P ISSN : 2618-1630
E ISSN : 2909-6130
Recognized Journal

Analysis of Code Vulnerabilities in Repositories of GitHub and RosettaCode: A Comparative Study

Original Article

Abdul Malik [1,2], Muhammad Shumail Naveed [1]
[1] Department of Computer Science & Information Technology, University of Balochistan, Quetta, Pakistan.
[2] Department of Computer Science, Government Postgraduate Science College Quetta, Pakistan.
* Correspondence: Abdul Malik: **amalikgspc@gmail.com**

Citation | Malik. A, Naveed. S. M, "Analysis of Code Vulnerabilities in Repositories of GitHub and RosettaCode: A Comparative Study". International Journal of Innovations in Science and Technology. Vol 4, Issue 2, 2022, pp: 499-511
Received | May 19, 2022; **Revised** | June 01, 2022; **Accepted** | June 10, 2022; **Published** | June 20, 2022.
DOI:https://doi.org/10.33411/IJIST/2022040219

Open-source code hosted online at programming portals is present in 99% of commercial software and is common practice among developers for rapid prototyping and cost-effective development. However, research reports the presence of vulnerabilities, which result in catastrophic security compromise, and the individual, organization, and even national secrecy are all victims of this circumstance. One of the frustrating aspects of vulnerabilities is that vulnerabilities manifest themselves in hidden ways that software developers are unaware of. One of the most critical tasks in ensuring software security is vulnerability detection, which jeopardizes core security concepts like integrity, authenticity, and availability. This study aims to explore security-related vulnerabilities in programming languages such as C, C++, and Java and present the disparities between them hosted at popular code repositories. To attain this purpose, 708 programs were examined by severity-based guidelines. A total of 1371 vulnerable codes were identified, of which 327 in C, 51 in C++, and 993 in Java. Statistical analysis also indicated a substantial difference between them, as there is ample evidence that the Kruskal-Wallis H-test p-value (.000) is below the 0.05 significance level. The Mann-Whitney Test mean rank for GitHub (Mean-rank=676.05) and Rosettacode (Mean-rank=608.64) are also different. The novelty of this article is to identify security vulnerabilities and grasp the nature severity of vulnerability in popular code repositories. This study eventually manifests a guideline for choosing a secure programming language as a successful testing technique that targets vulnerabilities more liable to breaching security.

Keywords: Software, Vulnerability, Software Security, Programming Portal, Vulnerability Severity.

Acknowledgment.
I would like to give a sincere thanks to GitHub, and Rosettacode, for freely providing source code and SourceForge for the Statistical Analysis tool Yasca setup.
Project details. Nil

CONFLICT OF INTEREST:
The author(s) declare that the publication of this article has no conflict of interest.

Author's Contribution
Conceptualization, methodology, visualization, and data curation by Malik A. Validation, statistical analysis, and supervision by Naveed S. All authors have read and agreed to the published version of the manuscript.

INTRODUCTION

According to different researchers, software consists of specialized formats, i.e., data structures that allow software developers to manipulate data frequently, as well as a set of electronic instructions executed on a machine. The software is also an accompanying document that explains how to set it up, use and run it [1]–[4]. Software products are employed in most aspects of an individual's life, [5]. A programming language is a tool for creating computer-coded instructions [6]. Code repositories provide a centralized location for hosting code and are a viable, cost-saving solution for software developers and the scientific community to cooperate. It is one of the guiding principles for rapid prototyping of software [7]. The open-source code is included in 99% of the commercial software [8]. Although open-source software (OSS) is widely used in software applications, it is seldom examined, and corporations have no concept of what they don't know about open source, as Luszcz [9] pointed out, illustrating the necessity to address that space. Many vulnerabilities have been reported in code that causes catastrophic software security problems [10]–[13]. In a decisive study, Reiss [14] deliberated those insecure applications are not only a threat to an individual but also a severe threat to people, organizations, and even national security at all levels. Three fundamental principles for security are Confidentiality, Integrity, and Availability [15], [16]. Common Vulnerabilities and Exposures (CVE) describes security vulnerabilities as severe issues for three of them. CVE defined "Vulnerabilities" as severe weaknesses in software and hardware components that an attacker can exploit to cause detrimental effects on the applications or its environment's security, confidentiality, integrity, or availability. Vulnerabilities, according to Russell et al. [17] and Zhou et al. [18], instigate by insecure code. Luszcz [9] and Brazhuk [19] stress in their work that software security is a crucial problem, and even a single vulnerability may have a significant impact on software security. However, only one of the software vulnerabilities might result in a successful destructive attack implementation. However, not all vulnerabilities have the same likelihood of exploitation, security issues, or maintenance requirements. Vulnerabilities have varying risk levels [20], and are categorized as critical, high, medium, low, or none. One of the frustrating aspects of security-sensitive bugs or vulnerabilities is that vulnerabilities manifest themselves in hidden ways that software developers are unaware of [17]. One of the crucial steps for assuring software security is the detection of vulnerabilities in the program's code before deploying the application product [12]. This study would comparatively analyze vulnerabilities in C++, C, and Java, which are available on GitHub and Rosettacode. The TIOBE index, which assesses the popularity of programming languages, puts C++ fourth, C second, and Java third [21], although the C, C++, and Java languages have severe Common Weakness Enumeration(CWE) type vulnerabilities, 87 in C++ [22], 83 in C [23], and 78 in Java [24] out of 924. This study's novelty would serve as a guideline for software developers to select a secure programming language and a portal in terms of vulnerabilities. This also familiarizes the code quality in open-source programming code hosted in code repositories by assessing the probable vulnerabilities and their severity risk scores. This study's ultimate objective would be to facilitate the development of secure software less likely to cause security breaches.

LITERATURE REVIEW

Security vulnerabilities are critical issues in open source code which is 99% in commercial software, and can undermine the development of software security [8], [10], [11], [25] Several vulnerabilities have been sorted in code that cause catastrophic software security issues [11], [12]. Furthermore, Zhao et al. [26] recognized vulnerabilities as inescapable security risks in open source code, one of four common evaluation perspective indicators: quality, reliability, maturity, and vulnerability. The National Vulnerability Database [27] publishes vulnerabilities that have been recognized as common vulnerabilities and include an identification number, at least one public reference, and a comprehensive explanation. It's getting worse, with 173267 vulnerabilities reported to date. A significant research effort has been devised to mitigate security issues of software systems as follows. In his groundbreaking research, Zhang et al. [28] reported the presence of vulnerabilities in code snippets hosted on Stack Overflow, a website that contains millions of solutions to programmer difficulties. Recklessly sharing such code snippets can expose the application or its environment to security risks. The sample size of 646,716 code snippets of the programming languages C/C++ was scanned for weaknesses, and 12998 Common Weakness Enumeration instances were identified, with 32 code weaknesses. The six most frequently occurring vulnerabilities among them are provided here CWE-908(54.2%), CWE-401(14%), CWE775 (5.2%), CWE-562(4.7%), CWE-119(4%) CWE-758(3.7%).Al-Boghdady, Wassif, and El-Ramly [29] also examined widely used open-source Internet of Things (IoT) OSs written in C/C++ such as RIOT, Contiki, FreeRTOS, and Amazon for vulnerabilities and scanned them with computational analysis tools Cppcheck, Flawfinder, and Rough Auditing Tool for Security(RATS). The results were disparate. The most prevalent vulnerabilities detected by computational analysis tools were: (CWE-119! /CWE-120), (CWE-120), and (CWE-126) with Flawfinder, (CWE-563, CWE-561, and CWE-398) with Cppcheck, and (CWE134, CWE-120, and CWE-119) with RATS in the IoT OS source code. These security vulnerabilities can cause security breaches as a consequence. Verdi et al. [11] examined the C++ security issue in crowded source code by analyzing the vulnerabilities in snippets of reusable source code posted on stack overflow and their spread in the program's repository from two perspectives: prevalence and propagation. Consequently, 69 vulnerable code snippets were reported among 72,483 examined code snippets in at least one project hosted on an online repository. These 69 vulnerable code snippets were shared in the same repository in 2859 projects. In this regard, Alnaeli et al. [30] undertook research to expose IoT software systems to the use of vulnerable source code and insecure functions in the programming languages C/C++, which can constitute a catastrophic security concern. Empirically, analysis was conducted on three open-source software: TinyOS, openWSN, and Contiki, with 1,000,981 lines of code. The presence of vulnerabilities and unsafe function usage was reported with a figure of 772 in TinyOS, 220 in openWSN, and 1,859 in Contiki. Furthermore, the research also reported that the IoT is growing with over a million devices and runs with millions of open-source software. The study also evident the security issues in open-source software, which causes security compromise and becomes a severe concern for stack holders, including individuals and groups. Vulnerable code is the most typical source of security problems [31]–[33]. In their study, Kaur and Nayyar [10] compared static code analysis methods for exposing vulnerabilities in Java and C/C++ programs, emphasizing the significance of vulnerability identification in guaranteeing software security, such as integrity, authenticity, and availability. They also favored vulnerability identification in the initial stages because it makes fixing easier for developers. They used static analysis tools such as FlawFinder, RATS, and cppcheck to scan Java and the C/C++ programming languages for certain types of vulnerabilities and report the presence of those chosen for investigation. Furthermore, they expressed concern that some vulnerabilities might go undetected without cutting-edge tools. In this study, Zahedi et al. [34] asserted that system connectivity to the internet raises the need for software security. Security vulnerabilities are recognized as flaws that allow attackers with malicious intent to breach system security. They studied the GitHub security concerns and discovered that around 2.98% are security vulnerabilities, with a significant weightage on encryption and identity management. The primary source of the investigation was contained within a single GitHub repository.

The foregoing literature shows that most past software vulnerability detection research has relied on reviewing a single repository. Concerning the comparative analysis of languages in terms of vulnerabilities, the datasets enclosed within the study were not applications of standard open-source code and are no longer of general view but instead were narrowed to addressing vulnerabilities in specific application domains. Another significant weakness of serious concern in the previous study was that it focused on the efficiency of a static analysis tool rather than the presence of security vulnerabilities in code. Furthermore, the fundamental elements that contribute to the formation of programs and cause various security vulnerabilities in programming languages have not been thoroughly investigated. Consequently, it may be deduced that previous studies did not contain adequate findings. This endeavor is probably geared toward mitigating the prevailing research gap by experimenting with the dataset of well-known programs and tasks from different study areas for various programming portals and languages. This research investigates the severity of vulnerabilities in code in more depth, and a comparative study is being undertaken to categorize programming portals and languages secure in the context of vulnerabilities.

DESIGN AND METHOD

This study analyzes probable vulnerabilities in popular programming languages like C, C++, and Java, hosted on well-known programming portals like GitHub and Rosettacode. This endeavor would be accomplished by scanning many well-known computer programs from the aforementioned programming languages and portals. Furthermore, the vulnerabilities are evaluated severity-wise. As illustrated in Figure 1, this approach is organized into steps for pursuing the goal. In the first step, notable algorithms and tasks such as Universal Turing Machine, sorting algorithms like the bubble sort, selection sort, searching algorithms like linear search, and so on, which are used as contributing factors in software development, are included in the study. Furthermore, comparable algorithms are employed in this research, equivalent both in function and structure. And the sample of code extracted is from different disciplines 43% for math, 2% for Graphics,6%String Manipulation, 32% for Data structure, 3% for

Routing, 3 % for Compression, 9%Encryption/Decryption, 8%Puzzle, and 12% are of the game.

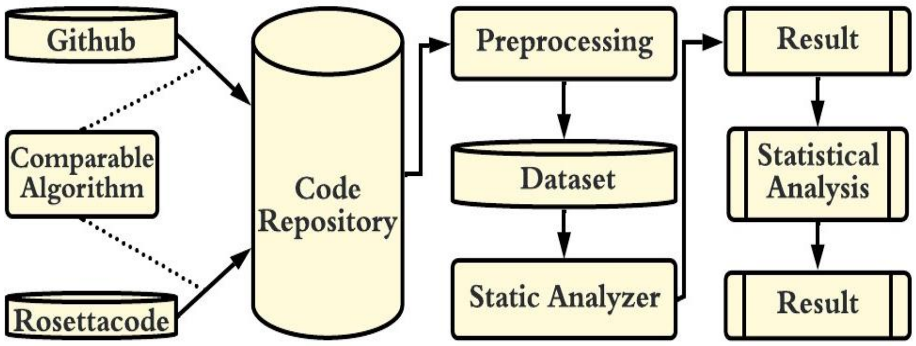


Figure 1. An Overview of the Research Approach

During the preprocessing, the extracted code repository is manually preprocessed as an updated dataset with the intent to equalize the task and make it appropriate for the static analyzer tool. The preprocessing activities comprise deleting extraneous source code from the file and integrating the remaining task source code from other files if the desired task is not completed in a file. Preprocessing also involves saving files with the required extension and comparing the outputs. The dataset is scanned for vulnerabilities using the computational analysis tool Yasca. Yasca is an open-source tool created in PHP in 2008–10 that scans source code for security vulnerabilities, code quality, performance, and compliance with best practices in various programming languages, including Python, Java, C/C++, and others. Each vulnerability detected by Yasca is assigned a risk level rating from 1 to 5, with 1,2,3,4,5, respectively, denoting Critical, High, Medium, Low, and Informational [35], [36].

To analyze the relationship between vulnerability presence in programming portals and programming languages for statistical significance, the Mann-Whitney Test, Kolmogorov-Smirnov Test, Wald-Wolfowitz Test, and Kruskal-Wallis H-test we conducted. The null and alternative hypotheses are verified or tested using statistical techniques to find their significance. The Null and alternative hypotheses might be formulated as follows:

Null hypothesis (H0): In terms of vulnerabilities, code hosted in code repositories has no statistical difference.

Alternative Hypothesis (H1): In terms of vulnerabilities, code hosted in code repositories has a statistical difference.

The p-value is used to test the hypotheses. The p-value is a factor of hypothesis testing theory and may be used to determine whether to accept or reject the null hypothesis. After calculating the p-value, we compared it with the value of 0.05, a long tradition for rejecting or accepting the null hypothesis. A very small p-value indicates evidence against the null hypothesis, whereas a large p-value indicates evidence favoring the null hypothesis.

RESULTS

The cumulative result of vulnerabilities identified in the code repositories GitHub and Rosettacode is shown in Table 1 and categorized by severity. The vulnerabilities discovered in the repositories differ in severity and programming portals. The RosettaCode repository has fewer vulnerabilities than GitHub.

Table 1. Vulnerabilities in Portals: A Comparative Analysis

Programming Portal	**Vulnerabilities Severity**					**Total**
	Critical	**High**	**Warning**	**Low**	**Informational**	
GitHub	2	24	60	96	799	981
RosettaCode	8	2	46	20	314	390
Total	10	26	106	116	1113	1371

For more insight and comparison, Figure 2 depicts the vulnerabilities of varying severity for the aforementioned portals. The RosettaCode repository contains fewer security vulnerabilities than GitHub. The figure also demonstrates that vulnerabilities of informational severity are more prevalent. However, of all the vulnerabilities of critical severity are the least reported. There is a significant disparity between the number of vulnerabilities detected in portals and the severity.

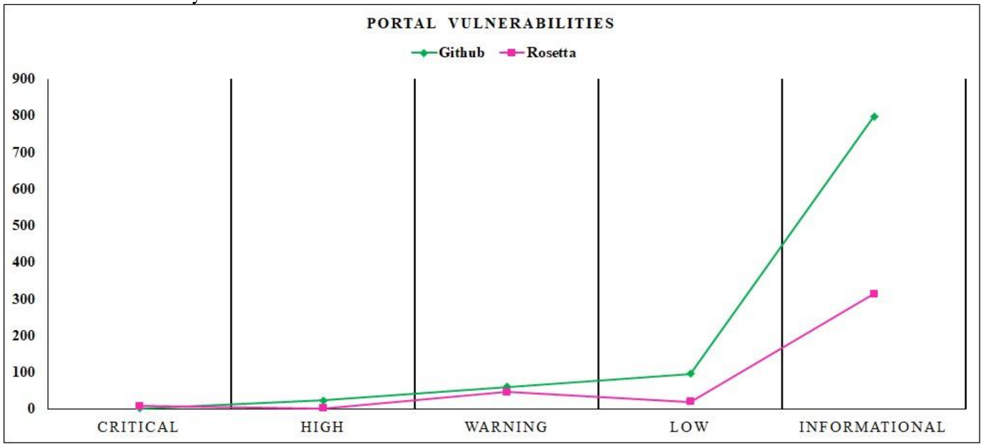


Figure 2. Portal Vulnerabilities Trends Severity Levels

Furthermore, vulnerabilities based on programming portals have been evaluated with the Mann-Whitney Test to gain some more insight into portal vulnerabilities. Table 2 shows the results of the test statistics. GitHub (Mean rank =676.05) has more vulnerabilities than Rosettacode (Mean rank =608.64). According to the Mann-Whitney test, this difference between programming portals and vulnerability is statistically significant.

Table 2. Mann-Whitney test statistics for vulnerability disclosure on Programming Portal

Vulnerabilities	**Frequency**	**Mean Rank**
Rosettacode	1113	608.64
GitHub	116	676.05

Moreover, the Kolmogorov-Smirnov Test has analyzed the relationship between vulnerability presence and programming portals. The calculated value of the Kolmogorov-Smirnov Test is 1.124, and the significance value is 0.159, which concluded that as far as the sampled programming corpus is concerned, there is no statistical difference between the Rosettacode and GitHub in terms of vulnerabilities.

Another test, named the Wald-Wolfowitz Test, has also been used for more confirmation of analyzing the relationship between vulnerability presence and programming portals. Test statistics are shown in Table 3. The Minimum Possible Exact P-value is $P_{1, min}$=.000 is less than 0.05, and the Maximum Possible exact P-value is $P_{1, max}$=0.345 which is more significant than 0.05. So, following the decision rules of the Wald-Wolfowitz Test, there is no statistical difference between the Rosettacode and GitHub.

Table 3. Wald-Wolfowitz Test statistics for vulnerability disclosure on Programming Portal

		Number of Runs	Z	Asymp. Sig. (1-tailed)
Portal Code	Minimum Possible	3	-34.792	.000
	Maximum Possible	233	3.661	1.000

The preliminary evaluation below examines vulnerabilities in the programming languages C, C++, and Java, and the result is displayed in Table 4. The vulnerabilities detected in repositories are different. In this analysis, Java has more vulnerabilities than any other language in this study, followed by the C language and C++.

Table 4. Vulnerabilities in Languages: A Comparative Analysis

Language	**Vulnerabilities Severity**					**Total**
	Critical	**High**	**Warning**	**Low**	**Informational**	
C	10	1	92	48	176	327
C++	0	0	7	0	44	51
Java	0	25	7	68	893	993
Total	10	26	106	116	1113	1371

For more insight and comparison, Figure 3 depicts the vulnerabilities of varying severity in the C, C++, and Java programming languages. As the figure illustrates, Java has more vulnerabilities than any other language in this study, followed by the C language and C++. The figure also demonstrates that vulnerabilities of informational severity are more prevalent. However, of all the vulnerabilities of critical severity are the least reported. There is a significant disparity between the number of vulnerabilities detected in programming languages and the severity.

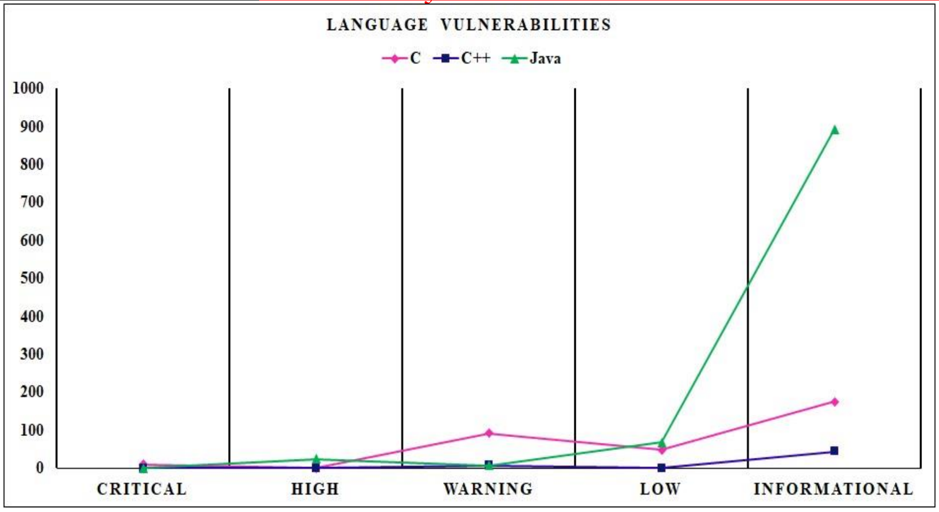


Figure 3. Language Vulnerabilities Trends Severity Levels

Moreover, the Kruskal-Wallis H-test has been used to analyze the relationship between vulnerability presence and programming language. The mean rank vulnerability score for the Kruskal-Wallis H test was 721.84 for C, 564.53 for C++, and 215.92 for Java, with Kruskal-Wallis H = 290.336 and a p-value (.000). The result suggests a statistically significant difference in code vulnerabilities between different programming languages at a significance level of 0.05, as there is ample evidence that the p-value is less than 0.05.

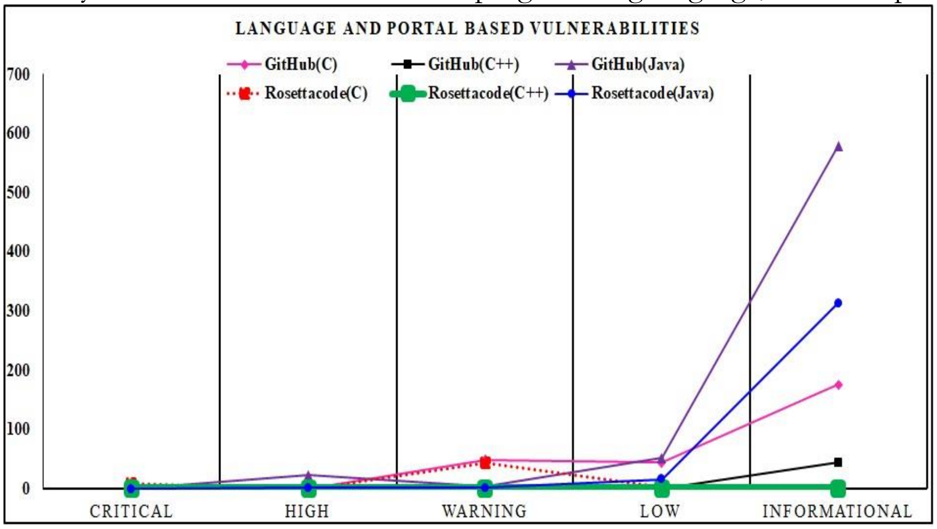
Moreover, code repositories and programming languages have examined the security vulnerabilities. Figure 4 shows the visual representation of the vulnerabilities identified in the programming languages C, C++, and Java, and in the code repositories, GitHub and Rosettacode are varied. GitHub (Java), GitHub (C), and GitHub (C++) are more prone to vulnerabilities than their corresponding Rosettacode (Java), Rosettacode (C), and Rosettacode (C++) programming languages. Furthermore, GitHub (Java) has more vulnerabilities than Rosettacode (Java), followed by GitHub (c), which has more vulnerabilities than Rosettacode (c). There are also security vulnerabilities present in GitHub (C++) and Rosettacode (C++), but they are less prevalent than in GitHub (Java) and Rosettacode (Java). Additionally, just one vulnerability in Rosettacode (C++) has been detected. The accompanying Figure 4 demonstrates that in comparison to the vulnerabilities in GitHub (Java, c, and C++) and Rosettacode (Java, C, and C++), the vulnerabilities in GitHub (Java) are higher than in any other language, followed by Rosettacode (Java), GitHub (C), Rosettacode (C), GitHub (C++), and Rosettacode (C++) lastly.

Figure 4. Language & Portal Vulnerabilities Trends Severity Levels

In Figure 4, the disparity in vulnerability severity is also evident, with vulnerabilities of informational severity being reported more frequently, followed by Low, Warning, High, and Critical.

DISCUSSION

Security vulnerabilities are one of the most frustrating aspects of computer security. Security vulnerabilities are severe software and hardware components weaknesses that constitute a danger to security, confidentiality, integrity, or availability when exploited. Vulnerabilities in software security have a catastrophic effect on software security. This study examined source code for popular programming languages such as C, C++, and Java hosted at two well-known repositories, GitHub and Rosettacode, in the context of security vulnerabilities under the umbrella of the Common Vulnerability Scoring System (severity-wise). This study will serve as a guideline for software developers to select a programming language and a secure portal in terms of vulnerabilities. This also familiarizes the code quality accessible in languages at programming portals by assessing the varied levels of probable vulnerabilities and their severity. This study will facilitate the development of secure software that is probably less likely to breach fundamental security aspects like integrity, authenticity, and availability with the selection of a secure programming language, a reliable programming hosting portal, and



appropriate source code. The well-known 118 comparable computer programs in C, C++, and Java hosted at Rosettacode and GitHub code repositories have been extracted to complete the research. The 708 programs are the dataset's complete size. A tool like Yasca is used to scan the dataset. The vulnerabilities present in C, C++, and Java are 23.8512035%, 3.71991247%, and 72.428884%, respectively, out of the total vulnerabilities, and their percentage has significant disparity. The language-based vulnerabilities with tests like the Mann-Whitney also confirm the difference as statistically significant. The percentage difference between vulnerabilities presents in the portals GitHub and Rosettacode is 43.11%, which is a significant disparity. The language-based vulnerabilities with tests like the Kruskal-Wallis tests also confirm the difference as statistically significant.

Further, the disparity is also evident in vulnerability severities, vulnerabilities with informational severity being reported more frequently. The vulnerabilities with warning and low severity being reported are significant figures, but critical and high severity have been reported in small numbers. This endeavor's finding demonstrates that the presence of vulnerabilities differs depending on the programming portal and programming languages used. So, the most probable preventative technique to secure an application or its environment, including hardware and software, from security risks caused by repeating code is to examine both the programming language and the programming portal.

In the future, work on developing a cutting-edge tool and scanning code with other tools to uncover vulnerabilities missed by selected tools will be necessary. Furthermore, one downside of this study would be that it only considers the programming languages C, C++, and Java, with a statistically limited sample size. Another possible future aspect of the current work is adding new algorithms, languages, and repositories to explore the influence of vulnerabilities on software, which may change the study's findings.

CONCLUSION

This endeavor explored and compared security vulnerabilities in the programming languages C, C++, and Java for well-known comparable computer algorithms posted on popular portals such as GitHub and Rosettacode. This research uncovered specific vulnerabilities present in the source code. GitHub (Java) is more prone to vulnerabilities, followed by Rosettacode (Java), GitHub (C), Rosettacode (C), GitHub (C++), and only one vulnerability in Rosettacode (C++). The disparity is also evident in vulnerability severities, vulnerabilities with informational severity being reported more frequently. The vulnerabilities with warning and low severity being reported are significant figures, but critical and high severity have been reported in small numbers. The Rosettacode (C++) is probably more secure regarding vulnerabilities for secure software development. This study probably manifests a guideline for choosing a secure programming language and a successful approach for testing that targets vulnerabilities that are more liable to compromise security.

REFERENCES

[1] "Software Engineering | Introduction to Software Engineering - GeeksforGeeks." https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/ (accessed Jun. 21, 2022).

[2] roger s Pressman and B. Maxim, "Sofware Enginering : A Practitioner's Approacch," p. 978, 2014.

[3] I. R. Imran, "A Study of Awareness and Practices in Pakistan's Software Industry towards DevOps Readiness," no. November 2021, 2022.

[4] "Requirements decision-making as a process of Argumentation: A Google Maps Case Study with Goal Model," vol. 3, pp. 15–33, 2021.

[5] J. P. Miguel, D. Mauricio, and G. Rodríguez, "A Review of Software Quality Models for the Evaluation of Software Products," *Int. J. Softw. Eng. Appl.*, vol. 5, no. 6, pp.

31–53, Nov. 2014, doi: 10.5121/IJSEA.2014.5603.
[6] "Computer Programming Basics: Introduction to Computer Programming." https://edu.gcfglobal.org/en/computer-programming-basics/introduction-to-computer-programming/1/ (accessed Jun. 21, 2022).
[7] Y. Zhang *et al.*, "H I G IT C LASS : Keyword-Driven Hierarchical Classification of GitHub Repositories."
[8] Synopsys, "Open Source Security and Risk Analysis Report," pp. 1–29, 2021.
[9] J. Luszcz, "Apache Struts 2: how technical and development gaps caused the Equifax Breach," *Netw. Secur.*, vol. 2018, no. 1, pp. 5–8, Jan. 2018, doi: 10.1016/S1353-4858(18)30005-9.
[10] M. Papamichail, T. Diamantopoulos, and A. Symeonidis, "User-Perceived Source Code Quality Estimation Based on Static Analysis Metrics," *Proc. - 2016 IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS 2016*, pp. 100–107, Oct. 2016, doi: 10.1109/QRS.2016.22.
[11] M. Verdi, A. Sami, J. Akhondali, F. Khomh, G. Uddin, and A. K. Motlagh, "An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples," *IEEE Trans. Softw. Eng.*, vol. 48, no. 5, pp. 1497–1514, Oct. 2019, doi: 10.1109/tse.2020.3023664.
[12] A. Kaur and R. Nayyar, "A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code," *Procedia Comput. Sci.*, vol. 171, pp. 2023–2029, Jan. 2020, doi: 10.1016/J.PROCS.2020.04.217.
[13] "A Smart Contract Approach in Pakistan Using Blockchain for Land Management," vol. 4, no. 2, pp. 425–435, 2022.
[14] S. P. Reiss, "Continuous Flow Analysis to Detect Security Problems," *arXiv*, no. July, 2019.
[15] L. Stosic and D. Velickovic, "Computer security and security technologies," *J. Process Manag. New Technol.*, vol. 1, no. 1, pp. 14–19, 2013, doi: 10.5937/jpmnt1301014s.
[16] F. Bukhari *et al.*, "Quack Finder: A Probabilistic Approach," vol. 4, no. 2, 2022.
[17] J. A. Harer *et al.*, "Automated software vulnerability detection with machine learning," no. October, 2018, [Online]. Available: http://arxiv.org/abs/1803.04497.
[18] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks," *Adv. Neural Inf. Process. Syst.*, vol. 32, Sep. 2019, doi: 10.48550/arxiv.1909.03496.
[19] A. Brazhuk, "Semantic model of attacks and vulnerabilities based on CAPEC and CWE dictionaries," *Int. J. Open Inf. Technol.*, vol. 7, no. 3, pp. 38–41, 2019.
[20] "Common Vulnerability Scoring System SIG." https://www.first.org/cvss/ (accessed Jun. 22, 2022).
[21] "TIOBE Index - TIOBE." https://www.tiobe.com/tiobe-index/ (accessed Jun. 22, 2022).
[22] "CWE - CWE-659: Weaknesses in Software Written in C++ (4.7)." https://cwe.mitre.org/data/definitions/659.html (accessed Jun. 22, 2022).
[23] "CWE - CWE-658: Weaknesses in Software Written in C (4.7)." https://cwe.mitre.org/data/definitions/658.html (accessed Jun. 22, 2022).
[24] "CWE - CWE-660: Weaknesses in Software Written in Java (4.7)."

https://cwe.mitre.org/data/definitions/660.html (accessed Jun. 22, 2022).
[25] M. A. Arshed, S. Mumtaz, O. Riaz, W. Sharif, and S. Abdullah, "A Deep Learning Framework for Multi-Drug Side Effects Prediction with Drug Chemical Substructure," *Int. J. Innov. Sci. Technol.*, vol. 4, no. 1, pp. 19–31, 2022.
[26] Y. Zhao, R. Liang, X. Chen, and J. Zou, "Evaluation indicators for open-source software: a review," *Cybersecurity*, vol. 4, no. 1, pp. 1–24, Dec. 2021, doi: 10.1186/S42400-021-00084-8/FIGURES/3.
[27] "CVE - CVE." https://cve.mitre.org/ (accessed Jun. 22, 2022).
[28] H. Zhang, S. Wang, H. Li, T. H. P. Chen, and A. E. Hassan, "A Study of C/C++ Code Weaknesses on Stack Overflow," *IEEE Trans. Softw. Eng.*, 2021, doi: 10.1109/TSE.2021.3058985.
[29] A. Al-boghdady, K. Wassif, and M. El-ramly, "The Presence, Trends, and Causes of Security Vulnerabilities in Operating Systems of IoT's Low-End Devices," *Sensors 2021, Vol. 21, Page 2329*, vol. 21, no. 7, p. 2329, Mar. 2021, doi: 10.3390/S21072329.
[30] S. M. Alnaeli, M. Sarnowski, M. S. Aman, A. Abdelgawad, and K. Yelamarthi, "Vulnerable C/C++ code usage in IoT software systems," *2016 IEEE 3rd World Forum Internet Things, WF-IoT 2016*, no. February 2019, pp. 348–352, 2017, doi: 10.1109/WF-IoT.2016.7845497.
[31] C. Kolias, A. Stavrou, J. Voas, … I. B.-I. S. &, and undefined 2016, "Learning internet-of-things security&quot; hands-on&quot;," *Ieeexplore.Ieee.Org*, [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7397713/.
[32] A. M. Gamundani, "An impact review on internet of things attacks," *Proc. 2015 Int. Conf. Emerg. Trends Networks Comput. Commun. ETNCC 2015*, pp. 114–118, Aug. 2015, doi: 10.1109/ETNCC.2015.7184819.
[33] R. K. McLean, "Comparing static security analysis tools using open source software," *Proc. 2012 IEEE 6th Int. Conf. Softw. Secur. Reliab. Companion, SERE-C 2012*, pp. 68–74, 2012, doi: 10.1109/SERE-C.2012.16.
[34] M. Zahedi, M. A. Babar, and C. Treude, "An empirical study of security issues posted in open source projects," *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, vol. 2018-Janua, pp. 5504–5513, 2018, doi: 10.24251/hicss.2018.686.
[35] C. Scripting, "A nalysis Tools A gainst Cross-site Scripting V ulnerabilities keywords :," pp. 125–142, 2021.
[36] "Yasca by scovetta." http://scovetta.github.io/yasca/ (accessed Jun. 22, 2022).