electronics

MDPI

Article

Security Analysis of Web Open-Source Projects Based on Java and PHP

Zhen Yin and Scott Uk-Jin Lee *

Department of Computer Science and Engineering, Hanyang University, Ansan 15588, Republic of Korea; thefrozensnow@naver.com

* Correspondence: scottlee@hanyang.ac.kr

Abstract: During website development, the selection of suitable computer language and reasonable use of relevant open-source projects is imperative. Although the two languages, PHP and Java, have been extensively investigated in this context, there are not many security test reports based on their open-source projects. In this article, we conducted separate security analyses on web-related open-source projects based on PHP and Java. To this end, different open-source frameworks and services are used to design websites used to test experimental attacks on 12 popular open-source filters available on GitHub, as well as investigate the use of Lightweight Directory Access Protocol (LDAP) in the Firefox browser environment. Using malicious payloads published by Open Web Application Security Project (OWASP) and others, Cross-site Scripting (XSS), Local File Inclusion (LFI), SQL injection, and LDAP injection are performed on the test targets. The experimental results reveal that although PHP-based open-source projects are more vulnerable to attacks than Java-based ones, there is significant room for improvement. Finally, a whitelist-based filtering scheme is proposed. This scheme filters the inline attributes of label elements so that the filter has an excellent detection rate of malicious payloads while having an excellent pass rate of benign payloads. Effective references and suggestions for web developers are also included to aid the selection of open-source web projects, and feasible solutions to improve filter performance are proposed.

Keywords: PHP; Java; web security; filter; open-source

Citation: Yin, Z.; Lee, S.U.-J. Security Analysis of Web Open-Source Projects Based on Java and PHP. *Electronics* **2023**, *12*, 2618. https://doi.org/10.3390/electronics12122618

Academic Editor: Andrei Kelarev

Received: 14 April 2023
Revised: 2 June 2023
Accepted: 8 June 2023
Published: 10 June 2023

1. Introduction

With the continuous development and improvement of computer languages, numerous excellent languages have emerged in the field of web development. PHP is currently the most-used language, followed by ASP.NET, Ruby, and Java [1]. The selection of a programming language is primarily based on each language's unique features and standard library, I/O operations, operating system, and programming language compatibility for simulations. Furthermore, whilst graphical performance is important, raw performance (with graphics turned off or disabled) is far more so; start-up speed is barely relevant at all [2]. However, many developers prioritize implementing functions and satisfying user requirements over web security, which may lead to vulnerabilities that could be exploited to cause irreparable loss. Besides the strength of the selected language, poor security knowledge among web developers and the blind use of open-source projects also increase vulnerabilities [3]. Although abundant literature and multiple organizations (e.g., Open Web Application Security Project (OWASP)) have argued that web vulnerabilities are a major problem in modern computer languages, most current research analyzes single programming languages or a few types of web attacks [4,5]. In addition, comparative studies have been conducted primarily for attack analysis of Cross-site Scripting (XSS) and SQL injection, and few extensive security performance tests have been performed on web programs written in different programming languages. In early research, the strengths and weaknesses of PHP and Java were compared, revealing that Java is more robust than

PHP, and data analysis was conducted on two common vulnerabilities of XSS and SQL injection [3]. It was concluded that web applications written using Java exhibit fewer vulnerabilities. A recent study evaluated the filtering capabilities of 12 popular XSS filters (a security mechanism to detect whether the user input content conforms to the specification) in terms of success rate and achieving comprehensive XSS filtering data analysis to help developers choose appropriate filters to ensure web safety [6]. However, with the continuous updating of language versions and mining of web vulnerabilities, a single attack test cannot guarantee the data security of web applications composed using specific programming languages.

To resolve this problem, in this study, a comprehensive web security test is performed on web-related open-source projects based on PHP and Java. To this end, different shooting range websites capable of carrying malicious payloads through the ten most popular open-source frameworks are designed [7]. The websites are equipped with the eight most popular filters and four SQL injection filters. The aforementioned experimental environment is used to conduct penetration tests comprising nearly 700 attack codes on XSS, Lightweight Directory Access Protocol (LDAP), Local File Inclusion (LFI), and SQL injection filter mechanisms. The security differences between web programs written in Java and PHP in open-source projects are evaluated by analyzing the filtering results of different shooting ranges corresponding to different attacks. This paper aims to (1) analyze web security differences between Java and PHP open-source projects, (2) provide suggestions on the use of web-related open-source projects for developers using different programming languages, and (3) propose an improved scheme based on the whitelist mechanism that can help web developers improve filter performance.

The remainder of this article is structured as follows. In Section 2, related works are discussed. Section 3 discusses the web attack considered in the experiment. In Section 4, the experimental procedure and results are presented. Finally, Section 5 presents a summary of the experiments and discusses directions for future research.

2. Related Works

This section summarizes the current state of research on open-source projects used in web applications in recent years.

Helmiawan et al. (2020) analyzed the security of open-source web applications through the top ten of OWASP. This study showed the vulnerability of the web application in the experiment and what security risks they had [8], but it does not conduct further research on these security risks and only makes superficial suggestions. In this study, we analyze the open-source web application layer by layer and give reasonable modification advice.

Talib et al. (2021) adjusted the configuration of 12 popular open-source filters and provided suggestions for developers to use filters more reasonably [9]. However, in their research, they only modified some rules for those filters and did not deviate from the strategies of filters, so the suggestions given can only be effective within a certain range. The scheme proposed in this paper can be used for security policy extension.

Shahriar et al. (2016) proposed a method to detect LDAP attacks and implemented it based on PHP [7]. However, the experimental object used in this test is a PHP application written for the experimental test, meaning it can't guarantee the effectiveness of real environment tests such as open-source applications. In order to provide effective suggestions for developers to use open-source applications, this paper selects some popular frameworks that support LDAP and conducts attack detection on the filtering rules of different frameworks to ensure the practicability of the suggestions.

Likaj et al. (2021) conducted the first security assessment of Cross-Site Request Forgery (CSRF) defense on popular web open-source frameworks, identified 16 defense measures and 18 security threats, and discussed the vulnerabilities of three of these frameworks [10]. The integration of popular frameworks in their research provided us with convenience. In this study, the top five frameworks in different languages from the framework rank were selected and used as test objects to conduct a more comprehensive attack test.

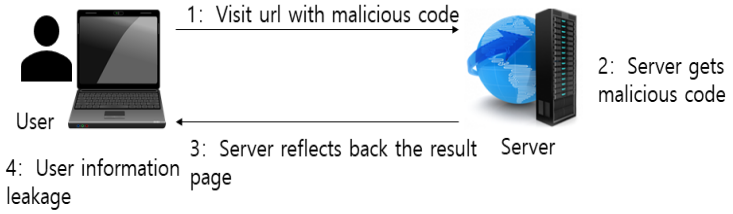
Che et al. (2021) demonstrated LFI vulnerability risks and implemented a LFI detection method based on Tor Proxy. The study pointed out that most of the current vulnerability scanners focus on SQL injection and XSS, so the security detection on LFI is relatively weak, and due to the difference in security awareness of developers, different artificial vulnerabilities will be generated [11]. The test experiment in this study uses the LFI testlevel of Damn Vulnerable Web Application (DVWA), considering the level of different developers, to verify the effectiveness of LFI attack from the attacker's perspective at different levels.

3. Background

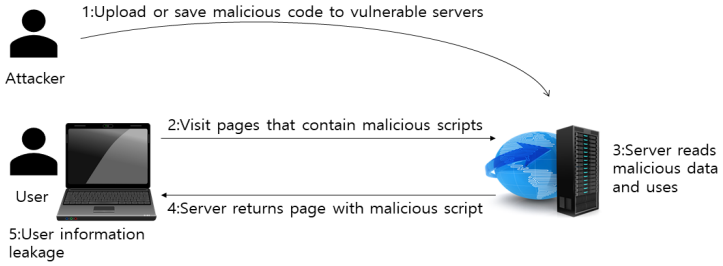
In this chapter, some attack examples relative to the experiment are introduced. These examples include three types of XSS—the demo of SQL injection, PHP and Java's LFI introduction, and the demo of the LDAP attack.

3.1. XSS

XSS remains one of the most prevalent security vulnerabilities in web applications to this day [12]. XSS can be roughly classified into three types of attack methods—reflected XSS, DOM-based XSS, and stored XSS. The attack processes are illustrated in Figure 1. The general methodology of reflected XSS is based on malicious connections. The attacker deploys a connection to the server comprising a JavaScript code to obtain the target's cookie value. The general methodology of stored XSS involves text box content input. Vulnerabilities are more common in social networking site message boards, comment areas, and other areas that allow users to input content freely. Further, in form submission areas, attackers often insert <img> or <script> tags within the content to steal information. Although most websites filter tags by limiting the number of strings in the filter or using regular expressions, code splitting is still active in several small websites. This scenario is discussed in the next section. DOM-type XSS is essentially a special type of reflective XSS—it is based on the DOM document object model. With the rapid development of single-page applications (SPAs), progressively more web programs are being dynamically handled by browsers, which increases the threat of such vulnerabilities. As illustrated in Figure 2, this type of XSS is implemented at the front end without bypassing the server. Thus, this type of vulnerability often exists in websites with tabbed navigation.

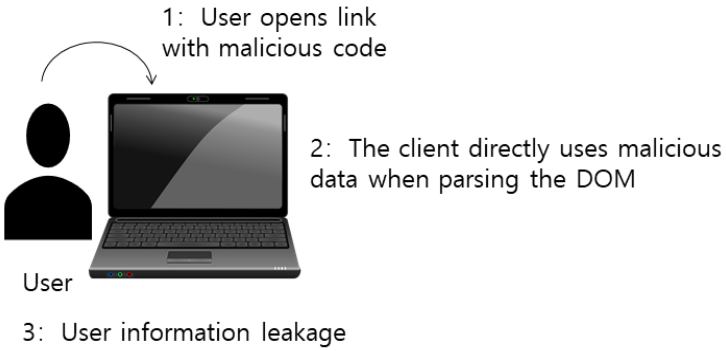


(**a**)



(**b**)

Figure 1. *Cont.*



(**c**)

Figure 1. Types of XSS attack methods. (**a**) Reflect-XSS; (**b**) Stored-XSS; (**c**) Dom-XSS.

```
<script type="text/javascript">
var s = location.search; //Get the query string
for the current URL.

s = "?url=javascript:alert('xsstest')";
s = s.substring)1,s.length); //trip the leading
part of the query string.

var url = "";  //initialize the variable url to an
empty string.

if(s.indexOf("url=")>-1){
   var pos = s.indexOf("url=")+4;
   url = s.substring(pos, s.length);
}else{
   url = "url is null";
}
</script>
<div id="test><a href=""></a></div>
<script type="text/javascript">

document.getElementByld("test").innerHTML
= 'my url is:<a href="'+url+'">"'+url+'"</a>';
</script>
```

Figure 2. Web application with a DOM-Based XSS.

3.2. SQL Injection

SQL injection is a type of attack that targets databases specifically. General web applications utilize databases that are often more likely to be exploited by attackers than XSS vulnerabilities because of their involvement with data storage. Alert Logic Cloud Security recently stated that SQL injection accounted for 55% of the detected attacks [13]. The general test method for this vulnerability involves the construction of simple test code in the username and password input boxes, such as:

$$username'or'1' ='1$$

$$password'or'1' ='1$$

This method is often used on online login pages by attackers. Alternatively, vulnerabilities can be mined by adding &id = 1 and id = '1' after the label in the URL bar. The existence

of vulnerabilities of this type is conformed if no 404 or 403 warning is displayed and the new page's content is identical to that of the old page. In addition, Boolean blind injection, timestamp injection, truncated injection and other methods can be used to determine the effectiveness of SQL injection on web applications.

3.3. LFI

LFI primarily affects web applications written in PHP [14]. When a file is imported through a PHP function, the incoming file name or path is not properly processed. Checking or operating unintended files may lead to accidental file leakage or even malicious code injection. The vulnerable functions in PHP are usually include(), require(), include_once(), and require_once(). In JSP/servlet, they are ava.io.File() and java.io.FileReader() functions instead.

3.4. LDAP

LDAP is a lightweight online directory access protocol; its information storage form is depicted in Figure 3. The usage of web applications has increased manifold recently, and the resources and data of these applications are distributed and stored in directories. Usually, different applications involve directories, called proprietary directories, dedicated to their related data. An increase in the number of proprietary directories leads to the formation of information islands (i.e., information systems that cannot interoperate or coordinate work with each other). This complicates the sharing and management of systems and resources. In this case, some web programs utilize LDAP to facilitate data query and processing, which may make them vulnerable to LDAP attacks [7,15]. LDAP injection is similar to SQL injection—parameters introduced by the user are used to generate LDAP queries, which can be divided into 'and' injection and 'or' injection, as depicted below:

$$(\&(parameter1 = value1)(parameter2 = value2))$$

$$(\|(parameter1 = value1)(parameter2 = value2))$$

If a server opens any of the ports 389, 636, or 3269, it is likely to exhibit LDAP vulnerability. An attacker can exploit this vulnerability to perform blind LDAP injection on test directory attributes and obtain document information [16].

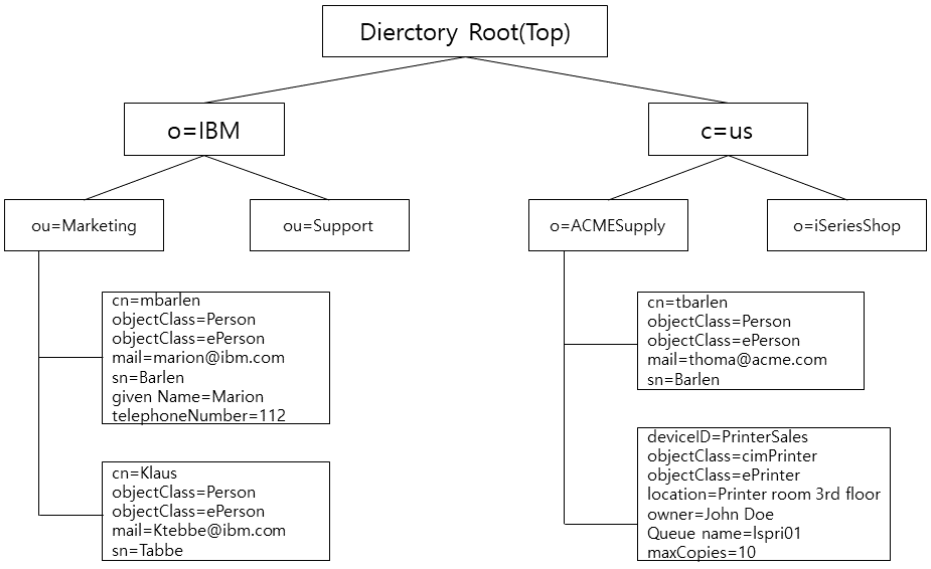


Figure 3. LDAP information tree.

4. Experiments

4.1. Range Construction

To ensure the controllability and effectiveness, five of the top ten Java and PHP web frameworks were selected as per the latest OpenSFF evaluation [10], as listed in Table 1.

Table 1. Experimental framework.

Framework	Version	Language
Spring	5.3.17	Java
Play	2.8.13	Java
Spark	3.1.3	Java
Vaadin	Vaadin23	Java
Vert.x-Web	v4.2.6	Java
Symfony	6.0	PHP
CakePHP	4.2	PHP
Slim	slim4.5.0	PHP
Laravel	9.x	PHP
Zend/Laminas	3.0.1	PHP

Web frameworks with LDAP enabled are indicated in cyan. A common configuration method is used for frameworks that are not equipped with LDAP services.

First, a shooting range was constructed based on the framework, as depicted in the Figure 4, which covers the easily attacked tag types, such as <a href>, <button>, <textarea>, <script>, <form>, <img>, and <onmouse>. Users are allowed to upload files and due to the popularity of HTML5, the <svg> and <canvas> tags may be vulnerable. Then, eight XSS filters and four SQL injection filters popular on GitHub were selected—these are listed in Table 2.

In subsequent experiments, we intend to combine frameworks and filters to evaluate the improvement in security performance by matching different filters after detecting the security levels of different web frameworks.

Table 2. Experimental Filter for XSS and SQL Injection.

XSS		
Filter	**Version**	**Language**
jsoup [17]	1.14.3	Java
Lucy-XSS [18]	1.6.3	Java
XSS HTML Filter [19]	1.5	Java
xssprotect [20]	0.1	Java
HTML Purifier [21]	4.11.0	PHP
PHP Anti-XSS [22]	1.2b	PHP
PHP-XSS-Filter [23]	1.1	PHP
xss_clean [24]	-	PHP
SQL Injection		
InjectionAttackFilter [25]	-	Java
sql-injection-filter [26]	-	Java
phpClassFilter [27]	-	PHP
Web-Security-Filter [28]	-	PHP

"-" means that the filter has no historical version.

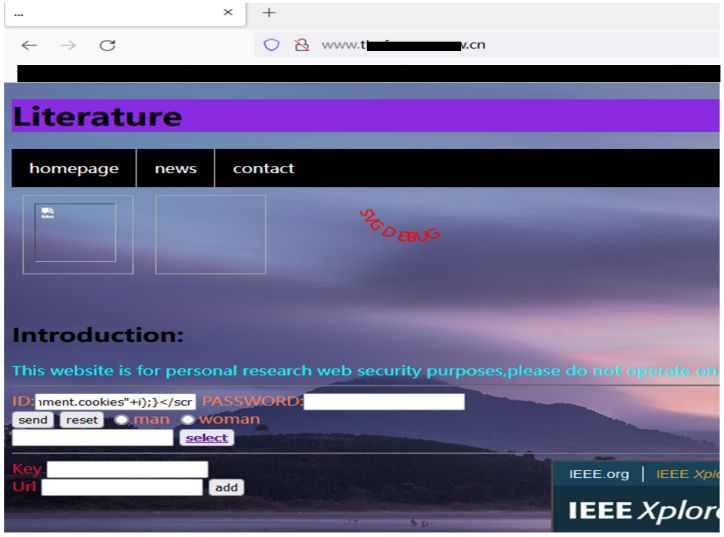


Figure 4. Test range example.

4.2. Data Collection

By combining the list of vulnerabilities provided on QWASP [29] and the open-source payload test code provided by OBB (openbugbounty) as of February 2022, with the cases reported in the existing literature, we collected 676 payloads in aggregate—219 involving XSS, 248 involving SQL, 79 involving LFI, and 130 involving LDAP.

4.3. Detection Method

Although some studies have demonstrated that web application detectors may yield erroneous data, some open-source detectors have been proven to be trustworthy [30]. To ensure the reproducibility of the experiment, we used SQLMAP and Xray, two popular detection tools, to scan the target for vulnerabilities. Xray detects XSS and LFI, and SQLMAP detects SQL injection. As open-source software available on the market does not include LDAP detection, detection via manual injection was performed based on the similarity between LDAP and SQL and the research of Hossain Shahriar et al. [7]. Detection for different purposes is achieved through the native detection of filters and changes to specific filtering rules. All experiments were performed on the Firefox browser.

4.4. Research and Analysis

First, a risk ranking was constructed by counting the past CVE data [31] of each framework (as of 10 January 2022) in Table 3.

Analysis revealed that, among the Java frameworks, the most popular spring framework ranks second in terms of the number of vulnerability submissions, and the attack reports involved in the experiment accounted for 19.04% of the figure. Thus, spring framework exhibits high web security risk, and it is not suitable for programmers who lack experience in web security. In contrast, the Vert.x-Web framework with relatively few users was observed to have exhibited only one vulnerability report in recent years, corresponding to the CSRF attack. Despite this disadvantage, Vert.x-Web is the leading open-source framework in terms of web security. The framework is mostly developer-friendly as it enables decent security even in the absence of relevant expertise; several important observations were noted in the case of PHP frameworks as well. According to Google's PHP framework ranking, Larravel and Symfony are the most used PHP frameworks by a significant margin. However, they were observed to rank second and third in terms of the number of vulnerabilities in Symfony, 23.08% of the vulnerabilities in the report were related to our experiment attack; this indicates that websites using the Symfony framework

are inefficient at filtering attacks, such as XSS. We believe that CakePHP is more suitable for web developers who are proficient in PHP, but not proficient in web security.

Table 3. Risk ranking for Java and PHP.

Java					
Framework Name	**Total Vulnerabilities**	**XSS**	**SQLInjection**	**LFI**	**LDAP**
#1 Spark	47	2	0	1	0
#2 SpringFramework	42	3	3	1	1
#3 Vaadin	22	1	0	3	0
#4 Play	10	0	0	1	0
#5 Vert.x-Web	1	0	0	0	0
PHP					
Framework Name	**Total Vulnerabilities**	**XSS**	**SQLInjection**	**LFI**	**LDAP**
#1 Zend/Laminas	85	6	7	3	1
#2 Symfony	72	7	4	4	1
#3 Laravel	53	9	1	1	0
#4 Slim	23	2	1	4	0
#5 CakePHP	13	2	0	1	0

To illustrate this argument, we conducted further research on Symfony. We configured the PHP-related filters listed in Table 2 sequentially in the Symfony framework used in the experiment and estimated the detection rate with its twig native template [32]. The results are depicted in Table 4.

Table 4. Detection rate testing of Symfony native twig templates and open-source filters.

XSS Detection	
Filter Name	**Detection Rate (%)**
HTML Purifier	97.69
PHP Anti-XSS	99.53
PHP-XSS-Filter	99.53
xss_clean	95.37
Twig1.x	94.44
Twig2.x	95.83
Twig3.x	96.29
SQL Injection Detection	
Filter Name	**Detection Rate (%)**
phpClassFilter	86.69
Web-Security-Filter	94.35
Twig1.x	93.78
Twig2.x	97.74
Twig3.x	98.26

It is evident that even with updated twig to three versions, the rate is less than the XSS detection rate of open-source filters. However, Symfony's twig template exhibits the best defense against SQL injection. Testing revealed that the old version of twig1.x contained reference instances such as '_self'=>' this', which could lead to the use of the '_self' variable to return the /Twig/Template instance; this is indicative of the env property of the Twig_Environment, rendering it vulnerable to attack. The new version changes the role of _self, which improves security, but lacks filtering of name values, resulting in problematic code, e.g., [0, 0] | reduce("system", "calc") detection can be bypassed. Thus, in web applications designed using Symfony, twig templates should be avoided or the

HTML Purifier filter should be used; this is the relevant advice in this article for PHP practitioners.

As the number of risk vulnerabilities related to Java frameworks is significantly smaller than those related to PHP frameworks, comparative tests between Java's native templates and open-source filters were omitted from this study. By Table 3, Java is more secure than PHP in terms of web development. Next, the detection rates of open-source filters in the two languages were compared. To reduce the influence of the framework on the detection rate, Vert.x-Web (JDK8 or above) and CakePHP were used owing to high framework security. Different filters were configured into the framework. For example, the PHP-XSS-Filter was configured into CakePHP by copying its xss_filter.class.php file to the routing directory in CakePHP and adding it through the DispatcheFactory class in config/bootstrap.php. Java filters rely on Maven packages. The filter rules were not changed; meanwhile, the following testing data were acquired.

The filtering and detection results of 216 XSS codes and 248 SQL injection codes using 12 filters are depicted in Table 5. Among the open-source filters in the Java language, the XSS HTML Filter exhibited the highest detection rate exceeding 97.69%, followed by jsoup with 97.22%. Corresponding to SQL injection, all filters used in the experiment exhibited detection rates exceeding 95%. Unfortunately, none of the current popular Java open-source filters were observed to exhibit a detection rate of 100%. In fact, the detection rate of Lucy-XSS was only 67.59%. Thus, these popular filters are not effective in defending against advanced malicious codes, and web developers should avoid using Lucy-XSS. In the experiment, Lucy-XSS filtered out the keywords of the executive function, such as alert(), but failed to filter out the payload executed by eval() plus encoding, as illustrated below:

```
<scr<script>ipt>eval(\u0061\u006c\u0065 \u0072\u0074(1))/></scr</script>ipt>
```

In contrast, among the open-source filters in the PHP language, two XSS filters, PHP-XSS-Filter and PHP Anti- XSS, exhibited 100% filtering under experimental detection. The other two filters also exhibited detection rates exceeding 95%. However, the detection rate of the filters involved in SQL injection detection were low, with the lowest detection rate of 86.69% exhibited by phpClassFilter. Therefore, the filtering rules of phpClassFilter were analyzed in depth and concluded that, although it cannot be injected directly owing to its lack of filtering of SQL method functions, it can be brute-force cracked after method testing. For example, using the following payloads:

```
sel<>ect count(*) from data where name='id' and len(password)<18)>0
```

the malicious code is used to test the length of the password, which is obtained after multiple cycles. After the password length is confirmed, brute-force cracking is performed using the password dictionary.

Although almost all scripting languages provide the function of file inclusion to facilitate programming, LFI attacks primarily exist in PHP programs, which is a drawback of the language design [33,34]. Therefore, in this experiment, the LFI vulnerability of PHP programs was tested. As LFI vulnerabilities may arise from poor coding habits of developers, false-positive data generated by personal factors using DVWA's LFI test were avoided. The injection test of 79 LFI codes corresponding to every level of DVWA was used to obtain the data listed in Table 6.

Testing data indicated the absence of any filtering mechanism at lower levels, while at the medium level, the pass rate following the blacklist mechanism was only 49.36%. At a high level, the whitelist mechanism was used, yielding a rate of only 18.98%. Finally, under the whitelist mechanism at the impossible level, complete defense against exploitable LFI vulnerabilities as of February 2022 was achieved. It is evident from the open-source code of the impossible level that effective defense against LFI is not difficult. As long as the allowed PHP file is not executed, an error is reported. Therefore, although LFI vulnerability is a great threat to web programs written in PHP, relevant defense is not as complicated as that against XSS attacks. Moreover, after updating PHP to version 8.0,

the string in php://filter .strip_tags is removed, which makes PHP-based web application less vulnerable to LFI.

Table 5. Open-source filter detection rate for XSS and SQL Injection.

XSS Detection		
Filter Name	**Language**	**Detection Rate (%)**
xssprotect	Java	96.26
XSS HTML Filter	Java	97.69
Lucy_XSS	Java	67.59
jsoup	Java	97.22
xss_clean	PHP	95.37
PHP-XSS-Filter	PHP	100
PHP Anti-XSS	PHP	100
HTML Purifiler	PHP	97.69
SQL Injection Detection		
Filter Name	**Language**	**Detection Rate (%)**
Sql-injection-filter	Java	97.96
InjectionAttackFilter	Java	95.56
Web-Security-Filter	PHP	94.35
phpClassFilter	PHP	86.69

Table 6. LFI injection test.

Grade	**Policy**	**Passing Rate (%)**
Low	No policy	100
Medium	Blacklist	49.36
Hight	Whitelist	18.98
Impossible	Whitelist	0

Finally, we used the framework with the LDAP service listed in Table 1 to perform the LDAP injection test. The LDAP rules defined in the official LDAP website [35] were followed to obtain the following statistics on the LDAP filter types of the LDAP open-source services of different frameworks:

Spring

PresenceFilters, EqualityFilters, Greater-Or-Equal Filters, OR Filters, AND Filters, NOT Filters, Extensible Match Filters, The String Representation of LDAP Filters.

Spark

PresenceFilters, EqualityFilters, Greater-Or-Equal Filters, OR Filters, AND Filters, NOT Filters, The String Representation of LDAP Filters.

Vert.x-Web

PresenceFilters, EqualityFilters, Greater-Or-Equal Filters, OR Filters, AND Filters, NOT Filters, Substring Filters, Less-Or-Equal Filters, Approximate Match Filters, The String Representation of LDAP Filters.

Symfony

PresenceFilters, EqualityFilters, Greater-Or-Equal Filters, OR Filters, AND Filters, NOT Filters, Extensible Match Filters, The String Representation of LDAP Filters.

Laravel

PresenceFilters, EqualityFilters, Greater-Or-Equal Filters, OR Filters, AND Filters, NOT Filters, Substring Filters, Less-Or-Equal Filters, Approximate Match Filters, The String Representation of LDAP Filters.

Zend

PresenceFilters, EqualityFilters, Greater-Or-Equal Filters, OR Filters, AND Filters, NOT Filters, The String Representation of LDAP Filters.

The data shown in the Figure 5 are obtained by manually injecting 130 malicious codes, and experimental results revealed that among the popular frameworks with LDAP services, only Vert.x-Web-related filtering mechanism completely filtered the malicious code used in the experiment. In terms of effectiveness, it was followed by PHP-based Laravel and Java-based Spring, which exhibited a detection rate of 98.46% and only allowed two malicious codes to pass. The detection rates of Laravel and Spring were equal, but they have different LDAP filtering mechanisms. Some filtering mechanisms may be particularly important, while others could have little effect. Thus, only six types of filtering mechanisms were retained—Presence Filters, Equality Filters, OR Filters, AND Filters, NOT Filters, and Substring Filters—and they were repeatedly tested. The detection rate of Spring was observed to be 95.38% and that of Laravel was 94.61%, which were almost identical to those of other open-source projects, except for Vert.x-Web; this indicates that some filtering mechanisms are particularly important for defense against LDAP attacks. This indicates that expertise in the effects of different filtering mechanisms and the most suitable filtering positions is essential in security personnel. The difference in detection rates between identical filtering mechanisms was attributed to the filtering range within the mechanism, which depends on the professional capacity of the developer. The security analysis of open-source web software has been presented, and security suggestions have been proposed for web developers using open-source projects. In turn, advice to maintain web security and improve the detection rate is presented in this study.

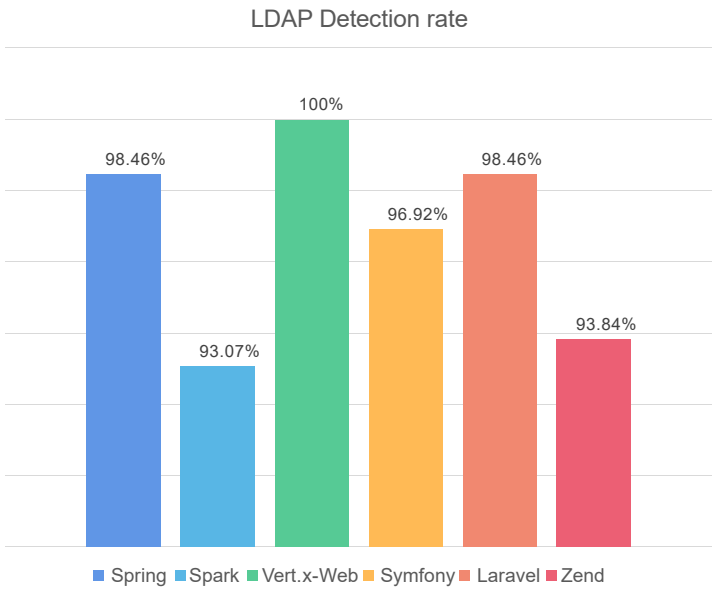


Figure 5. Framework that supports LDAP services and the detection rate of LDAP based on open-source projects.

In the aforementioned experiments, the effect of both PHP- and Java-based filter mechanism were better in case of whitelisted ones than blacklisted ones. However, in practical filters, prevention of the transmission of malicious code must coexist with the transmission of normal code [36]. For example, if the <img> tag is added to the blacklist, users will be blocked from uploading images freely, while if it is added to the whitelist, it will become an injection point for further processing. To make the experimental results more obvious, Lucy-XSS and PHP-Anti-XSS were selected using the whitelist mechanism and PHP-XSS-Filter using the blacklist mechanism, and manually entered 100 benign codes, such as <img src='picture'>, into the system. The transmission rate test was carried out, and the results are listed in the following Table 7.

Table 7. Pass rate detection of benign codes.

Filter	Lucy-XSS	PHP-XSS-Filters	PHP-Anit-XSS
Pass rate (%)	99	60	10

The results indicated that, although the detection rate of Lucy-XSS was low, its transmission rate for benign codes was close to 100%. However, the detection rate of PHP-Anti-XSS was 100%, but benign codes were almost completely filtered out. In actual web applications, allowing free inputs by users is nearly impossible. The transmission rate of PHP-XSS-Filters was 60%—thus, it is the most suitable open-source filter for actual web applications. To improve the transmission rate of this filter, some contents of the PHP XSS filters blacklist were altered to a whitelist. However, this strategy failed as the priority of the blacklist is higher than that of the whitelist [37], and consequently, even when previously blacklisted content is whitelisted, the blacklist still takes precedence. Conversely, if the content in the blacklist is deleted directly, the detection rate falls short of 100%. For example, deleting the case check causes '.php' and '.PHP' to generate file parsing errors and form an attack vulnerability. From this perspective, it is not advisable to improve the filter performance using blacklist mechanisms with a 100% detection rate. Next, we attempted to improve the performance of Lucy-XSS and PHP-Anti- XSS using the whitelist mechanism. The blacklist and whitelist mechanisms of PHP-Anti-XSS were opened, and the test results were compared when only the whitelist mechanism was opened. The results are presented in Table 8.

Table 8. PHP-Anti-XSS test.

Mechanism	Detection Rate (%)	Pass Rate (%)
Whitelist	100	10
W&B	100	8

The test results reveal a transmission rate of 8% and a detection rate of 100%. Therefore, when a whitelist mechanism is used to achieve a 100% detection rate, the use of a blacklist does not affect its detection rate of malicious payloads. However, it may have a negative impact on the transmission rate of benign payloads.

Motivate by the excellent transmission rate for benign payloads and weak detection rate for malicious payloads of the Lucy-XSS filter, the experimental results of the experiments filters were compared, and corresponding code analysis. It is assumed that tag elements such as <img>, <a>, <button> cannot be placed on the blacklist because this would restrict the relevant embedded elements and attributes greatly, degrading the transmission rate for benign payloads. Elements capable of carrying malicious payloads, such as attribute elements, should be blacklisted or screened. To establish this, a simple blacklist was added to the Lucy-XSS filter, add all attribute elements such as 'href' in the whitelist, to the blacklist. The results were as expected—the detection rate of the Lucy-XSS filter increased to 100%, but its transmission rate decreased to 74%. It was found that most of the benign payloads that were not transmitted were <a href=""> and <img src=>. However, most users use image and link functions. To resolve these disadvantages, 'src' and 'herf' were re-labeled and inserted back into the whitelist, 'http://' was added to the content after 'herf', and a strip_tags() function was processed after the 'src' element, so that the elements in it can only return such as 'png'. The code is given in Figure 6. After re-testing, the detection rate of malicious payloads was observed to be 100%, and the transmission rate of benign payloads was 96%, which satisfies the expectations of our experiment. It is important to indicate that this detection rate is affected by the experimental environment and has limitations. This method to has not yet been applied to other open-source filters, and the difference in payloads will also lead to fluctuations in the detection rate, but the two-way improvement of the filter is certain.

```
Step1: a[]<- Match the content after 'src'

Step2:  Judging the content of a
      if  a Include .png  then
          bgein
            b<- indexOf(.png);// Java retrieval and insertion methods
            a<-  insert(b,">");
          else if a include .jpg then
                b<- indexOf(.jpg);
                a<-  insert(b,">");
          else
                a<-strip_tags(a,"src");

          end
```

Figure 6. Design idea to improve the performance of the filter to build on the feasibility of the Java function representation used to improve Lucy-XSS in the experiment in replacing other language functions, such as PHP.

5. Conclusions

In this work, four attack methods that threaten web security were investigated—XSS, SQL injection, LFI, and LDAP. First, the security of web-related open-source software developed based on PHP and Java was analyzed, yielding unexpected results. The most popular PHP-based framework, Laravel, and the most popular Java-based framework, Spring, performed poorly in terms of web security (at least in terms of the attacks involved in the experiment). Vert. x-Web, with relatively fewer users, was observed to be the most secure web development framework. Moreover, although some frameworks provide supporting templates to improve the security of web applications, their scope of defense is limited, making them unsuitable for web applications with high-security requirements. Furthermore, some open-source filters were observed to have the potential to completely filter malicious payloads. However, maximizing the transmission rate of benign payloads to improve the interaction between users and web applications remains to be addressed in this context. In addition, experiment results also revealed that the performance improvement space of the whitelist mechanism is bigger than that of the blacklist mechanism, which implies that it can more effectively identify and validate the allowed operations or objects, thereby enhancing the system's detection accuracy. However, blacklist mechanisms may require more frequent updates and adjustments to identify newly emerging forbidden content, which could potentially lead to a decrease in detection rates. Moreover, for tag elements and attribute elements, the selection of an appropriate filtering algorithm is essential to improve filter performance. In general, this study demonstrates that the popularity of open-source projects does not guarantee high web security. Instead, optimal choice and implementation of open-source projects and improvement of the security of web applications is dependent on the web security knowledge and understanding of diverse web attack methods of developers. This research has experimental limitations, and results rely on attack testing in experimental test environments. However, the result shows that developers cannot blindly choose popular frameworks and rely on open-source filters, but must have a comprehensive understanding of them. At the same time, the traditional black-and-white list mechanism is not enough to defend against attacks. The defense detection mode combined with artificial intelligence (AI) is the current tendency. A more secure framework should have its own AI detection API, which can directly deploy defense mechanisms during development. In the future, we intend to continue to improve the bidirectional performance of the filter to provide better security strategies for web developers. Simultaneously, following the current trend of combining web security and AI, we intend to construct a network attack detection model based on Natural Language Processing.

Author Contributions: Conceptualization, Z.Y.; methodology, Z.Y.; software, Z.Y.; validation, Z.Y.; formal analysis, Z.Y.; investigation, Z.Y.; resources, S.U.-J.L.; data curation, Z.Y.; writing—original draft preparation, Z.Y.; writing—review and editing, S.U.-J.L.; visualization, Z.Y.; supervision, S.U.-J.L.; project administration, S.U.-J.L.; funding acquisition, S.U.-J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2022-00155885, Artificial Intelligence Convergence Innovation Human Resources Development (Hanyang University ERICA)).

Data Availability Statement: The publicly available data analysed in this study can be found in GitHub at https://github.com/yinzhen1996/paperdata, accessed on 9 June 2023.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Marashdih, A.W.; Zaaba, Z.F.; Suwais, K. Cross Site Scripting: Investigations in PHP Web Application. In Proceedings of the 2018 International Conference on Promising Electronic Technologies (ICPET), Deir El-Balah, Palestine, 3–4 October 2018; pp. 25–30.
2. Tushnytskyy, R.; Levus, Y.; Branec, I. Computer language benchmarks tool. In Proceedings of the VIIth International Conference on Perspective Technologies and Methods in MEMS Design, Polyana, Ukraine, 11–14 May 2011; pp. 211–212.
3. Seixas, N.; Fonseca, J.; Vieira, M.; Madeira, H. Looking at web security vulnerabilities from the programming language perspective: A field study. In Proceedings of the 2009 20th International Symposium on Software Reliability Engineering, Mysuru, India, 16–19 November 2009; pp. 129–135.
4. Wang, Y.; Sang, Y. An Empirical Study of Security Risks of PHP Open-Source Software. *Int. J. Softw. Eng. Knowl. Eng.* **2014**, *24*, 1073–1086.
5. Choudhary, S.; Kaur, P. A Study of Security Vulnerabilities on Java Web Application Frameworks. In Proceedings of the 2018 3rd International Conference on Computing Sciences (ICCS), Wuxi, China, 11–13 June 2018; pp. 167–171.
6. Weinberger, J.; Saxena, P.; Akhawe, D.; Finifter, M.; Shin, R.; Song, D. A systematic analysis of XSS sanitization in web application frameworks. In Proceedings of the 16th European Symposium on Research in Computer Security, Leuven, Belgium, 12–14 September 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 150–171.
7. Shahriar, H.; Haddad, H.; Bulusu, P. OCL Fault Injection-Based Detection of LDAP Query Injection Vulnerabilities. In Proceedings of the 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, GA, USA, 10–14 June 2016; pp. 455–460.
8. Helmiawan, M.A.; Firmansyah, E.; Fadil, I.; Sofivan, Y.; Mahardika, F.; Guntara, A. Analysis of web security using open web application security project. In Proceedings of the 2020 8th International Conference on Cyber and IT Service Management (CITSM), Pangkal, Indonesia, 23–24 October 2020; pp. 1–5.
9. Talib, N.A.A.; Doh, K.G. Assessment of Dynamic Open-source Cross-site Scripting Filters for Web Application. *KSII Trans. Internet Inf. Syst. (TIIS)* **2021**, *15*, 3750–3770.
10. Likaj, X.; Khodayari, S.; Pellegrino, G. Where We Stand (or Fall): An Analysis of CSRF Defenses in Web Frameworks. In Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses, San Sebastian, Spain, 6–8 October 2021; pp. 370–385.
11. Ku Mohd Sahidi, K.A.H.H.B.C.; Ariffin, M.A.M.; Ramli, M.I.; Kasiran, Z. Local File Inclusion Vulnerability Scanner with Tor Proxy. In Proceedings of the 2021 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuala Terengganu, Malaysia, 13–15 September 2021; pp. 244–249.
12. Cui, Y.; Cui, J.; Hu, J. A survey on xss attack detection and prevention in web applications. In Proceedings of the 2020 12th International Conference on Machine Learning and Computing, Shenzhen, China, 15–17 February 2020; pp. 443–449.
13. Rai, A.; Miraz, M.M.I.; Das, D.; Kaur, H. SQL Injection: Classification and Pre- vention. In Proceedings of the 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, UK, 28–30 April 2021; pp. 367–372.
14. Available online: https://owasp.org/www-community/vulnerabilities/PHP_File_Inclusion (accessed on 10 April 2023).
15. Alonso, C.; Bordón, R.; Beltrán, A. LDAP Injection & Blind LDAP Injection in Web Applications. Available online: https://www.blackhat.com/presentations/bh-europe-08/Alonso-Parada/Whitepaper/bh-eu-08-alonso-parada-WP.pdf (accessed on 10 April 2023).
16. Vazquez, A. OpenLDAP Replication. In *Practical LPIC-3 300*; Apress: Berkeley, CA, USA, 2019; pp. 157–180.
17. Available online: https://github.com/jhy/jsoup (accessed on 10 April 2023).
18. Naver Corp. Lucy-XSS. Available online: https://github.com/naver/lucy-xss-filter (accessed on 10 April 2023).
19. Available online: https://github.com/finn-no/xss-html-filter (accessed on 10 April 2023).
20. Toonstra, G. xssprotect. Available online: https://code.google.com/archive/p/xssprotect/ (accessed on 10 April 2023).
21. Yang, E.Z. HTML Purifier-Filter Your HTML the Standards-Compliant Way! Available online: http://htmlpurifier.org/ (accessed on 10 April 2023).

22. Available online: https://github.com/voku/anti-xss (accessed on 10 April 2023).
23. Mario. PHP-XSS-Filter. Available online: https://github.com/JBlond/PHP-XSS-Filter (accessed on 10 April 2023).
24. Bijon, M. xss_clean. Available online: https://gist.github.com/mbijon/1098477 (accessed on 10 April 2023).
25. Available online: https://github.com/acslocum/InjectionAttackFilter (accessed on 10 April 2023).
26. Available online: https://github.com/stu17682/sql-injection-filter (accessed on 10 April 2023).
27. Available online: https://github.com/aaronmx/phpClassFilter (accessed on 10 April 2023).
28. Available online: https://github.com/aboutstudy/Web-Security-Filter (accessed on 10 April 2023).
29. Owasp. Available online: https://owasp.org/www-project-top-ten/ (accessed on 10 April 2023).
30. Bursell, M. Open Source and Trust. In *Trust in Computer Systems and the Cloud*; Wiley: Hoboken, NJ, USA, 2022; pp. 211–231.
31. Available online: https://cve.mitre.org/ (accessed on 10 February 2023).
32. Available online: https://twig.symfony.com/ (accessed on 10 February 2023).
33. Begum, A.; Hassan, M.M.; Bhuiyan, T.; Sharif, M.H. RFI and SQLi based local file inclusion vulnerabilities in web applications of Bangladesh. In Proceedings of the 2016 International Workshop on Computational Intelligence (IWCI), Dhaka, Bangladesh, 12–13 December 2016; pp. 21–25.
34. Yenduri, R.; Al-khassaweneh, M. PHP: Vulnerabilities and Solutions. In Proceedings of the 2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), Cairo, Egypt, 8–9 May 2022; pp. 391–396.
35. Available online: https://ldap.com/ldap-filters/ (accessed on 10 February 2023).
36. Hendler, J.; Shadbolt, N.; Hall, W.; Berners-Lee, T.; Weitzner, D. Web science: An interdisciplinary approach to understanding the web. *Commun. ACM* **2008**, *51*, 60–69. [CrossRef]
37. Kim, D.; Lee, J. Blacklist vs. whitelist-based ransomware solutions. *IEEE Consum. Electron. Mag.* **2020**, *9*, 22–28. [CrossRef]