I.J. Information Technology and Computer Science, **2023, 3, 26-41**
Published Online on June 8, 2023 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2023.03.03

Detecting and Preventing Common Web Application Vulnerabilities: A Comprehensive Approach

Najla Odeh*
Palestine Technical University – Kadoorie / Department of Computer Science, Tulkarm, P.O Box 305, Palestine
E-mail: najlaa.odeh@ptuk.edu.ps
ORCID iD: https://orcid.org/0000-0003-1089-9243
*Corresponding Author

Sherin Hijazi
Palestine Technical University – Kadoorie / Department of Computer Science, Tulkarm, P.O Box 305, Palestine
E-mail: s.hijazi@ptuk.edu.ps, sherinhijazi@yahoo.com
ORCID iD: https://orcid.org/0000-0003-2411-5681

Received: 09 February 2023; Revised: 20 March 2023; Accepted: 17 April 2023; Published: 08 June 2023

Abstract: Web applications are becoming very important in our lives as many sensitive processes depend on them. Therefore, it is critical for safety and invulnerability against malicious attacks. Most studies focus on ways to detect these attacks individually. In this study, we develop a new vulnerability system to detect and prevent vulnerabilities in web applications. It has multiple functions to deal with some recurring vulnerabilities. The proposed system provided the detection and prevention of four types of vulnerabilities, including SQL injection, cross-site scripting attacks, remote code execution, and fingerprinting of backend technologies. We investigated the way worked for every type of vulnerability; then the process of detecting each type of vulnerability; finally, we provided prevention for each type of vulnerability. Which achieved three goals: reduce testing costs, increase efficiency, and safety. The proposed system has been validated through a practical application on a website, and experimental results demonstrate its effectiveness in detecting and preventing security threats. Our study contributes to the field of security by presenting an innovative approach to addressing security concerns, and our results highlight the importance of implementing advanced detection and prevention methods to protect against potential cyberattacks. The significance and research value of this survey lies in its potential to enhance the security of online systems and reduce the risk of data breaches.

Index Terms: Security System, Websites Attacks, Application Security, SQL Injection, XSS.

1. Introduction

With the spread of the Internet and the emergence of many Internet-connected applications, the importance of web application security has become more and more critical. Every few years, the Open Web Application Security Project (OWASP) publishes a list of security threats to web applications. In 2017, the injection attack took the number one spot, which proved to be both important and harmful. In 2021, injections are the third most dangerous after Broken Access Control and Cryptographic Failures [1].

According to Edgescan's 2020 Vulnerability Statistics Report, over 8 billion records were breached in 2019, with many of these breaches caused by web application layer vulnerabilities that might have been avoided if good security development practices and visibility had been followed. According to the report also, the number of high-risk or critical vulnerabilities found in external web applications rose from 19.2% in 2018 to 34.78% in 2019. In addition, the number of high-risk or serious vulnerabilities discovered in network layer systems has greatly increased more than doubled in 2019 [2].

Detecting vulnerabilities is always an essential requirement for doing a decent job of network security [4]. Hackers attempt to obtain information about the database schema, including the database name, column names, and data types by exploiting vulnerabilities in websites, such as SQLI technology. If the SQLI attack is successful, hackers may be able to extract a large amount of stored data and may gain administrative access to the databases if the databases are not properly protected. As a result, a website's back-end DBMS must be able to detect and prevent malicious SQLI attacks on stored databases.

Developers are more concerned with providing functionality and satisfying user requirements rather than security concerns. As a result, we have numerous security concerns. To ensure security and secure web applications, developers must apply coding best practices, do code security reviews, conduct vulnerability detection, and employ vulnerability analysis tools [5].

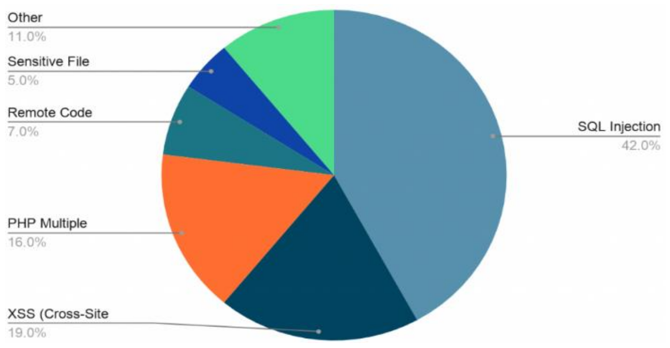


Fig.1 Most Common Critical Vulnerabilities in 2019 [3]

Web application security is a critical concern due to the increasing prevalence of web applications. While a number of techniques and tools have been created to find potential vulnerabilities, they usually only focus on one topic and do independent vulnerability scanning, limiting the efficiency of vulnerability detection. The main objective of this paper is to develop a comprehensive approach to detecting and preventing four common web application vulnerabilities: SQL injection, cross-site scripting attacks, remote code execution, and fingerprinting of backend technologies. To address this problem, we have developed a new vulnerability detection and prevention system that will provide an automated, efficient, and effective means of securing web applications. The main contributions of this paper are: clarifying the working principle of system-detected vulnerabilities, examining the website and searching for vulnerabilities, and preventing these vulnerabilities through the use of defense and protection methods. In addition, the system has been evaluated through practical application on a website affected by many security vulnerabilities. The significance of this research lies in its potential to improve the security of web applications and protect sensitive data from malicious attacks. We hope that this research will raise awareness about the importance of web application security and inspire further research in this area.

The structure of this paper is summarized as follows: the second section provides a background on the vulnerabilities that the system discovers and how they work. The third section presents the methodology used in this paper. In the fourth section, related works are presented. In the fifth section, the experimental method, proposed system operations plan, and analysis of both the detection and prevention phases are presented. In the last section, our system interface is shown and applied to a website and shows the results.

2. Background

The background section of this paper provides an overview of various web vulnerabilities, including SQL injection, XSS, RCE, and fingerprinting of backend technologies. These vulnerabilities can cause significant harm to web applications and require effective defense and protection methods to prevent them. The objectives of this paper include clarifying the working principle of system-detected vulnerabilities, examining websites for vulnerabilities, and preventing these vulnerabilities using defense and protection methods. Therefore, the background section is directly related to the research objectives, as it lays the foundation for understanding the vulnerabilities that the research aims to prevent.

2.1. SQL Injection Vulnerability

SQLI is one of the most dangerous web vulnerabilities. By submitting forged input data to the website, attackers can change the communication between a web server and a SQL database. Attackers can extract data from a database that they should not be allowed to retrieve by manipulating the SQL query invoked by a server-side script. Attackers can alter databases permanently or even use the SQLI vulnerability to transmit remote commands to the website server for execution. Detecting SQLI vulnerabilities is a critical responsibility for ethical hackers and legitimate penetration testers when it comes to securing a system [6].

SQLI can overcome authentication, data loss, denial of access, and it may harm the entire database or cause a host takeover when data arrives from an untrusted source or build dynamic queries based on incoming data [7]. The key event behind SQLI is the direct insertion of user input into parameters, which are then concatenated into SQL statements and executed.

2.2. XSS Vulnerability

The cross-site scripting vulnerability, abbreviated as XSS to distinguish it from cascading style sheets, is a type of vulnerability that can put web applications at risk by introducing malicious code. Fishing attacks and form hijacking caused by exploiting XSS vulnerabilities result in enormous losses for businesses, according to the Internet security threat report for 2019 [8]. In 2018, Symantec detected more than 3.7 million hijacking attacks. The XSS vulnerability is one of the most common and widespread online vulnerabilities. Exploiting XSS flaws can result in a slew of major issues.

Attackers construct a large number of URLs with malicious code and use them to entice visitors to click on them. An attacker can obtain cookies from users by clicking on these URLs, and then use those cookies to enter into victims' accounts. Vulnerabilities to XSS Classification [9]:

- DOM-Based XSS is caused by dangerous client-side code rather than server-side code. This type of issue can surface on pages containing JavaScript code, such as document. eval () or write (). The attacker delivers a URL containing malicious JS code to the victim. When the victim clicks the link, he will receive an email with no malicious code. The malicious code is executed on the client side, providing the attacker with access to the victim's personal information.
- Stored XSS is a prevalent issue on websites such as forums and blogs. The malicious code is involved in the attacker's website submissions, which are promptly kept in databases by the websites. When the user browses these contents, the XSS attack will begin.
- Reflected XSS attacks are similar to DOM Based XSS attacks in that malicious code is incorporated into the website's response. Malicious scripts are commonly put into URLs in order to attract users to click on them. When the user clicks on the link, the website responds with malicious code. The attacker will have access to sensitive information if malicious code is executed. Unlike DOM-based XSS vulnerabilities, reflected XSS is a server-side flaw. Malicious code is processed on the server rather than on the client.

2.3. RCE Vulnerability

RCE indicates an attacker's ability to gain access to another person's computer and modify it, anyway of where that machine is physically situated. Vulnerabilities can allow an attacker to run malicious code and take complete control of a system with the user's rights to run the program. Attackers frequently seek to boost their privileges after getting access to the system.

RCE has been identified as one of the most dangerous web applications dangers. Although RCE is a specific type of cross-site scripting attack, it has some variations, such as requiring both server and client state consideration, string and non-string alteration of client inputs, and numerous requests to multiple server-side scripts [10]. According to data research, RCE vulnerabilities are more common in software systems than other types of vulnerabilities. As a result, programmers are frequently called upon to remedy RCE vulnerabilities by adding or altering lines of code in their software systems such as adding updates, and patches to mitigate RCE vulnerabilities [11].

A person's computer can be accessed and changes made over the Internet using remote code execution. In order for the attacker to execute server commands on a remote server. Many exploit techniques are meant to give clients client-level access to the root level of the machine. A web application attacker exploits the RCE vulnerability by using the request-based field, which is the URL base parameter in the request for the input field parameter. The attacker creates malicious scripts that aid in exploiting the RCE vulnerability in the target website, such as echo "hello world"; this code is then delivered to the server via the RCE-based vulnerable website. Malicious code runs on the remote server and responds to messages sent to the attacker by the server. It will be considered an RCE vulnerable website if this message is relevant to the attacker's needs [12].

2.4. Fingerprinting of Backend Technologies Vulnerability

To obtain as much information about targets as possible, hackers utilize fingerprinting as the initial step in their attack. Cyber security fingerprinting is a collection of data that can be used to recognize network protocols, operating systems, physical devices, and software. Attackers find weaknesses in targeted systems that they can exploit using this vulnerability.

Banner grabbing, eliciting responses to faulty requests and employing automated tools to execute more powerful detect that use a combination of strategies are all techniques used for web server fingerprinting. All of these strategies are based on the same core concept. They are all trying to get a response from the web server that can be matched to a recognized server type by comparing it to a database of known responses and behaviors [13].

While exposed server information is not necessarily a vulnerability in and of itself, it is the knowledge that can aid attackers in exploiting other vulnerabilities. Exposed server information can also lead to the discovery of version-specific server flaws that can be exploited on unpatched servers. As a result, it is advised that precautions be taken.

3. Methodology

The methodology of this research helps to examine web applications and discover pages affected by security vulnerabilities and then provides means of prevention and protection from them. The methodology of this work is divided into four steps. First, analysis of previous studies related to techniques and tools for detecting and preventing security vulnerabilities. Second, designing a new proposed system for vulnerability detection and prevention system, the system identifies many website vulnerabilities like cross-site scripting attacks, SQL injection, remote code execution, and back-end fingerprinting. Third, developing the proposed system. Fourth, validating the proposed system through a practical application on a website, and showing the results. Overall, the proposed methodology facilitates achieving research objectives by providing a systematic approach to detecting and preventing security vulnerabilities in web applications. By combining a literature review, system design, system development, and system validation, the researchers can ensure that their methodology is effective, reliable, and achieves its objectives.

4. Review of Previous Studies

Because a website carries basic information about an organization, website security is crucial. The problem is that the results of the last few decades are all from a safety standpoint, thus redesigning and developing them is impossible. Preventing attacks and predictability is the standard solution. It entails embedding some specific software in an existing network framework or operating system in order to uncover existing vulnerabilities and then updating and upgrading to close the gaps in order to avoid security problems.

Many techniques and tools have been proposed to protect against security vulnerabilities and defense methods, including: In the study [14] they made a tool that detects vulnerabilities such as XSS, and SQLI using Python language. The program is run automatically to send reports to the concerned users. The vulnerabilities were categorized as accessible. It is relevant to the research objectives as it provides a technique for detecting common website vulnerabilities.

The vulnerability detection efficiency could be improved by using AJAX crawl as outlined in [15]. Scanning news aggregation, a web program named Gregorius was used to test this strategy. This approach can be used to detect AJAX application vulnerabilities. This study proposes the use of AJAX crawl to improve vulnerability detection efficiency. It is relevant to the research objectives as it suggests a technique for detecting vulnerabilities in AJAX applications.

The authors [16] present an approach Identification and Mitigation Tool for SQLI Attacks. It enables software testers to detect SQLI vulnerabilities in web applications during the testing process. The proposed method is based on parameterized queries and the validation of user input. Their findings reveal that the tool is completely accurate and efficient in identifying and mitigating SQL issues. It is relevant to the research objectives as it provides a technique for detecting and preventing SQLI vulnerabilities, which is one of the vulnerabilities targeted by the proposed system.

Ali, Abdullah, and Alostad [17] developed an SQLI vulnerability detection tool for the automatic creation of SQLI attacks, the tool automates the penetration testing process to make it easier even for those who are not familiar with hacking methods allowing the effective performance of penetration testing on PHP to find SQL vulnerabilities in web server databases that are hidden. The tool uses a combination of attacking patterns, vectors, and modes to help web developers who are not familiar with hacking techniques perform penetration testing on their web database servers. This study proposes an SQLI vulnerability detection tool for the automatic creation of SQLI attacks. It is relevant to the research objectives as it provides a technique for automating penetration testing to find SQL vulnerabilities in web server databases.

According to Zhang et al. [18] proposed An Automated Composite Detection Tool with multiple vulnerabilities, the tool includes a combination of three types of vulnerability detection, including cross-site scripting attacks, SQL injections, and directory actions. The principle of each type of vulnerability and the detection method will be discussed. Then details about the defenses of each type will be outlined. The advantages are as follows: first, manual testing costs are less. Second, work efficiency is considerably enhanced. And third, the network runs safely for the first time. It is relevant to the research objectives as it provides a technique for detecting multiple vulnerabilities, similar to the proposed system.

Alsmadi and Mira [19] Suggested a method for classifying eigenvectors based on genetic ambiguities and rules. They devised a model to study how websites behave while faced with incorrect inputs. Invalid inputs should be recognized and discarded as soon as feasible from a security standpoint. They suggested a set of indications for determining how incorrect inputs are handled. This model will be implemented using a tool. They put the model to the test by examining a variety of websites that were chosen at random. It is relevant to the research objectives as it provides insights into how websites handle incorrect inputs, which can inform the development of the proposed system.

ETSS Detector is a generic and modular online vulnerability detection created by Rocha and Souto that automatically examines web applications for XSS vulnerabilities. It can find and evaluate all of the application's data entry points, as well as create code injection tests for each of them. The findings revealed that correctly completing the input fields with only the proper information improves the effectiveness of the tests, increasing the rate of XSS assault detection. This study proposes an online vulnerability detection tool that automatically examines web applications for

XSS vulnerabilities. It is relevant to the research objectives as it provides a technique for detecting XSS vulnerabilities, which is one of the vulnerabilities targeted by the proposed system [20].

For automatically detecting RCE vulnerabilities in PHP-based platforms, a path, and context-sensitive inter-procedural analysis model approach was presented. The prototype looked at ten real-world PHP applications and discovered 21 actual RCE flaws. This study proposes a path and context-sensitive inter-procedural analysis model approach for automatically detecting RCE vulnerabilities in PHP-based platforms. It is relevant to the research objectives as it provides a technique for detecting RCE vulnerabilities, which is one of the vulnerabilities targeted by the proposed system [10].

Table 1 shows a comparison between the proposed system and other systems in terms of the vulnerability detection process. The proposed system detects four vulnerabilities at once, which are as follows: SQL injection, cross-site scripting attacks, remote code execution, and backend fingerprinting techniques.

Table 1. A comparison between the Proposed System and Other Systems in Terms of Detecting Security Vulnerabilities

Systems/ Vulnerabilities Detection	SQLI	XSS	RCE	Fingerprinting
Rankothge et al [16]	✓	x	x	x
Ali et al [17]	✓	x	x	x
Zhang et al [18]	✓	✓	x	x
Proposed system	✓	✓	✓	✓

Table 2 shows a comparison between the proposed system and other systems in terms of the vulnerabilities preventing process and protecting against it. The proposed system prevents four security vulnerabilities, namely: SQL injection, cross-site scripting attacks, remote code execution, and backend fingerprinting techniques.

Table 2. A Comparison between the Proposed System and Other Systems in Terms of Preventing Security Vulnerabilities

Systems/ Vulnerabilities Prevention	SQLI	XSS	RCE	Fingerprinting
Rankothge et al [16]	x	x	x	x
Ali et al [17]	x	x	x	x
Zhang et al [18]	✓	✓	x	x
Proposed system	✓	✓	✓	✓

Table 3 shows the additional features in the proposed system, such as its support for Windows and Linux systems, and its provision of a full report. In addition, it detects security vulnerabilities in sites programmed using PHP and ASP.

Table 3. Additional advantages of the proposed system

Extra features/ Systems	Rankothge et al [16]	Ali et al [17]	Zhang et al [18]	Proposed system
Submit a full report	Full Report	Full Report	Full Report	Full Report
Support websites (PHP, ASP)	PHP	PHP	PHP & ASP	PHP & ASP
Support systems (Linux, Windows)	Linux	N/A	N/A	Linux & Windows

5. Experimental

We developed a new system to detect and prevent the security vulnerabilities that put web applications at risk. The system is developed in two phases: the detection phase and the prevention phase.

First, the proposed system detects SQLIA vulnerabilities on every page of the website by analyzing the source code, URL, and user input forms. It starts preparing a payload to inject the website, then injects it with malicious code and prints the report. This is then followed by an XSS penetration examination and fingerprinting technology and RCE, as well as a report on the presence of these vulnerabilities at the website.

For our system to automatically detect SQLI vulnerabilities, XSS vulnerabilities, remote code execution, and fingerprinting attacks, the website to be detected must be analyzed. Figure 2 shows the Vulnerability detection and prevention flowchart. At first, we enter the URL of the website to be examined. The system checks the entered link and makes sure that they are correct, and then the process of checking the website for security vulnerabilities begins. To test if a website has a vulnerability or not, the website is injected with a set of malicious code and various payloads, and then we check the result of this injection. If it leads to damage to the website and the emergence of a specific error through which this vulnerability can be exploited and the website hacked. In the context of web application security, RCE (Remote Code Execution) is a vulnerability that allows an attacker to execute arbitrary code on the web server or application. This type of vulnerability can be extremely dangerous, as it allows an attacker to take complete control over the affected system. The proposed system includes a module that tests for RCE vulnerabilities on the target website. The proposed system also includes a fingerprinting module, which can be used to gather information about the target website and help identify potential vulnerabilities.

Once security vulnerabilities and errors are discovered, the system suggests a series of instructions that the programmer must follow to prevent attacks on the website. Overall, the system aims to provide a comprehensive approach to detecting and preventing security vulnerabilities in web applications by identifying potential vulnerabilities and providing guidance on how to mitigate them.

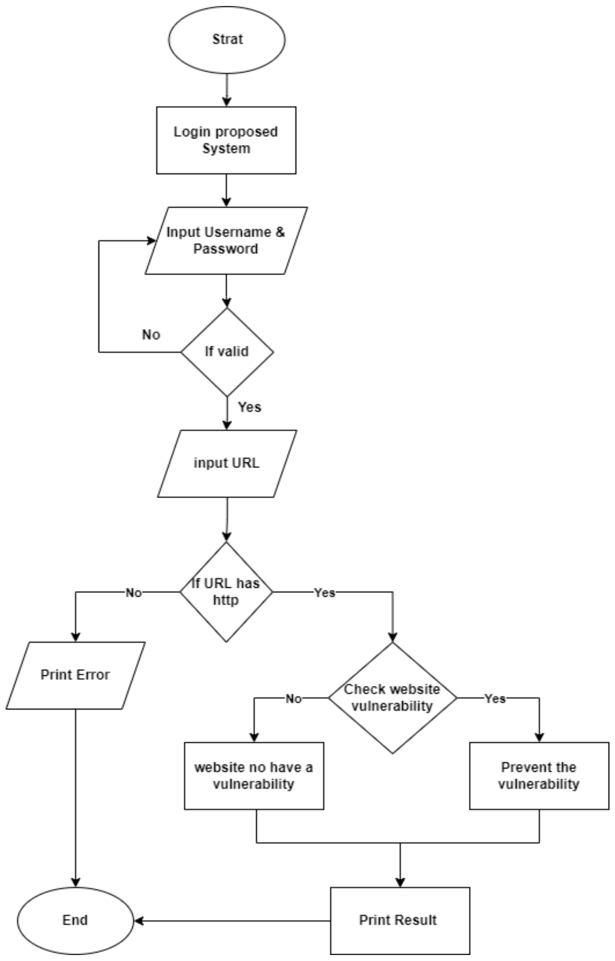
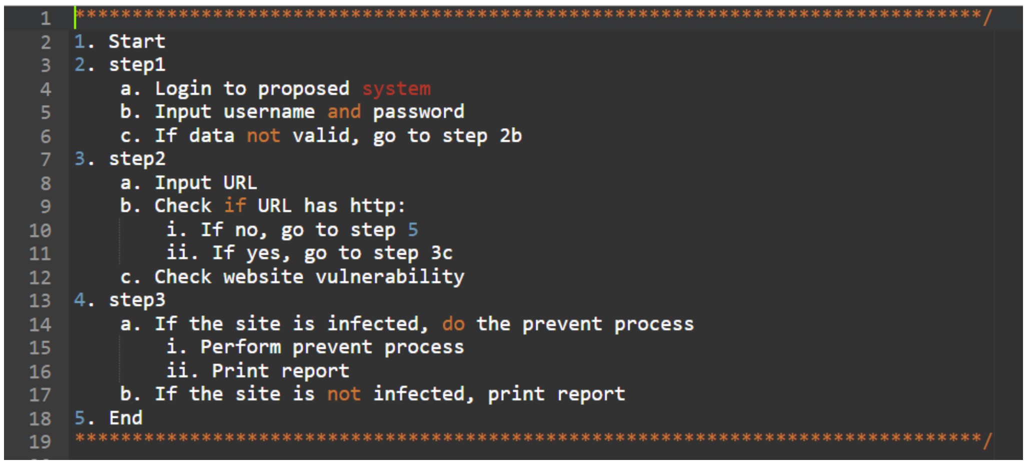

Strat
Login proposed System
Input Username & Password
No
If valid
Yes
input URL
No
If URL has http
Yes
Print Error
No
Check website vulnerability
Yes
website no have a vulnerability
Prevent the vulnerability
End
Print Result

Fig.2. The Proposed System Flowchart



```
*******************************************************************************/
1. Start
2. step1
    a. Login to proposed system
    b. Input username and password
    c. If data not valid, go to step 2b
3. step2
    a. Input URL
    b. Check if URL has http:
        i. If no, go to step 5
        ii. If yes, go to step 3c
    c. Check website vulnerability
4. step3
    a. If the site is infected, do the prevent process
        i. Perform prevent process
        ii. Print report
    b. If the site is not infected, print report
5. End
*******************************************************************************/
```

Fig.3. Proposed System Algorithm

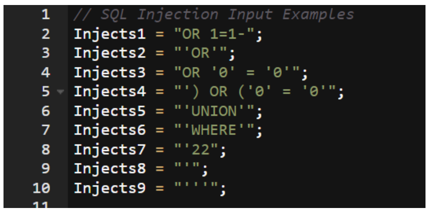
5.1. Detection Phase

In this stage of the proposed system, we detect security vulnerabilities affecting websites such as SQLI, cross-site scripting attacks, remote code execution, and fingerprinting of backend technologies.

A. SQLI Vulnerability Detection

Detecting for SQL injection vulnerabilities on the website, a single quotation (') symbol is used as the input for each form field. If the targeted website is vulnerable, some of the form parameters will be used to build a SQL query without being sanitized first. In this situation, the injected quote symbol is likely to change the query's syntax so that it no longer conforms to proper SQL syntax. If the website does not handle exceptions or server problems, the return page includes a SQL error explanation [21].

The login system is typically the most prevalent attack that threatens the database system. When it comes to the login screen, most attackers will try every possible combination to guess the password. SQLI is a technique that is quite prevalent and commonly exploited by attackers. If the website is vulnerable to SQLI, the attacker will make multiple malicious code entries, which will lead to database problems and error messages. Figure 3 shows examples of SQLI entries.



```
// SQL Injection Input Examples
Injects1 = "OR 1=1-";
Injects2 = "'OR'";
Injects3 = "OR '0' = '0'";
Injects4 = "') OR ('0' = '0'";
Injects5 = "'UNION'";
Injects6 = "'WHERE'";
Injects7 = "'22";
Injects8 = "'";
Injects9 = "''''";
```

Fig.4. SQLI Input Examples

At this point, the proposed system provides SQLI attack detection. Figure 4 shows a successful SQLI attack detected by the login page.

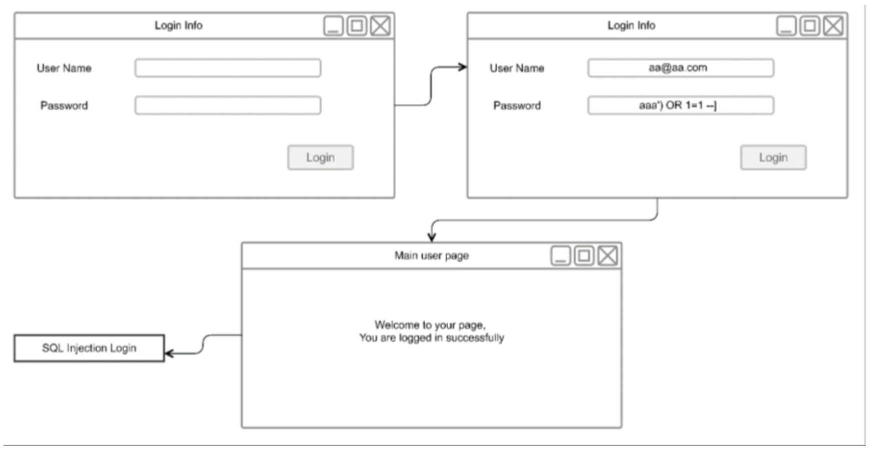
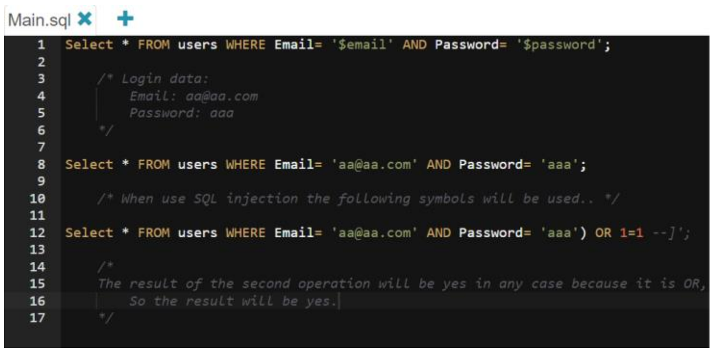


Fig.5. SQLI Attack Detection

If the website is vulnerable to SQLI, the attacker will make many malicious code entries, causing database issues and error messages. For example, it adds the symbols "1" or "1" = "1" in the name and password fields. If the website does not have any SQLI defense, an attacker can access the website without permission. Figure 5 shows unauthorized access to the database system after a successful SQLI attack.



Main.sql

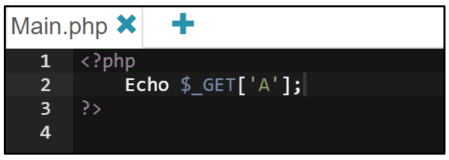
```
Select * FROM users WHERE Email= '$email' AND Password= '$password';

    /* Login data:
        Email: aa@aa.com
        Password: aaa
    */

Select * FROM users WHERE Email= 'aa@aa.com' AND Password= 'aaa';

    /* When use SQL injection the following symbols will be used.. */

Select * FROM users WHERE Email= 'aa@aa.com' AND Password= 'aaa') OR 1=1 --]';

    /*
    The result of the second operation will be yes in any case because it is OR,
        So the result will be yes.
    */
```

Fig.6. Unauthorized Access by SQLI Attack

B. XSS Vulnerability Detection

XSS can cause a wide range of issues for end-users, ranging from little annoyance to complete account breach. The most serious XSS attacks expose the user's session cookie, allowing an attacker to hijack the user's session and seize control of the account. Other harmful assaults include the revealing of end-user files, the installation of Trojan horse programs, the redirection of the user to another website or website, and the alteration of content presentation. An XSS flaw that allows an attacker to change the content of a press release or news item could hurt a company's stock price or erode consumer confidence [22].

At this point, the proposed system detects an XSS attack by injecting a suitable malicious code into a website that the user visits to run a malicious attack in the user's browser. For example, we inject a PHP code like "Echo $_Get ['A']", without any filtering, the value of input A is output immediately and gets a message error. When the server parses the response body, echo will fully output <Script>alert("Error") </script> and the browser will parse and execute the JavaScript dangerous script.



```
<?php
    Echo $_GET['A'];
?>
```

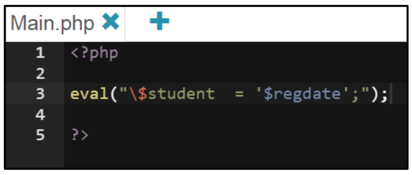
Fig.7. PHP Code- input A without a Filter

In a typical XSS attack, there are two stages:

- An attacker must first discover a technique to inject malicious code (payload) into a web page that the victim visits in order to run malicious JavaScript code in the victim's browser.
- The victim must then browse the malicious code-infected web page. If the attacker wants to target a specific victim, he can employ social engineering or phishing to send the victim a malicious URL.

C. RCE Vulnerability Detection

At this point, the proposed system detects the RCE attack by executing remote instructions and commands. For example, sometimes-variable names are generated dynamically for each user and their registration date is stored. PHP example: As long as the username student is controlled by the user's input, an attacker may create a named student like this:



```
<?php

eval("\$student  = '$regdate';");

?>
```

Fig.8. PHP Code- Dynamic Variable

x = 'ahmad';phpinfo(); The resulting PHP code would look like this:
$x = 'ahmad';phpinfo();// PHP Version => 8.0.9;

When the attacker can change the value of the variable, he can use a semicolon to construct a new command (;). Now he may finish the rest of the string. He will not have any syntax issues in his work this way. The output of phpinfo will be displayed on the website as soon as he runs this code.

Another Example of a code REC attack: When a web server gets a request from a client, it runs code to process the request, using the data in the request as parameters that influence the response provided. In requesting the URL http://www.website.com?usernum=1&pageid=7 a parameter of "1" has been passed in for the username which is used as part of an operating system command. If a remote user can send in a parameter that breaks out of the intended context and influences the command performed by the server instead of a "1" (data – an integer), then remote code execution has occurred. For example, injecting Meta characters such as backslashes, dollar signs, single quotations, and semi-colons, in a parameter that would be included in an operating system command without sanitization can allow us to run our own instructions.

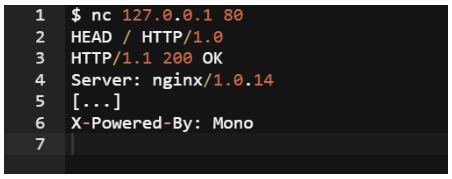
D. Fingerprinting of Backend Technologies Vulnerability Detection

At this point, the proposed system detects the Fingerprinting attack by sending customized packets to obtain information about the target's network. The attackers can identify the OS, software, and protocols. This enables them to create the attack to cause the most damage to the targeted systems [23]. Sending an HTTP request to the web server and inspecting the response headers is how a banner grab is done. This can be done with a variety of technologies, such as telnet for HTTP queries and OpenSSL for SSL requests.

Well-known web applications contain HTML headers, cookies, and directory structures that can be counted to identify the application. Furthermore, most web frameworks include tags on those websites that aid an attacker or tester in identifying them. A tag is searched for from a Preset location and then compared to a database of known signatures. Several tags are typically used to improve accuracy.

In the proposed system, we focused on HTTP headers, The X-Powered-By field in the HTTP response header is the simplest basic way to identify a web framework. Example of an HTTP request-response:

The web application framework is most likely Mono, according to the X-Powered-By field. We can also see that the content is served by a certain version of Nginx. There are sometimes more HTTP headers pointing to a certain framework. This allows penetration testers to increase the number of attack paths available to them. As a result, each HTTP header should be thoroughly examined for leaks during fingerprinting.



```
$ nc 127.0.0.1 80
HEAD / HTTP/1.0
HTTP/1.1 200 OK
Server: nginx/1.0.14
[...]
X-Powered-By: Mono
```

Fig.9. HTTP Request-Response

5.2. Prevention Phase

The proposed system introduced the following vulnerabilities prevention methods:

A. Prevent SQLI Vulnerabilities

When hackers attack websites that use SQL in a back-end DBMS to manage private and sensitive data, they use an SQLI attack to gain access to valuable and sensitive information such as credit card information, account names, and passwords. Therefore, the proposed system introduces the necessary mechanism to prevent SQLI attacks. Figure 9 shows the instructions.

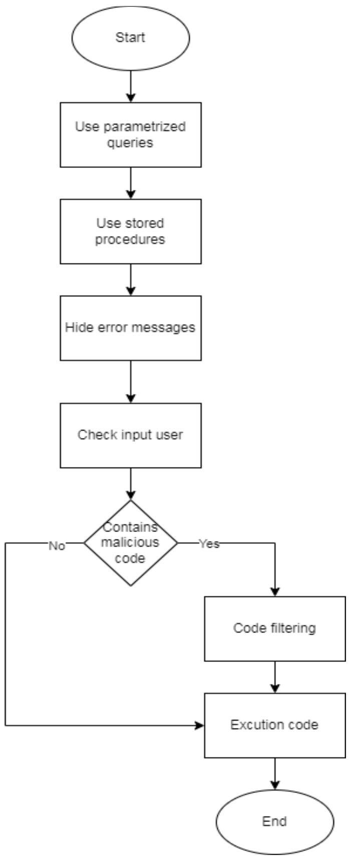


Fig.10. Prevent SQLI Vulnerability

The best mechanism for preventing SQL injections is to utilize safe programming techniques such as parameterized queries (prepared statements) and stored procedures. It is also important not to trust user entries; they should always be sanitized before they can be used in dynamic SQL statements. Regular expressions can be used to detect and eliminate potentially dangerous code before running SQL statements. User access permissions for the database connection must be specified, and the accounts used to connect to the database must have only the access privileges that they need. It is important that error messages do not reveal important information or locate the error. Instead of displaying which SQL queries caused the error, simple custom error messages can be used.

B. Prevent XSS Vulnerabilities

XSS assaults are one of the most popular types of website attacks. The proposed system provides a mechanism to prevent an XSS attack. Figure 10 shows the instructions.

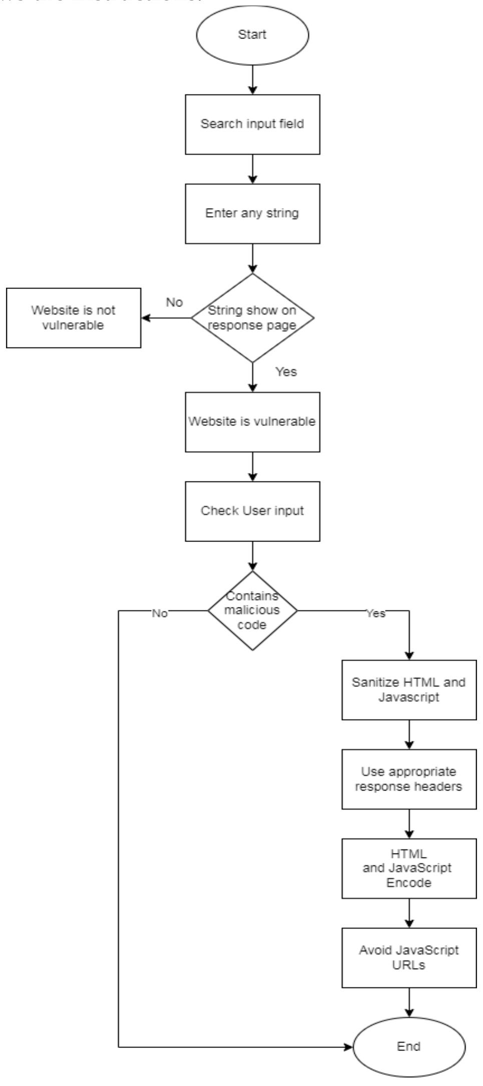

Start
Search input field
Enter any string
String show on response page
No
Website is not vulnerable
Yes
Website is vulnerable
Check User input
Contains malicious code
No
Yes
Sanitize HTML and Javascript
Use appropriate response headers
HTML and JavaScript Encode
Avoid JavaScript URLs
End

Fig.11. Prevent XSS Vulnerability

To prevent XSS vulnerability, the application must check all input data, guarantee that only allowable data is permitted, and confirm that any variable output in a website is encrypted before it is provided to the user. It is necessary to use addresses that are relevant to the response. Content-Type and X-Content-Type-Options headers can be used to avoid XSS in HTTP answers that are not intended to contain any HTML or JavaScript. They ensure that browsers read responses the way you want them to. The content security policy must also be followed. A Content Security Policy (CSP) can be used as a last line of defense to mitigate any remaining vulnerabilities in XSS.

C. Prevent RCE Vulnerabilities

A remote code execution attack involves a remote attacker executing code on the server. An attacker exploits a vulnerability to gain admission to and execute commands on a device or server from anywhere in the globe – or from wherever the attacker is. RCE attacks, in general, rely on exploiting some kind of vulnerability. It might be an error in network defenses, a security flaw in one of the apps, or a vulnerability in the operating system that has not been fixed. The proposed system provides a mechanism to prevent an RCE attack. Figure 11 shows the instructions.

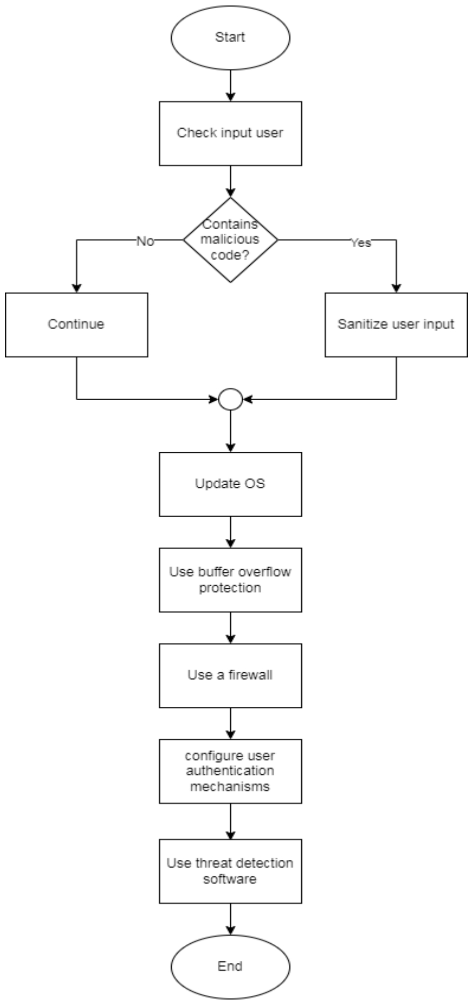


Fig.12. Prevent RCE Vulnerability

To prevent RCE vulnerability, must ensure that user input is sanitized, and it is best to avoid using user input in the evaluated code. The user must not be allowed to modify the content of files parsed by the respective programming language. It is best to keep the operating system up to date and use buffer overflow protection. Also, use a firewall and timely detection of threats.

D. Prevent Fingerprinting Vulnerabilities

The process of preventing the fingerprinting vulnerability is done through security and obscurity, which is the process of enforcing secrecy and confidentiality of a system's internal design architecture in order to implement security within it. The goal of security by obscurity is to safeguard a system by concealing or masking its security weaknesses. The proposed system provides a mechanism to prevent a fingerprinting attack. Figure 12 shows the instructions.

To prevent Fingerprinting vulnerability, the necessary measures must be taken: Ensure that web servers, firewalls, intrusion prevention systems, and intrusion detection systems are properly configured and monitored. Files should be checked on a regular basis for any unexpected activities. Consider using the Apache Modifier header module to hide web server information in headers (removing known headers). It is preferable to add an extra layer of security between the web server and the Internet by using a hardened reverse proxy server. Web servers should be updated with the latest software and security fixes. Use a variety of cookie names (by changing configurations).

5.3. Experimental Implementation

We have developed a new system using Python to achieve the detection and prevention of security vulnerabilities that affect web applications; we have made user interfaces using the Tkinter library to facilitate work on the system so that the average user can use it easily and smoothly. The system consists of the login interface using the username and password, the detect interface that includes entering the website link, in addition to the vulnerability prevention interface, which includes steps and methods of prevention. Figure 13 shows the main interface of the system.

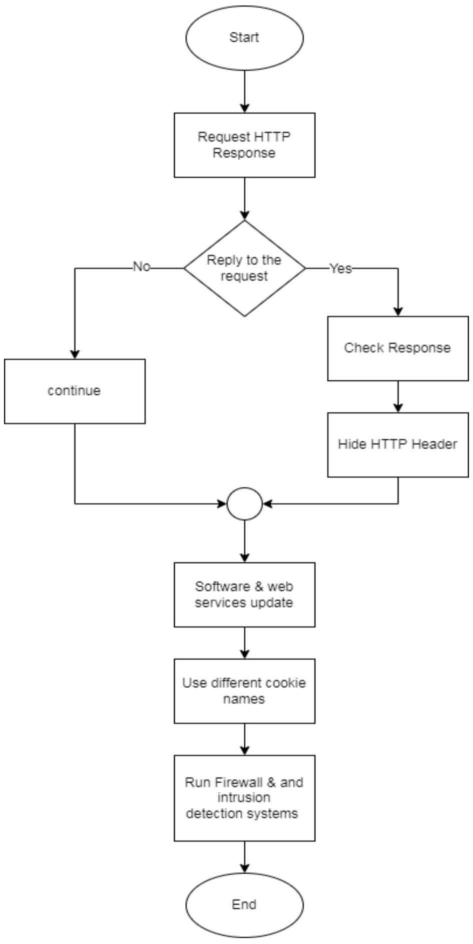

Start
Request HTTP Response
Reply to the request
No
Yes
Check Response
continue
Hide HTTP Header
Software & web services update
Use different cookie names
Run Firewall & and intrusion detection systems
End

Fig.13. Prevent Fingerprinting Vulnerability

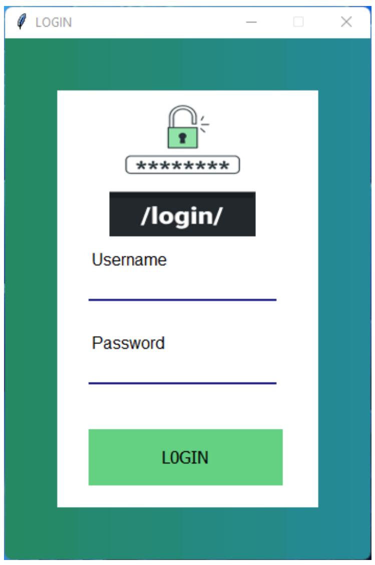


Fig.14. Main System Interface

In case the password is entered incorrectly, a message appears on the screen that the login process failed, as shown in Figure 14, and if the information is correct, a message appears stating that the process of entering the system was successful, Figure 15 shows entering the system.

In this interface, Figure 16, the user enters the link of the website he wants to check, and he has to enter the full link of the site starting with HTTP followed by an internal page of the website. Example: http://testphp.vulnweb.com/listproducts.php?cat=1. Then click on the check icon to start the process of detecting security vulnerabilities.

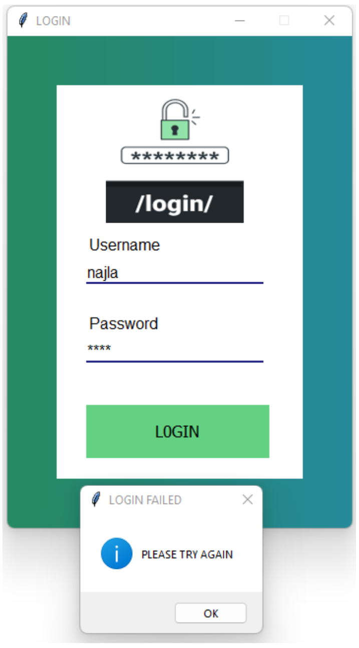


Fig.15. Error Login Interface

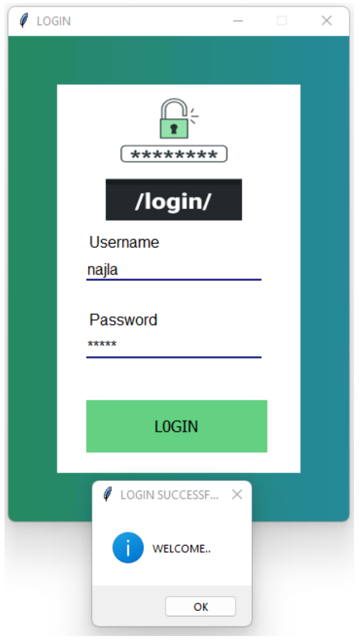


Fig.16. Successfully Login Interface

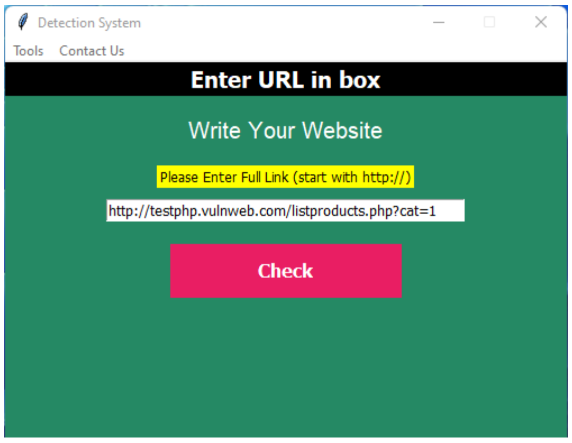

Detection System
Tools
Contact Us
Enter URL in box
Write Your Website
Please Enter Full Link (start with http://)
http://testphp.vulnweb.com/listproducts.php?cat=1
Check

Fig.17. Security Vulnerabilities Check Interface

6. Results and Discussions

During the detecting process, the system crawls every page of the online application, looking for vulnerable pages that can be exploited. The system examines the web application for vulnerabilities by using various payloads. The following is an example of SQLI testing: URL: http://testphp.vulnweb.com/listproducts.php?cat=1 We can identify SQLI by utilizing special characters such as a single quote ('), double quote ("), or queries such as "and false" and so on. Example: URL: http://testphp.vulnweb.com/listproducts.php?cat=1'.

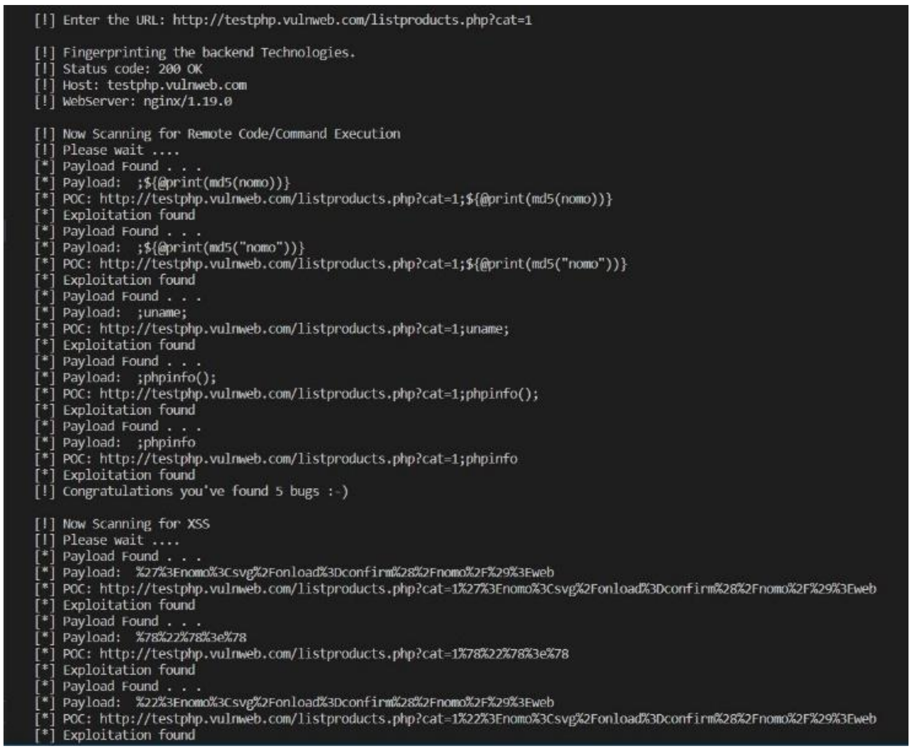
If the website uses MySQL as a backend database and is vulnerable to SQLI, error messages such as mysql_num_rows(), mysql_fetch_array(), and so on will be displayed. When a database error message appears on a website, it becomes vulnerable to SQLI.

We decide if there is a vulnerability in the response based on the content of the response to the malicious code with which the website is injected. When an XSS vulnerability is found, for example, we look for a script alert (XSS) script string in the HTTP response, and if it is present, the URL holds the XSS vulnerability. Otherwise, there is no XSS vulnerability.

To detect RCE vulnerability, the system can use payloads that contain commands that could be executed on the server. If the web application is vulnerable to RCE, the commands will be executed, and the output will be displayed in the response. The system can also detect RCE vulnerability by analyzing the behavior of the web application when the payloads are injected. For example, if the web application responds with an error message that indicates that the command was executed, it indicates that the web application is vulnerable to RCE. An attacker may exploit an RCE vulnerability by injecting malicious code into a web application. For example, if the application has a file upload feature that does not properly validate user input, the attacker could upload a file containing malicious code that is executed on the server. Once the code is executed, the attacker can gain control of the server and perform actions such as stealing sensitive data or installing malware.

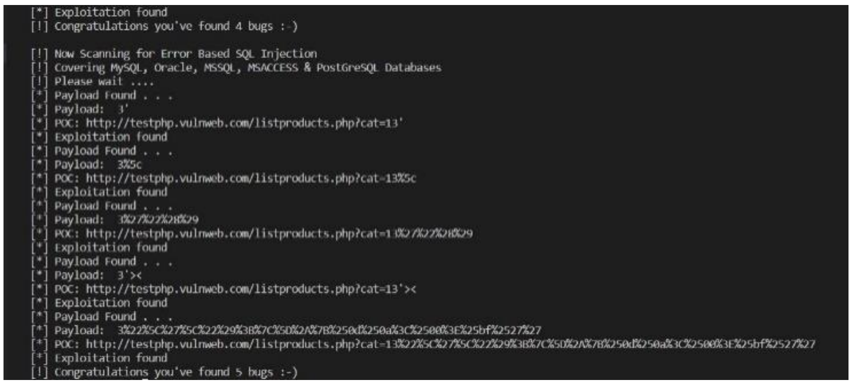
Fingerprinting is essential in penetration testing because it provides information about the technology used by the web application, which can be used to identify vulnerabilities and exploit them. To perform fingerprinting, the system can use various techniques such as analyzing the HTTP headers, analyzing the HTML source code, and using specialized tools such as Nmap, WhatWeb, and Wappalyzer. The system can also analyze the behavior of the web application to gather information about its technology. For example, the system can send various payloads and analyze the response to identify the technology used by the web application. A common example of fingerprinting is when a website uses cookies to track user behavior. If the website does not properly secure the cookies, an attacker could intercept them and use them to identify the user and their behavior on the website. Another example is when a website sends unique identifiers in HTTP headers or other metadata, allowing an attacker to track a user's behavior across multiple requests.

The results show the types of vulnerabilities the website has been infected with by injecting the website with a variety of malicious payloads. Figures 18 and 19 show the results of the assay.



```
[!] Enter the URL: http://testphp.vulnweb.com/listproducts.php?cat=1

[!] Fingerprinting the backend Technologies.
[!] Status code: 200 OK
[!] Host: testphp.vulnweb.com
[!] WebServer: nginx/1.19.0

[!] Now Scanning for Remote Code/Command Execution
[!] Please wait ....
[*] Payload Found . . .
[*] Payload:  ;${@print(md5(nomo))}
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=1;${@print(md5(nomo))}
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  ;${@print(md5("nomo"))}
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=1;${@print(md5("nomo"))}
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  ;uname;
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=1;uname;
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  ;phpinfo();
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=1;phpinfo();
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  ;phpinfo
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=1;phpinfo
[*] Exploitation found
[!] Congratulations you've found 5 bugs :-)

[!] Now Scanning for XSS
[!] Please wait ....
[*] Payload Found . . .
[*] Payload:  %27%3Enomo%3Csvg%2Fonload%3Dconfirm%28%2Fnomo%2F%29%3Eweb
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=1%27%3Enomo%3Csvg%2Fonload%3Dconfirm%28%2Fnomo%2F%29%3Eweb
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  %78%22%78%3e%78
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=1%78%22%78%3e%78
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  %22%3Enomo%3Csvg%2Fonload%3Dconfirm%28%2Fnomo%2F%29%3Eweb
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=1%22%3Enomo%3Csvg%2Fonload%3Dconfirm%28%2Fnomo%2F%29%3Eweb
[*] Exploitation found
```

Fig.18. The Results



```
[*] Exploitation found
[!] Congratulations you've found 4 bugs :-)

[!] Now Scanning for Error Based SQL Injection
[!] Covering MySQL, Oracle, MSSQL, MSACCESS & PostGreSQL Databases
[!] Please wait ....
[*] Payload Found . . .
[*] Payload:  3'
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=13'
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  3%5c
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=13%5c
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  3%27%22%28%29
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=13%27%22%28%29
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  3'><
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=13'><
[*] Exploitation found
[*] Payload Found . . .
[*] Payload:  3%22%5C%27%5C%22%29%3B%7C%5D%2A%7B%250d%250a%3C%2500%3E%25bf%2527%27
[*] POC: http://testphp.vulnweb.com/listproducts.php?cat=13%22%5C%27%5C%22%29%3B%7C%5D%2A%7B%250d%250a%3C%2500%3E%25bf%2527%27
[*] Exploitation found
[!] Congratulations you've found 5 bugs :-)
```

Fig.19. The Results

The proposed system was validated through a practical application on a website, where vulnerabilities were identified and the type of vulnerability was determined. Prevention methods were suggested, and the effectiveness of the detection and prevention methods used were found to be excellent. Future research directions include a comprehensive evaluation of the proposed system's performance against other vulnerabilities and programming prevention action to automatically fix errors. By detecting and preventing common web application vulnerabilities, the proposed approach reduces testing costs, increases efficiency and safety, and contributes to the overall security of web applications.

7. Conclusions and Future Work

In conclusion, this paper has presented a system for detecting and preventing vulnerabilities in web applications, specifically focusing on four common types: SQL injection, cross-site scripting attacks, remote code execution, and backend technology fingerprinting. The paper has provided detailed explanations on how each type of vulnerability works and how the detection method can be applied to identify them. Moreover, the paper has offered preventive measures to control each type of vulnerability. The research has scientific justification, as it addresses the critical need to enhance web application security by preventing and detecting vulnerabilities that can compromise confidential information. Future research can expand on this work by evaluating other types of vulnerabilities and incorporating automated system fixes for errors. The advancements in this field will continue to be critical as web applications become increasingly prevalent and complex.

References

[1] OWASP, OWASP Top 10 - 2021: Top 10 Web Application Security Risk. [Online]. Available: https://owasp.org/Top10/, Accessed on: Nov. 20, 2021.

[2] EdgeScan, Edgescan’s 2020 Vulnerability Stats Report Released. [Online]. Available: https://www.edgescan.com/edgescans-2020-vulnerability-stats-report-released/, Accessed on: Nov. 22, 2021.

[3] Socradar, Top 5 Remote Code Execution (RCE) Attacks in 2020. [Online]. Available: https://socradar.io/top-5-remote-code-execution-rce-attacks-in-2020/, Accessed on: Nov. 20, 2021.

[4] C.Tankard, "Advanced persistent threats and how to monitor and deter them", Network security, Vol.2011, No.8, pp.16-19, 2011. DOI: 10.1016/S1353-4858(11)70086-1.

[5] N.Laranjeiro, M. Vieira, and H. Madeira, "A learning-based approach to secure web services from SQL/XPath Injection attacks," 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, pp.191-198, 2010. DOI: 10.1109/PRDC.2010.24.

[6] L.Erdodi, Å.Å. Sommervoll, and F.M. Zennaro, "Simulating SQL Injection Vulnerability Exploitation Using Q-Learning Reinforcement Learning Agents", Journal of Information Security and Applications, Vol.61, No.102903, 2021, DOI: 10.1016/j.jisa.2021.102903

[7] H.Alsobhi and R. Alshareef, "SQL Injection Countermeasures Methods", 2020 International Conference on Computing and Information Technology (ICCIT-1441), pp.1-4, IEEE, 2020. DOI: 10.1109/ICCIT-144147971.2020.9213748.

[8] SymantecCorporation, Threat Report 2018. [Online]. Available: https://docs.broadcom.com/doc/istr-23-2018-en, Accessed on: Dec 12, 2021.

[9] M.Liu et al., "A survey of exploitation and detection methods of XSS vulnerabilities", IEEE Access, Vol.2019, No.7, pp.182004-182016, 2019. DOI: 10.1109/ACCESS.2019.2960449.

[10] Y.Zheng and X. Zhang, "Path sensitive static analysis of web applications for remote code execution vulnerability detection", 2013 35th International Conference on Software Engineering (ICSE), pp.652-661, IEEE, 2013. DOI: 10.1109/ICSE.2013.6606611.

[11] S.Bier et al., "Mitigating Remote Code Execution Vulnerabilities: A Study on Tomcat and Android Security Updates", 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), pp.1-6, IEEE, 2021. DOI: 10.1109/iemtronics52119.2021.9422666.

[12] S.Biswas et al., "A study on remote code execution vulnerability in web applications", International Conference on Cyber

Security and Computer Science (ICONCS 2018), pp.50-57, 2018.

[13] OWASP, Fingerprint Web Server. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server, Accessed on: Dec. 11, 2021.

[14] R.Abirami, D. J. W.Wise, R.Jeeva, and S.Sanjay, "Detecting Security Vulnerabilities in Website using Python", 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), pp.844-846, IEEE, 2020. DOI: 10.1109/ICESC48915.2020.9155908.

[15] K. J.Koswara and Y. D. W. Asnar, "Improving Vulnerability Scanner Performance in Detecting AJAX Application Vulnerabilities", 2019 International Conference on Data and Software Engineering (ICoDSE), pp.1-5, IEEE, 2019. DOI: 10.1109/ICoDSE48700.2019.9092613.

[16] W.H.Rankothge, M. Randeniya, and V. Samaranayaka, "Identification and Mitigation Tool for Sql Injection Attacks (SQLIA)", 2020 IEEE 15th International Conference on Industrial and Information Systems (ICIIS), pp.591-595, IEEE, 2020. DOI: 10.1109/ICIIS51140.2020.9342703.

[17] A.B.M.Ali, M.S. Abdullah, and J. Alostad, "SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks", Procedia Computer Science, Vol.2011, No.3, pp.453-458, 2011. DOI: 10.1016/j.procs.2010.12.076.

[18] X.Zhang et al., "An Automated Composite Scanning Tool with Multiple Vulnerabilities", 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), pp.1060-1064, IEEE, 2019. DOI: 10.1109/IMCEC46724.2019.8983828.

[19] I.Alsmadi and F. Mira, "Website security analysis: variation of detection methods and decisions", 2018 21st Saudi Computer Society National Computer Conference (NCC), pp.1-5, IEEE, 2018. DOI: 10.1109/NCG.2018.8592962.

[20] T.S.Rocha and E. Souto, "ETSSDetector: a tool to automatically detect Cross-Site Scripting vulnerabilities", 2014 IEEE 13th International Symposium on Network Computing and Applications, pp.306-309, IEEE, 2014. DOI: 10.1109/NCA.2014.53.

[21] S.Kals, E.Kirda, C.Kruegel, and N.Jovanovic, "Secubat: a web vulnerability scanner", in Proceedings of the 15th international conference on World Wide Web, pp.247-256, 2006. DOI: 10.1145/1135777.1135817.

[22] KirstenS. Cross Site Scripting (XSS). [Online]. Available: https://owasp.org/www-community/attacks/xss/, Accessed on: Dec. 5,2021.

[23] Cyware, What Is Cybersecurity Fingerprinting?. [Online]. Available: https://cyware.com/news/what-is-cybersecurity-fingerprinting-de718f94, Accessed on: Dec. 11,2021.

Authors' Profiles

Najla Odeh received the B.S. degree in computer information system from the Al-Quds Open University, Tulkarm, Palestine, in 2011. She is currently working toward the M.S. degree in Computer Science at Palestine Technical University Kadoorie PTUK, Tulkarm, Palestine. She was a Programmer and web designer in IDEX Company, Tulkarm, Palestine. From 2011 to 2015. Currently, she is working as Technical Support at PTUK. Her specific areas of research interest mainly focus on deep learning, network technologies, information system, and network security.

Sherin Hijazi received her B.S. degree in Management Information System from An-Najah National University, Nablus, Palestine, in 2005. She finished her M.S. degree in Computer Information System from Al-Yarmouk University, Irbid, Jordan, in 2012. She was granted a Scholarship from Palestine Technical University Kadoorie (PTUK) to pursue her PhD in Computer Science at the University of Jordan, Amman, in 2015. She finished her Ph.D. degree in Computer Science from the University of Jordan, Amman, Jordan, in 2020. From 2005 to 2007, she was an Administrative Assistant with the IT Department in Palestine Securities Exchange (PSE). From 2007 to 2011, she was a Programmer with PTUK, Tulkarm, Palestine. From 2011 to 2013, she was the Head of programming with the Computer Center, PTUK. From 2013 to 2020, she was a Lecturer with the Department of Applied Computing, PTUK. From 2020 to 2022, she has been a Professor Assistant with the Department of Applied Computing, PTUK. From 2022 until now, she has been a head of Computer Science Department with the Information Technology Faculty, PTUK. From 2022 until now, she has been a head of Information System Department with the Information Technology Faculty, PTUK. She has eight publications in various fields of computer science. Her research interests are in artificial intelligent, knowledge representation, network security, software engineering, database systems, and information system. Dr. Hijazi received the IEEE Systems Journal Best Paper Award in 2020. Dr. Hijazi received the Reviewer Certificate of Security and Privacy in 2020 and 2021. Dr. Hijazi received the Reviewer Certificate from IEEE Access in 2020-2022.

How to cite this paper: Najla Odeh, Sherin Hijazi, "Detecting and Preventing Common Web Application Vulnerabilities: A Comprehensive Approach", International Journal of Information Technology and Computer Science(IJITCS), Vol.15, No.3, pp.26-41, 2023. DOI:10.5815/ijitcs.2023.03.03