International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Evaluating Security and Community Health Metrics of FOSS Repositories

Niraj Salvi[1], Amal Thundiyil[2], Seema Supe[3], Nataasha Raul[4]

[1,2,3]Student, Department of Information Technology, Sardar Patel Institute of Technology
[4]Assistant Professor, Department of Computer Engineering, Sardar Patel Institute of Technology

Abstract

The extensive use of open-source packages in software development has greatly increased output and effectiveness. However, this development presents a challenging security environment in which vulnerabilities found in these packages can spread to other projects. To address this challenge, we suggest creating an open-source security assessment tool that has been painstakingly designed. The purpose of this tool is to assess security risks related to third-party dependencies and packages that are available on npm and GitHub.

Concerns are raised by the lack of a thorough assessment because seemingly innocuous packages could be hiding vulnerabilities that could lead to significant financial losses, service interruptions, and data breaches. Within the dynamic realm of open-source packages, developers often struggle to stay up to date with the ever changing security landscape. The main difficulty with this problem is figuring out which secure packages are kept up to date and which are either showing signs of poor maintenance or contain latent vulnerabilities.

Therefore, it becomes necessary to have a methodical, data-driven security evaluation tool so that developers can make informed choices about which packages to install. This project uses a wide range of parameters in an attempt to meet this requirement. These parameters allow for a quantitative evaluation of a package's security posture. They include metrics like stars, forks, resolved issues, and community engagement.

Our project aims to strengthen software security measures and mitigate potential risks associated with using third-party packages by giving developers actionable insights into the security status of their dependencies.

Keywords: Free and Open Source Software (FOSS), Security Assessment, Dependency Analysis, Community Health Metrics, Software Supply Chain, Metrics Calculation, System Architecture, Data Analysis, Vulnerability Assessment, Monitoring Tools, Data Collection, Qualitative Analysis, Continuous Integration/Continuous Deployment (CI/CD), Package Managers, Repositories.

1. Introduction

The development of open-source software confronts significant obstacles, mostly related to security threats. The absence of effective tools for identifying and mitigating dependents' weaknesses exposes projects to possible intrusions. Moreover, the lack of a uniform methodology for evaluating and improving code quality in repositories has a substantial effect on overall integrity. Legal issues may arise from difficult licence compliance, especially when confirming dependencies' licencing obligations.

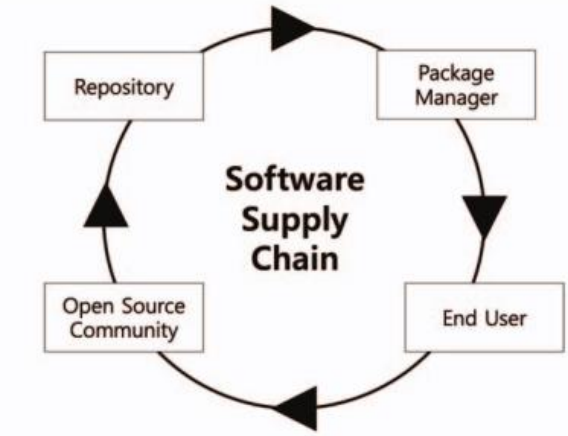
It is difficult to evaluate community involvement, which hinders the development of a cooperative developer community. Projects can become vulnerable due to inadequate tools for thorough dependency risk assessment, which could jeopardise their security and stability. Furthermore, the lack of defined metrics to track the sustainability of projects creates questions about their long-term viability and contributions to the software ecosystem.

In order to support open-source projects' security, quality, compliance, engagement, risk management, and sustainability, these issues must be resolved. The goal of this research is to create a solid framework for assessing community health and security metrics in FOSS (Free and Open Source Software) repositories. This study attempts to offer a solution that improves the open-source software environment overall by tackling these important problems in the field of open-source software development.

2. Background

In this section, the structure of the software supply chain is introduced.

Figure 1 : Structure of the Software Supply Chain



Open-Source Communities: Package managers, end users, repositories, and the open-source community comprise the four essential stages of the software supply chain in open-source communities. Software developers write the source code for their projects during the first phase and submit it to a repository. Within the community, code development and sharing are encouraged by this cooperative atmosphere.

Repositories: Repositories are organised data storage systems; well-known examples are GitHub and GitLab. Repositories, powered by version control systems such as Git, meticulously log every software change. Prominent features encompass forking, which facilitates project replication for tailored development, and branching, which allows software development across multiple streams to contribute to particular projects.

Package Managers: In the software supply chain, package managers are essential because they enable end users to download software in packages. Some examples are the popular app stores like Apple's AppStore and Google's PlayStore, as well as npm for JavaScript, PyPI for Python, Maven for Java, and RubyGems for Ruby. These managers make it easier to distribute and use software packages.

End Users: End users who make use of the developed software are involved in the last stage. It's interesting to note that an end user can act as both a developer and an end user by downloading a package from the manager and participating in its development, illustrating how interconnected different chains are within the software supply chain.

A rise in software supply chain attacks could result from the software supply chain becoming more complex due to the growing use of software libraries. Reducing the number of software libraries and selecting safe libraries for development are crucial steps in mitigating this.

3. Related Works

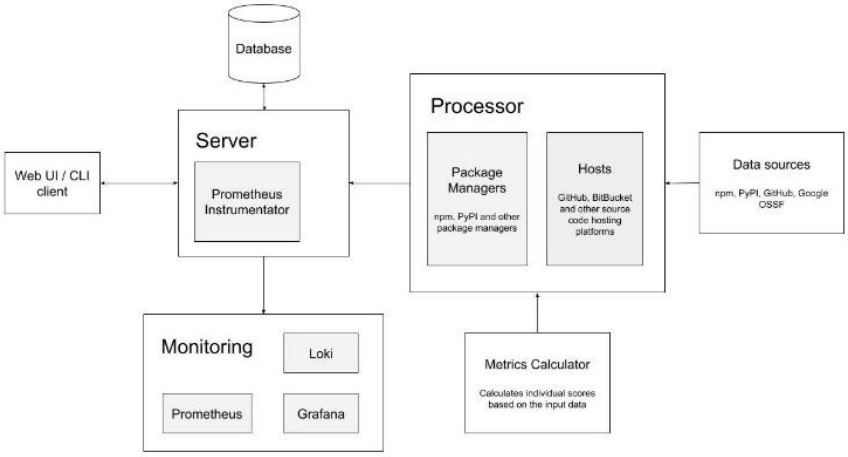
Researchers Pashchenko et al.'s study [1] examined how developers handle FOSS dependencies and found that they typically update software based on serious vulnerabilities and give priority to popular dependencies. Nonetheless, the study emphasised the necessity of safe dependency management by highlighting the underutilization of automation technologies and suggesting approaches for resolving unfixed vulnerabilities. Researchers in [2] introduced AutoMetric, an automated method for measuring OSS security metrics. This study suggested a standardised method for OSS security assessment and discovered associations between fewer vulnerabilities and frequent updates, commits, and shorter idle periods. A system for precisely counting susceptible dependencies was proposed by Vuln4Real [3], which also greatly reduced false alerts and gave developers insightful information. Furthermore, Cox et al.'s research [4] offered metrics to measure how fresh a dependent is, highlighting the need for regular updates and dependency management to reduce security and compatibility issues. Furthermore, to highlight the importance of testing and continuous integration (CI) in security analysis, VulinOSS [5] created a dataset that connected software metrics to security vulnerabilities. The research scripts and dataset were also made available to the public by the study. SoK [6] carried out a thorough taxonomy in the area of assaults on open-source software supply chains, offering insights into protective measures for enhancing OSS supply chain security.

Developers have noted that Dependabot lacks focus on security evaluation and community health metrics, and instead concentrates mostly on automating dependency updates when it comes to tools analysis. Snyk, on the other hand, is an enterprise-focused technology that works with CI/CD pipelines but depends on vulnerability data that is made public. Its reliance on publicly known vulnerabilities and the resource-intensive nature of scanning big codebases, however, raise red flags.

4. Methodology

The architecture employed in this project comprises key components tailored for evaluating the security posture of open-source projects:

Figure 2 : Illustration of the system architecture components



- **Database:** Manages open-source project dependency data, including security scores and metrics for historical analysis.
- **User Interfaces:** Provides a web client and a CLI for user interaction and integration into CI/CD pipelines.
- **Processor:** Functions as the core engine running heuristics to assess dependency security by processing data from diverse sources.
- **Data Sources:** Utilizes multiple repositories (e.g., npm and PyPI) and version control systems (e.g., GitHub) for comprehensive dependency information retrieval.
- **Metrics Calculator:** Calculates scores based on collected data to evaluate the security posture of dependencies.
- **Monitoring Tools:** Equipped with monitoring tools like Loki, Grafana, and Prometheus for operational monitoring and logging.

The methodology applied in this project encompasses the CHAOSS project, which is aimed at defining community health metrics. It involves a meticulous process that includes data collection from various open-source community sources such as code repositories, mailing lists, issue trackers, forums, and social media platforms (Figure 3). This data undergoes cleaning and preparation to remove duplicates, filter irrelevant information, and format it for analysis. Subsequently, metrics are defined according to specific criteria, facilitating consistent and comparable measurements across different contexts. The prepared data is then analyzed using statistical analysis, trend analysis, sentiment analysis, and other methods to extract meaningful insights. Visualization of the analyzed data is achieved through charts, graphs, and dashboards to aid in interpretation (Figure 4). Finally, qualitative insights are employed to understand the contextual significance of the metrics within the community.

Figure 3 : Chaoss' [1] high-level overview of metric calculation

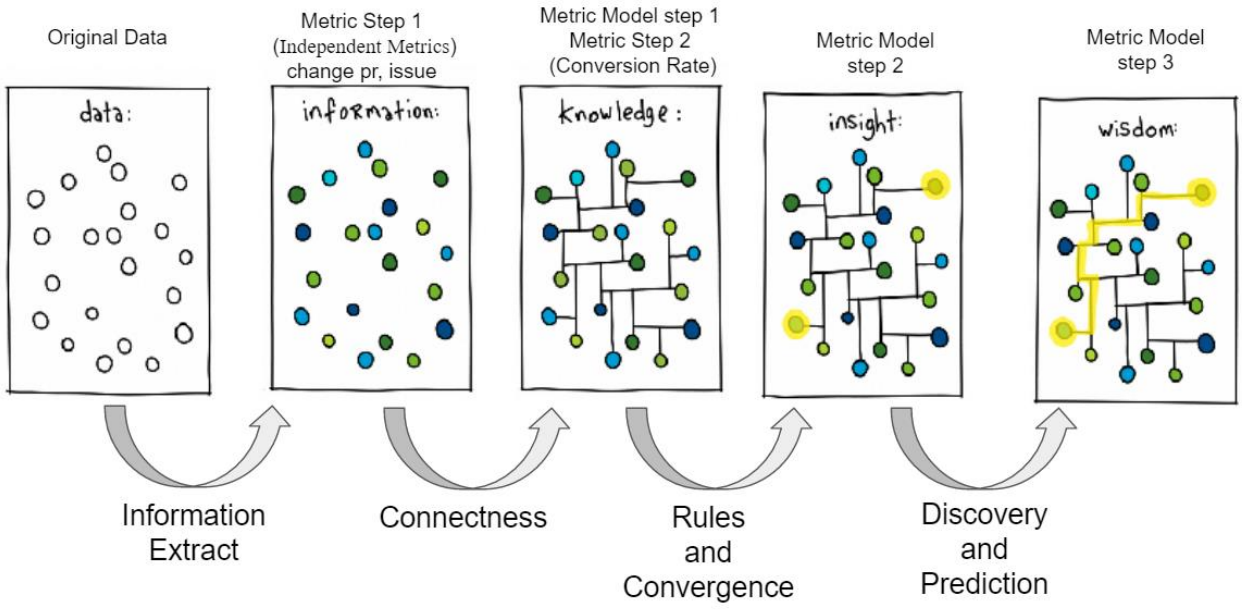
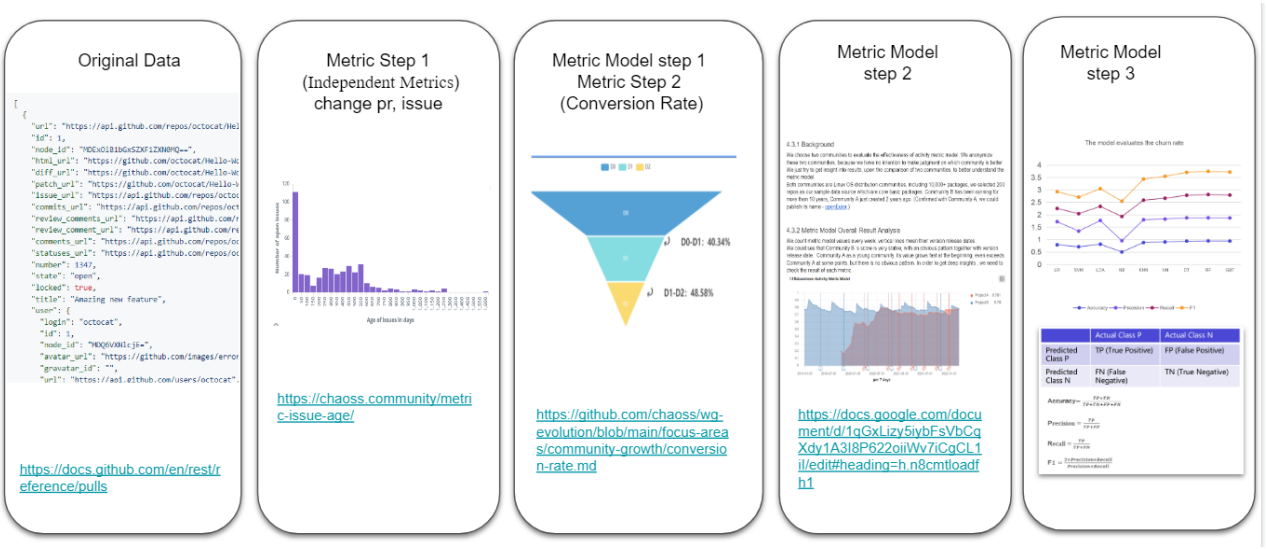


Figure 4 : Example of Chaoss' [1] high-level overview of metrics calculation



Additionally, the project integrates the OpenSSF Scorecard, an automated tool designed to assess the security posture of open-source projects. The Scorecard conducts checks on various aspects of the software supply chain, evaluating vulnerabilities present in source code, build processes, dependencies, testing, and project maintenance. It employs a scoring system that assigns scores and risk levels based on adherence to security best practices (Figure 5). These assessments are then used to compute an overall score that provides a snapshot of the project's security health. Furthermore, the tool offers recommendations for fixing identified issues (Figure 6).

Figure 5 : Risk levels and corresponding scores of OpenSSF Scorecard [2]

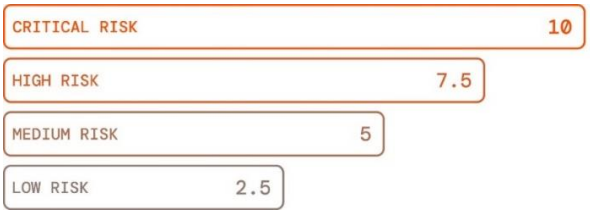
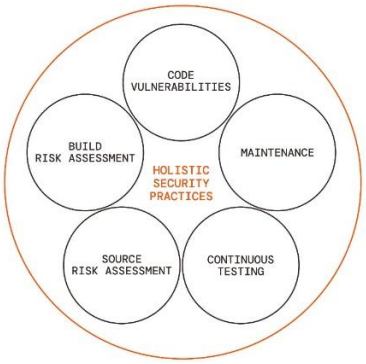


Figure 6 : OSSF scorecard [2] security checks



The integration of these methodologies, CHAOSS and the Open SSF Scorecard, within the Architecture allows for a comprehensive evaluation and enhancement of the security aspects of Open-source projects.

5. Implementation Details

The system assesses open-source projects through a series of metrics, each targeting specific aspects of project health. These metrics fall into categories such as risk levels, maintenance, community engagement, popularity, and security.

The integration of these methodologies, CHAOSS and the OpenSSF Scorecard, within the architecture allows for a comprehensive evaluation and enhancement of the security aspects of open-source projects.

A. Maintenance Metrics

Project maintenance is evaluated using various indicators:

Code Changes Commits:

- commit frequency: Average number of commits per week over the last 90 days.

Activity Dates and Times:

- updated since: Time since the last update in months.
- created since: Time since creation in months.
- comment frequency: Average comments per issue in the last 90 days.
- updated issues count: Number of issues updated in the last 90 days.
- downloads: Number of releases in the last year.

Change Request Reviews:

- code review count: Average review comments per pull request in the last 90 days.

Issues Closed:

- closed issues count: Number of issues closed in the last 90 days.
- issue age: Age of each open issue.

B. Community Metrics

Metrics reflecting community engagement include:

- contributor count: Active authors and participants in the past 90 days.
- maintainer count: Average number of maintainers per repository.
- org count: Distinct organizations contributors belong to.
- License, Code of Conduct, Bus Factor: Factors including project licensing, community standards, and risk evaluation.

C. Popularity Metrics

Popularity and engagement metrics consist of:

- Technical Forks, Reactions: Measures such as forks, stars, and other reaction counts.
- Followers, Watchers, Downloads: Number of users actively following the project and download counts.

D. Security Risk Levels

Each check is associated with a risk level and carries a distinct weight:

- Low Risk: Weighted at 10.
- Medium Risk: Weighted at 7.5.
- High Risk: Weighted at 5.
- Critical Risk: Weighted at 2.5.

E. Security Metrics

Critical security checks include:

- Security best practices such as CII-Best-Practices, Fuzzing, and more.

- Dependency and release management checks like Dependency-Update-Tool and Signed-Releases.
- Codebase checks including SAST and Branch-Protection.
- Community-related checks such as Code-Review and Contributors.

Each security metric returns a score from 0 to 10, with 10 indicating the highest level of security adherence. The aggregate score is a weighted average based on the individual checks' risk levels.

F. Algorithm for Aggregate Score

The algorithm for computing the aggregate score is adapted from the criticality score system referenced in the Open Source Security Foundation's Criticality Score algorithm [4]. Each check returns a score of 0 to 10, with 10 representing the best possible score. The aggregate score, denoted as $C_{project}$ is a weight-based average of the individual checks weighted by risk.

The algorithm is represented as follows:

$$C_{project} = \frac{1}{\sum_i \alpha_i} \sum_i \alpha_i \frac{\log(1 + S_i)}{\log(1 + \max(S_i, T_i))}$$

- S_i: Represents individual signals or metrics associated with a package's importance within the packaging syst em. These signals, such as download counts or number of dependents, contribute to assessing the package's significance.
- T_i: Denotes the threshold value for each signal S_i, defining the point at which a signal is considered maximally critical. It provides a boundary beyond which the signal's impact on the package's criticality becomes significant.
- α_i: Signifies the weight or relative importance assigned to each signal S_i. These weights allow the algorithm to scale signals based on their significance in determining the overall criticality of a package.

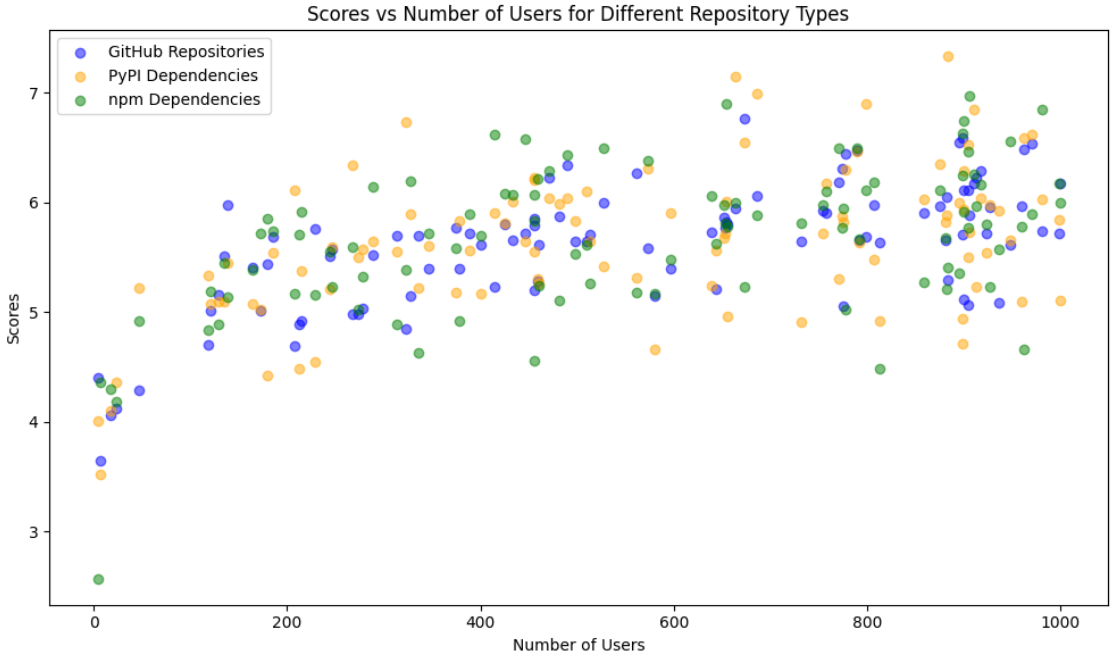
This algorithm aims to provide a singular numeric representation of a package's criticality within its respective packaging system. It considers various quantitative signals or metrics associated with the package's importance, scaling them based on their relative significance (α_i). The logarithmic operation in the formula allows for non-linear scaling, giving more significance to smaller changes for larger values, thereby preventing unbounded growth in criticality. The summing process aggregates the scaled signals, producing an overall score that represents the package's criticality.

6. Observations

The evaluation of Security and Community Health Metrics within Free and Open Source Software (FOSS) repositories involved an examination of the relationship between user count and assigned scores in GitHub repositories, PyPI dependencies, and npm dependencies.

Figure 7 illustrates a scatter plot representing the relationship between the scores of GitHub, npm, and PyPI dependencies concerning the number of users.

Figure 7: Score of GitHub, npm, PyPi dependencies vs Number of Users



The scatter plot analysis reveals a discernible pattern showcasing the positive influence of user count on the assigned scores. It is apparent that as the number of users increases, there is a notable inclination towards higher scores. This trend indicates a correlation wherein repositories or dependencies with a larger user base tend to receive comparatively higher scores.

This observation suggests a potential relationship between the magnitude of user engagement and the perceived security and community health metrics of the repositories and dependencies in question. The trend implies that projects or dependencies with a greater number of users might exhibit more robust security measures or healthier community interactions, resulting in elevated scores within these evaluation metrics.

Further investigation and statistical analysis could provide deeper insights into the nature and strength of this relationship, contributing to a more comprehensive understanding of the factors influencing the security and community health metrics of FOSS repositories.

7. Conclusion and Further Works

This work establishes a foundational approach to assessing the health of open-source projects. However, it's crucial to acknowledge the current limitations and areas for future enhancement.

A. Limitations

The process of data acquisition and analysis presents challenges. Initial data processing and presentation can be time-consuming, partly due to rate limits and varying speeds of vendor APIs crucial in sourcing the data. Moreover, the current scope predominantly focuses on well-known platforms hosting the bulk of publicly accessible repositories, limiting the breadth of analysis.

B. Future Scope

Looking ahead, several avenues for expansion exist:

- **Monetization Strategies:** One potential development involves monetizing the tool by providing actionable insights to enhance package health, adding a commercial dimension to its utility.

- **Security Beyond Code Repositories:** Expanding the tool's capabilities to assess cloud infrastructure security, accommodating other package managers, and integrating with additional source code hosting platforms would significantly broaden its applicability.
- **Integration with Development Workflows:** Further work could involve creating seamless integrations with developer tooling and workflows. Embedding the tool within automation pipelines could facilitate continuous monitoring and improvement of project health.

In conclusion, while the current system marks a significant step towards understanding and improving open-source project health, there's a compelling opportunity for further work to enhance its capabilities and reach.

References

1. I. Pashchenko, D. -L. Vu and F. Massacci, "Preliminary Findings on FOSS Dependencies and Security: A Qualitative Study on Developers' Attitudes and Experience," in 2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Seoul, Korea (South), 2020, pp. 284-285.
2. AutoMetric: An Automatic Technique for Measuring Security Metrics in Open-Source Software Repositories, in 2023 IEEE/ACM International Conference on Automation of Software Test (AST), Melbourne, Australia, 15-16 May 2023, doi: 10.1109/AST58925.2023.00009.
3. Vuln4Real: A Methodology for Counting Actually Vulnerable Dependencies in Free Open-Source Software Ecosystems, IEEE Transactions on Software Engineering, vol. 48, no. 5, pp. 1592-1609, May 2022, doi: 10.1109/TSE.2020.3025443.
4. J. Cox, E. Bouwers, M. van Eekelen, and J. Visser, "Measuring Dependency Freshness in Software Systems," in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 16-24 May 2015, doi: 10.1109/ICSE.2015.140.
5. A. Gkortzis, D. Mitropoulos, and D. Spinellis, "VulinOSS: A Dataset of Security Vulnerabilities in Open-Source Systems," in MSR '18: Proceedings of the 15th International Conference on Mining Software Repositories, May 2018, pp. 18-21, doi: 10.1145/3196398.3196454.
6. P. Ladisa, H. Plate, M. Martinez, and O. Barais, "Taxonomy of Attacks on Open-Source Software Supply Chains," in 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2023, pp. 1509-1526, doi: 10.1109/SP46215.2023.00010.
7. Dependabot Official Website, [Online]. Available: https://github.com/dependabot .
8. Snyk Official Website, [Online]. Available: https://snyk.io/ .
9. Chaoss, GitHub - chaoss/wg-metrics-models: Working Group for Metrics Model. Available: https://github.com/chaoss/wg-metrics-models .
10. Ossf, GitHub - ossf/scorecard: OpenSSF Scorecard - Security health metrics for Open Source. Available: https://github.com/ossf/scorecard .
11. npm | Home. Available: https://www.npmjs.com/ .
12. Ossf, GitHub - ossf/criticality score: Gives criticality score for an open source project. Available: https://github.com/ossf/criticality_score .
13. PyPI. PyPI · The Python Package Index , Available: https://pypi.org/ .
14. GitHub, GitHub API , GitHub, Inc. Available: https://api.github.com .