Towards Unveiling Vulnerability Remediation Tactics from OSS Community

Lyuye Zhang*, Jiahui Wu* Chengwei Liu*, Kaixuan Li† Sen Chen‡, Yang Liu*
zh0004ye@e.ntu.edu.sg, jiahui004@e.ntu.edu.sg, chengwei001@e.ntu.edu.sg, kaixuanli@stu.ecnu.edu.cn
*College of Computing and Data Science, Nanyang Technological University, Singapore
†Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China
‡College of Cryptology and Cyber Science, Nankai University, China

2025 IEEE/ACM International Workshop on Automated Program Repair (APR) | 979-8-3315-2585-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/APR66717.2025.00010

***Abstract*—In the rapidly evolving landscape of software development, addressing security vulnerabilities in open-source software has become highly important. However, existing research and tools from both academia and industry mainly relied on limited solutions, such as vulnerable version adjustment and adopting patches, to handle identified vulnerabilities. However, far more flexible and diverse countermeasures have been actively adopted in the open-source communities, which potentially complement the traditional remediation tactics provided by modern vulnerability databases. To this end, we conducted an empirical study to unveil the remediation tactics used in the GitHub community towards CVEs. Then we compared the tactics with those from modern vulnerability databases to underscore a critical gap in modern vulnerability databases, where 54% of CVEs lack fixing suggestions. This gap can be significantly mitigated by leveraging the 93% of actionable solutions provided through GitHub issues.**

***Index Terms*—Software Security**

I. Introduction

With the advantages of flexibility and transparency, open-source software (OSS) has been increasingly integrated into software projects. However, the adoption of OSS also brings challenges, particularly related to security issues. For instance, in 2021, a devastating vulnerability in the widely-used library Log4j [1] was reported to affect millions of devices worldwide. The Log4j incident highlights the far-reaching consequences security issues in OSS can have, which should be carefully remediated. Remediation, in the context of OSS security, involves a series of operations aimed at mitigating the risks posed by vulnerabilities [2], [3], [4], [5]. This process is crucial for both OSS developers and downstream users [6], [7], [8], [9], [10]. Remediation can ensure the security of OSS by fixing vulnerabilities, providing downstream users with stable and secure applications, and allowing OSS developers to maintain and improve the reliability of the software supply chain.

Researchers from both academia and industry have put efforts into investigating and improving vulnerability remediation. Some researchers focus on the lifecycles of vulnerabilities in OSS projects to reveal the effectiveness and timeliness of remediation [8], [11], [7], [6]. Other researchers concentrate on the prioritization of vulnerabilities for remediation due to concerns on cost-efficiency [9], [12], [13]. Moreover, the specific vulnerability remediation solutions, such as vulnerable version adjustment [14], [15], [4], vulnerability patching [16], [17], [18], and workarounds [19] are also well-studied by existing research. However, existing studies focus on mainstream remediation tactics in academia, overlooking the diverse solutions employed within the OSS community and leaving numerous approaches to security issue remediation unexplored.

As for remediation tools, many tools and advisories have been developed to assist the process of security issue remediation, which are usually incorporated in Software Composition Analysis (SCA) tools, and vulnerability databases. Eclipse Steady [20] and Coral [15] aid developers in identifying and addressing vulnerabilities in open-source libraries by upgrading or downgrading libraries to their secure versions. For industry tools, OSV-fix [21] and Snyk [22] streamline the process of fixing vulnerabilities in OSS dependencies directly in users' Software-Bill-of-Material (SBOM) files by modifying versions. Vulnerability databases, such as OSV [23] and GitHub Advisory [24], may also provide remediation suggestions for reference for a limited number of vulnerabilities. Nevertheless, as demonstrated by our study, existing solutions are still largely limited to well-known and straightforward solutions, such that they either heavily rely on upgrading the affected versions and applying patches or rarely offers remediation suggestions. Therefore, there is a clear shortage of solutions incorporated in existing tools.

To this end, we aim to conduct a comprehensive study to investigate the remediation solutions for Common Vulnerabilities and Exposures (CVEs) and their corresponding GitHub issues. Specifically, we summarized the categories of Remediation Tactics (RT) for both vulnerability databases and GitHub issues to identify their distinctions. We found a significant gap of 54% of CVEs lacking remediation solutions, which could be mitigated by tactics from the community.

II. Related Works

A. Vulnerability Fixing for OSS

Fixing vulnerabilities in OSS has been considered the last mile for vulnerability management [25]. Many research works have been conducted to investigate the RT of OSS vulnerabilities from various aspects. Specifically, some researchers [26], [27], [28], [29] focused on the lifecycles of vulnerabilities in OSS projects to reveal vulnerable library management. Pan et al. [8] distinguished the lifespans of critical and non-critical software vulnerabilities and found that critical vulnerabilities receive higher priority. Forootani et al. [11] explored the

979-8-3315-2585-9/25/$31.00 ©2025 IEEE
DOI 10.1109/APR66717.2025.00010

timeline of self-fixed vulnerabilities in OSS projects and found that they are resolved more quickly than non-self-fixed ones. Bandara et al. [7], [6] identified that 18% of vulnerability fixing commits introduced new vulnerabilities. Moreover, some researchers also focus on the criteria and prioritization of vulnerabilities to optimize remediation tactics for projects. Piantadosi et al. [12] compiled statistics on practitioners, timelines, and commits involved in remediation in two Apache projects. Shah et al. [13] studied the vulnerability selection criteria for remediation based on Multiple Attribute Value Optimization.

Moreover, many researchers also focus on detailed countermeasures and effects of vulnerability remediation. First, as the most adopted solution, **adjusting vulnerable versions** has been studied by many researchers. Liu et al. [14] proposed DTReme to exhaustively explore all possible dependency resolutions to find possible version-adjustment-based remediation tactics for NPM projects, Zhang et al. [15], [30] joint major concerns, such as compatibility, vulnerability reachability, and dependency conflicts, to propose an integrated solution for vulnerability remediation for Java projects. They also inspected the phenomenon of vulnerability persistence in most Java projects and proposed Ranger to remediate vulnerability impact from the ecosystem perspective. **OSS Patching** is also widely studied, Li et al. [16] found that security patches typically have a smaller footprint in code bases compared to non-security bug patches. Farris et al. [17] introduced the VULCON, taking various metadata information of vulnerabilities to prioritize patching. **Workaround** [31], considered a special form of remediation, also provides temporary and non-ideal fixes for vulnerabilities. Huang et al. [19] proposed Talos to generate workarounds for rapid vulnerability response.

III. Data Collection Methodology

GitHub, being the largest source code hosting platform, hosts numerous popular OSS projects with significant developer involvement. Thus we selected GitHub as the primary source for our base dataset. Our objective is to explore human-involved solutions to CVE-specific security issues, which extends beyond traditional RT. Note that pull requests were excluded from the scope of our study as they are solely associated with code-level changes, whereas our research scope extends beyond these to encompass a broader range of strategies. We primarily focused on *Issue* with free discussions, which allows for flexible RT proposals, included references to *PR* and *commit* in events of *Issue* to enrich the diversity of RT in our study.

GitHub Issue selection criteria have been systematically established by previous work, such as filtering out unresolved open issues [9], [15], filtering issues by community engagement (i.e., activeness) [32], [15], and excluding noisy information introduced by bots [33], [34]. Honoring the criteria from previous work, we detail our selection of issues as follows: to avoid including unresolved issues, we crawled *closed issues* from GitHub. As our focus is the fixing tactics regarding security issues, we selected the related keywords for searching via GitHub API, including *security/vulnerability/issue/bug & fix/remediation/tweak/repair/resolve* to encompass as many relevant issues as possible.

Additionally, to guarantee that the issue has been actively discussed, conservatively at least 3 comments and 2 distinct participants should be involved. Since our focus is on human-involved issues and human-developed RT, issues created by bots were subsequently filtered out. Furthermore, we assumed highly starred repositories are more likely to have a complete and robust issue-resolving mechanism, issues from repositories with fewer than 10k stars were filtered out. Within these issues, we further focused on those related to CVEs by mentioning CVE-IDs. By strictly filtering out issues not mentioning CVE-IDs at comments and titles, $4,462$ issues were derived. Specifically, we employed a two-step approach to pinpoint CVE-related issues. (1) We retrieved issues that mentioned at least one CVE-ID in their titles or comments. (2) Subsequently, the first two authors manually reviewed these issues to exclude issues that did not explicitly address the mentioned CVEs. The specific procedures were ① Reading the sentences mentioning CVE-IDs to determine if the CVEs are fixing targets or simply referenced as related vulnerabilities. ② For the seemingly fixing targets CVEs, we approached NVD entries for them to cross-check if the vulnerabilities were discussed in the issues. If either condition was not satisfied, the issues were filtered out. Eventually, $3,262$ out of $4,462$ issues were collected. In total, $3,529$ CVE-IDs were identified and saved.

IV. Empirical Study

This study aims to reveal the RT from the GitHub community and highlight the difference between it and the RT provided by modern vulenrability databases (VDBs). We first explored the prevalence of RT availability in modern VDBs. Then we labeled and analyzed the RT of both the 3,262 issues and corresponding CVEs for comparison.

A. Remediation Tactics Availability in VDBs

Modern VDBs, such as NVD [35], Snyk Vulnerability DB [36], Mend [37] OSV [23], GitHub Advisory [24], VulDB [38], VulnDB [39], Seclists [40], ExploitDB [41], and IBM XForce [42], may offer various RT. We focused on universal VDBs that are not specific to certain platforms, excluding those like UbuntuDB [43] focusing on Ubuntu-related issues. Additionally, DBs limited to access restrictions, such as VulnDB, were excluded. To quantitatively reveal the RT availability in these DBs, considering that most of them have no public API for automated crawling, we could only manually scrutinize samples of entries from their websites. For the diversity, we reviewed 100 entries (5 random entries for each year in the last two decades) for each DBs.

Table I lists the support of RT in popular VDBs. ◐ means that not all entries include the feature. ○ refers to no support. It is shown that only GitHub Advisory and commercial DBs have designated sections displaying remediation suggestions. Accordingly, all DBs except for ExploitDB may mention the remediation suggestions including workarounds in the vulnerability descriptions. However, the presence of RT is

TABLE I: Remediation Tactics Availability inx VDBs

DB name	Remediation	Workaround	Desc	Ref link
NVD [35]	○	○	◐	◐
GitHub Adv [24]	◐	◐	◐	○
OSV [23]	○	○	◐	○
SecList [40]	○	○	◐	○
ExploitDB [41]	○	○	○	○
***SnykVulDB** [36]	◐	◐	◐	◐
***Mend** [37]	◐	○	◐	○
***VulDB** [38]	◐	○	◐	○
***IBM XF** [42]	◐	○	◐	○

Desc *indicates if a remediation suggestion is included in the vulnerability description.* ***Ref link*** *indicates the involvement of designated links for remediation or patches. Commercial DBs are prefixed by *.*

not guaranteed. In fact, most of the entries have no RT as revealed in the next subsection IV-C.

Notably, GitHub Advisory and SnykVulDb have a designated section of workaround that provides alternative solutions for users if the remediation suggestions are not feasible. For example, the fixed version has breaking changes. Only NVD and SnykVulDB have the designated links for fixes including suggestions and patches, which could help users swiftly locate the external resources of remediation. Overall, the support of RT is not prevalent in modern VDBs, impeding the prompt fixes for downstream users.

B. Remediation Tactics from GitHub Issues

We employed the Hybrid Card Sorting [44] with majority voting to label the RT in GitHub issues. First, the first three authors seperately went through the issue titles and bodies to summarize the tactics with a phrase as a card and individually placed it into the group. Note that no pre-defined cards were provided to allow free ideation. In the spirit of Hybrid Card Sorting, the annotators may label an issue with an existing card from the group if they find it proper. After the individual labeling, 204 unique cards were generated, we then discussed together to de-duplicate them based on synonyms and similar concepts. Eventually, 6 cards were distilled. Then issues having conflict cards were reviewed. In case two identical cards were assigned to a issue, the label was settled as the card with major votes. In case three cards are different, the card was determined by a discussion of the authors. The labeling resulted in the 6 major categories of RT.

- **Version adjustment** (48.62%) usually refers to adjusting the versions of software components or dependencies used in the software projects by changing the versions in Bill-of-Material (BOM) files [45], such as Project Object Model (POM) [46] for Maven [47]. This category does not involve changes in source code files but only BOM files.
- **Patching vulnerabilities** (22.90%) refers to patching the source code or binary of the software projects instead of external configurations. The GitHub issues in this major category mostly involve file changes by mentioning related commits or PRs in the issue events.
- **Using alternative secure libraries** (0.74%) involves substituting an independent library or component within software projects. It is important to note that this category does not encompass the replacement of files or code snippets, which are classified under *Patching vulnerabilities*. The component or library being replaced should have identifiable names, and the replacement process can be executed either through BOM files or by replacing a group of files.
- **Changing settings or configurations** (13.58 %) is not the same as *Patching vulnerabilities* as it does not involve code change. It commonly changes the external configuration files or configurations on the application before and at runtime.
- **Bypassing vulnerabilities** (4.72%) refers to running the program in a different way to avoid executing the vulnerable part. Since vulnerability bypassing is pure subtractive, it is considered an independent category.
- **Re-initializing** (2.64%) involves starting anew with fresh components and data arrangements. This category encompasses three common subcategories: components, data, and building. We labeled an issue as Re-initializing based on comments suggesting that the program should be restarted or refreshed.
- **No actionable solution** (6.83%) does not belong to the categories of RT. It refers to suggestions that do not directly address the security issues immediately, such as *Wait for upstream fixes* and *Ignoring non-critical issues*.

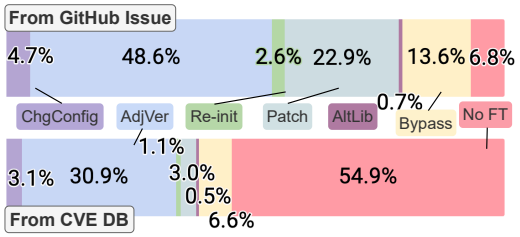

Fig. 1: Distributions of RT of VDBs and GitHub issue

C. Remediation Tactics from Vulnerability DB

To statistically understand the RT provided by VDBs, we manually analyzed the RT for three non-commercial DBs. We fetched the CVE descriptions and relevant sections from these DBs and categorized the RT in a similar manner as before. Due to the accessibility, we only focused on non-commercial DBs. Considering the diverse and abundant descriptions, NVD, GitHub Advisory, and OSV were included in our dataset. To prepare the data, we crawled the descriptions from DBs for the 3,044 CVEs that linked to GitHub issues.

The categories of RT in VDBs is depicted in Figure 1. The lower bar depicts the distribution of RT major categories from VDBs, and the upper one is from the GitHub issue. It is observed that 54.85% of CVEs lack remediation suggestions, underscoring the need for additional sources of RT, such as community-driven platforms like GitHub. Fortunately, 93.26% of issues have actionable RT for these CVEs. Towards the categories that are rarely present in VDBs, such as Patching vulnerabilities, Bypassing vulnerabilities, and those substantially fewer than GitHub Issue, such as Version adjustment, RT from GitHub Issue could introduce much more RT to achieve much better coverage of remediation for CVEs.

D. Threats of Validity

The main threat to our study is the use of manual labeling and categorization, which can inevitably introduce biases. To counteract these biases and maintain a balance between efficiency and accuracy, we employed a majority voting to compensate for the inaccuracy of labeling.

Another threat lies in the sample scrutinization for VDBs. Due to the limited access to DBs and implicit RT statements, a manual examination had to be conducted. The limited size of samples inevitably results in an incomplete overview of RT. We have involved entries published in the last two decades to increase the diversity as much as possible.

V. Conclusion

Our study on OSS security remediation has filled significant research gaps by analyzing 3,262 GitHub issues. The RT for both GitHub issues and VDB suggestions were summarized. We revealed that the absent RT for 54% highlights the need for ongoing enhancements and support for RT by VDBs to address the evolving landscape of OSS vulnerabilities.

References

[1] "Log4j Vulnerability News," https://www.securityweek.com/one-year-later-log4shell-remediation-slow-painful-slog/, 2023.

[2] Y. Wu, Z. Yu, M. Wen, Q. Li, D. Zhou, and H. Jin, "Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem," in *45th International Conference on Software Engineering*, 2023, pp. 1–12.

[3] Y. Wang, P. Sun, L. Pei, Y. Yu, C. Xu, S.-C. Cheung, H. Yu, and Z. Zhu, "Plumber: Boosting the propagation of vulnerability fixes in the npm ecosystem," *IEEE Transactions on Software Engineering*, 2023.

[4] L. Zhang, C. Liu, S. Chen, Z. Xu, L. Fan, L. Zhao, Y. Zhang, and Y. Liu, "Mitigating persistence of open-source vulnerabilities in maven ecosystem," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 191–203.

[5] Y. Sun, D. Wu, Y. Xue, H. Liu, W. Ma, L. Zhang, Y. Liu, and Y. Li, "Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning," *arXiv preprint arXiv:2401.16185*, 2024.

[6] V. Bandara, T. Rathnayake, N. Weerasekara, C. Elvitigala, K. Thilakarathna, P. Wijesekera, and C. Keppitiyagama, "Fix that fix commit: A real-world remediation analysis of javascript projects," in *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2020, pp. 198–202.

[7] V. Bandara, T. Rathnayake, N. Weerasekara, C. Elvitigala, K. Thilakarathna, P. Wijesekera, K. De Zoysa, and C. Keppitiyagama, "Large scale analysis on vulnerability remediation in open-source javascript projects," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2447–2449.

[8] S. Pan, L. Bao, J. Zhou, X. Hu, X. Xia, and S. Li, "Unveil the mystery of critical software vulnerabilities," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 2024, pp. 138–149.

[9] S. Panichella, G. Canfora, and A. Di Sorbo, ""won't we fix this issue?" qualitative characterization and automated identification of wontfix issues on github," *Information and Software Technology*, vol. 139, p. 106665, 2021.

[10] V. Piantadosi, S. Scalabrino, and R. Oliveto, "Fixing of security vulnerabilities in open source projects: A case study of apache http server and apache tomcat," in *2019 12th IEEE Conference on software testing, validation and verification (ICST)*. IEEE, 2019, pp. 68–78.

[11] S. Forootani, A. Di Sorbo, and C. A. Visaggio, "An exploratory study on self-fixed software vulnerabilities in oss projects," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 90–100.

[12] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.

[13] A. Shah, K. A. Farris, R. Ganesan, and S. Jajodia, "Vulnerability selection for remediation: An empirical analysis," *The Journal of Defense Modeling and Simulation*, vol. 19, no. 1, pp. 13–22, 2022.

[14] C. Liu, S. Chen, L. Fan, B. Chen, Y. Liu, and X. Peng, "Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 672–684.

[15] L. Zhang, C. Liu, Z. Xu, S. Chen, L. Fan, L. Zhao, J. Wu, and Y. Liu, "Compatible remediation on vulnerabilities from third-party libraries for java projects," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 2540–2552. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00212

[16] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2201–2215.

[17] K. A. Farris, A. Shah, G. Cybenko, R. Ganesan, and S. Jajodia, "Vulcon: A system for vulnerability prioritization, mitigation, and management," *ACM Trans. Priv. Secur.*, vol. 21, no. 4, jun 2018. [Online]. Available: https://doi.org/10.1145/3196884

[18] L. Zhang, K. Li, K. Sun, D. Wu, Y. Liu, H. Tian, and Y. Liu, "Acfix: Guiding llms with mined common rbac practices for context-aware repair of access control vulnerabilities in smart contracts," *arXiv preprint arXiv:2403.06838*, 2024.

[19] Z. Huang, M. DAngelo, D. Miyani, and D. Lie, "Talos: Neutralizing vulnerabilities with security workarounds for rapid response," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 618–635.

[20] "Steady," https://projects.eclipse.org/proposals/eclipse-steady, 2023.

[21] "Osv-scanner fix," https://google.github.io/osv-scanner/experimental/guided-remediation/, 2024.

[22] "Snyk," https://snyk.io/, 2023.

[23] "Osv," https://osv.dev/, 2023, (Accessed on 02/17/2023).

[24] "Github Security Advisory," https://github.com/advisories, 2023.

[25] "Eliminating the last mile between security data and decision making - cyber defense magazine," https://www.cyberdefensemagazine.com/eliminating-the-last-mile-between-security-data-and-decision-making/, 2024, (Accessed on 08/03/2024).

[26] J. Hu, L. Zhang, C. Liu, S. Yang, S. Huang, and Y. Liu, "Empirical analysis of vulnerabilities life cycle in golang ecosystem," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[27] J. Wu, Z. Xu, W. Tang, L. Zhang, Y. Wu, C. Liu, K. Sun, L. Zhao, and Y. Liu, "Ossfp: Precise and scalable c/c++ third-party library detection using fingerprinting functions," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 270–282.

[28] L. Zhao, S. Chen, Z. Xu, C. Liu, L. Zhang, J. Wu, J. Sun, and Y. Liu, "Software composition analysis for vulnerability detection: An empirical study on Java projects," in *Proceedings of the 2023 31th acm sigsoft international symposium on foundations of software engineering*, 2023.

[29] F. Zhang, L. Fan, S. Chen, M. Cai, S. Xu, and L. Zhao, "Does the vulnerability threaten our projects? automated vulnerable api detection for third-party libraries," *IEEE Transactions on Software Engineering*, 2024.

[30] L. Zhang, C. Liu, Z. Xu, S. Chen, L. Fan, B. Chen, and Y. Liu, "Has my release disobeyed semantic versioning? static detection based on semantic differencing," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023, pp. 1–12. [Online]. Available: https://doi.org/10.1145/3551349.3556956

[31] A. Khazaei, M. Ghasemzadeh, and C. Meinel, "Vuwadb: A vulnerability workaround database," *International Journal of Information Security and Privacy (IJISP)*, vol. 12, no. 4, pp. 24–34, 2018.

[32] J. Jiang, D. Lo, X. Ma, F. Feng, and L. Zhang, "Understanding inactive yet available assignees in github," *Information and Software Technology*, vol. 91, pp. 44–55, 2017.

[33] M. Wessel, A. Abdellatif, I. Wiese, T. Conte, E. Shihab, M. A. Gerosa, and I. Steinmacher, "Bots for pull requests: The good, the bad, and the promising," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 274–286.

[34] M. Wessel, I. Wiese, I. Steinmacher, and M. A. Gerosa, "Don't disturb me: Challenges of interacting with software bots on open source software projects," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW2, pp. 1–21, 2021.

[35] “National vulnerability database,” https://nvd.nist.gov/, 2023.
[36] “Snyk Vulnerability Database,” https://security.snyk.io/, 2023.
[37] “Mend,” https://www.mend.io/, 2024.
[38] “VulDB,” https://vuldb.com/, 2024.
[39] “VulnDB,” https://vulndb.flashpoint.io/, 2024.
[40] “SecLists,” https://seclists.org/, 2024.
[41] “ExploitDB,” https://www.exploit-db.com/, 2024.
[42] “IBM XForce,” https://exchange.xforce.ibmcloud.com/ip/, 2024.
[43] “Ubuntu CVE,” https://ubuntu.com/security/cves.json, 2024.
[44] “Hybrid card sorting,” https://support.optimalworkshop.com/en/articles/2626850-choose-between-an-open-closed-or-hybrid-card-sort, 2021.
[45] “Bill of Materials,” https://en.wikipedia.org/wiki/Bill_of_materials, 2024.
[46] “Project object model,” https://maven.apache.org/guides/introduction/introduction-to-the-pom.html, 2023.
[47] “Maven,” https://maven.apache.org/, 2023.