Security Tool for Evaluation of Web Applications for Vulnerabilities as listed by OWASP

[1]Vankadara Veera Harshith, [2]Priyadharshini, [3]Tejas R M, [4]Tatireddy Goutham, [5]V S K Anudeep Reddy

Authors affiliation, Department of Computer Science and Engineering, REVA University

Abstract.

Web Application developers in their hurry to deliver the applications quickly could be overlooking the various security vulnerabilities in their application. OWASP has listed vulnerabilities of the web applications This paper has tried to identify possible security breaches in the web applications by developing a comprehensive tool that can detect the infirmities in the web application. The tool is built using python which detects vulnerabilities in the web applications named Quick Scanner. It can detect five vulnerabilities SQL Injection, Cross site scripting, Open redirection, Vulnerable default pages and Local file inclusion vulnerability. This security breaches may lead to leak of sensitive data like usernames, passwords, personal identifiable information, social security number, credit card details, health information etc. This tool takes very less time to scan and generate report on the web application.

Keywords. Vulnerability scanner, SQL Injection, Cross site scripting, local file inclusion, default vulnerable pages, open redirection.

1. INTRODUCTION

Web services are used for increasing various kind of services available from different websites and web-applications which can be accessed from range of devices starting from desktops and laptops to tablets and smartphones. And not to mention about the technological development that we have witnessed in the past few years is quite immense.

According to a survey, there were about 23 million web developers in the US as of the year 2018. During the Covid pandemic, it was observed that there were higher search queries in google regarding how to create or design a web application. Though the applications are developed by individual or web developers, these applications are not designed to take care of various security threats that may be incurred on the data and systems during various stages of the network transaction

A report in 2019, found that security breaches had increased by 67% over the last five years. Similarly, a survey as of 2020, the average cost a data breach is $3.86 million and the average time to identify a breach was 207 days (sources from IBM) and we might be thinking or assuming that our systems are secured and defended by firewalls and antivirus software's. But the fact is that 73% of black hat hackers have stated that the traditional firewalls and antivirus security systems are irrelevant or obsolete.

Many new variants of vulnerabilities are discovered every day, and OWASP (Open Web Application Security Project) lists top 10 vulnerabilities every 3-4 years, just to mention a few vulnerabilities like Cross-Site Scripting (XSS), Injection (such as SQL, NoSQL, OS, and LDAP), Broken Authentication, Local File Inclusion [LFI], etc. Measures to protect web application from these attacks. But to build the web application that is robust, reliable in spite of hostile web environment there is a need to carry out or perform penetration testing and find out vulnerabilities. It is a useful tool for web developer as well as the consumer of services of the web application. There are few tools that could be put to service but are costly and may not be available to all developers. They are a few that consume lot of memory resources, while some are manual or automated. With the wide range and ease of usage of libraries in python, we have designed this automated tool that is optimized in terms of memory resources consumed and execution time.

2. AUTOMATED AND MANUAL TESTING

Manual testing is testing of software where the test is carried out by professionals manually. It consumes more time compared to automatic testing. Efficacy of manual testing is more compared to automatic, as the scope of the automatic testing is limited within the vulnerabilities defined by the developer. In this process, the tester carries out different test cases and generate report manually without the help of any automatic devices. Manual testing is preferred in case of more complex applications.

Automatic testing is testing of software where the test is carried out automatically by automatic tools. The main goal of automated testing is to reduce the time consumption. The tool performs different test cases and generates report automatically. Automated testing is preferred in case of simple applications.

3. METHODOLOGY

The Quick Scanner tool developed will accept URL and cookie as input. The tool checks the system has stable internet connection and verifies if the URL and cookie entered is valid. Scanner will collect all the information about the target like server type, connection, and content type. It will crawl over the target site and find all the links linked to the target so that vulnerability testing can be performed on web sites linked to target. Later the target is further processed for vulnerability detection and user can choose specific vulnerability to perform the vulnerability detection and generate report. As shown in Figure 1. The tool is implemented using different module that is Scanner module, Attack module which contains vulnerabilities and Report generation.

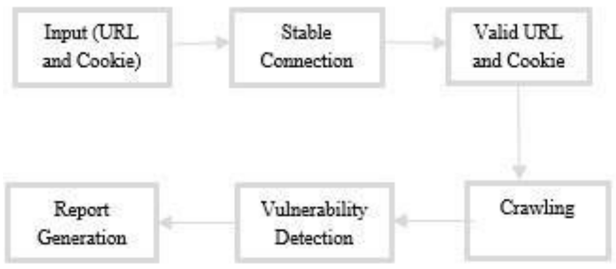


FIGURE 1. Implementation Details

3.1. Scanner

In the scanning process Quick scanner will accept URL and cookie as input. Checks if the system has stable internet connection and verifies if the URL and cookie entered is valid or not. Scanner will find all the information about the target like server type, connection, and content type. It will crawl over the target site and find all the links linked to the target so that vulnerability testing can be performed on sites linked to target. Later the target is further processed for vulnerability detection and user can choose specific vulnerability to perform the vulnerability detection.

3.2. SQL Injection

SQL Injection is also known as 'SQLI', this is one of the top 10 OWASP vulnerabilities. This uses malicious SQL to access the sensitive data like passwords, usernames, etc. which are stored in the database and not supposed to be displayed in frontend.

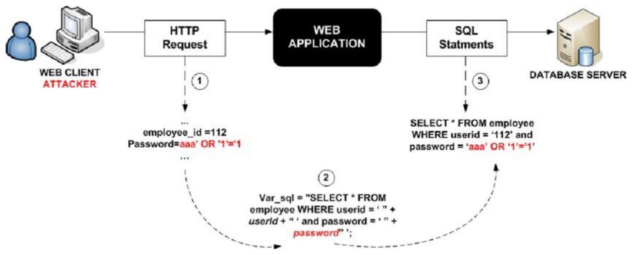

WEB CLIENT
ATTACKER
HTTP Request
WEB APPLICATION
SQL Statments
DATABASE SERVER
1
... employee_id =112 Password=aaa' OR '1'='1 ...
2
Var_sql = "SELECT * FROM employee WHERE userid = ' " + userId + " ' and password = ' " + password" ';
3
SELECT * FROM employee WHERE userid = '112' and password = 'aaa' OR '1'='1'

Figure.2 Diagrammatic Representation of SQL Injection

3.2.1. SQL Injection Problems

The main cause for SQLI is many web application developers do not apply validation for inputs and they are not aware about consequences that may arise due to that. So, this is nothing but welcoming the attackers to access the sensitive data from the database by executing malicious SQL commands. The most advantage to attacker here is easy to implement and disadvantage to developer is difficult to avoid like these injections.

3.2.2. SQL Injection Scanner

Here we define “scan_ sql_injection ()” for detecting SQL injection. Here we give the respective URL and dictionary of cookies as input. First URL will be parsed for getting query, then by using requests. Get () function we will get the result, then by using function “is vulnerable ()” whether vulnerability present or not, this function checks for errors like ‘errors in SQL syntax’, ’warning: MySQL’, ‘unclosed quotation mark after the character string’, ‘quoted string not properly terminated’, 'Error: You have an error in your SQL syntax. In this step we check any vulnerability is present in the URL or not.

After URL is verified, we will fetch all the forms from the URLs by using “get_all_forms ()”. Now we will find the details such as form method, input type, name, buttons, text area and option of each form. Now again as mentioned above by using “is vulnerable ()” we will check vulnerability in all the web pages.

3.2.3. SQL Injection Prevention

- Authenticate user inputs
- Restrict special characters from input fields
- Enforce prepared statements and parameterization
- Use stored procedures in the database
- Raise Virtual or Physical firewalls
- Use whitelists, not blacklists
- Establish appropriate privileges and strict access

3.3. Cross Site Scripting (XSS)

Cross site scripting is one of the top 10 OWASP vulnerability found in web applications. Cross site scripting will allow users to inject scripts from client side into web pages which are viewed by the other users. The users will trust the script and execute the script. The script sent by the attacker can access any cookies, sensitive data, and session cookies of the used user on that particular site.

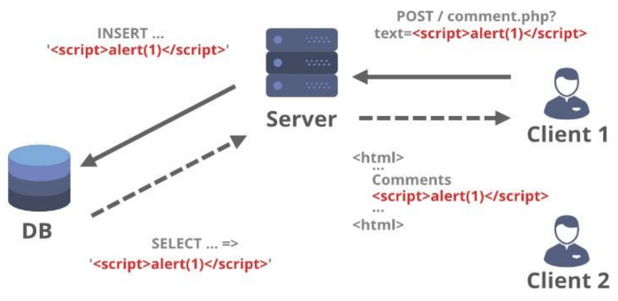

INSERT ...
'<script>alert(1)</script>'
Server
POST / comment.php?
text=<script>alert(1)</script>
Client 1
DB
<html>
...
Comments
<script>alert(1)</script>
...
<html>
SELECT ... =>
'<script>alert(1)</script>'
Client 2

Figure.3 Diagrammatic Representation of Cross Site Scripting

3.3.1. Cross site scripting problems

Cross site scripting mainly occurs at the arrival of input, in response headers and in content security policy. The proper input validation as to be done and filter the input based on the requirement. The output data should be encoded to prevent active interactions of the attackers. We should use Content-Type and X-Content-Type-Options headers to prevent XSS in headers. And can use Content security policy as the last line security in web applications.

3.3.2. Cross site scripting scanner

Here we have defined “scan_xss()’’ function which takes header and URL has the parameter. The function checks if cross site scripting vulnerability exists in that URL. First, we get all the forms used in the website using the function “get_all_forms()’’ and pass header and URL as parameter. In this function we get forms used on the website using beautiful soup library in python and get all the input details like type and name. And store the details of action, method, and inputs separately in a list. The list of payloads will be made based on JavaScript that can be used for the cross-site scripting exploitation. We read each line from the payload file and submit that payload in the input of the form using the “submit form ()’’ function. In the “submit form ()’’ function we pass “form details ()’’, URL, payload, and headers as parameters. Based on the method of form (post or get) we send the request to the URL using the payload sent to the form and store the response from the server. Based on the response we define if vulnerability exists in that URL. Similarly same process is continued for all the payloads present in the list. If vulnerability exists, the response would be 200 OK and the pop-up message would be displayed as mentioned in the script else the response will be 404 error which implies there is no XSS vulnerability.

3.3.3. Cross site scripting prevention

- Filter inputs on arrival.
- HTML encode before placing Untrusted data into HTML element content.
- Use suitable response headers.
- Use HTTP Only cookie flag.
- Evaluate content security policy

3.4. Open Redirection

Open redirection vulnerability is found in web applications which incorporates user data into the application about the user. An attacker can make users redirect to external domain by developing a URL within the web application. And hence the attacker gets the sensitive data about the user like email, password, and other personal information. So, we should not allow the user to the redirect page of the application.

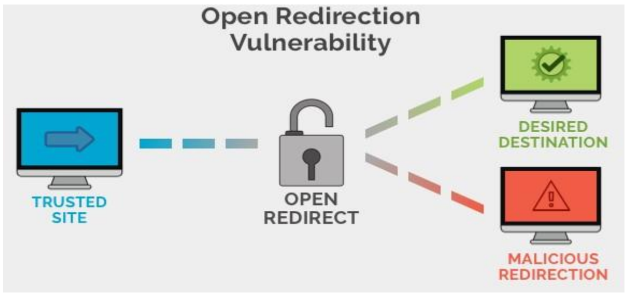


Figure.4 Diagrammatic Representation of Open Redirection

3.4.1. *Open redirection Problem*

Open redirection vulnerability exists when the application redirects the user entered data form to the other page. We should use an ID internally in the code to the respective URL so that page does not redirect to the other URL or domain specified by the attacker. The user should validate whether the URL starts with http:// or https// and invalidate all other URLs other than http:// and https://. The other method is to use the server-side URLs to list them all and permit only listed URLs to redirect.

3.4.2. *Open Redirection Scanner*

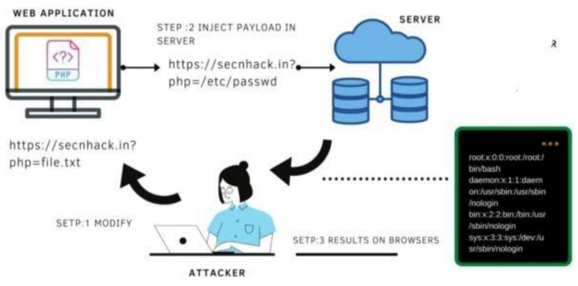
Here we have defined "scan ()'' function which takes URL and header(cookie) as parameter. We will also have a list of payloads which contains the redirection extension for the URL. From the payload file we read a single line of payload and add the payload to the URL. Later we send request for each URL with different payload and store the response. If the response contains 404 status code (i.e., Page not found) which implies the page does not redirect to the other page for that payload and hence we check for the vulnerability for other URLs. If response contains the status code 301(Permanently redirect status response) or 302(the resource requested has been temporarily redirected to the URL) which implies that the application is redirecting to the other page without user's knowledge. This states that the web application as the Open redirection vulnerability. We store the data about the URL and payload for which redirection exits and report it in the final step for the developer about their web application.

3.4.3. *Open Redirection Prevention*

- Block offsite redirects.
- Validate the referrer when doing redirects.
- Do not allow the URL as user input for the destination
- Filter input by creating a list of trusted URLs (list of hosts or a regex).
- Force all redirects to first go through a page notifying users.

3.5. Local File Inclusion Vulnerability

Local file inclusion is one of the top OWASP vulnerability found in web applications. It is a technique where attackers make users to run or expose files on a web server. It leads to exposure of sensitive data and in some cases, it leads to cross-site scripting vulnerability. It mainly occurs when the application takes any type of files or images as input.


WEB APPLICATION
STEP :2 INJECT PAYLOAD IN SERVER
https://secnhack.in?php=/etc/passwd
SERVER
https://secnhack.in?php=file.txt
SETP:1 MODIFY
SETP:3 RESULTS ON BROWSERS
ATTACKER

```
root:x:0:0:root:/root:/
bin/bash
daemon:x:1:1:daem
on:/usr/sbin:/usr/sbin
/nologin
bin:x:2:2:bin:/bin:/usr
/sbin/nologin
sys:x:3:3:sys:/dev:/u
sr/sbin/nologin
```

Figure.5 Diagrammatic Representation of Local File Inclusion Vulnerability

3.5.1. Local file inclusion problem

Local file inclusion vulnerability occurs when the application takes files or images as input. This provides the attacker to inject their own scripted code into the application and exploit it. The web application developer should use IDs to save their file paths in the secure database and use ID for different files. Can also use whitelisting method where the only verified and secured files should be used, and the remaining files must be rejected. The file upload must not be uploaded on the web server instead a separate database must be used to store the files uploaded by the user. The paths passed should be properly sanitized. This vulnerability will also lead to exposure of passwords if stored locally in the application.

3.5.2. Local file Inclusion Scanner

Here we have defined "main ()'' function to check for local file inclusion scanner and pass URL and header (Cookie) as the parameter. First, we check for the PHP filter using function "test_php_filter ()'' passing same parameter to this function. Based on the test cases for the PHP filter we assign the payload for the URL and send the request to that address using the request function and store the response we get. Then we filter the response using "checkPayload ()'' function and if we get any Keyword as defined in the list then return true. Hence it states that local file inclusion vulnerability exists in that application. Next, we check for the PHP data only if there is no vulnerability in PHP filter. In PHP data we call the function "php_data ()'' function by passing URL and header as parameter. Like "php_filter ()'' case we define a payload that is related to the PHP data and add the payload to the URL and send the requests. Then we check response using the "checkPayload ()'' function and state if the application as local file inclusion vulnerability or not. Then check for the PHP except where we pass same parameters and like the PHP data and filter functions, we again add payload, send request and check the response. If all the three cases fail, then we check for the wordlist using the function "test wordlist ()''.

Same process is used in this function as above three functions and response are stored. If all the four cases fail which implies there is no local file inclusion vulnerability in that application.

3.5.3. Local file Inclusion Prevention

- Never trust user input.
- Use a whitelist of allowed file names and locations.
- Make sure that none of the file can be replaced by the attacker using file upload functions.
- Do not include file on a web server that can be compromised, use a separate database instead.

3.6. Vulnerable Default Pages Vulnerability

Vulnerable default pages [12] are a security misconfiguration vulnerability. This attack is found in web applications with default settings and pages which leads to misconfiguration. Consequences of this vulnerability is it leads to sensitive data disclosure and the attacker may even change the user's data in the application.

3.6.1. Vulnerable default pages problem

The important way to prevent vulnerable default page vulnerability is to disable the use of default accounts, passwords, and pages in the applications. The application must be scanned regularly to detect any misconfiguration is present in it. The developer must also disable the administrative interfaces and debugging option in the application. The server configuration must be done to prevent unauthorized access and directory listings.

3.6.2. Vulnerable default pages scanner

Here we have defined vulnerable pages function to detect the vulnerability. The function takes URL and headers as arguments. We have a list of payloads and add the payload to URL and send it to test function. Later we send the request of the URL with payload and check the response. If the status code is 200 which means the default page vulnerability is present else there is no default vulnerable page vulnerability in the application.

3.6.3. Vulnerable default pages Prevention

- Keep the software up to date.
- Disable all the default accounts and change passwords regularly.
- Develop strong app architecture and encrypt data which has sensitive information.
- Make sure that the security settings in the framework and libraries are set to secured values.
- Perform regular audits and run tools to identify the holes in the system.

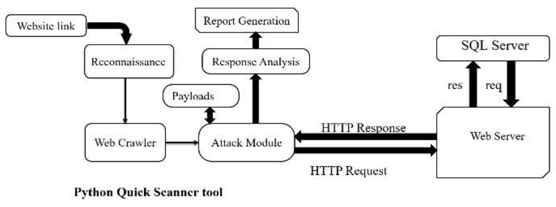


Figure.6 Low Level Design of the Quick scanner tool

4. MODULES IMPLEMENTED

1. Vulnerability Module - This module contains the code of for probing all short coming of the website. Code for testing of vulnerabilities like SQL injection, cross site scripting, open redirection, local file inclusion and default vulnerabilities.
2. Payload Module - It contains the list of payloads required for detecting vulnerability. Example for cross site scripting it contains all scripts for checking if there is any vulnerability.
3. Main Module - It provides user interface; it takes URL and cookie as input and proceed for the vulnerability detection and finally produce report based on the vulnerabilities detected.
4. Python Library - It contains the list of python modules required for the execution of the scanner.
5. Report Generation – Module stores all the data obtained during detection process and at last it generates the report about each vulnerability.

5. EXPERIMENTAL RESULTS

The Quick Scanner tool prompts to generate report after scanning. The report is generated in pdf format. It contains four main tables in the report (Figure 14 and Figure 15) as follows:

1. Summary of Alerts - This table contains two columns 'Risk Level' and 'Number of Vulnerabilities'. Risk level indicates how many vulnerabilities are present and how much risk they are high, medium, low and information to the website. Number of vulnerabilities contains count of them with respect to the risk level. As shown in figure 14.
2. Vulnerabilities Count - This table contains two columns 'Vulnerability Name' and 'Count'.
   Vulnerability name contains all the vulnerabilities like SQL injection, Cross site injection, Local File inclusion etc which are scanned by QS tool. Count contains the number of vulnerabilities present in a website with respect to the vulnerability name. As shown in figure 14.
3. Server Information - This table contains the server information connection type

and content type related to that website and target link will be highlighted here. As shown in figure 15.

4. More Information - This table contains links to the target web pages, local photos and internet photos identified inside the target. As shown in figure 15.

After all these details the report contains alert details, here tables will be differentiated with three different colours red, orange, and yellow. Red indicates that vulnerability's level of risk is high, orange indicates that vulnerability's level of risk is medium and yellow indicates that vulnerability's level of risk is low.

Here each table contains description related to that vulnerability, URL of the particular web page, method, attack parameters, some prevention methods which should be considered by the web developers to protect their website from particular vulnerability and finally the table contains references links through which developers can get some information related to that particular vulnerability as follows.

1. Target URLs – This page contains various web pages that are found while spidering and crawling.
2. Local Images - This page contains links of the local images found inside the target URL.
3. Internet Images – This page contains links of the internet images found inside the target URL.

6. Comparative Study

This module is about the differences between proposed tool and existing tools. The main advantage of our tool is less time complexity, less space complexity and consumes very less RAM for execution. When we compare our tool with Nessus, it takes more time, consumes lot of RAM and also more data is stored. Wapiti is the vulnerability scanner which scans only the URL specified but our tool will scan for all web-links or URLs linked to given target.

For comparative study purposes we developed a website and hosted in the localhost as well as containerized and hosted in docker to perform isolated or sandbox testing. We tested the website developed for vulnerability scanning with Nessus tool and with our tool Quick Scanner. Both tools detected vulnerabilities in the websites, but Nessus took more time compared to Quick Scanner.

7. Conclusion

Security issues related to web-application is evolving always. Building and deploying web application seems to be easy, but web application needs to be developed with lot considerations to the security issues involving different stake holders. Whereas major responsibility lies in maintaining and securing the application from being attacked and losing user information or data. In secure web application may be security risk for the host as well as the consumer of its services due to loss of personal data. Every day new technologies pose new challenges and risk and the tools to check the vulnerabilities need to be constantly upgraded.

8. FUTURE WORK

This study can further be enhanced and pave ways for future development through these possible ways:

- Could use information gathering techniques to gather information from the web about the latest threats and vulnerabilities, which could be useful for developers.
- Build or design a compiler not just to find and debug errors, but also to warn a developer or user, signs of weak practices or loose ends, to safeguard while developing an application.
- Work upon scanning for vulnerabilities, other than the ones which are defined in this study work

9. REFERENCES

[1] N. Anantharaman and B. Wukkadada, "Identifying the Usage of Known Vulnerabilities Components Based on OWASP A9," 2020 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2020, pp. 88-91, doi:10.1109/ESCI48226.2020.9167645.

[2] M. Buchler, J. Oudinet and A. Pretschner, "Spa Cite -- Web Application Testing Engine," 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, Montreal, QC, Canada, 2012, pp. 858859, doi: 10.1109/ICST.2012.187.

[3] Y. Khera, D. Kumar, Sujay, and N. Garg, "Analysis and Impact of Vulnerability Assessment and Penetration Testing," 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 2019, pp. 525-530, doi: 10.1109/COMITCon.2019.8862224.

[4] S. Huang, H. Lu, W. Leong, and H. Liu, "CRAXweb: Automatic Web Application Testing and Attack Generation," 2013 IEEE 7th International Conference on Software Security and Reliability, Gaithersburg, MD, USA, 2013, pp. 208-217, doi:10.1109/SERE.2013.26.

[5] B. Wang, L. Liu, F. Li, J. Zhang, T. Chen, and Z. Zou, "Research on Web Application Security Vulnerability Scanning Technology," 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chengdu, China, 2019, pp. 1524-1528, doi: 10.1109/IAEAC47372.2019.8997964.

[6] Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu and N. Almashfi, "Web Application Security Tools Analysis," 2017 IEEE 3rd international conference on big data security on cloud (bigdatasecurity), IEEE international conference on high performance and smart computing (hpsc), and IEEE international conference on intelligent data and security (ids), Beijing, China, 2017, pp. 237-242, doi:33 10.1109/BigDataSecurity.2017.47. 7.

[7] D. Gol and N. Shah, "Detection of web application vulnerability based on RUP model," 2015 National Conference on Recent Advances in Electronics & Computer Engineering (RAECE), Roorkee, India, 2015, pp. 96-100, doi: 10.1109/RAECE.2015.7510233.

[8] L. Dukes, X. Yuan, and F. Akowuah, "A case study on web application security testing with tools and manual testing," 2013 Proceedings of IEEE Southeastcon, Jacksonville, FL, USA, 2013, pp. 1- 6, doi: 10.1109/SECON.2013.6567420.

[9]. Antony Vijay, J., H. Anwar Basha, and J. Arun Nehru. "A dynamic approach for detecting the fake news using random forest classifier and NLP." In *Computational methods and data engineering*, pp. 331-341. Springer, Singapore, 2021.