Beware of The Vulnerability!

How Vulnerable Are GitHub's Most Popular PHP Applications?

Ahmed Ibrahim [1,2]
[1] Mentor, A Siemens Business
78, El Nozha Street, Heliopolis, Cairo
11361, Egypt
[2] Faculty of Computers and Artificial Intelligence, Cairo University
ahmed_ibrahim@mentor.com

Mohammad El-Ramly
Faculty of Computers and Artificial Intelligence, Cairo University
5, Ahmed Zewail Street, Dokki, Giza
12613, Egypt
m.elramly@fci-cu.edu.eg

Amr Badr
Faculty of Computers and Artificial Intelligence, Cairo University
5, Ahmed Zewail Street, Dokki, Giza
12613, Egypt
a.badr@fci-cu.edu.eg

***Abstract*—The presence of software vulnerabilities is a serious threat to any software project. Exploiting them can compromise system availability, data integrity, and confidentiality. Unfortunately, many open source projects go for years with undetected ready-to-exploit critical vulnerabilities. In this study, we investigate the presence of software vulnerabilities in open source projects and the factors that influence this presence. We analyzed the top 100 open source PHP applications in GitHub using a static analysis vulnerability scanner to examine how common software vulnerabilities are. We also discussed which vulnerabilities are most present and what factors contribute to their presence. We found that 27% of these projects are insecure, with a median number of 3 vulnerabilities per vulnerable project. We found that the most common type is injection vulnerabilities, which made 58% of all detected vulnerabilities. Out of these, cross-site scripting (XSS) was the most common and made 43.5% of all vulnerabilities found. Statistical analysis revealed that project activities like branching, pulling, and committing have a moderate positive correlation with the number of vulnerabilities in the project. Other factors like project popularity, number of releases, and number of issues had almost no influence on the number of vulnerabilities. We recommend that open source project owners should set secure code development guidelines for their project members and establish secure code reviews as part of the project's development process.**

Keywords— Software Security, Open Source, Software Vulnerability, Vulnerability Detection, RIPS, CVSS, PHP, GitHub, Scanning, Cross-Site Scripting

I. Introduction

Information Technology systems consist of three overlapping circles: IT infrastructure, software, and people. Each circle could be subject to certain types of attacks that aim at penetrating the system and exploiting it. Infrastructure (consisting of networks, routers, servers, etc.) can be subject to denial of service, man-in-the-middle, and other types of attacks. People can be manipulated by social engineering to obtain unauthorized access to otherwise restricted computing systems. Software can be targeted by numerous types of attacks that exploit the presence of vulnerabilities in them. A software vulnerability is a shortcoming in the software that provides an attacker with the opportunity to compromise the system's integrity, availability, or confidentiality. A vulnerability can be exploited by criminals to steal money, compromise sensitive data, or cause other forms of harm.

Unfortunately, software is an easy layer to attack. Vulnerability exploitations are easy to implement in many cases; e.g., just through simple manipulation of the inputs injected to the system. The good news is that this layer is also easy to defend and protect. Secure code development practices, automated vulnerability scanners, prevention techniques, secure code reviews and security testing, among others, are all well-known and established practices to secure software systems. However, many developers and their project managers are not aware of the problem and solutions or do not systematically apply these solutions under work pressures and deadlines. Nevertheless, tackling the problem seriously saves time, cost, resources and reputation.

The first step to remedy this situation is understanding the scale of the problem, its severity and the factors that contribute to it. This study is one step in that direction. It analyzes the currently existing software vulnerabilities in the top 100 applications in PHP hosted in GitHub to find out the scale of the problem and investigate its severity and causes.

The rest of the paper is organized as follows: Section 2 presents the motivation and the research questions we answer in this work. Section 3 presents some related works. Section 4 describes the experimental settings used, the experiment conducted and the results obtained. Section 5 is an analysis and discussion of the results, along with very brief recommendations. Section 6 concludes the paper and presents the future work directions to further extend and validate the obtained results.

II. Motivation and Research Questions

A. Research Motivation

Software vulnerabilities cause major concerns when it comes to securing software systems. If exploited, they can cause major harm to system owners and users alike. Unfortunately, most of such vulnerabilities stem from poor secure software development practices. As recognizing the problem is the first step to treat it, it is essential to understand how common vulnerabilities are in software systems, how severe they are and what factors contribute to their presence. This topic is still under-researched.

To that end, we conducted an experimental study on the current presence of undetected software vulnerabilities in PHP open source projects. Our focus was on PHP projects because PHP applications are used intensively on daily basis. They are an integral part of modern daily life as e-commerce sites, educational sites, online banking systems, online gaming sites, etc. Security vulnerabilities in web applications

978-1-7281-5052-9/19/$31.00 ©2019 IEEE

may result in stealing of confidential data, harming data integrity and/or hindering web application availability. [1]

While PHP applications are only one segment of the software spectrum, the study gives indicative results of the level of the problem and its severity in general and in open source and web applications in particular.

B. Research Questions

Based on the above motivation, we carried out an investigation of the presence of software vulnerabilities in the top 100 open-source PHP applications in GitHub to answer the following research questions.

1) ***Qeustion 1****. How vulnerable are PHP open source software applications?*

2) ***Question 2****. What vulnerabilities are most common in PHP open source software applications? And to what extent? What is their impact?*

3) ***Question 3****. What factors contribute to open source project vulnerability (and hence to its lack of security)?*

4) ***Question 4****. How does the experience of the contributors and number of contributors influence the presence of the vulnerabilities in vulnerable files?*

III. Related Work

Software vulnerabilities and secure software practices received a great deal of attention in research. Numerous studies are directed to identifying, classifying, detecting and fixing various types of software vulnerabilities, using various program analysis techniques. Describing these studies is out of the scope of this research. The same applies to open source systems, which are intensively studied from various aspects, including studies on the presence of specific kinds of bugs in open source software like performance bugs, concurrency bugs, etc.

On the other hand, little research was done to study how common undetected vulnerabilities are in current releases of open source projects, what kind of vulnerabilities are most common and what factors impact the presence of vulnerabilities. Most studies rely on using readymade datasets of known vulnerabilities like the National Vulnerability Database or using software repository metadata like bug reports, commits, etc., instead of diving into the source code in search for undetected vulnerabilities as we did in this work.

Edwards and Chen [2] examined the history of open source releases of Sendmail, Postfix, Apache httpd, and OpenSSL using static source code analysis. They show that the change in number and density of issues reported by the analyzer is indicative of the change in rate of discovery of exploitable bugs for new releases. This shows that software quality does not always improve with each new release. They found that the rate of discovery of exploitable bugs starts to drop 3 to 5 years after the initial release.

Zhou and Sharma [3] used machine learning to develop an automated real-time vulnerability identifier based on project commits and bug reports. They trained it on bug reports and commits collected from thousands of open source projects. It is capable of identifying undisclosed bugs with good recall and precision.

Li and Paxon [4] conducted a large-scale empirical study of security patches, evaluating over 4,000 security fixes across 682 software projects. Their focus was studying the relation between security patches and vulnerability detection. They found shortcomings in addressing security vulnerabilities timely after publically disclosing them, providing windows of opportunity for attacker exploitation.

Bosu *et al*. studied the characteristics of vulnerable code changes, and identified the characteristics of developers likely to introduce vulnerabilities. They found that the less experienced contributors' changes to code were 1.8 to 24 times more likely to be vulnerable than changes done by experienced developers. They also found that modified files are more likely to contain vulnerabilities than new files. [5]

Ben Othmane *et al.* found, through a survey, that the expertise and knowledge of developers is one of the major factors, among others, that influence the time and effort required to fix a security vulnerability. [6]

Nunes *et al.* found that combining the outputs of several free Automated Static Analysis Tools (ASATs) does not always improve the vulnerability detection performance. Thus, the best solution can be a single tool or a combination of tools that may not include all the tools under evaluation [7]. Results of this study support our decision to use one static analysis tool and analyze as many projects as we can instead of using multiple tools and analyzing fewer projects.

IV. Experiment

A. Choice of Projects

We analyzed the latest release of the top 100 PHP applications in GitHub with the highest number of stars, as of December 2018. Stars indicate the project's popularity. We chose PHP projects as PHP is currently the most popular server-side language, with 79% of websites using it [8]. Plus, there is no shortage of vulnerability detection tools for PHP. The full list of the projects analyzed is included in Appendix A.

B. Choice of Vulnerability Scanner and Scoring System

We chose a static analysis tool as opposed to dynamic analysis tools, which are also known as Dynamic Application Security Testing Tools (DAST). This is because dynamic analysis exposes vulnerabilities by examining the information attained during program execution. So, DAST tools require the system to be deployed and running in order to scan it for vulnerabilities. While static analysis works by examining the source code. We analyzed the source code of each project in search of potentially exploitable vulnerabilities.

We used OWASP's selection criteria for source code analysis tools [9]. We inspected various tools from this perspective and chosen one that fulfills these criteria.

To detect vulnerabilities, we used RIPS (Re-Inforce PHP Security) [10]. RIPS is a popular static code analysis tool to automatically detect vulnerabilities in PHP applications. RIPS has 2 versions, open-source version (PHP) and a commercial version (Java and PHP).

We built our experiment using the open-source version which tokenizes PHP code based on PHP's tokenziers extension and perform semantic analysis to build a program model. Its strength is the ability to scan PHP applications very fast for PHP-specific vulnerabilities. It supports the detection of 15 different vulnerability types, including Cross-

Site Scripting, SQL Injection, Local File Inclusion, and others. For each vulnerability, an integrated code viewer can be opened in order to highlight the affected code lines in the original source code for easy remediation. The weakness of the open-source version are false positives due to no use of an abstract syntax tree or control flow graph. So, to avoid false positives we reviewed the code for each vulnerability manually to build our results on real vulnerabilities and exclude duplicated and false ones. We followed the below guidelines to analyze each vulnerability reported by RIPS:

1. If the detected vulnerability is clear and straightforward, it is reported as a real vulnerability.
2. If the code marked as containing a vulnerability is not certain to cause a vulnerability or to be reached, we apply manual data flow analysis to ensure that it is a real reachable vulnerability that can be exploited. Then, the vulnerability is reported.
3. If the vulnerability reported by RIPS doesn't pass either test point 1 or 2 so we consider it as a false positive result and reject it.

To further analyze these discovered vulnerabilities in terms of severity, we used the Common Vulnerability Scoring System (CVSS) version 3.0 [11, 12] and an available online CVSS version 3.0 calculator, provided by the National Vulnerability Database (NVD) [13]. CVSS is a numerical measure that reflects the severity of the analyzed type of vulnerability, based on its principal characteristics. One of a set of pre-defined values is assigned to each characteristic based on experts' judgment. We did this exercise based on our experience, to the most common vulnerability only; i.e. XSS.

C. Experiment and Results

The chosen projects were scanned with RIPS version 0.5 to detect any software vulnerabilities in them. The results of the analysis are shown in Tables I and II. Table I shows that, in total, 193 vulnerabilities were discovered with average of 7.15 (5.38 without outliers, which are values that fall beyond three standard deviations from the mean [14]) vulnerabilities per unsecured application. This average is biased by a small number of projects with a high number of vulnerabilities. However, a more representative measure is the median value of 3 vulnerabilities per vulnerable project. It is important to note that no SQL Injection (SQLI) vulnerabilities were discovered in these almost 7 MLOC.

The most dominant vulnerability was XSS and 6 of the 13 discovered vulnerabilities are on Open Web Application Security Project (OWASP) top 10 list [15]. Table II shows the 13 vulnerabilities discovered in these projects, the number and percentage of occurrences of each one, the number of projects suffering from this vulnerability and whether this vulnerability is on OWASP top 10 list or no. Table III shows the correlation coefficients of project factors (e.g., size, number of commits, number of releases, etc.) with the number of vulnerabilities found in it. In the next section, we analyze these results in details.

Seven projects with a number of vulnerabilities above average were selected. These are Matomo (Web Analytics), OwnCloud Core (Web server), NextCloud Server, YOURLS (URL Shortener), Typecho, Yii PHP Framework and OpenCart. Table IV shows the properties of the projects.

For each vulnerable file in these seven projects, Table V shows the numbers of vulnerabilities, contributors and average years of experience of the contributors for this file. Experience metric is taken to be the number of years of contribution on the project from the date the contributor joined the project to the current date. Then we averaged this measure for all contributors to that specific file.

Table V also shows conflicting values of the correlation coefficient between the number of contributors and the number of vulnerabilities present in it, ranging from –ve to +ve to insignificant. One can conclude from this that there is no evidence that the number of contributors influences the presence of vulnerabilities.

TABLE I. VULNERABILITY SCANNING RESULTS OF TOP 100 PHP PROJECTS

	Attribute	Number
1.	Number of scanned projects	100
2.	Number of scanned files (non PHP file are ignored)	83666
3.	Number of scanned Lines of Code (LOC)	6,803,499
4.	Number of vulnerable projects	27
5.	Number of vulnerabilities found	193
6.	Median Number of vulnerabilities per vulnerable projec	3
7.	Avg Number of vulnerabilities per vulnerable project	7.15 (or 5.38)*
8.	Average types of vulnerabilities per vulnerable project	2.56
9.	Average number of vulnerabilities per file	0.0023
10.	Average number of vulnerabilities per KLOC	0.0283

* After excluding an outlier project with 53 vulnerabilities.

TABLE II. LIST OF VULNERABILITIES DETECTED IN THE TOP POPULAR 100 PHP PROJECTS IN GITHUB (AS OF DECEMBER 2018)

	Name of Vulnerability	Num. of Vul. of This Type	%	Num. of Projects Affected	In OWASP Top 10 Vul.?
1.	Cross-Site Scripting (XSS)	84	43.51	12	Yes
2.	Possible Flow Control (PFC)	24	12.41	12	
3.	File Manipulation (FM)	20	10.41	10	
4.	File Disclosure (FD)	17	8.81	5	
5.	HTTP Response Splitting (HRS)	12	6.21	6	
6.	Code Execution (CD EX)	11	5.71	5	Yes
7.	Command Execution (CMD EX)	7	3.61	4	Yes
8.	Session Fixation (SF)	7	3.61	7	Yes (Broken Authentication)
9.	Protocol Injection (PI)	4	2.11	3	
10.	PHP Object Injection (POI)	3	1.61	2	Yes
11.	LDAP Injection (LDAPI)	2	1.00	1	Yes
12.	File Inclusion (FI)	1	0.50	1	
13.	Reflection Injection (RI)	1	0.50	1	
	Total	**193**	**100**	**27***	

* Some projects have multiple types of vulnerabilities.

TABLE III. CORRELATION COEFFICIENT BETWEEN PROJECT FACTORS AND THE NUMBER OF VULNERABILITIES

	Factor	Corr. Coef.		Factor	Corr. Coef.
1.	Number of Issues	0.26	7.	Number of Contributions	0.27
2.	Number of Pull Requests	0.46	8.	Number of Lines of Code	0.38
3.	Number of Forks	0.23	9.	Number of Scanned Files	0.17
4.	Number of Commits	0.40	10.	Number of Stars	-0.05
5.	Number of Branches	0.51	11.	Number of Watches	0.09
6	Number of Releases	0.22			

TABLE IV. PROPERTIES OF SELECTED VULNERABLE PROJECTS

Project	Stars	Commits	# Cont	# Pull Req	# Vul
Matomo	10308	25416	217	50	9
OwnCloud	6421	39680	459	151	10
NextCloud	6159	47424	574	100	11
Typecho	5178	1108	48	29	21
Yii	4858	6582	266	2	10
OpenCart	4559	9215	267	192	19
YOURLS	4432	1432	43	7	11

TABLE V. PROPERTIES OF VULNERABLE FILES AND CORRELATION WITH THE NUMBER OF VULNERABILITIES IN THE 7 SELECTED PROJECTS

	NextCloud Server (rel 461)						OwnCloud (rel 401)						
Vul File	1	2	3	4	5	6	1	2	3	4	5	6	7
# Vul	4	1	1	2	1	2	4	1	1	1	1	1	1
# Cont	2	12	20	11	1	2	3	15	14	2	7	12	7
Av Yr Exp	3	1.5	6.1	3.6	3	1.7	1.3	4.4	5	2	3	2.5	1
Corr Coeff	Avg. Yrs. Exp.				**-0.48**		Avg. Yrs. Exp.				**-0.42**		
	# Contributors				**-0.48**		# Contributors				**-0.47**		

	YOURLS (rel 13)						TypechoFramework (rel 11)						
Vul File	1	2	3	4	5	6	1	2	3	4	5	6	7
# Vul	2	3	1	2	2	1	3	5	1	1	1	1	1
# Cont	2	5	2	7	2	15	4	6	2	4	2	2	1
Av Yr Exp	7.5	4.6	8	5.2	8	3.6	5	4.2	6	6	2	2	6
Corr Coeff	Avg. Yrs. Exp.				**-0.18**		Avg. Yrs. Exp.					**0.08**	
	# Contributors				**-0.34**		# Contributors					**0.85**	

	Matomo Files (rel 535)						Yii Framework(rel 42)			
Vul File	1	2	3	4	5	6	1	2	3	4
# Vul	1	3	2	1	1	1	1	1	4	4
# Cont	10	5	6	1	1	4	1	1	21	7
Av Yr Exp	6.3	7.8	3.8	2	3	3.8	10	9	5.3	3.6
Corr Coeff	Avg. Yrs. Exp.					**0.67**	Avg. Yrs. Exp.			**-0.96**
	# Contributors					**0.18**	# Contributors			**0.80**

	OpenCart (rel 39)					
Vul File	1	2	3	4	5	6
# Vul	1	1	3	1	12	1
# Cont	3	4	2	1	1	2
Av Yr Exp	2.33	1.75	2	2	1	1.5
Corr Coeff	Avg. Yrs. Exp.				**-0.77**	
	# Contributors				**-0.51**	

V. ANALYSIS OF THE RESULTS

In this section we discuss the results of the experiment, answer the research questions, discuss the limitations of this study and provide brief recommendations.

A. Discussion of the Results

This section discusses the results of this experiment.

- First, one in four PHP projects includes at least one vulnerability. But for vulnerable projects, the median value was 3 vulnerabilities per project. These are undetected untreated ready-to-exploit vulnerabilities. Many of these projects are tools and frameworks used to develop further web applications. This means that their vulnerabilities propagate to further web applications. This is a disturbing result.
- Injection vulnerabilities are the dominant category, including XSS, CD EX, CMD PI, POI, LDAPI and RI. In this category, the attacker injects something (code, command or data) to temper system's behavior.
- XSS comes at the top of the list at 43.5% despite of being well known, well-researched and easy to remedy. Twelve percent (12%) of the projects are infected with it.
- Surprisingly, SQL Injection (SQLI) did not come up in the scan of almost 7 MLOC. This reflects much awareness of SQLI among open source web developers but far less awareness of XSS.

B. Answers to Research Questions

Question 1. *How vulnerable are PHP open source software applications?* At the first glance, 73% of the projects were vulnerability-free. This sounds like great news. However, if you flip the coin, this means that about 1 in 4 projects is subject to currently undetected and untreated vulnerabilities in its most recent release. These vulnerabilities can be exploited to cause harm, steal money, compromise data integrity or system availability, etc. The vulnerable projects included cloud servers, e-commerce tools and framework, a blogging framework, and popular PHP frameworks, tools and libraries used to develop web applications. While attackers need only one backdoor to get into the system, a vulnerable system had a median value of 3 back doors to exploit. As such, we can conclude that open source projects are less secure than desirable and that open source community needs more awareness and better secure code development practices and processes.

Question 2. *What vulnerabilities are most common in PHP open source software applications? And to what extent? What is there impact?* It is surprising that SQLI, the number one software vulnerability, was not found in the huge volume of code analyzed. This shows big awareness among developers of SQLI and the harm it can cause. Numerous serious attacks were carried out with SQLI that exploited IT systems and caused the loss of huge sums of money and very sensitive data. On the other hand, XSS, despite being a serious threat to web applications too, is the most common vulnerability with 12% of the projects suffer from it with a median value of 3 occurrences per vulnerable project. XSS made 43.5% out of all detected vulnerabilities. Next to XSS are file related vulnerabilities (FM + FD + FI) which add up to about 20% of the vulnerabilities. Next are other types of injection vulnerabilities (other than XSS) which allow attackers to inject something (code, command or data) to temper the system's behavior. These include CD EX, CMD EX, PI, POI, LDAPI and RI and they add up to 14.5%. Next are PFC vulnerabilities at 12.4%. Due to space limitations, we only analyzed XSS using CVSS version 3.0 to determine the severity of XSS and the urgency and priority of the response, given that it is the most common vulnerability. NVD CVSS calculator gave a base score of 9.3 which maps to a critical vulnerability that requires immediate attention not only in the open source project but also in any system that was built on it.

Question 3. *What factors contribute to open source project vulnerability (and hence to its lack of security)?* Table III indicates that popularity factors (stars and watches) have no impact on the number of vulnerabilities detected. Factors related to the number of releases, number of issues and number of contributions also have trivial impact. Factors related to project activities are the most influential with moderate correlation with the number of vulnerabilities detected in the project. Activities are represented by the numbers of branches, pull requests and commits. Close to that is the total project size in LOC. One concludes from this result that the busier the project is, the more subject to vulnerabilities it is. The explanation could be that, intuitively, if you work more, you err more.

Question 4. *How does the experience of the contributors and number of contributors influence the presence of the vulnerabilities in vulnerable files?* By close examination of the files containing vulnerabilities and statistical analysis using linear correlation, Table V shows moderate to strong negative correlation between the average years of experience of contributors to the file and the number of vulnerabilities found in it in 4 projects, +ve correlation in one project and insignificant result in two projects. Despite not fully conclusive, this result suggests that for the majority of the projects the experience factor reduces vulnerabilities. The more experienced the contributors, the less the number of vulnerabilities present in the file. While intuitive, our experiment proves this result statistically. Table V also shows conflicting values of the correlation coefficient between the number of contributors and the number of vulnerabilities present in it, ranging from –ve to +ve to insignificant. One can conclude from this that there is no evidence that the number of contributors to a vulnerable file influences the presence of vulnerabilities.

C. Limitations of The Results

The focus of this study was the most popular PHP open source projects in GitHub. Vulnerability scanning was done using static analysis with RIPS which discovers a pre-defined list of threats. While these choices are reasonable given the research questions of this paper, it is dangerous to over generalize the results. Each of these choices imposes a limitation on the results of the study, and possibly a threat to its validity.

While being the most popular server side web development language, PHP has a low entry barrier, making it a favorite language for web development beginners. This results in big amounts of naïve insecure code. On the other hand, open source projects usually are open to many eyes for inspection. So, it is fair to assume that the selected sample of projects fairly represent web-based open source projects.

Open source projects are only one type of software. Being open is double-edged sword. Source code is open to both critiques and attackers alike. If it is likely for vulnerabilities to be spotted faster, it is also likely to exploit them faster and at a larger scale. Would the findings of this study apply to closed source software? This needs further research and experiments. Some studies suggest that security, reliability and the severity of vulnerabilities found of both types are indifferent. [16, 17]

Static analysis is one way of finding vulnerabilities through examination of one aspect of the software, namely the source code. Dynamic analysis, symbolic execution and other techniques exist. Static analysis is most suitable when it is not desirable or possible to run the target system. Other methods can provide different insights and further enrich the results by exploiting other aspects of the software.

D. Recommended Actions

While detailed recommendations are out of the scope of this paper, it is important to advise at least a very brief remedy to this "vulnerable" state of secure software development in open source projects. In particular, XSS is very well studied and can be treated very easily. Many scanners are available for PHP code for example that can easily detect vulnerabilities and propose remediation. Remediation techniques themselves are well known and very easy to deploy, e.g. encoding (escaping) and validation (rejection or sanitization) in case of XSS and other injection attacks. Additionally secure code reviews are well established and many secure code checklists are available. We recommend the following for open source project owners and contributors:

1- Establish secure code development guidelines for the project and require project contributors to follow them.
2- Develop an automatic or manual secure code review process to review the code in search for vulnerabilities before releasing it, or even better, before committing it.
3- Scan existing code bases by suitable scanners, depending on the nature of the project in search for currently undetected vulnerabilities to fix them.

VI. Conclusion and Future Work

In this paper we analyzed the 100 top popular open source PHP applications, which are the most popular PHP projects in GitHub. Our goals were to find to what extent software vulnerabilities exist in these prestigious projects, which vulnerabilities are most common and what factors influence the presence of vulnerabilities.

We found that 27% of these projects are vulnerable to attacks that can exploit the poorly secured parts of the code, with a median value of 3 vulnerabilities per project of the vulnerable projects. Furthermore, we found that the most common vulnerability type is injection vulnerabilities with XSS at the top of the list. Project popularity, number of releases and issues hardly influence the number of vulnerabilities detected. While project activities (branching, pulling and committing) and project size have moderate correlation with the number of vulnerabilities.

We found moderate to strong –ve correlation between the average years of experience of the contributors and the presence of vulnerabilities, in the majority of the projects. The number of contributors of a file had no impact on the presence of vulnerabilities.

Further extensions and validations of these research findings are needed. Future work can explore multiple directions. This can include repeating the same experiment on top popular open source projects in other popular web languages like JavaScript and on other types of applications, other than web applications, like mobile and desktop applications, etc. It also includes using other vulnerability scanners based on dynamic analysis. It also includes repeating the same study on closed and commercial software.

REFERENCES

[1] A. Petukhov and D. Kozlov, "Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing". Proceedings of the Application Security Conference, 2008.

[2] N. Edwards, L. Chen, "An historical examination of open source releases and their vulnerabilities", Proceedings of the 2012 ACM SIGSAC Conference on Computer and Communications Security (CCS 2012), pp. 183-194, 2012.

[3] Y. Zhou, A. Sharma, "Automated identification of security issues from commit messages and bug reports", Proc. 2017 11th Joint Meeting Foundations of Software Engineering, pp. 914-919, 2017.

[4] F. Li, V. Paxson, "A Large-Scale Empirical Study of Security Patches", Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017), pp. 2201-2215, 2017.

[5] A. Bosu, J. Carver, M. Hafiz, P. Hilley, P. and D. Janni, "Identifying the characteristics of vulnerable code changes: an empirical study". In Proceedings of 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014), pp. 257-268, 2014.

[6] L. ben Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, A. Brucker and P. Miseldine, "Factors impacting the effort required to fix security vulnerabilities". Proceedings Of International Information Security Conference, pp. 102-119, Springer, Sep. 2015.

[7] P. Nunes, I. Medeiros, J. Fonseca, N. Neves, M. Correia, and M. Vieira. "An Empirical Study on Combining Diverse Static Analysis Tools for Web Security Vulnerabilities Based on Development Scenarios". (Computing, 101(2)), pp. 161–185, 2019.

[8] W3 Techs, https://w3techs.com/technologies/details/pl-php/all/all

[9] OWASP, https://www.owasp.org/index.php/Source_Code_Analysis_Tools

[10] RIPS, http://rips-scanner.sourceforge.net

[11] CVSS, https://www.first.org/cvss

[12] M. Schiffman, G. Eschelbeck, D. Ahmad, A. Wright and S. Romanosky, "CVSS: A Common Vulnerability Scoring System," National Infrastructure Advisory Council (NIAC), 2004.

[13] NVD, https://nvd.nist.gov/vuln-metrics/cvss

[14] C. Mertler and R. Vannatta, "Advanced and Multivariate Statistical Methods: Practical Application and Interpretation", 3rd Edition, Pyrczak, Los Angeles, 2005.

[15] OWASP: OWASP Top 10 – 2017: The Ten Most Critical Web Application Security Risks. The Open Web Application Security Project, 2017.

[16] A. Boulanger, "Open Source versus Proprietary Software: Is One More Reliable and Secure than the Other", IBM Systems J., vol. 44, no. 2, pp. 239, 2005.

[17] G. Schryen, "Is open source security a myth", Communications of the ACM, vol. 54, no. 5, pp. 130-140, 2011.

APPENDIX A. LIST OF PHP PROJECTS USED IN STUDYING VULNERABILITY IN OPEN SOURCE WEB APPLICATIONS AND THEIR STAR RATINGS, NUMBER OF FILES AND VULNERABILITIES DETECTED (AS OF DECEMBER 2018)

	Project (GitHub Repo)	Rel. #	Stars	Scanned Files	# Vul.		Project (GitHub Repo)	Rel. #	Stars	Scanned Files	# Vul.
1.	Laravel (Framework for web artisans)	88	47670	55	0	**51.**	Slim Framework	84	9435	47	0
2.	The Symfony Framework	396	19258	4235	4	**52.**	Symfony Console	371	5696	176	3
3.	Faker (PHP Library)	10	18552	570	0	**53**	Phpstan (Static analysis tool)	35	5635	381	0
4.	Composer (PHP Dependency Manager)	57	17425	515	2	**54.**	Twig	129	5634	698	0
5.	CodeIgniter	36	16769	281	2	**55.**	Prophecy Framework	28	5501	89	0
6.	DesginPatternsPHP	N/A	16321	165	0	**56.**	EmailValidator	29	5478	63	0
7.	Laravel/framework (Core framework)	374	15125	1220	0	**57.**	Dompdf	13	5453	78	0
8.	Guzzle	116	14821	65	3	**58.**	Php-pm	6	5439	29	0
9.	Yii 2 Framework	30	12516	949	1	**59.**	Symfony thanks	10	5406	3	0
10.	Phpunit	529	12292	529	4	**60.**	Http-foundation	371	5359	142	0
11.	PHPMailer	51	12198	82	2	**61.**	Random_compat	59	5301	10	0
12.	Carbon	78	11058	443	5	**62.**	Predis	37	5286	288	0
13.	PHPExcel	14	10595	386	0	**63.**	Php-code-coverage	130	5255	93	0
14.	Phabricator	N/A	10337	5759	2	**64.**	Laravel-permission	99	5239	24	0
15.	Matomo (Web Analytics)	535	10308	2542	9	**65.**	Typecho	11	5178	204	21
16.	Grav (Flat-file CMS)	150	9908	241	1	**66.**	Event-dispatcher	371	5141	18	0
17.	Parsedown (PHP Parser)	74	9495	6	0	**67.**	Symfony finder	371	5116	41	0
18.	Cphalcon Framework	91	9338	668	0	**68.**	Symfony debug	323	5084	52	0
19.	Cachet	51	8976	1075	0	**69.**	Polyfill-mbstring	15	5023	5	0
20.	Whoops	46	8878	51	0	**70.**	Pagekit CMS	32	5001	516	1
21.	PHP-Parser	50	8652	445	0	**71.**	Http-kernel	371	4984	247	0
22.	Laravel Debugbar	92	8489	39	0	**72.**	PhpSpreadsheet	14	4974	325	0
23.	Flarum	9	8376	2	0	**73.**	Woocommerce	336	4949	815	53
24.	Image (Image Manipulation)	102	8369	195	0	**74.**	Symfony process	371	4916	26	0
25.	Phpdotenv	22	8349	11	0	**75.**	Symfony routing	371	4903	141	0
26.	Mobile-detect	64	8333	43	0	**76.**	Yii PHP Framework	42	4858	1800	10
27.	Flysystem	113	8096	89	0	**77.**	Respect/Validation	95	4839	940	0
28.	October CMS	125	7738	551	1	**78.**	Var-dumper	250	4819	83	0
29.	CakePHP framework	336	7724	1091	2	**79.**	Sylius Framework	86	4636	2769	0
30.	UUID (PHP library)	46	7559	77	0	**80.**	Php-timer	11	4572	4	0
31.	Wechat	129	7402	280	0	**81.**	OpenCart	39	4559	2910	19
32.	Jwt-auth	41	7166	67	0	**82.**	Humhub (Social Network)	54	4556	15119	3
33.	Workerman Framwork	52	7016	26	0	**83.**	Symfony/Translation	371	4521	116	0
34.	Swiftmailer (Mailing	60	6956	288	0	**84.**	Ratchet (WebSocket server)	24	4491	66	0
35.	PHP-CS-Fixer	127	6893	381	0	**85.**	Css-selector	371	4466	75	0
36.	Goutte (Web Scraper)	22	6880	3	0	**86.**	YOURLS (URL Shortener)	13	4432	198	11
37.	Mockery (Mocking framework)	22	6831	154	0	**87.**	Php-file-iterator	3	4425	4	0
38.	Psysh (A REPL for PHP)	63	6611	220	1	**88.**	Php-token-stream	40	4413	29	0
39.	Reactphp (Event driven in PHP)	18	6602	1	0	**89.**	Oauth2-server	83	4397	45	0
40.	Magento2	115	6535	22774	8	**90.**	Php-text-template	9	4366	1	0
41.	Php-ml (Machine Learning)	13	6516	129	0	**91.**	Sebastian-comparator	20	4360	36	0
42.	OwnCloud core	401	6421	2241	10	**92.**	Sebastian-exporter	10	4269	2	0
43.	Deployerphp (Deployment tool)	61	6376	117	0	**93.**	Sebastian-environment	23	4251	6	0
44.	Monica CRM php	55	6341	1190	0	**94.**	Sebastian-version	9	4245	1	0
45.	Laravel-Excel	118	6234	96	0	**95.**	Botman Framework	53	4234	71	0
46.	Nextcloud-server	461	6159	3071	11	**96.**	Wechat-php-sdk	1	4231	27	3
47.	ThinkPHP5 Framework	52	6066	19	1	**97.**	Manifest	4	4191	84	0
48.	Clean-code-php	N/A	5905	1	0	**98.**	Flarum-core	11	4178	484	0
49.	PHP_CodeSniffer	71	5817	813	0	**99.**	Recursion-context	6	4165	4	0
50.	Google-api-php-client	34	5667	20	0	**100**	Mledoze-countries	21	4098	10	0