Computer Systems Science & Engineering
DOI:10.32604/csse.2022.019427
Article

Tech Science Press

Web Security: Emerging Threats and Defense

Abdulwahed Awad Almutairi[1], Shailendra Mishra[2,*] and Mohammed AlShehri[1]

[1]Department of Information Technology, College of Computer and Information Sciences, Majmaah University, Majmaah, 11952, Saudi Arabia
[2]Department of Computer Engineering, College of Computer and Information Sciences, Majmaah University, Majmaah, 11952, Saudi Arabia
[*]Corresponding Author: Shailendra Mishra. Email: s.mishra@mu.edu.sa
Received: 13 April 2021; Accepted: 09 June 2021

Abstract: Web applications have become a widely accepted method to support the internet for the past decade. Since they have been successfully installed in the business activities and there is a requirement of advanced functionalities, the configuration is growing and becoming more complicated. The growing demand and complexity also make these web applications a preferred target for intruders on the internet. Even with the support of security specialists, they remain highly problematic for the complexity of penetration and code reviewing methods. It requires considering different testing patterns in both codes reviewing and penetration testing. As a result, the number of hacked websites is increasing day by day. Most of these vulnerabilities also occur due to incorrect input validation and lack of result validation for lousy programming practices or coding errors. Vulnerability scanners for web applications can detect a few vulnerabilities in a dynamic approach. These are quite easy to use; however, these often miss out on some of the unique critical vulnerabilities in a different and static approach. Although these are time-consuming, they can find complex vulnerabilities and improve developer knowledge in coding and best practices. Many scanners choose both dynamic and static approaches, and the developers can select them based on their requirements and conditions. This research explores and provides details of SQL injection, operating system command injection, path traversal, and cross-site scripting vulnerabilities through dynamic and static approaches. It also examines various security measures in web applications and selected five tools based on their features for scanning PHP, and JAVA code focuses on SQL injection, cross-site scripting, Path Traversal, operating system command. Moreover, this research discusses the approach of a cyber-security tester or a security developer finding out vulnerabilities through dynamic and static approaches using manual and automated web vulnerability scanners.

Keywords: SQL injection attack; cross-site scripting attack; command injection attack; path traversal attack

1 Introduction

A web application vulnerability enables an intruder to crack into the web application to execute unwanted things on specific victim's sites. The most advanced security vulnerabilities are found in the systems, networks, and application programs of present-day applications. Web applications are open to the public by description, including malicious attackers [1]. Moreover, input to web applications is via Hypertext Transfer Protocol (HTTP) request and response for transferring data over the web, and accurately processing the input can be highly challenging. In recent years, these attacks on web applications have been on the top of the list of hazardous attacks. The most significant and noteworthy examples are Cross-Site Scripting (XSS), which is a type of injection where malicious scripts are injected, and Structured Query Language injection (SQLi) that uses malicious SQL code for backend database manipulation [2].

The end users are using their systems more creatively in the present day than in the past decade. A maximum number of applications are executed online over the network since web applications can replace desktop applications. The information of users that is connected to the internet; can be accessed from any location [3]. Moreover, the developer is aware of the version used by the end-user and hence, conveniently providing support. However, despite having these advantages, there are a few disadvantages. When the server is offline or the end-user does not have access to the internet, the service is unavailable, and the end-user cannot access their information. Furthermore, the end-user should also trust that the web application will protect confidential information, both in terms of availability and confidentiality. It would ensure that the information would not disappear and be accessible to the people he provides authorizations [4].

The user should also consider the major disadvantages. When the server is offline or the end-user does not have accessibility to the internet, the service would be unavailable, and the end-user would not get access to information. However, the web application itself also has to be highly secured. The web application's security is within the developer's skills, and hence, he has to be knowledgeable of all the exploits and the approach of evading them. It can be a challenging job in small web applications and for helping the duty security testing tools be built and examining the web application for security flaws. This paper examines the security measures in web applications by addressing techniques that can be done by different tools, manual processes, and a few vulnerabilities that web applications might experience.

One of the most distinct goals that impact the determination to research this area; is to promote an approach that would significantly influence the fight against web application vulnerabilities, especially cases that concern applications written within PHP and JAVA. A reduced rate of exploits is observed by focusing on the web application vulnerabilities within web applications. The success of this specific objective is possible by producing an appropriate approach towards securing web applications in the source code in PHP and JAVA applications and even selecting the appropriate web application scanner to make the applications more secure.

To prevent malicious access, the web application should be secured. The web application is protected within the developer's skill set, and therefore, they require to know about all the exploits and the approach to work around them. It can be challenging for small web applications and examine the web application for security vulnerabilities [5,6]. Vulnerabilities and attack detection processes are reported based on features' selection [7]. The drawback of this approach is its subjective nature due to ranking. In the paper [8], the authors have presented a planning-based attack model. The disadvantage of this model was that the representation is extremely small. This research aims to explore and analyze four vulnerabilities from the most common and critical vulnerabilities found in a web application in both PHP and JAVA code. A comprehensive description of the vulnerabilities, exploitation, and testing is discussed

in [8] using manual and automated web vulnerability scanners. Moreover, this research examines different security measures in web applications.

The main contributions of this research are summarized as follows:

- This research explores and analyzes vulnerabilities found in a web application in both PHP and JAVA codes.
- It examines the security measures in web applications by using techniques that can be performed by the tools manually and examination of some of the vulnerabilities that web applications might experience.
- The research also aims to promote an approach that would significantly influence the fight against web application vulnerabilities, especially cases concerning applications written in PHP and JAVA.
- It has proposed an appropriate approach towards securing web applications in source code in PHP and JAVA applications and choosing a suitable web application scanner to make the applications more secure.

The main part of the paper is organized as follows:

i) Related work in web security, threats attacks, and countermeasures are discussed in section two.

ii) Research Methodology is illustrated in section three.

iii) A detailed discussion of the implementation and result analysis is discussed in section four.

iv) The paper is concluded in section five.

2 Related Work

A literature review has been conducted to explain the modern state-of-the-art automated web application security scanners using dynamic and static analysis. It has also intended to explain the primary issues of the subject and summarize the essential concepts of existing challenges and attacks, such as SQL injection attacks, OS command Injection, Cross-Site Scripting, and directory traversal [9–11]. The advanced development methods include specific frameworks and libraries for regulating effective and quick approaches to improve market requirements. A web application vulnerability allows an intruder to penetrate the web application for performing unwanted aspects on the specified pages of the user. In Durai et al. [12], the authors explained the differences between vulnerability assessment and exploitation and the approach of an assessment to be performed for SQL injection attacks, cross-site scripting attacks, and cross-site request forgery attacks. They even mention secured code reviews and penetration testing in the development lifecycle to identify and mitigate significant vulnerabilities in web applications.

The white-box testing method uses an examination of the programming code to detect major exploitations in the applications. The examination of the programming code can be done either in the traditional style of testing, i.e., manually, or with the help of tools. The black-box testing process, on the contrary, analyzes the execution of an application and detects the issues. This particular examination process is termed penetration testing, in which the scanner sends massive, predefined HTTP requests. Similar to white box testing, black box testing can be done either manually or with tools. In [13], the authors explain a system that automatically detects and mitigates the attacks. Web applications are inefficient in checking the correctness of the data request. For this reason, intruders can exploit their power by inserting the malicious data into the application, bypassing the website's security attributes. The inserted parameters should be examined for their effectiveness in terms of their information representation and field labeling. Patterns and values examine the legitimacy of the parameters for the occurrence of null values, duplicate values.

In [14], the authors have discussed the Injection Attack (IA) classification, which classifies SQLi and is exploited in PHP. The standard and extensive learning-based IA algorithms have appeared to teach and estimate the classification types with the help of testing and validation features of data input from the code source data. A prototype has been taught using a Convolutional NN, and it produced the most significant accuracy of 95.4%. In comparison, another prototype based on Multilayer Perceptron has achieved the highest recall of 63.7% and the most significant measure of 74.6%. The main functionalities of this specific vulnerability scanning system have been realized, and the scanning results show the probability of the technical analyses. In [15], the authors have examined a couple of open and freely available static investigation tools, with a couple of weakly designed applications for vulnerability identification. These are made of insecure applications as evidence after the testing process. Both of them produce results after running the tests.

These results are documented and examined using the Open Web Application Security Project's (OWASP) WAP and static code analysis tool RIPS to detect vulnerabilities in PHP source code. The XSS exploits are classified into two classes, called inside and outside. The inside-cross-site scripting occurs within the corresponding website, where the exploit is to be inserted. In this class, the exploit is also executed on the machine when that inserted page is being clicked. For the outside XSS, the exploit is executed through the browser. Paper [16] has addressed identification of characteristic exploitation in PHP code. It extends the traditional custom design with a component called cleans. It refers to designing of different sanitation methods. An inactive and behind-the-information investigation procedure is provided to discover technical exploits based on the unique design. The tool of POSE performs this process, and the investigation results have proved that the procedure is highly efficient for discovering techniques in web application exploits. The input given by the end-user is included in an HTTP request created by the web server or takes place somewhere in the HTML pages of the Document Object Model (DOM). DOM is a programming interface for HTML.

Cross-site scripting exploits can be broken into web-server-side exploits and client-side vulnerabilities. The web-server-side XSS exploits mostly covered "reflected XSS" and "stored XSS". The client-side exploit is applicable to DOM Cross-site scripting [17]. It distinguishes and examines advanced study outcomes on XSS discovery, breaks them into three vital sections to various tools, and these three sections include static examination, dynamic examination, and hybrid examination. They list 30 discovery methods, make overall similar analyses, and note on powers and flaws of discovered XSS exploits. Sqlmap is a free exploit discovery and pen-testing tool that finds out SQL injection attacks. This tool also automates identification and impacting SQLi exploits and allows for getting across the databases. Havij GUI is an established SQLi tool that benefits all pen testers to identify and exploit SQL injection weakness in a web application [18]. The study reported in [18] is successfully converged on popular and common vulnerabilities like SQLi, XSS, and Cross-Site Request Forgery (CSRF) and explaining such exploitations of the exploits with DVWA. Benefits of the web exploit with discovery and security refers to a reliable web access policy plan.

Machine learning (ML) exceeds difficulties that people can determine. Various ML types, such as Logistic Regression, information description, are significant for design performance. It is essential to collect and choose the convenient characteristics that are utilized to describe any information situation. Deep learning (DL) types, rather than defining the features, are likely to study them concurrently with the primary task. In [19], the authors have performed an extensive learning type capable of analyzing PHP parts as vulnerable or not to the SQL injection. This strategy is meant to suit the workflow of web applications' support written in server-side programming languages like PHP, JSP, and ASP. In [20], the authors have proposed a strategy that combines the software support lifecycle with system examination. All unprotected codes are recreated for showing the recommended encoder. It is validated with the help of a valuating censure method to obtain a conventional replacement. Evaluation of this method has remained successful for an open-source pharmaceutical application written in JSP.

In [21], the authors have performed a combined examination strategy for examining web applications related to the concerns of SQLi exploits and proposed a SQL-INJECTION tool capable of automatic SQLi exploits examination in web applications. In [22], the authors have built the first level RegExps for SQL Injection discovery from network traffic. To identify malicious SQL Injection of web traffic, it uses legitimate expressions obtained from popular SQL Injection tools. SQLi is amongst the traditional protection warnings to cloud databases that work by expanding applications that install their settings on the cloud. Session Hijacking happens when an intruder gains entrance to the session of a particular end-user. The intruder slices a legitimate session identifier applied to go inside the system and take the information. Privilege Escalation refers to the end-user getting the rights of different users. These rights can be utilized to remove data, observe confidential data, or even place undesired applications like viruses. A directory traversal is additionally recognized as path traversal. Directory traversal exploits mainly inadequate validations of browser inputs from end-users.

The algorithm Vul-Scan is applied to create a deception method for mutating examination data automatically. The algorithm chooses methods that meet the victim system and scan class, do not break the conflicted relationship, and follow the hierarchy relationship. The input is victim system data, the most evasion techniques, scan type, and evasion methods database. The authors have developed a web scanner, VulScan, which creates analysis data by utilizing a mixture of deception methods. It avoids filters and Web application firewalls (WAFs) to expose SQLi and XSS exploits [23].

The discovery of XSS vulnerability starts with the Pixy tool to investigate the origin codes for illustrating the control flow graph (CFG) and obtain the PHP source code's entire paths. A few paths of the Control Flow Graph are introduced as unseen routes, which means they will not perform. Consequently, they have pushed the unseen routes to provide reliable outcomes, and they utilized the GA generator on the unseen routes completely. The GA generator begins to initialize an arbitrary group of irregular Cross-site scripting hateful scenarios as inputs. The convenience function estimates the outcomes of the population in every generation. This paper has developed the discovery strategy of XSS exploits in PHP by implanting that elimination step to the discovery strategy as a whole to identify and eliminate Cross-site scripting. The outcomes have proved the ability of the suggested strategy to identify and eliminate Cross-site scripting exploits in PHP [24]. A logic exploit appears when programmers have produced a relevant mistake in their applications. It might be extremely easier, similar to the decreasing price in calculating it or any aspect, which has occurred completely ignored. This study explains the aspects of entirely automated and standard pen-testing techniques that result in identifying certain exploits.

3 Research Methodology

This research aims to explain the high-risk vulnerabilities in web applications and helping the security developers and penetration testing teams have secure code in their applications. It will also compare the web application scanner statically and dynamically for all selected vulnerabilities. It is needed to set up a workstation on Windows and Linux with all the tools to analyze the vulnerable codes of PHP and JAVA. After setting up the workstation, the vulnerable web application will be examined and explained, in which the vulnerability occurs in the code, and the approach of it can be secured through the code review process. Tools are also run and analyzed statically and dynamically in PHP and JAVA. After completing the vulnerable web application analysis and explaining how the exploits work and how to secure them, the web application scanner will scan the vulnerable web application for any kind of vulnerability and then compare the results of different tools. The benefits associated with the investigation include an in-depth analysis of the selected vulnerabilities within the source codes, gaining insights into the existing vulnerabilities, learning remediation strategies for reducing exposure to the identified vulnerabilities, and providing assistance to the security developer and penetration tester selecting the right scanner tool. The research process is shown in Fig. 1.

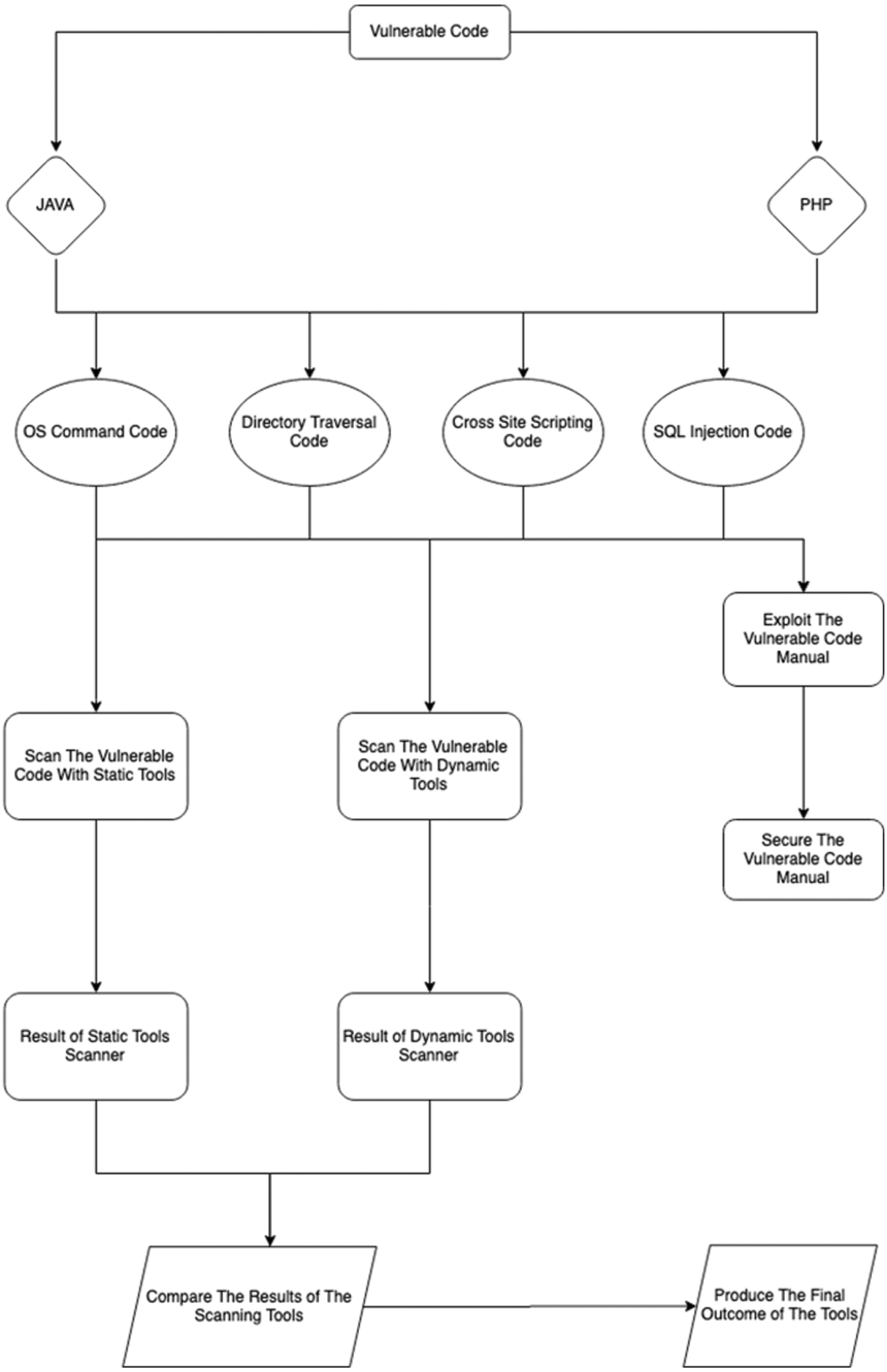

Figure 1: Research process

4 Experimental Setup and Analysis

There are a variety of tools for static as well as dynamic source code scanners (Acuntix [25], NetSparker [26], BurpSuitePro [27], SonarCloud [28], Cobra [29], VulnyCode [30], ASST [31], AppSpider 7 [32], FindBug [33]). Some of them are commercial, and others are free and open source. The support of scanning PHP and JAVA applications allows different vulnerabilities that this study focuses on: SQL, XSS, Path Traversal, and OS Command. It also focuses on the popularity of the tools and accuracy in finding the vulnerability. A basic script to find vulnerabilities in a PHP source code; uses Regular Expression to find vulnerabilities. It can also find out bugs and implements a program that uses static analysis to look for bugs in Java code. Experimental Setup to scan the vulnerability in the web application is shown in Fig. 2. Vulnerability in the web application is scan with the help of static and dynamic web scanners.

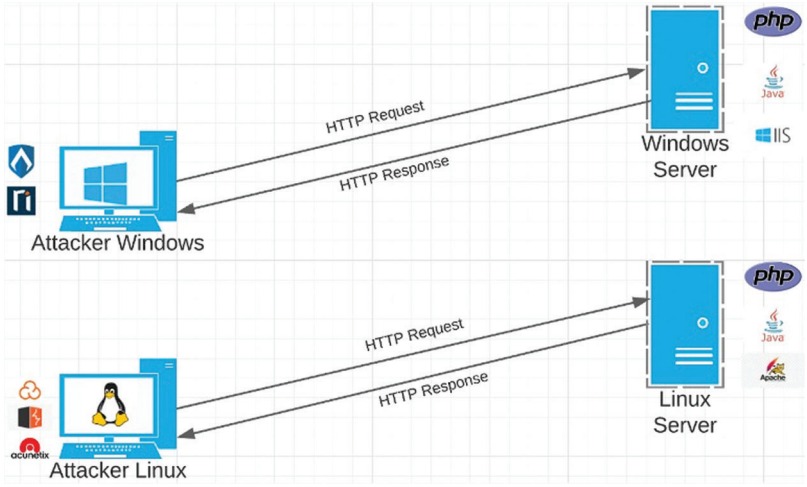

Figure 2: Experimental setup

4.1 Vulnerabilities and Attack Analysis

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries of an application to its database. It usually enables an attacker to view data that is usually unrecoverable. This might be related to data for other users or other data that the application itself can reach. In many cases, an attacker can modify or delete this data, permanently altering the content or performance of the application. SQL injection consistently appears in the Open Web Application Security Project's (OWASP) list of top 10 security risks [34].

4.1.1 SQL Injection in PHP Code

The SQLi attack occurs when an SQL statement selects a specific user with a specified user ID. Fig. 3 shows the PHP SQLi Vulnerable Code 1. Fig. 4 generates a SELECT statement by adding a variable “id” to a select string. The variable is fetched from user input: When there is no validation to prevent a user from entering “malicious” input, the user might enter a "smart" input such as 'AND 1=1'. This SQL statement is correct and returns ALL rows from the Users table because AND 1=1 is always TRUE in the SQL statement. It also indicates that the SQL queries can bypass ACL by avoiding regular authorization and authentication checks, and the distinct SQL queries even may provide a way to host OS-level commands.

```php
<?php
// SQL Injection
if(isset($_GET['id'])){
$user = $conn->query("SELECT * FROM users where id =".$_GET['id']);
?>
```

Figure 3: PHP SQLi vulnerable code 1

```
SELECT * FROM Users WHERE UserId = 105 AND 1=1;
```

Figure 4: PHP SQLi vulnerable code 2

The primary purpose of the code was to build an SQL statement for selecting a user with a provided user id. When there is no validation to prevent a user from entering "malicious" input; the user can enter some "clever" input like provided below:

4.1.2 SQL Injection in JAVA Code

SQL Injection attacks work because, in various applications, the most distinct method is to perform a provided computation and dynamically produce code that is run by another system or component. When we use untrusted data without proper sanitization, we often drop an open door for the intruders to exploit in generating this code. Let us take a view of how this occurs in the following example shown in Fig. 5.

```java
@RequestMapping("/jdbc/vuln")
    public String jdbc_sqli_vul(@RequestParam("username") String username) {

        StringBuilder result = new StringBuilder();

        try {
            Class.forName(driver);
            Connection con = DriverManager.getConnection(url, user, password);

            if (!con.isClosed())
                System.out.println("Connect to database successfully.");

            // sqli vuln code
            Statement statement = con.createStatement();
            String sql = "select * from users where username = '" + username + "'";
            logger.info(sql);
            ResultSet rs = statement.executeQuery(sql);

            while (rs.next()) {
                String res_name = rs.getString("username");
                String res_pwd = rs.getString("password");
                String info = String.format("%s: %s\n", res_name, res_pwd);
                result.append(info);
                logger.info(info);
            }
            rs.close();
            con.close();
```

Figure 5: JAVA SQLi vulnerable code 1

The problem of this specific code is apparent. We have put the *username* value into the query with no validation. Exploiting this query code is trivial, and all the attackers have to send a condition that, when concatenated with the fixed part of the query, it is needed to modify the expected performance:

The SQL statement above (Fig. 6) is correct and will return ALL rows from the "Users" table since OR a=a is always TRUE in the SQL statement.

```
select * from users where username = 'abdo' OR 'a'='a'
```

Figure 6: JAVA SQLi vulnerable code 2

4.1.3 Cross-Site Scripting in PHP Code

Cross-site scripting (XSS) attacks are injections in which malicious scripts are being inserted into trusted websites. XSS happens when an attacker uses a web application to push malicious code, usually on a browser's page, to another end user. Vulnerabilities that allow attacks to work are quite well known and take place in every location. A web application accepts input from a user within the output it produces without validating or encrypting it. When there would be a variable that outputs some data directly to the browser without validation, it can lead to the execution of XSS attacks. An attacker exploits this vulnerability by intercepting the POST or GET requests and injecting the variables with harmless JavaScript codes. In the following example (Fig. 7), a sample of a vulnerable code to XSS in PHP is provided.

```
<?php

foreach($_SESSION['cart'] as $isbn => $qty){
                    $conn = db_connect();
                    $book = mysqli_fetch_assoc(getBookByIsbn($conn, $isbn));
            ?>
            <tr>
                <td><?php echo $book['book_title'] . " by " . $book['book_author']; ?></td>
                <td><?php echo "$" . $book['book_price']; ?></td>
                <td><input type="text" value="<?php echo $qty; ?>" size="2" name="<?php echo $isbn; ?>"></td>
                <td><?php echo "$" . $qty * $book['book_price']; ?></td>
            </tr>
            <?php } ?>

?>
```

Figure 7: PHP XSS vulnerable code 1

The vulnerable variables are ***$gty*** and ***$isbn***. The code echoes the number of books directly in the cart and the book ISBN without sanitization of validation. It results in the execution of an XSS (Fig. 8).

```
http://localhost/bookstore/cart.php

978-1-484216-40-8=<script>alert(1)</script>&save_change=Save+Changes
```

Figure 8: PHP XSS vulnerable code 2

An attacker will exploit this vulnerability by intercepting the ***POST*** request and injecting the variables with harmless javascript codes. In the following example (Fig. 9), a sample of a vulnerable code to XSS in JAVA is provided.

The parameter XSS is unescaped or sanitized. It will print any input it receives from the user. A malicious attacker can inject JavaScript code into the application (Fig. 10).

An attacker will exploit this vulnerability by injecting the ***GET*** request with harmless JavaScript codes. In the following example, the script passes the ***$file_name*** variable an unvalidated/unsensitized HTTP request value directly to the ***$file_dir*** variable to get the path to upload the file to it.

```
@RequestMapping("/xss")
public class XSS {

    * @param xss unescape string

    @RequestMapping("/reflect")
    @ResponseBody
    public static String reflect(String xss) {
        return xss;
    }
```

Figure 9: JAVA XSS vulnerable code 1

```
http://localhost:8080/xss/reflect?xss=<script>alert(1)</script>
```

Figure 10: JAVA XSS vulnerable code 2

4.1.4 Directory Traversal in JAVA Code

A path traversal attack (identified as directory traversal) aims to reach files and directories that are stored outside the Webroot directory [34]. In the example (Fig. 11), a web service receives a parameter named ***filepath*** that corresponds to the name of a file previously in the server's directory. Different attackers can also control the parameter to read files in the system leading to information disclosure. The attacker would initiate the request as following to read files in the system. By manipulating variables that reference files or foldersfile path../) sequences and their changes, or by using absolute file paths, it can likely reach out arbitrary files and directories stored on the system OS, including application source code or important system files (Fig. 12)

```
public class PathTraversal {

    protected final Logger logger = LoggerFactory.getLogger(this.getClass());

    @GetMapping("/path_traversal/vul")
    public String getImage(String filepath) throws IOException {
        return getImgBase64(filepath);
    }
```

Figure 11: JAVA directory traversal vulnerable code 1

```
http://localhost:8080/path_traversal/vul?filepath=../../../../../etc/passwd
```

Figure 12: JAVA directory traversal vulnerable code 2

4.1.5 OS Command Injection in JAVA Code

Command injection attack executes arbitrary commands on the operating system through a vulnerable program or a web application. Command injection attacks are possible when an application passes the insecure user-supplied data to a specific system shell. The OS commands provided by the attacker; are regularly executed with all vital privileges of the vulnerable application [34]. This specific application passes a cmd parameter and executes it without any kind of validation or cleanup, which is vulnerable to command injection via the CommandExec method that results in the execution of OS.

The above example (Fig. 13) is a java application that is vulnerable to command injection; since the method, ***CommandExec*** is passing a ***cmd*** parameter and executing it without validation or sanitization, leading to OS command execution. The attacker would send the request as following for executing commands in the system.

```
public String CommandExec(String cmd) {
        Runtime run = Runtime.getRuntime();
        StringBuilder sb = new StringBuilder();

        try {
            Process p = run.exec(cmd);
            BufferedInputStream in = new BufferedInputStream(p.getInputStream());
            BufferedReader inBr = new BufferedReader(new InputStreamReader(in));
            String tmpStr;

            while ((tmpStr = inBr.readLine()) != null) {
                sb.append(tmpStr);
            }
```

Figure 13: JAVA OS command vulnerable code 1

4.2 Vulnerabilities Mitigations in PHP and JAVA

More than 60% of all codebases used by enterprises contain at least one specific vulnerability from open source elements, according to the “Open-Source Security and Risk Analysis” (OSSRA) report [35]. The main aspect is to check whether the application is exclusive code or open-source code; it will indeed have vulnerabilities. However, the experts overwhelmingly admit that the open-source code libraries ' "components" are highly secured than the commercial applications. The dilemma is not within the use of open-source libraries. Application vulnerabilities thrive because it's complicated to write secure code. As a result, many companies are dependent on open source projects. We will list the processes of mitigating the web applications from the vulnerabilities of XSS, SQL, Path traversal, OS command injection in PHP and JAVA, based on the OWASP Cheat Sheet Series. We will also demonstrate the process of mitigating the vulnerability in the source code below the images. The OWASP Cheat Sheet Series was created to present a dense collection of high-quality knowledge on specific application security issues [36].

To protect the code from SQLi, we can use a function called mysql_real_escape_string in PHP. It is responsible for escaping certain characters in a string for use in a SQL declaration to avoid SQL injection attacks. The developer can use the "PreparedStatement" function to protect and escape the SQL query from SQL injection attacks to avoid SQL injection attacks in JAVA code. Prepared Statements help stop SQL injection because the values that are entered into a SQL query are transmitted to the SQL server; after the actual query is sent to the server. We can use the “htmlspecialchars” function to turn special characters into HTML entities to prevent XSS attacks in PHP codes. Developers can also use a custom method to convert special characters into HTML entities.

The developer uses the "basename ()" function to return only the filename part of a given path to prevent path traversal attacks. However, to make the code more secure, we can use another function called realpath() with the basename() function. It is needed to return the canonicalized absolute pathname and avoid path traversal attacks in PHP codes. Moreover, they can also use a customized method to protect their application from path traversal attacks in codes to check if the parameter contains ".." or "/." The developer can use a function escapeshellarg() in PHP, and in Java to prevent OS Command Injection, and they should use every method that executes a system command and prevent the end-user from supply their inputs in it.

4.3 Comparison with Static and Dynamic Tools

Comparing the tools has been conducted by installing the vulnerable applications in both PHP and java language. They are also vulnerable to SQL injection and cross-site scripting and directory traversal, and os command injection. After deploying the applications in windows server with Java tomcat and Linux server with PHP interpreter scanned vulnerable applications with the dynamic tools like AppSpider, Burp Suite Pro, with a windows machine. Examining the result from these scanners after it finishes scanning, Results are in a report format with all the details, *i.e.*, the vulnerability found, the time it takes to scan the application, etc. Based on the report, a comparison takes place between scanners. In the static approach, we run the static tools scanner locally in Linux machine and using the time command combined with the static tools in a terminal and scan the vulnerable application statically after it finishes the scanning and produces the report of the finding, we compare the result based in the information we get for these scanners in static mode. The sonar cloud scanner is a web-based static scanner. We upload the vulnerable applications to it and scan them. After scanning, the scanner produces a report of the result with all the information (time, the vulnerability found), etc. The FindBug is a plugin tool that scans the vulnerable applications in IDE software and shows a report if there is a vulnerability in the codes (statically). Recorded the time of the plugin when it finishes and if it found the vulnerability in the code.

4.4 Discussion

There are a variety of tools for static source code scanners. A few of them are commercial, and others are free and open source. The selected five tools based on their scanning PHP and JAVA code features to focus on, mainly include SQL, XSS, Path Traversal, OS Command. It is needed to focus on the tools' popularity and the accuracy in finding out the vulnerability. Cobra is a source code security audit tool that supports the detection of major security issues and vulnerabilities in the source code of several development languages. SonarCloud is cloud-based, and it supports twenty-three languages Java, JS, C#, C/C++, Objective-C, and more. It also supports Deep Code analysis to examine all source files. ASST is an open-source developed with JavaScript. It uses Regular Expression to find out vulnerabilities. FindBugs is a program that uses static analysis to look for bugs in Java code.

All dynamic tool scanners are commercial as the large companies put a lot of effort into them. In this study, we have selected four dynamic tools to scan the vulnerable application; they support scanning PHP and JAVA applications, and these are SQL, XSS, Path Traversal, OS Command". Moreover, we have shown a compression between Static and Dynamic Tools in Tab. 1. It also shows that the dynamic tools are excellent in finding vulnerabilities; however, it is slower than static tools. The top vulnerability type found by static and dynamic tools ranks accordingly by SQL Injection, Cross-Site Scripting, Path Traversal, and OS Command Injection. Regarding the above analysis of the tools, we found that the top tools for static tool scanners are FindBug,Vulny, and the top tools for dynamic tool scanners are AppSpider, NetSparker. Fig. 14 shows the comparison between static and dynamic tools against speed and vulnerability. Analysis speed of Cobra, FindBug, Sonar, ASST, and Vulny are higher than others.

Table 1: Compression between static and dynamic tools

Tool Name	Analysis Style	Language Support	Cost Factor	Testing Style	Method	Analysis Speed	Vulnerability Found	Vulnerability Top Language
Acuentix	Dynamic	PHP-JAVA	Commercial	Black	Runtime	Slow	4	PHP&JAVA
AppSpider	Dynamic	PHP-JAVA	Commercial	Black	Runtime	Slow	5	JAVA
Cobra	Static	PHP-JAVA	Free	White	Source	Fast	1	JAVA
FindBug	Static	JAVA	Free	White	Source	Fast	3	JAVA
Sonar	Static	PHP-JAVA	Free	White	Source	Fast	2	PHP
BurpSuitePro	Dynamic	PHP-JAVA	Commercial	Black	Runtime	Slow	4	PHP&JAVA
ASST	Static	PHP	Free	White	Source	Fast	0	None
Vulny	Static	PHP	Free	White	Source	Fast	3	PHP
NetSparker	Dynamic	PHP-JAVA	Commercial	Black	Runtime	Slow	5	JAVA

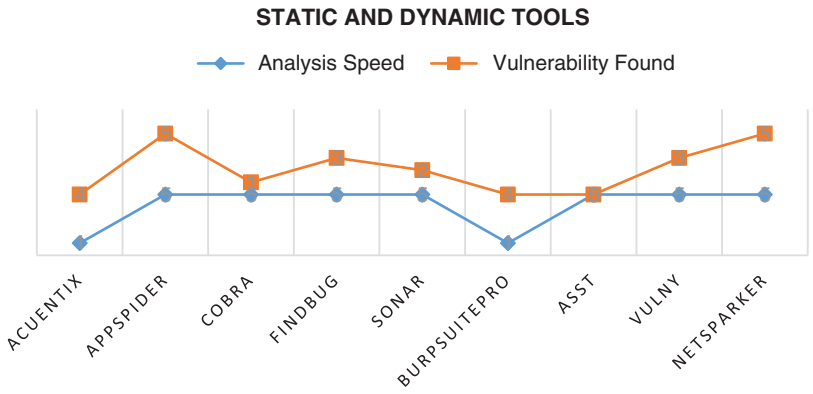

Figure 14: Comparison of Static *vs*. Dynamic tools against analysis speed and vulnerability

5 Conclusions

There are a large number of vulnerabilities and security flaws in applications, which interact on the internet. Developers and penetration testers need a reference point to create or verify that a web application is secure. OWASP is the most reliable standard to refer to for guidelines in the present scenario, and the tools draw one of the best ways to make web application security. New vulnerabilities are found every day, and even the most trivial vulnerability, if properly exploited, can cause significant damage to an organization. SQL injection is the most prevalent in this study, along with path traversal and OS command injection. Moreover, XSS can also lead to serious consequences. In many organizations, application security is the most vital aspect of the application lifecycle that is usually ignored, and cybersecurity is seen as a cost rather than an investment. Experimental results indicate that SQL queries can bypass ACL by avoiding regular authorization and authentication checks, and seldom SQL queries even may provide a way to host OS-level commands. *SQLi in JAVA Code* works because, for various applications, the only way to perform a given computation is to dynamically produce code that is, in turn, run by another system or component. When it uses untrusted data without effective sanitization, it drops an open door for attackers to exploit in generating this code. XSS attack happens when an attacker uses a web application to push malicious code, usually on a browser's page. An attacker exploits this vulnerability by intercepting the POST or GET request and injecting the variables with harmless JavaScript codes.

A path traversal attack aims to reach out to files and directories stored outside the Webroot directory. Attackers can control this parameter to read files in the system leading to information disclosure. Command injection attacks are possible when an application passes insecure user-supplied data to a system shell. The application passes a cmd parameter and executes it without validation or cleanup, which is vulnerable to command injection *via* the CommandExec method, resulting in the execution of OS. To protect the code from SQLi, we can use a function called mysql_real_escape_string in PHP, responsible for escaping certain characters in a string for use in a SQL declaration to avoid SQL injection attacks. To avoid SQL injection attacks in JAVA code, the developer can use the "PreparedStatement" function to protect and escape the SQL query from SQL injection attacks. The developer uses the "basename ()" function to return only the filename part of a given path to prevent path traversal attacks. The users should use every method that executes a system command and prevent the end-user from supplying their inputs to prevent OS Command Injection. The developer can use the function escapeshellarg( ) in PHP and Java.

The four security risks are implemented in the web application, used as an analysis set for evaluating the effectiveness of Acunetix vulnerability scanners, Netsparker vulnerability scanners, Burp Suite Pro vulnerability scanners, AppSpider vulnerability scanners, Cobra vulnerability scanners, Sonar vulnerability scanners, FindBug vulnerability scanners, ASST vulnerability scanners, and Vulny vulnerability scanners. The evaluation of the reputed web application vulnerability scanners is done by examining the results obtained while running web scanners for the vulnerable web application in PHP and JAVA and then comparing the number of vulnerabilities detected.

Several vulnerability scanners might comprise of multiple reports, and they even have several techniques to test particular types of vulnerabilities. The comparison of the tools shows that the top languages in which the tools find security vulnerabilities are JAVA and PHP. It also shows that dynamic tools excel at finding security vulnerabilities but are slower than static tools. The vulnerabilities found by both static and dynamic tools are SQLi XSS, Path Traversal, and OS Command Injection. FindBug and Vulny are the best static, and AppSpider, NetSparker are the best dynamic tools. Therefore, developers and testers should attempt to use more scanners to discover web vulnerabilities. When we exploit vulnerable applications, we can imagine from our views. While possible attackers are evermore more creative than developers, they may have talented ideas to attack the system. Therefore, the evaluation result of these vulnerabilities may not be permanent. In the future, focus on finding complex vulnerabilities such as insecure direct object references related to PHP and JAVA applications. That strategy is referred to suit the workflow of web applications written in server-side programming languages like PHP, JSP, and ASP and even develops an automatic detection and mitigation of XSS vulnerabilities, SQL injection, and other attacks based on machine learning techniques.

Acknowledgement: The authors sincerely acknowledge the support from Majmaah University, Saudi Arabia for this research.

Funding Statement: The authors would like to thank the Deanship of Scientific Research at Majmaah University for supporting this work under Project Number No -R-14xx-4x.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Mishra, S. K. Sharma and M. A. Alowaidi, “Analysis of security issues of cloud-based web applications,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 3, no. 1, pp. 50, 2020.

[2] D. Mitropoulos, P. Louridas, M. Polychronakis and A. D. Keromytis, "Defending against web application attacks: Approaches, challenges and implications," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 188–203, 2019.

[3] H. Tabrizchi and M. K. Rafsanjani, "A survey on security challenges in cloud computing: Issues, threats, and solutions," *Journal of Supercomputing*, vol. 76, no. 12, pp. 9493–9532, 2020.

[4] P. Martins, S. I. Lopes, A. M. R. D. Cruz and A. Curado, "Towards a smart & sustainable campus: An application-oriented architecture to streamline digitization and strengthen sustainability in academia," *Sustainability*, vol. 13, no. 6, pp. 1–25, 2021.

[5] A. Tekerek, "A novel architecture for web-based attack detection using convolutional neural network," *Computers & Security*, vol. 100, no. 2, pp. 102096, 2021.

[6] A. Gkortzis, D. Feitosa and D. Spinellis, "Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities," *Journal of Systems and Software*, vol. 172, no. 2, pp. 110653, 2021.

[7] S. Mishra, M. A. Alowaidi and S. K. Sharma, "Impact of security standards and policies on the credibility of e-government," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 1–12, 2021.

[8] J. Bozic and F. Wotawa, "Planning-based security testing of web applications with attack grammars," *Software Quality Journal*, vol. 28, no. 1, pp. 307–334, 2020.

[9] Y. B. Zikria, R. Ali, M. K. Afzal and S. W. Kim, "Next-generation internet of things (IoT): Opportunities, challenges, and solutions," *Sensors*, vol. 21, no. 4, pp. 1174, 2021.

[10] Z. Zhang, H. Ning, F. Shi, F. Farha, Y. Xu *et al.,* "Artificial intelligence in cyber security: Research advances, challenges, and opportunities," *Artificial Intelligence Review*, vol. 54, pp. 1–25, 2021.

[11] A. Aljumah and T. A. Ahanger, "Cyber security threats, challenges and defence mechanisms in cloud computing," *IET Communications*, vol. 14, no. 7, pp. 1185–1191, 2020.

[12] K. N. Durai, R. Subha and A. Haldorai, "A novel method to detect and prevent SQLIA using ontology to cloud web security," *Wireless Personal Communications*, vol. 117, no. 4, pp. 2995–3014, 2021.

[13] S. Mishra, S. K. Sharma and M. A. Alowaidi, "Multilayer self-defense system to protect enterprise cloud," *Computers, Materials & Continua*, vol. 66, no. 1, pp. 71–85, 2020.

[14] P. Tang, W. Qiu, Z. Huang, H. Lian and G. Liu, "Detection of SQL injection based on artificial neural network," *Knowledge-Based Systems*, vol. 190, pp. 1–10, 2020.

[15] A. Kaur and R. Nayyar, "A comparative study of static code analysis tools for vulnerability detection in c/c++ and java source code," *Proc. Computer Science*, vol. 171, no. 13, pp. 2023–2029, 2020.

[16] A. Gupta, B. Suri, V. Kumar and P. Jain, "Extracting rules for vulnerabilities detection with static metrics using machine learning," *International Journal of System Assurance Engineering and Management*, vol. 12, pp. 65–76, 2020.

[17] M. Liu, B. Zhang, W. Chen and X. Zhang, "A survey of exploitation and detection methods of XSS vulnerabilities," *IEEE Access*, vol. 7, pp. 182004–182016, 2019.

[18] O. C. Abikoye, A. Abubakar, A. H. Dokoro, O. N. Akande and A. A. Kayode, "A novel technique to prevent SQL injection and cross-site scripting attacks using knuth-morris-pratt string match algorithm," *EURASIP Journal on Information Security*, vol. 14, pp. 1–14, 2020.

[19] A. Fidalgo, I. Medeiros, P. Antunes and N. Neves, "Towards a deep learning model for vulnerability detection on web application variants," in *2020 IEEE Int. Conf. on Software Testing, Verification and Validation Workshops*, Porto, Portugal, pp. 465–476, 2020.

[20] C. Li, Y. Wang, C. Miao and C. Huang, "Cross-site scripting guardian: A static XSS detector based on data Stream input-output association mining," *Applied Sciences*, vol. 10, no. 14, pp. 1–20, 2020.

[21] D. E. Simos, J. Zivanovic and M. Leithner, "Automated combinatorial testing for detecting SQL vulnerabilities in web applications," in *2019 IEEE/ACM 14th Int. Workshop on Automation of Software Test*, Montreal, QC, Canada, 55–61, 2019.

[22] H. Gu, J. Zhang, T. Liu, M. Hu, J. Zhou *et al.,* "DIAVA: A traffic-based framework for detection of SQL injection attacks and vulnerability analysis of leaked data," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 188–202, 2020.

[23] 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.

[24] V. K. Malviya, S. Rai and A. Gupta, "Development of web browser prototype with embedded classification capability for mitigating Cross-Site Scripting attacks," *Applied Soft Computing*, vol. 102, no. 3, pp. 106873, 2021.

[25] Acunetix [Online]. Available: https://www.acunetix.com.

[26] NetSparker [Online]. Available : https://www.netsparker.com.

[27] Burp Suite Pro [Online]. Available: https://portswigger.net/burp.

[28] SonarCloud [Online]. Available: https://sonarcloud.io.

[29] Cobra [Online]. Available: https://github.com/WhaleShark-Team/cobra.

[30] VulnyCode [Online]. Available: https://github.com/swisskyrepo/Vulny-Code-Static-Analysis.

[31] ASST, OWASP [Online]. Available: https://github.com/OWASP/ASST.

[32] AppSpider7 [Online]. Available: https://www.rapid7.com/products/appspider.

[33] FindBug [Online]. Available: http://findbugs.sourceforge.net.

[34] Owasp top 10 [Online]. Available: https://owasp.org/www-project-top-ten.

[35] OSSRA report [Online]. Available: https://www.synopsys.com/software-integrity/resources/analyst-reports/2020-open-source-security-risk-analysis.html.

[36] OWASP Cheat Sheet Series [Online]. Available: https://cheatsheetseries.owasp.org/index.html.