JCBI
Journal of Computing & Biomedical Informatics Volume 04 Issue 01
ISSN: 2710 - 1606 2022

*Research Article*
https://doi.org/10.56979/401/2022/111

# Correlation Between GitHub Stars and Code Vulnerabilities

**Muhammad Shumail Naveed[1]***

[1]Department of Computer Science & Information Technology, Quetta, 87300, Pakistan.
*Corresponding Author: Muhammad Shumail Naveed. Email: mshumailn@gmail.com.

**Received:** November 01, 2022 **Accepted:** November 28, 2022 **Published:** December 29, 2022.

**Abstract:** In the software industry, open-source repositories are widely utilized to speed up software development. GitHub is a big source of open-source repositories and offers users to star the code repository. Stars are used in GitHub to represent appreciation and popularity. Studies have revealed that repositories may be of lower quality and may have vulnerabilities that hackers may exploit. It is not known whether the popularity of the GitHub repositories in terms of stars confirms the security and invulnerability of the program code. This paper analyzed the correlation between stars of GitHub's code repositories and the vulnerabilities in their code by using static code analyzer. The study examined the vulnerabilities in ten popular C++ source repositories on GitHub and discovered 3487 vulnerabilities in the dataset, which were split into four categories based on severity. There was not a single repository in the dataset that was free of flaws. On the detected vulnerabilities, a Kruskal-Wallis H test reveals a significant difference between the different code repositories of the dataset. The Spearman's rank correlation coefficient test found no correlation between repositories' stars and the frequency of vulnerabilities, implying that the popularity of code repositories on GitHub in terms of high star ratings does not imply their security integrity. Overall, the findings suggest that code repositories should be thoroughly evaluated before being used in software development. The novelty of this paper resides in the development of new knowledge as well as the study pattern that may be used to other investigations.

**Keywords:** GitHub; C++; Vulnerabilities; Static Code Analyzer.

## 1. Introduction

In today's world, computers and communication devices have become commonplace, and thereby software plays a vital role in modern civilization. The wide need of software has sped up the development of software. The number of job opportunities for development is likely to increase in the future [1].

One of the key goals of software development is to develop high-quality software in a cost-effective manner. However, programming from scratch is incredibly expensive [2], and programming is itself difficult to learn [3-5]. As a result, developers frequently reuse code to save time and efforts. Code reuse is a well-known and essential way for making development easier. In software development code reuse is a form of knowledge reuse which is critical to innovation in a variety of domains [6].

The core concept and paradigm of software development are fast changing; today, software development is a collaborative and distributed process in which success is dependent on the ability to systemize social and technical resources [7]. Code repositories shared on coding platforms are used by novice and even professional developers to reuse code in software development. GitHub is a well-known social coding platform and a modern forerunner of software forges. It has evolved into a hub for open-source projects [8]. It improves developers' development and collaboration by hosting projects and allowing them to share code. In 2022, there are approximately 94 million developers on GitHub and 414 million open-source contributions [9]. Git is a distributed version control system (DVCS) that was created in response to the commercialization of BitKeeper, the version control system that was used to build the Linux kernel. Git is a

piece of software that programmers use to keep track of all changes made to a software's source code. Basically. Git is an autonomous software that can be used by other systems like GitLab, Bitbucket, and SourceForge, but it is not a core part of GitHub. GitHub may be conceived as a Git cloud platform that developers use to manage their open-source projects when used in its most basic form. Version control systems aid programmers in managing concurrent changes to a shared code base by synchronizing all code contributions. [10]. Each repository on GitHub is linked to a collection of meta data. The open GitHub API offers information about the repository such as its size, the people who have starred it, and other statistics.

The major features of GitHub that allow code reuse are Gist and advanced search. It has a number of features for teamwork, collaboration, and sustained project discussion. GitHub has a "fork & pull" framework, where programmers create their own version of a repository and send a request to the maintainer when they need the maintainer to merge their changes into the core branch, allowing individuals to easily evaluate code surveys [11]. Alternatively, each repository can use GitHub's issue tracking framework to notify and investigate errors and other issues.

Social coding environments like GitHub not only captivate developers to develop cost-effective software, but they also entice attackers to diffuse the malicious code [12]. Other software developers can simply fork harmful repositories that are purposefully hosted on GitHub. Given the high level of interaction between GitHub and other social sites [13], malicious repositories might easily be spread to develop the production software. GitHub offers a security bug bounty for reporting code vulnerabilities. The IT infrastructure of businesses and organizations is at risk due to security vulnerabilities. Attackers exploit a vulnerability, affecting millions of users and compromising enormous amounts of financial data [14,15].

GitHub users express indebtedness to projects by adding stars to them. Therefore, the number of stars of a repository is a direct measure of its popularity [16,17]. Staring on GitHub is similar to "liking" on other social media sites. For open-source communities, popularity is advantageous since it encourages new contributors and informs the repository's authors that their product is being used. However, to our knowledge, there is no study that examines whether the popularity of GitHub repositories in terms of stars confirms the security and invulnerability of program code. The GitHub has some tools for semantic code analysis, but they are relatively limited. To promote secure code development and collaboration, a thorough examination of popular GitHub repositories is required.

The main objective and motivation of this article is to analyze the correlation between popularity of GitHub repositories in terms of stars and code vulnerabilities. This kind of rigorous analysis of code repositories in GitHub can provide esteemed insights into the development and maintenance of high-quality software. The article is organized as follows. Section 2 provides background information and a review of the literature. Section 3 discusses the design and methodology. Sections 4 contains the results and discussion, while section 5 concludes the article and identifies directions for further study.

## 2. Background & Related Work

This section is virtually divided into two sections. The first section covers the basics of vulnerabilities, while the second section gives a quick rundown of noteworthy studies on examining vulnerabilities in various code repositories, including GitHub.

Weaknesses in software architecture, design, and coding can allow hackers to gain unauthorized access to a network or system [18]. Vulnerabilities are caused by flaws in software or hardware. By removing the coding flaw, a large number of vulnerabilities can be eliminated. The vulnerability being investigated is typically published on a community-developed list known as the common weakness enumeration. The common weakness enumeration (CWE) is a list of different types of hardware and software flaws. It serves as a standard language, a metric for security tools, and a guideline for identifying, reducing, and avoiding vulnerabilities [19].

Vulnerabilities are flaws in program code that can be exploited to conduct unaccredited activities [20]. The severity of these operations might range from Denial of Service to Arbitrary Code Execution. In vulnerability databases, the CWE is often used to define the security flaw within studied vulnerabilities. This association is studied Mell and Gueye [21], in investigating vulnerabilities to find the critical flaws that lead to vulnerabilities. During the study, CWE taxonomies are combined with relevant data, and mashup views and graphs are created. These graphs are then used to create metrics that identify the most critical types of weaknesses.

Internally revealed vulnerabilities are seldomly published; nevertheless, publicly disclosed vulnerabilities are normally assigned a label number known as the Common Vulnerability and Exposure ID. Common Flaws and Exposures (CVE) is a set of standard names for publicly disclosed security vulnerabilities that has been widely used by organizations in order to provide effective coverage, easy interoperability, and enhanced security. The MITRE Corporation maintains CVE data, which are now sponsored by the Cybersecurity and Infrastructure Security Agency (CISA) [19]. Chen [22] established a CVE classifier, a classification framework for Common Vulnerabilities and Exposures (CVEs) that converts a list of CVEs into a classifier and categories them based on systematic criteria. The CVE classifier also assesses the general trend in vulnerability progression. The CVE classifier uses supervised learning to create learning models for taxonomic features based on training data extracted from the vulnerabilities database. The study discovered that a small number of services are responsible for the majority of security flaws and gaps.

Only a few studies have focused into the security flaws in GitHub projects. Fan et al. [23], established a C/C++ source code vulnerability dataset. The dataset was created using GitHub projects. The dataset is formulated using the Common Vulnerabilities and Exposures database and its associated repositories. According to the analysis, the dataset contains 3,754 code vulnerabilities.

Zhang et al. [24] developed GitCyber, a tool for detecting malicious GitHub repositories. GitCyber is a deep neural network-based system. The GitHub code repositories are used for learning in GitCyber, and structural HIN with the concept of Metapath is used to model domain knowledge. The early results of working with GitCyber are really promising.

Rokon et al. [25] presented a machine learning-based method to pin down malicious code repositories from GitHub by realizing that public archives have an astounding figure of malware repositories. The study used Multivariate Event Model being trained with LD137 and applied on RD137 dataset and identified 8644 malware repositories.

Stack overflow is a serious security flaw that is frequently exploited by advanced persistent threats to gain unauthorized access to a computer application. Rahman et al. [26] examined three open-source projects on GitHub and discovered that the selected projects have code segments that can overrun the stack and push malicious script that can disrupt normal program execution.

The use of GitHub makes it easier to collaborate and communicate while developing software. The forking feature of GitHub's concept allows users to edit a copy of the base repository. Between users of the base repository and those of the forked repositories, communication channels are opened by the forking process. Brisson et al. [27] analyzed communications between software repositories and found a statistically significant correlation between repository stars and fork depth, the number of individuals who have contributed to several repositories belonging to the same family, followers from outside the family, familial pull requests, and reported issues. The study also shed light on the significance of communication within a software family and how it affects individual repository star counts.

The scientific cyberinfrastructure community primarily uses open internet-based platforms like GitHub for resource sharing and teamwork. Monitoring GitHub for disclosed vulnerabilities can reduce costs and prevent exploits and attacks on cyber infrastructure. Lazarine et al. [28] used unsupervised graph embedding methods to find vulnerable communities within scientific cyberinfrastructure and found that the most important of them contain security flaws related to secret leaks and unsafe coding habits for high-impact genomics research.

Productivity, work quality, creativity, rapport in the group, and job happiness are all significantly impacted by emotions. Guzman et al. [29] used lexical sentiment analysis to examine the emotions expressed in the commit comments of various open-source projects on GitHub and determined how these emotions related to various variables like the programming language used, the time and day of the week the commit was made, the team composition, and project approval. The study examined the association between the average emotion score of each project and its number of stars but found no correlation.

## 3. Materials and Methods

The study aimed to analyze the vulnerabilities in the top projects of C++ hosted on GitHub. For software development there are hundreds of programming languages in which C, C++, Java, and Python being among the most prominent. For the current study, C++ was selected because it is straightforward and close to the C language. Perhaps, in addition to the notable characteristics of object-oriented programming, C++

supports the essential features of the C language. The syntax and denotational structure of Java and several other programming languages were heavily impacted by C++. Apart from its simplicity and object-oriented support, C++ has other distinguishing advantages such as a comprehensive library, low-level memory management and speed; therefore, thousands of code repositories are hosted on GitHub. The research methodology followed in the present study is outlined in Figure 1.


**Figure 1.** Research Methodology

On GitHub, a large number of code repositories are hosted, in which the popular C ++ projects (code repositories) were selected for study to analyze the possible correlation between the stars of repositories and vulnerabilities in code. On March 21, 2022, the relevant data of repositories was collected, and the details are shown in Table 1.

**Table 1.** Detail of Selected Projects

| Project Code | Project Name | Stars | Commits | Total Files | Source Files |
|---|---|---|---|---|---|
| 1 | PowerToys | 48245 | 4695 | 22408 | 198 |
| 2 | AirSim | 11134 | 2351 | 1257 | 118 |
| 3 | Spdlog | 10616 | 3493 | 153 | 34 |
| 4 | Vowpal Wabbit | 7393 | 8993 | 1645 | 18 |
| 5 | DeepSpeech | 16270 | 3331 | 2138 | 7 |
| 6 | Json | 22700 | 3948 | 999 | 286 |
| 7 | Arrow | 6857 | 8291 | 5116 | 85 |
| 8 | Carla Simulator | 5449 | 4756 | 1760 | 440 |
| 9 | ImGui | 26406 | 6228 | 172 | 36 |
| 10 | FoundationDB | 10194 | 13508 | 1518 | 309 |

C++ projects with more than 5,000 stars were selected for the study. The source programs, as well as other supporting files, were included in the actual code repository. As a result, the selected repositories were first preprocessed, with the appropriate source files scraped and kept as the actual dataset (programming corpus). The programming corpus was then lexically evaluated to identify lines of code, whitespaces, and comments.

A static code analyzer has been used to investigate the vulnerabilities in the programming corpus. For vulnerability scanning, a variety of static analysis techniques are readily available. YASCA is a static analysis tool [30] that was created to aid in quality assurance and vulnerability scanning. It is widely acknowledged as a reliable and versatile tool. YASCA supports a variety of programming languages and aggregates data from a variety of static analysis tools. The vulnerabilities in the dataset were examined using YASCA. It discovered the flaws and classified them according to their severity. The results of the vulnerability study were statistically examined using SPSS and R, which are popular packages for statistical analysis. The results of the analysis as well as a comprehensive discussion are presented in section 4.

## 4. Results & Discussion

The primary data of the study includes the source programs as well as supporting files. During preprocessing, the irrelevant files were eliminated and the metainformation for each project is kept separate for study. The study was carried out in two phases. The basic information about the dataset was collected in the first phase, wherein the lines of code, whitespaces, and comments were collected, and the results are shown in Table 2.

**Table 2.** Elementary Statistics of Results

| Project | Project Name | Lines of | Whitespace | Comments |
|---|---|---|---|---|
| 1 | PowerToys | 38360 | 6313 | 3588 |
| 2 | AirSim | 31367 | 4984 | 2232 |
| 3 | Spdlog | 2868 | 584 | 238 |
| 4 | Vowpal Wabbit | 2186 | 552 | 153 |
| 5 | DeepSpeech | 1048 | 166 | 98 |
| 6 | Json | 234269 | 40025 | 38974 |
| 7 | Arrow | 42216 | 6436 | 11131 |
| 8 | Carla Simulator | 92414 | 16045 | 14957 |
| 9 | ImGui | 28668 | 4506 | 7021 |
| 10 | FoundationDB | 153399 | 26051 | 20206 |

A large difference is observed in the size of selected projects. Similarly, there is a significant difference in the frequency of whitespaces and comments in the selected C++ projects as shown in Figure 2.

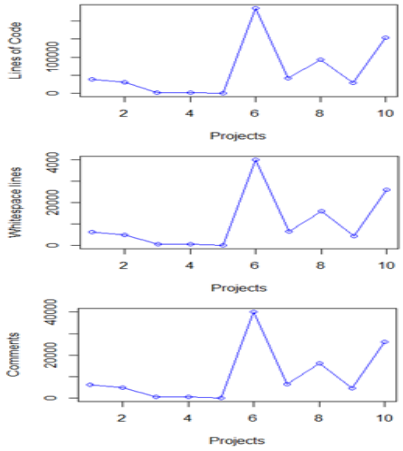


**Figure 2.** Line Charts of Elementary Information

The line charts in Figure 2 clearly indicate a wide range of projects, demonstrating the diversity of code repositories included in the dataset. The detection performance of present methodologies has to be increased due to numerous vulnerabilities in programs [31]. So, in the second phase, the current study used a static code analyzer to examine ten notable C++ source repositories hosted on GitHub.

The possible vulnerabilities in the constructed dataset are examined by analyzing the source code of each project. Static analysis is a vital approach to detect vulnerabilities [32], and thereby static code analysis tool (YASCA) was used to analyze the vulnerabilities in the dataset. The results obtained after scanning is shown in Figure 3.

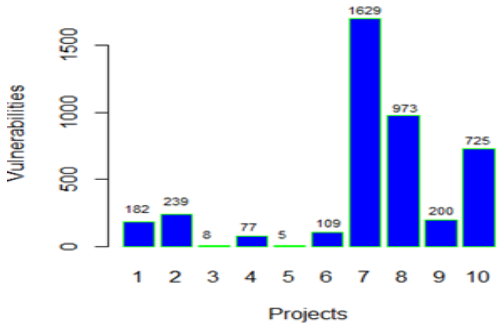


**Figure 3.** Detail of Vulnerabilities

In total, 1534 source files were scanned and 3487 flaws were identified during the vulnerability analysis. The flaws were identified in all the code repositories of dataset, however the frequency of observed vulnerabilities in each project was fairly different. In order to make the analysis more systematic, the identified vulnerabilities are further categorized into different categories according to their severity and details are shown in Table 3.

**Table 3.** Categories of Vulnerabilities in GitHub Projects

| Project | Project Name | Critical | High | Low | Info | Total Vulnerabilities |
|---|---|---|---|---|---|---|
| 1 | PowerToys | 0 | 6 | 120 | 56 | 182 |
| 2 | AirSim | 0 | 6 | 132 | 101 | 239 |
| 3 | Spdlog | 0 | 0 | 4 | 4 | 8 |
| 4 | Vowpal Wabbit | 0 | 0 | 10 | 67 | 77 |
| 5 | DeepSpeech | 0 | 0 | 1 | 4 | 5 |
| 6 | Json | 0 | 9 | 70 | 30 | 109 |
| 7 | Arrow | 0 | 0 | 392 | 1302 | 1694 |
| 8 | Carla Simulator | 0 | 22 | 641 | 310 | 973 |
| 9 | ImGui | 2 | 0 | 92 | 106 | 200 |
| 10 | FoundationDB | 1 | 10 | 499 | 215 | 725 |

In terms of severity, 3.3% of discovered vulnerabilities in the first project (PowerToys) of the programming corpus were high, 65.93% were low, and 30.77% were information. In the second project (AirSim), 2.51% of vulnerabilities detected were high, 55.23% were low, and the remaining 42.26% were information. In the third project (Spdlog), 50% of the vulnerabilities discovered were low in severity, while the remaining 50% were information. In the fourth project (Vowpal Wabbit), 12.99% of vulnerabilities were low and 87.01% were information; similarly, 20% of vulnerabilities were low and 80% were information in the fifth project (DeepSpeech). According to the findings of the study, 8.26% of vulnerabilities in the sixth project (Json) of the programming corpus were high, 64.22% were low, and 27.52% were information. In the seventh project (Arrow), 23.14% of vulnerabilities were low and the remaining 76.86% were information. Similarly, 2.26% of vulnerabilities were high, 65.88% were low, and the remaining 31.86% were information in the eighth project (Carla Simulator). In the ninth project of programming corpus, 1% of vulnerabilities was critical, 46% were low and 53% was information. In the tenth project of dataset, there were 0.14% critical vulnerabilities, 1.38% high vulnerabilities, 68.83% low vulnerabilities, and 29.65% information vulnerabilities.

For better illustration, the results of vulnerability analysis are represented with donut charts and shown in Figure 4.

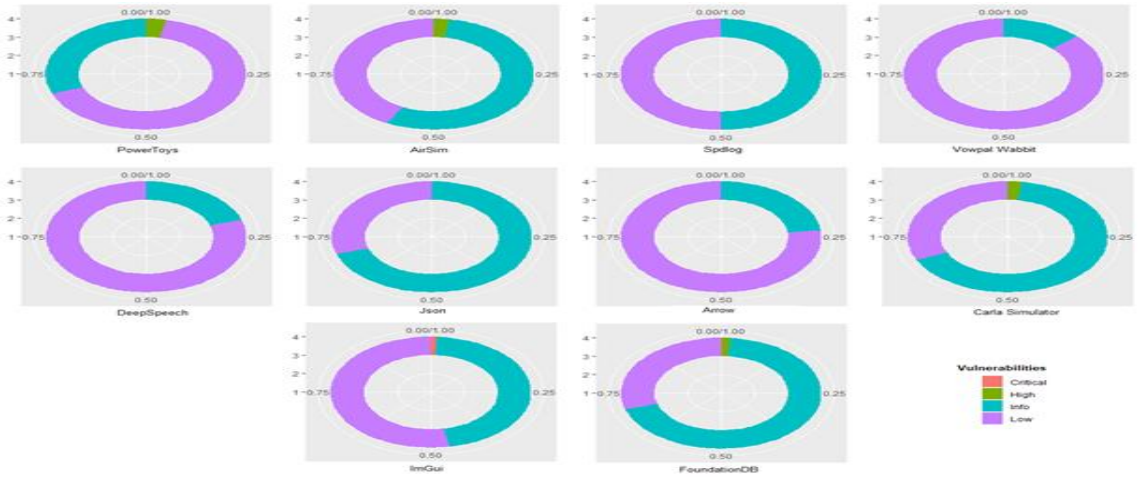


**Figure 4.** Donut Charts on Severity of Vulnerabilities in C++ projects

The highest number of vulnerabilities was observed in Arrow, followed by Carla Simulator and FoundationDB. For further analysis on vulnerabilities identified in the data, the normality tests are performed with R and SPSS. The Anderson-Darling test conducted on the identified vulnerabilities in code repositories of dataset does not confirm the normal distribution, A=1.0171, p=0.0064. Similarly, Shapiro-Wilk normality test conducted on the identified vulnerabilities does not confirm a normal distribution, W(10) = .766, p = .006. Likewise, Kolmogorov-Smirnov also identified the non-normal distribution, W(10) = .330, p = .003.
Due to non-normal distribution in data, the rank-based nonparametric test was conducted on the different categorizes of vulnerabilities analyzed in the dataset and results of ranks are shown in Table 4.

**Table 4.** Ranks of Projects

| Project Code | Project Name | Vulnerabilities | Mean Rank |
|---|---|---|---|
| 1 | PowerToys | 182 | 1643.19 |
| 2 | AirSim | 239 | 1889.87 |
| 3 | Spdlog | 8 | 2076.00 |
| 4 | Vowpal Wabbit | 77 | 2845.13 |
| 5 | DeepSpeech | 5 | 2699.40 |
| 6 | Json | 109 | 1525.78 |
| 7 | Arrow | 1694 | 2634.14 |
| 8 | Carla Simulator | 973 | 1676.29 |
| 9 | ImGui | 200 | 2127.99 |
| 10 | FoundationDB | 725 | 1637.92 |

The Kruskal-Wallis H test conducted on the identified vulnerabilities shows a significant difference between the different code repositories of the dataset, H (9) = 840.71, $p < .05$. Similarly, Fischer's exact test conducted on frequencies of different categories of vulnerabilities identified in the dataset shows a significant difference between the code repositories observed at $p < 0.05$.

During the analysis, 1534 files were examined, and YASCA was used to scan 626795 lines of code, yielding 3487 vulnerabilities. As a result, one vulnerability is estimated to exist in every 180 lines of code in a dataset. Table 5 shows the average size of a code section that has a single vulnerability by repository.

**Table 5.** Segment wise detail of Vulnerabilities in Dataset

| Project | Project Name | Lines of Code | Vulnerabilities | Average Size of |
|---|---|---|---|---|
| 1 | PowerToys | 38360 | 182 | 211 |
| 2 | AirSim | 31367 | 239 | 131 |
| 3 | Spdlog | 2868 | 8 | 359 |
| 4 | Vowpal Wabbit | 2186 | 77 | 28 |
| 5 | DeepSpeech | 1048 | 5 | 210 |
| 6 | Json | 234269 | 109 | 2149 |
| 7 | Arrow | 42216 | 1694 | 25 |
| 8 | Carla Simulator | 92414 | 973 | 95 |
| 9 | ImGui | 28668 | 200 | 143 |
| 10 | FoundationDB | 153399 | 725 | 212 |

Although all of the code repositories in a dataset include vulnerabilities, the number of vulnerabilities in each repository varies significantly. The identified vulnerabilities are categorized in different groups

according to their severity. The dataset contains 2 critical vulnerabilities (0.06% vulnerabilities), 43 high vulnerabilities (1.23%), 1462 low vulnerabilities (41.93%), and 1980 information vulnerabilities (56.78%).

It has been noticed that developers take into account the popularity of GitHub repositories in terms of stars when choosing a coding project. As a result, this study analyzed the correlation between stars, vulnerabilities, lines of code and other code repository attributes.

Spearman's rank correlation coefficient was run to determine the relationship between the GitHub stars and the identified vulnerabilities. There was no monotonic correlation between stars and vulnerabilities ($r_s = -.46$, $n = 10$, $p = .19$). This implies that the popularity of code repositories as measured by the number of stars given to GitHub projects does not imply that the code is secure and free of vulnerabilities. As a result, the study encourages developers to avoid choosing code repositories based on their GitHub star rating.

In order to identify the association between lines of code and vulnerabilities, Spearman's rank was conducted and the result identifies no correlation between lines of code and vulnerabilities ($r_s = .14$, $n = 10$, $p = .69$). This means that principally the size of code does not affect the frequency of vulnerabilities in GitHub repositories.

The correlation between vulnerabilities and commits was examined and the Spearman's ranks identifies no association between these two variables ($r_s = .44$, $n = 10$, $p = .20$). According to this, excessive revision of code repositories does not ensure the mitigation of vulnerabilities in their code.

During the study, the relationship between the popularity of repositories and the total number of code revisions is investigated. Spearman's rank determines that there is no correlation between commits and stars of GitHub repositories ($r_s = -.41$, $n = 10$, $p = .24$). This implies that the number of stars assigned to GitHub projects has no relevance on the frequency with which the source files are revised. Similarly, frequency of revision has no impact the stars of GitHub projects

The source code repositories on GitHub effectively facilitate quick software development; yet, the shared code may have security flaws that might lead to various risks. The vulnerabilities found in the dataset suggest that, despite their high star ratings, GitHub code repositories are likely to have a variety of vulnerabilities. Deep study of code and revealed vulnerabilities indicates that vulnerabilities expressed in code repositories are caused by debilitated programming found in the code. For instance, the critical vulnerabilities in the dataset are caused by weak credentials like infirm passwords. High vulnerabilities are mainly triggered by uninitialized variables, null pointer deference, division by zero and shifting by a negative value. Unnecessary use of consecutive return, break, continue, GOTO and throw statements caused the low vulnerabilities. In the same way, the definition of unused or high scoped variables, a wrong pointer casting and duplicate branches of selection structure provoked the low vulnerabilities. Similarly, the use of Boolean results of the bitwise operations and zero execution call of defined functions mainly caused the info vulnerabilities.

The correlational study conducted on the attributes of GitHub repositories suggests several points, i) selection of code repositories should not be considered on high stars because it does not ensure the absence of vulnerabilities, ii) According to the findings, the prevalence of vulnerabilities in repositories is not associated to the size of code. As a result, choosing repositories from GitHub should not be based solely on code size, as this does not guarantee the presence or lack of vulnerabilities, and iii) the frequency of vulnerabilities and the volume of commits are unrelated. So, while selecting repositories from GitHub, it is essential to realize that a large number of code revisions does not necessarily imply a high level of security. It is worth noting that GitHub's open repositories reduced the time and cost of software development. Excessive and haphazard use of repositories, on the other hand, is unsafe and could lead to significant vulnerabilities. The study obliquely recommends the detailed security examination of open code repositories before being used in real projects.

The study presented in this article is unique because it deeply analyzed the code in the popular repositories of C++ on GitHub. Similarly, the correlational study conducted in the present work introduce a new knowledge that could be useful to researchers and programmers. However, in current form the study has several limitations. Single static analyzer has used and only ten projects of C++ were analyzed in the study. So, the different pattern of results could be drawn by considering the other class of code repositories and static analyzer.

The study described in this article is unique in that it examined the code of prominent C++ GitHub repositories. Similarly, the correlational study undertaken in this article add to the existing body of knowledge that academics and programmers can use. However, the study contains significant limitations in its current form. The study only analyzed the ten C++ applications and employed a single static analyzer. Considering the other types of code repositories and static analyzers, a distinct pattern of results may be established.

## 5. Conclusions

Modern software development is heavily reliant on reusable code, which may be found in online repositories such as GitHub. The abundance and accessibility of code repositories on GitHub has piqued the interest of researchers. GitHub is a massive source code repository that has played a crucial role in the development of advanced social coding platforms. Despite the obvious advantages of hosting code repositories for reuse, the security risk of propagating insecure code has largely been overlooked. In this article, the static code analyzer was used to evaluate ten popular C++ source repositories available on GitHub. The analysis discovered that high-ranking C++ code repositories contain a variety of vulnerabilities, with no code repository being completely devoid of flaws. According to a correlational study of different attributes of GitHub repositories, the popularity of code repositories has no effect on their security integrity. There was no association found between the size of code and vulnerabilities, and no correlation was observed between the prevalence of vulnerabilities and the frequency of commits. The study certainly suggests that code repositories be investigated thoroughly before use in software development. In the future, i) more code repositories will be mined, ii) other static analyzers compliant with CWE and CVE will be used for a more detailed and extensive analysis, iii) code repositories for other languages will be examined, and iv) other coding platforms such as Google Code, BitBucket, and CodePlex will be used for dataset development.

**Funding:** This research received no external funding.

**Acknowledgments:** The author would like to express his gratitude to Muhammad Tahaam for his encouragement and valuable suggestions during data analysis. Special thanks to Muhammad Aayaan for his insightful comments on the final draft of this article.

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

**References**


1. Naveed, M.S., & Sarim, M. Pedagogical Significance of Natural Language Programming in Introductory Programming. Journal of Basic and Applied Sciences 2018, 14: 62-71.
2. Gharehyazie, M., Ray, B., Keshani, M., Zavosht, M.S., Heydarnoori, A., & Filkov, V. Cross-project code clones in GitHub. Empirical Software Engineering 2019, 24:1538–1573.
3. Naveed, M.S., & Sarim, M. Analyzing the Effects of Error Messages Presentation on Debugging and Programming. Sukkur IBA Journal of Computing and Mathematical Sciences 2020, 4(2):38-44.
4. Naveed, M. S., Sarim, M., & Nadeem, A. C in CS1: Snags and Viable Solution. Mehran University Research Journal of Engineering & Technology 2018, 37(1):1-14.
5. Naveed, M. S., Sarim, M., & Ahsan, K. Learners Programming Language a Helping System for Introductory Programming Courses. Mehran University Research Journal of Engineering & Technology 2016, 35(3): 347-358.
6. Haefliger, S., Krogh, G.V., & Spaeth, S. Code Reuse in Open Source Software. Management Science 2008, 54(1):180-193.
7. Joblin, M., Apel, S., & Mauerer, W. Evolutionary trends of developer coordination: a network approach. Empirical Software Engineering volume 2017, 22:2050-2094.
8. Cosentino, V., Izquierdo, J., & Cabot, J. A Systematic Mapping Study of Software Development with GitHub. IEEE Access 2017, 5:7173-7192.
9. (https://octoverse.github.com/) (Last Access: January 6th, 2022).
10. Vale, G., Schmid, A., Santos, A. R., Almeida, E. S., & Apel, S. On the relation between GitHub communication activity and merge conflicts. Empirical Software Engineering 2020, 25:402-433.
11. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. An in-depth study of the promises and perils of mining GitHub. Empirical Software Engineering 2016, 21:2035-2071.
12. Ye, Y., Hou, S. Chen, L., Li, X., Zhao, L., Xu, S., Wang, J., & Xiong, Q. ICSD: An Automatic System for Insecure Code Snippet Detection in Stack Overflow Over Heterogeneous Information Network. In Proceedings of the 34th Annual Computer Security Applications Conference 2018, 542-552.
13. Baltes, S., & Diehl, S. Usage and attribution of Stack Overflow code snippets in GitHub projects. Empirical Software Engineering 2019, 24:1259-1295.
14. Zhang, J., Chen, X., Xiang, Y., Zhou, W., & Wu, J. Robust network traffic classification. IEEE/ACM Transactions on Networking 2014, 23(4): 1257–1270.
15. Lin, g., Xiao, W., Zhang, L. Y., Gao, S., Tai, Y., & Zhang, J. Deep neural-based vulnerability discovery demystified: data, model and performance. Neural Computing and Applications 2021, 33:13287–13300.
16. Borges, H., Hora, A., & Valente, M. T. Predicting the Popularity of GitHub Repositories. In proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering 2018, 1-10.
17. Borges, H., Hora, A., & Valente, M. T. Understanding the Factors That Impact the Popularity of GitHub Repositories. In proceeding of IEEE International Conference on Software Maintenance and Evolution 2016, 334-344.
18. Park, K. Choi, S., Park, K. & Ki, C. Usability of Software Weakness Discovery based on the Binary File Visualization. In proceeding of the 3rd International Conference on Next Generation Computing 2017, 155-157.
19. https://cve.mitre.org/cve/ (Last Accessed on 5th January, 2022).
20. Schiappa, M., & Chantry, G., & Garibay, I. Cyber Security in a Complex Community: A Social Media Analysis on Common Vulnerabilities and Exposures. In proceeding of sixth International Conference on Social Networks Analysis, Management and Security 2019, 13-20.
21. Mell, P., & Gueye, A. A Suite of Metrics for Calculating the Most Significant Security Relevant Software. In proceeding of IEEE 44th Annual Computers, Software, and Applications Conference 2020, 511-516.
22. Chen, Z., Zhang, Y., & Chen, Z. A Categorization Framework for Common Computer Vulnerabilities and Exposures. The Computer Journal 2010, 53(5): 551–580.
23. Fan, J., Li, Y., Wang, S., & Nguyen, T.N. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In proceedings of the 17th International Conference on Mining Software Repositories 2020, 508-512.
24. Zhang, Y., Fan, Y., Hou, S., Ye, Y., Xiao, X., Li, P., Shi, C., Zhao, L., & Xu, S. Cyber-guided Deep Neural Network for Malicious Repository Detection in GitHub. In proceeding of IEEE International Conference on Knowledge Graph 2020, 458-465.
25. Rokon, M. D. F., Islam, R., Darki, A., & Papalexakis, E. E. SourceFinder: Finding Malware Source-Code from Publicly Available Repositories in GitHub. In 23rd International Symposium on Research in Attacks, Intrusions and Defenses 2020, 149-163.
26. Rahman, M. M., Satter, A., & Hossain, B. M. M. An Empirical Study on Stack Overflow Security Vulnerability in Well-known Open Source Software Systems. International Journal of Computer Applications 2020, 176(39): 11-16.
27. Brisson, S., Noei, E., & Lyons, K. We are family: analyzing communication in GitHub software repositories and their forks. In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering 2020, 59-69.
28. Horawalavithana, S., Bhattacharjee, A., Liu, R., Choudhury, N., O. Hall, L., & Iamnitchi, A. Mentions of security vulnerabilities on Reddit, Twitter and GitHub. In proceeding of IEEE/WIC/ACM International Conference on Web Intelligence 2019, 200-207.
29. Guzman, E., Azócar, D., & Li, Y. Sentiment analysis of commit comments in GitHub: an empirical study. In Proceedings of the 11th working conference on mining software repositories 2014, 352-355.

30. Kronjee, J., Hommersom, A., & Vranken, H. Discovering software vulnerabilities using data-flow analysis and machine learning. In proceedings of the 13th International Conference on Availability, Reliability and Security 2018, 1-10.
31. Yuan, X., Lin, G., Tai, Y., & Zhang, J. Deep Neural Embedding for Software Vulnerability Discovery: Comparison and Optimization. Security and Communication Networks 2022, 2022:5203217.
32. Zou, D., Zhu, Y., Xu, S., Li, Z., Jin, H., & Ye. H. Interpreting Deep Learning-based Vulnerability Detector Predictions Based on Heuristic Searching. ACM Transactions on Software Engineering and Methodology 2018, 37(4):111:2-111:32.