuse again to authenticate to victim's account without his username or password

2.4 Open Redirect Attack

One of the most common and largely overlooked vulnerabilities by web developers is Open Redirect (also known as "Unvalidated Redirects and Forwards"). A website is vulnerable to Open Redirect when parameter values (the portion of URL after "?") in an HTTP GET request allow for information that will redirect a user to a new website without any validation of the target of redirect.

An example of a vulnerable website link could look something like this:
https://www.example.com/login.html?**RelayState**=
http%3A%2F%2Fexample.com%2Fnext

In this example, "RelayState" parameter indicates where to send user upon successful login (In our example it is "http://example.com/next"). If website doesn't validate the "RelayState" parameter value to make sure that target web page is legitimate and intended, attacker could manipulate that parameter to send a victim to a fake page crafted by attacker:
https://www.example.com/login.html?**RelayState**=
http%3A%2F%2Fattacker.com

Open Redirect vulnerabilities don't get enough attention from developers because they don't directly damage website and do not allow an attacker to directly steal data that belong to the company. However, that doesn't mean that Open Redirect attacks are not a threat. One of the main uses for this vulnerability is to make phishing attacks more credible and effective.

When an Open Redirect is used in a phishing attack, the victim receives an email that looks legitimate with a link that points to a correct and expected domain. What the victim may not notice, is that in a middle of a long URL there are parameters that manipulate and change where the link will take them. To make identification of the Open Redirect even more difficult, redirection could take place after victim provides login on a legitimate website first. Attackers have found that an effective way to trick a victim is to redirect him to a fake website after they enter their credentials on a legitimate page. The fake website would look identical to a legitimate website, and it would ask the victim to re-enter their password. After the victim re-enters their password it would be recorded by the attacker and victim would be redirected back to a valid website. If done correctly, victim would think that he mistyped password once and would not notice that his username and password were stolen.

III. METHODS TO IDENTIFY

3.1 SQL Injection (SQLI)

In this category of Scanner module we try to identify SQL Injection attacks using some basic SQL Injection Payloads which helps us to check whether the application responds to the SQL queries or not i.e. if the application is vulnerable to

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 4, Issue 1, ISSN No. 2455-2143, Pages 132-136
Published Online May 2019 in IJEAST (http://www.ijeast.com)

SQLI or not proposed by Katkar Anjali S and Kulkarni Raj et al [5]

BOOLEAN BASED SQL INJECTION

In this type of SQL Injection we try to detect SQLI on the basis of the response returned from the server. Our scanner detects a difference between a TRUE and a FALSE response and based on that we find if SQL Injection is present or not.

Some Boolean Based SQLI Payloads:

- **999999 or 1=1 or 1=1**
- **' or 1=1 or '1'='1**
- **" or 1=1 or "1"="1**
- **999999) or 1=1 or (1=1**
- **') or 1=1 or ('1'='1**
- **") or 1=1 or ("1"="1**
- **999999)) or 1=1 or ((1=1**
- **')) or 1=1 or (('1'='1**
- **")) or 1=1 or (("1"="1**
- **999999))) or 1=1 or (((1**
- **'))) or 1=1 or (((1'='1**
- **"))) or 1=1 or (("1"="1**

TIME BASED SQL INJECTION

In this type of SQL Injection we try to detect SQLI on the basis of the time delays in the response returned from the server. Our scanner detects a difference between a TRUE and a FALSE response based on the time delays introduced by successful evaluation of sleep( ) or similar functions depending on the DBMS.

Some Time Based Blind SQLI Payloads:

- **999999 or sleep(10) or 1=1**
- **' or sleep(10) or '1'='1**
- **" or sleep(10) or "1"="1**
- **999999) or sleep(10) or (1=1**
- **') or sleep(10) or ('1'='1**
- **") or sleep(10) or ("1"="1**
- **999999)) or sleep(10) or ((1=1**
- **')) or sleep(10) or (('1'='1**
- **")) or sleep(10) or (("1"="1**
- **999999))) or sleep(10) or (((1**

Step 1. Parse the HTML Response of the URL to scan and extract all 'name' attributes from the 'input' tags

Step 2. Try fuzzing all the parameters with the SQL Injection payloads and check if the server responded correctly given the SQL query based on the HTTP Response or the Response Time.

3.2 Cross-Site-Scripting (XSS)

In this submodule of Scanner, we are trying to detect HTML Injections and XSS vulnerabilities using some XSS payloads. The basic approach used to identify this class of vulnerability is, we insert HTML Tags into every possible input fields in the page as well as query string parameters(if any) and try to see if the Tags are reflected as is, in the HTTP Response. If the tags are reflected Invalidated we could confirm that application is vulnerable to XSS explained by Jeremiah Grossman et al [6].

Step 1. Parse the HTML Response of the URL to scan and extract all 'name' attributes from the 'input' tags

Step 2. Try fuzzing all the parameters we extracted with HTML Tags and XSS Payloads.

Step 3. If the response contained un-sanitized HTML Tags or our XSS payloads then the application is vulnerable to XSS.

3.3 CRLF Injection

In this part, we try to find if the user input is reflected inside the HTTP Response Headers and if we are able to insert un-encoded sequence of Carriage Return and Linefeed i.e. \r\n or %0D%0A which allows us to inject arbitrary HTTP Response Headers and by injecting 2 sequence of <CR><LF> we are able to inject directly into their response.

Step 1. Parse the HTML Response of the URL to scan and extract all 'name' attributes from the 'input' tags

Step 2. Try fuzzing all the parameters we extracted with <CR><LF> sequence

Step 3. If the HTTP Response headers contains the un-encoded our new header that we injected via CRLF then the application is vulnerable to CRLF

3.4 Open or Invalidated Redirect

In this sub-module, we try to fuzz parameters by sending a URL to a website and check if there is a 3XX HTTP response code and compare the **Location** HTTP Response Header with the parameter we sent.

Step 1. Parse the HTML Response of the URL to scan and extract all 'name' attributes from the 'input' tags which have a HTTP GET Method

Step 2. Try fuzzing all the parameters we extracted with a URL and check if there is a HTTP Redirect in the Response.

IV. CONCLUSION

The Major contribution to this Web Scanner is to demonstrate the ease and the simplicity of

identifying common security vulnerabilities in web applications. The tool has been built in such a way that it can be easily upgraded to add many more functionalities. In this regard, following is the summary of some of the future works that can be done for enhancing the tool. In this regard, following is the summary of some of the future works that can be done for enhancing the tool.

1. Apart from detecting the current vulnerabilities, more modules can further be added for the detection of other major vulnerabilities such as File Inclusion, XXE, Insecure Deserialization, Buffer overflows, OS command injections etc.

2. Modules for penetration testing of these vulnerabilities can also be added to make it more powerful.

3. The algorithms and techniques currently used can be modified or replaced to more advanced and efficient ones for better accuracy of results. 4. Port scanner can be made threaded to enhance the speed and efficiency of the scan.

V. REFERENCE

1. Dafydd Stuttard , Marcus Pinto "The Web application Hacker‟s Handbook Finding an Exploiting Security Flaws " second edition ©2011
2. David Shelly, Randy Marchany, Joseph Tront "Closing the Gap: Analyzing the Limitations of Web Application Vulnerability Scanners" Virginia Polytechnic Institute and State University
3. Acunetix Ltd. Acunetix Web Vulnerability Scanner. http://www.acunetix.com/, 2005.
4. David Shelly, Randy Marchany, Joseph Tront "Closing the Gap: Analyzing the Limitations.
5. Katkar Anjali S and Kulkarni Raj B, "Web Vulnerability Detection and Security Mechanism", International Journal of Soft Computing and Engineering(IJSCE),ISSN: 2231-2307, Volume-2, Issue-4, p.-237-241
6. Jeremiah Grossman WhiteHat Security founder & CTO "Website Vulnerabilities Revealed " WhiteHat Security Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad " SecuBat: A Web Vulnerability Scanner" Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic Secure Systems Lab, Technical University of Vienna.Nuno Antunes and Marco Vieira , "Defending against Web Application Vulnerabilities",
7. V. Suhina, S. Groš and Z. Kalafatić, " Detecting vulnerabilities in Web applications by clustering