An Empirical Study on Vulnerability Disclosure Management of Open Source Software Systems

SHUHAN LIU, Zhejiang University, Hangzhou, China
JIAYUAN ZHOU, Waterloo Research Center, Huawei, Markham, Ontario, Canada
XING HU, Zhejiang University, Hangzhou, China
FILIPE ROSEIRO COGO, Centre for Software Excellence, Huawei, Markham, Ontario, Canada
XIN XIA, Software Engineering Application Lab, Huawei Technologies Co Ltd, Hangzhou, China
XIAOHU YANG, Zhejiang University, Hangzhou, China

Vulnerability disclosure is critical for ensuring the security and reliability of open source software (OSS). However, in practice, many vulnerabilities are reported and discussed on public platforms before being formally disclosed, posing significant risks to vulnerability management. Inadequate vulnerability disclosure can expose users to security threats and severely impact the stability and reliability of software systems. For example, prior work shows that over 21% of CVEs are publicly discussed before a patch is released. Despite its importance, we still lack clarity on the vulnerability disclosure practices adopted by open source communities and the preferences of practitioners regarding vulnerability management. To fill this gap, we analyzed the vulnerability disclosure practices of 8,073 OSS projects spanning from 2017 to 2023. We then conducted an empirical study by surveying practitioners about their preferences and recommendations in vulnerability disclosure management. Finally, we compared the survey results with the actual vulnerability practice observed within the OSS projects. Our results show that while over 80% of practitioners support Coordinated Vulnerability Disclosure (CVD), only 55% of vulnerabilities conform to CVD in practice. Although only 20% of practitioners advocate discussions before disclosure, 42% of vulnerabilities are discussed in issue reports before their disclosure. This study reveals the vulnerability management practices in OSS, provides valuable guidance to OSS owners, and highlights potential directions to improve the security of OSS platforms.

CCS Concepts: • **Security and privacy → Vulnerability management**;

Additional Key Words and Phrases: Empirical Studies, Vulnerability Disclosure Model, Practitioner's Expectations

This work was supported by the National Key R&D Program of China (No. 2024YFB4506400) and Ningbo Natural Science Foundation (No. 2023J292).

Authors' Contact Information: Shuhan Liu, Zhejiang University, Hangzhou, China; e-mail: liushuhan@zju.edu.cn; Jiayuan Zhou, Waterloo Research Center, Huawei, Markham, Ontario, Canada; e-mail: jiayuan.zhou1@huawei.com; Xing Hu (corresponding author), Zhejiang University, Hangzhou, China; e-mail: xinghu@zju.edu.cn; Filipe Roseiro Cogo, Centre for Software Excellence, Huawei, Markham, Ontario, Canada; e-mail: filipe.roseiro.cogo1@huawei.com; Xin Xia, Software Engineering Application Lab, Huawei Technologies Co Ltd, Hangzhou, China; e-mail: xin.xia@acm.org; Xiaohu Yang, Zhejiang University, Hangzhou, China; e-mail: yangxh@zju.edu.cn.

ACM 1557-7392/2025/8-ART214
https://doi.org/10.1145/3716822

ACM Reference format:
Shuhan Liu, Jiayuan Zhou, Xing Hu, Filipe Roseiro Cogo, Xin Xia, and Xiaohu Yang. 2025. An Empirical Study on Vulnerability Disclosure Management of Open Source Software Systems. *ACM Trans. Softw. Eng. Methodol.* 34, 7, Article 214 (August 2025), 31 pages.
https://doi.org/10.1145/3716822

1 Introduction

From small-scale projects to critical applications deployed on a global scale, **Open Source Software (OSS)** is widely used in the software industry [18, 57]. OSS offers numerous advantages [53] (e.g., low cost and high quality [60]), but the use of OSS also brings security risks in the form of vulnerabilities [15, 16, 35, 46, 62]. Therefore, vulnerability management is important in practice. To manage vulnerabilities, OSS developers need to put in place a process that involves the identification, prioritization, and remediation of vulnerabilities. Documentation resources from OSS projects, such as Wiki, `Readme`, or the project's Web site, often offer guidance on reporting and managing vulnerabilities, for which a commonly followed model is the **Coordinated Vulnerability Disclosure (CVD)** [14, 29, 32, 44]. The CVD recommends that vulnerabilities are reported privately to software maintainers, ensuring that key information remains confidential until maintainers are ready to make it public (e.g., after the vulnerability is fixed), potentially preventing attackers from exploiting the reported vulnerability before a patch is provided and its availability is officially disclosed [22]. However, in practice, numerous factors, including the lack of expertise in the security domain [21] and limited knowledge in security management [1], often lead reporters not to comply with CVD leading to the premature disclosure of vulnerabilities and providing attackers with opportunities such as zero-day attacks [2, 8].

Recently, OSS vulnerability management has received considerable attention. Several works studied the creation of disclosure histories model [30] and the analysis of vulnerability lifecycle [3, 20, 54]. Researchers have also evaluated the efficacy of vulnerability bounty programs and highlighted dangerous behaviors during the vulnerability disclosure process [8, 36, 43]. Despite numerous studies on vulnerability management in OSS, few studies [17, 28] have investigated practitioners' preferences regarding disclosure models. In particular, to the best of our knowledge, none of the previous research has studied the practitioners' preferences of reporting channels, pre-disclosure or post-disclosure discussions, and disclosure roles. Furthermore, there is a lack of research on vulnerability disclosure practices within open source platforms (e.g., GitHub) and the discrepancies between practitioners' preferences and modus operandi. Acquiring insights into the perspectives of practitioners and how they behave in practice is vital to improving the vulnerability disclosure process and discerning the key factors in issue reporting. These insights can guide OSS owners when selecting appropriate security policies and vulnerability disclosure management approaches.

To comprehensively understand the prevalence of vulnerability disclosure practices in OSS, we conducted a large-scale empirical study of 21,501 vulnerabilities linked to GitHub repositories, combining data from the **National Vulnerability Database (NVD)** [58] and **GitHub Advisory (GHSA)** [25]. We also surveyed 770 professional practitioners from 2,000 GitHub repositories that had previously reported **Common Vulnerabilities and Exposures (CVEs)** [41] entries, gathering insights into their preferences and recommendations throughout the vulnerability disclosure process. Finally, we conducted a comprehensive comparison between real-world practices and practitioners' preferences to explore the gap between these two aspects regarding vulnerability management. In particular, we investigated the following two research questions:

RQ1: What are the prevalent vulnerability disclosure practices within the open source platform?

The main objective of RQ1 is to investigate the practices of vulnerability disclosure management in OSS. We first analyzed the vulnerability handling guidance (e.g., security policy or vulnerability report channels) provided by OSS. We then studied the OSS vulnerability practices by analyzing the vulnerability response (e.g., report, patch, and disclosure) throughout the lifecycle of a vulnerability. We found that 62% of the studied OSS lack information about vulnerability handling guidance. Furthermore, we observed several dangerous practices in OSS. Specifically, 42% of CVEs are publicly discussed before their disclosure, and 34% of CVEs are fixed directly (e.g., direct commits to the main branch) and explicitly (e.g., describe the vulnerability in the commit message), which means attackers could discover the vulnerability before disclosure. Additionally, the decision whether to use **Pull Requests (PRs)** to merge fixes depends on the conventions of the projects and is correlated with the severity of the vulnerability. In the cases of severe vulnerabilities, the tendency to use PRs decreases, leading to faster fixes. Furthermore, in terms of vulnerability disclosure models, 56% of the studied vulnerabilities adhere to a CVD approach, while 38% follow a full disclosure approach. Finally, vulnerabilities that are publicly reported but left unfixed are often associated with small-scale and low-active projects, highlighting risks for the consumers of these types of projects.

RQ2: What are the preferences of practitioners in the OSS community regarding vulnerability disclosure management?

RQ2 investigates practitioners' personal preferences in the realm of vulnerability disclosure. Among the respondents, 65% have experience in handling security reports, while only 16.6% have ever requested a CVE. In terms of their preferences for a vulnerability disclosure model, over 81% of the respondents prefer CVD, while 10% favor private disclosure and 7% prefer full disclosure. However, only 57% of the respondents believe that public discussions should occur after the publication of the CVE, which contradicts the proportion of supporters of CVD. Regarding the role of individuals opening CVEs, most of the respondents prefer security experts. As for the channels for reporting vulnerabilities, most of the respondents favor direct contact with the organization (73%) or using an existing issue tracking system (61%).

To investigate the gap between actual practices and preferences, and gain deeper insights, we conducted a comparative analysis of vulnerability disclosure management practices and practitioners' preferences. We have uncovered some disparities: while over 80% of practitioners support the usage of CVD, only 56% of the vulnerability disclosures conform to CVD. Although only 20% of practitioners advocate publicly discussing the vulnerability before disclosure, 42% of the vulnerabilities are discussed in issue reports before their disclosure. This discrepancy can be attributed to the fact that 60% of the repositories lack documentation about security policies and private contact channels (see Sections 6.1 and 6.2). Additionally, vulnerabilities that are publicly reported but left unfixed are often associated with small-scale and low-active projects. To solve this problem, practitioners can implement time limits for CVD. If a vulnerability remains unpatched beyond the specified time limit, full disclosure of the vulnerability should be initiated to inform users about the existence of the vulnerability.

This article makes the following contributions:

- A comprehensive investigation of how practitioners approach vulnerability disclosure in practice, based on 21,501 NVD and GHSA vulnerabilities disclosed within the past 5 years, along with 8,073 GitHub projects that have disclosed vulnerabilities.
- A survey with 770 professional practitioners from 2,000 distinct GitHub repositories that had previously disclosed CVEs to shed light on practitioners' preferences, including their views on various aspects of vulnerability management.
- A set of empirically grounded guidelines practitioners can adopt when implementing vulnerability disclosure management in their OSS projects.

— To facilitate research replicability and verifiability, we provide a replication package[1] that includes the datasets and code used in our study, as well as a partial analysis of the questionnaire results.

2 Background

Vulnerability Disclosure Model: Vulnerability disclosure models serve as guidelines for vulnerability discovers and maintainers to report and handle vulnerabilities in OSS projects. There are three common vulnerability disclosure models [42]: (1) *Full Disclosure:* The full details of the vulnerability are made public as soon as they are identified. The full details may contain the exploit code. Thus, the full vulnerability details are usually disclosed before a patch is available, exposing them to potential exploitation by attackers. (2) *Private Disclosure:* The vulnerability is reported privately to the organization. The organization may choose to publish the details of the vulnerabilities or not. This decision is made by the organization, not the vulnerability discoverers, meaning that many vulnerabilities may never be made public. (3) *CVD* [29]*:* The initial report is made privately. However, once a patch has been made available, full details of the vulnerability should be published. Sometimes, publishing the vulnerability details could be delayed to allow more time for the patches to be installed. It is widely adopted within the security community, including organizations such as Apache [5], Microsoft [39], and Project Zero of Google [27].

CVE: CVE provides a standardized method to identify, define, and catalog publicly disclosed vulnerabilities [41]. The CVE platform assigns a unique ID to each vulnerability, enabling users to recognize and track specific vulnerabilities across different databases. Each CVE record includes a brief and project-oriented description of the vulnerability and the related references (e.g., vulnerability reports, patches, and discussions) [48].

NVD: NVD [58] is one of the most popular security advisories [63]. NVD fully syncs with the CVE list, ensuring that once a CVE record is published, it promptly appears in NVD. In addition to the information included in CVE records, NVD provides enhanced vulnerability information including the **Common Vulnerability Scoring System (CVSS)** score [19] and the weakness type (CWE [40]). The external links in CVE entries related to GitHub may include various references, such as issue reports, commits, or PR links.

GHSA: GHSA [25] is a platform-specific resource provided by GitHub, focusing on vulnerabilities discovered in projects hosted on this platform. When security researchers or others discover vulnerabilities in a project, they can report them to GitHub. The GitHub team manually verifies the reported vulnerabilities to confirm their validity and severity. Once a vulnerability is confirmed and the project maintainers agree to publish a security advisory, GitHub creates a corresponding entry in GHSA. It offers a curated database of vulnerabilities, including project information, affected versions, and CVSS score. GHSA also provides links to related CVE entries and external resources.

The Lifecycle of a Vulnerability: The lifecycle of a vulnerability in an OSS project includes four steps: (1) *Discovery:* The vulnerability is identified through internal testing or external research. (2) *Report:* The vulnerability is reported to the responsible organization, either privately or publicly. Private reporting allows the organization to address the issue before disclosing it. In GitHub, public report information can be found in the issue report. (3) *Fix:* The organization develops a fix to address the vulnerability. Patch information can be found in commits which can originate from various sources, including forks or other branches. Some organizations will use PRs to merge the patch. (4) *Disclosure:* The vulnerability is publicly disclosed on platforms like NVD or GHSA. When disclosing, vulnerability response events, such as the issue report, commit and PRs, can also be provided.

[1] https://github.com/MapleQwQ/Empirical-on-Vulnerability-in-OSS.

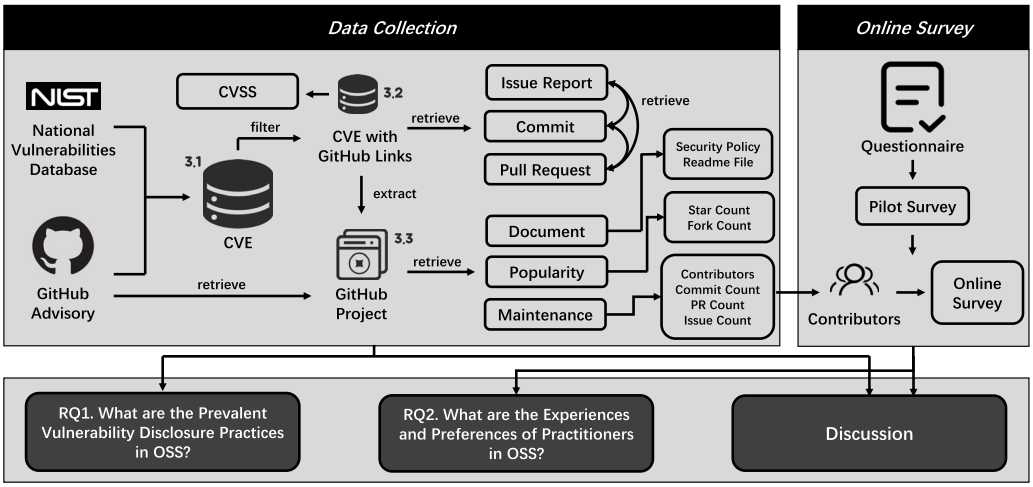

Fig. 1. An overview of our data collection and online survey process.

Vulnerability Response: We categorize the actions related to vulnerabilities throughout their lifecycle as vulnerability responses. Vulnerability responses are various actions project maintainers take while discovering and fixing a vulnerability. In the GitHub context, the generated events during a vulnerability response typically include issue reports, commits, PRs, and public disclosures.

3 Data Collection

In this section, we present our data collection procedure, whose steps are depicted in Figure 1. To explore the vulnerability disclosure practices of OSS, we first collected vulnerabilities related to OSS projects and then the relevant pieces of information about these vulnerabilities. Our dataset primarily consists of vulnerability information (e.g., CVE entries), vulnerability response events (e.g., issues, commits, and PRs), and OSS project information related to vulnerabilities (e.g., security-related documentation files).

3.1 Disclosed OSS Vulnerability Information

We utilized two primary sources, the NVD [58] and GHSA [25], as essential references to identify publicly disclosed vulnerabilities. NVD offers a broader perspective on vulnerabilities across the software landscape, and GHSA focuses on vulnerabilities specifically within OSS projects hosted on GitHub. This dual approach allows us to gain a more comprehensive dataset of vulnerability disclosure management strategies for OSS projects than selecting a single data source. We used two key datasets for our analysis: the NVD JSON dataset [59], obtained from the September 7, 2023 snapshot, and the GHSA JSON dataset [26], also sourced from the September 7, 2023 snapshot. We extracted data from the dataset spanning the years from 2017 to 2023, which served as our initial dataset, containing a total of 160,594 CVE vulnerabilities. This dataset includes 160,317 NVD entries and 11,952 GHSA entries. We further filter data related to GitHub based on this dataset in the following steps.

3.2 Vulnerability Response Events

From the collected vulnerability dataset (Section 3.1), we retained all vulnerabilities with CVE IDs. Among 160,594 CVEs disclosed in NVD or GHSA, 21,501 CVEs include GitHub links in their external links, representing approximately 14% of the total. To better understand the characteristics

of each CVE (e.g., severity and vulnerability responses), we also obtained detailed information about CVEs associated with OSS. In particular, we selected CVEs with external references linking to GitHub issue reports, commits, or PRs. To determine the severity and potential impact of these CVEs, we obtained their disclosure time and CVSS scores from NVD and GHSA. Then, to collect vulnerability responses related to CVEs, we utilized the external links in the CVE entries to find additional relevant information. We employed links between issues, commits, and PRs to retrieve related information. PRs often include links to issue reports and commits. We traced and retrieved the associated issues and commits using these links. Similarly, we identified relevant PRs and issue reports through commit links.

Following this procedure, we established a network of relationships for each CVE. We classify the obtained vulnerability responses into three distinct lifecycle phases of a vulnerability:

— *Report*: Information related to the initial reporting of vulnerabilities is collected from issue links. For instance, Qwaz [51] reported a memory-safety vulnerability in rust-smallvec by opening an issue. The issue contained a detailed description and reproduction steps for the vulnerability. The report time for each CVE is determined based on the earliest timestamps associated with these issues.
— *Fix*: This category contains commits and PRs related to the CVE. Analyzing the earliest timestamps of these commits and PRs helps us determine the fixed time. When a vulnerability is fixed through multiple commits, we use the timestamp of the earliest commit. To examine if attackers could discover the fix earlier, we checked if the vulnerability was fixed explicitly or directly. This means an attacker could leverage the fix information to discover the vulnerability before the fix gets into production, especially when not every downstream user updates to the version that includes the fix. To determine if the fix was explicit, we examined the commit message to identify vulnerability-related keywords using specific regular expression patterns. We expanded our regular expressions based on the work by Zhou and Sharma [64] to cover a wide range of vulnerabilities. These regular expressions include almost all possible expressions and keywords associated with security issues, including terms such as security, vulnerability, attack, CVE, DoS, and XSS (please refer to the replication package for a list of the set of rules). To ensure the accuracy of our automated classification, we conducted a manual validation by reviewing a random sample of 200 commit messages. During this validation, we found that commits identified as explicit fixes were almost always correctly classified. However, among the 125 commits identified as non-explicit fixes, we discovered 14 that should have been classified as explicit fixes. This validation confirmed that the overall accuracy of the explicit fix classification is reliable, with only a slight underestimation of explicit fixes. Then, we investigated the origin of commits, checking if they originated from other branches or were forked from other repositories to determine if the commits were made directly.
— *Disclosure*: Disclosure times are derived from the publication dates obtained from the NVD and GHSA databases data. This information offers insights into the public disclosure of CVEs.

In total, we collected 21,501 CVEs and their related information.

3.3 OSS Project Information

We followed two steps to obtain the OSS projects related to vulnerabilities. First, we extracted GitHub projects from the external links provided in the NVD and the GHSA. These links led to the corresponding GitHub projects associated with reported vulnerabilities. Second, we collected GitHub projects from the GHSA database, as each vulnerability documented in GHSA was inherently linked to a GitHub project. We merged the projects from both steps, ensuring the removal of duplicates to preserve data integrity and eliminate redundancy.

Our data collection encompassed a comprehensive set of project-specific details, including:

— *Documentation*: We gathered each project's "Security Policy" section and `Readme` files, which furnished essential information about the project, including guidelines for security practices. Furthermore, the "Security Policy" section allows maintainers to configure a private vulnerability reporting button and allows finders to report vulnerabilities in a private manner [24]. We also verified the presence of this configuration.
— *Popularity*: To evaluate the significance and activity of the projects within the OSS ecosystem, we employed the GitHub API to retrieve metrics including star counts and fork counts. These metrics provided valuable indicators of the popularity of the projects and community involvement.
— *Maintenance*: To elucidate the maintenance status of the projects and facilitate subsequent questionnaire surveys, we collected information on project contributors, including their names and e-mail addresses. Additionally, we collected metrics including commit counts, PR counts, and issue counts, providing insights into maintenance activities and project engagement levels.

In total, we gathered 8,073 GitHub projects related to vulnerabilities and their relevant details.

4 RQ1: What Are the Prevalent Vulnerability Disclosure Practices in OSS?

4.1 Motivation

Although CVD recommends reporting vulnerabilities privately and disclosing them after they have been fixed, not all OSS projects adhere to the CVD process. For example, Li and Paxson [36] observed that 21.2% of CVEs were publicly discussed but not yet fixed. They also highlighted variations in vulnerability disclosure practices within the OSS community [6, 30, 36, 37]. Some reporters use public forums where they disclose all information about a vulnerability, including technical details and sometimes even exploit code, even if the vendor has not yet released a patch [6]. This raises the question: What are the real-world practices of vulnerability disclosure in OSS? By addressing this question, we can gain insight into the various strategies used by OSS projects to address vulnerability and contribute to improving vulnerability management practices in the OSS community.

4.2 Approach

To investigate the OSS vulnerability disclosure practices, we first analyzed the vulnerability handling guidance of OSS at the repository level and then examined the vulnerability responses of the collected vulnerabilities from both individual behaviors and their combination sequences.

4.2.1 Vulnerability Handling Guidance of OSS. We analyzed vulnerability handling guidance from the perspective of security policy and private contact information.

A comprehensive investigation was conducted to evaluate the implementation of security policies in OSS projects. Security policy information can be found in the "Security Policy" section of a project. We examined the existence of the "Security Policy" section in each project. However, some projects lack a "Security Policy" section and instead include security-related information in the `Readme` file. We used an iterative keyword-based approach to identify the presence of security policies within these `Readme` files of the projects. To get the keywords related to security policy, we manually reviewed `Readme` files of 200 projects. Initially, we identified 16 keywords, such as "security policy," "security issue," and "vulnerability reporting." These keywords were used to filter projects by checking whether their `Readme` files contained identified security policy keywords. Subsequently, we divided the projects into two groups based on the presence or

absence of these keywords. From the filtered set of projects containing the keywords, we randomly selected 100 projects for further examination to assess the accuracy and relevance of the keywords. Additionally, we examined another 100 randomly selected projects that did not contain the identified keywords to uncover any potential missing terms. This iterative process was repeated five times, each iteration refining and improving the keyword list The final iteration resulted in 21 keywords with their corresponding frequencies of occurrence, documented in the replication package.

To check private contact information, we investigated whether `Readme` files and the "Security Policy" sections of OSS projects contained e-mail addresses. Then, we verified the presence of a private report button within the "Security Policy" section.

4.2.2 Vulnerability Responses. We are interested in four types of responses: issue reports, commits, PRs, and disclosure. We first analyzed them individually, then we combined them and analyzed the sequence of vulnerability responses. For each type of response, we examined the response time and calculated the percentage of vulnerabilities associated with the particular response.

For the issue report, we categorized reported vulnerabilities into two groups based on the timing of the reporting and disclosure. We then analyzed the characteristics and the impact of the report in each category. For commits, we examined whether there were issue reports before the commits to determine if the vulnerabilities were privately reported. We then analyzed whether the commits were explicit or direct to examine whether they could have been discovered earlier by attackers. For PRs, we checked the time interval between PR and disclosure and analyzed the relationship between the use of PR and repository or the severity of vulnerability. For disclosure, we want to investigate the correlation between the inclusion of patches when disclosing and the severity of vulnerability or the repository conventions.

Finally, we studied the sequence of vulnerability responses in each phase of the lifecycle to summarize different types of vulnerability disclosure in practice. We analyzed the sequence of these responses to determine the order in which fix, report, and disclosure activities occurred. For each cycle, we only considered the creation date of the earliest responses. We designated the earliest timestamp of the issue report as the reporting time. Considering that both commits and PRs are intended to fix vulnerabilities, we determined the earliest timestamp between them as the fix time. We categorized vulnerabilities based on the sequence of these responses and conducted a detailed analysis for each category.

4.3 Result

4.3.1 Vulnerability Handling Guidance of OSS. The adoption of security policies varies among OSS. This section focuses on investigating the existence of security policies and discussing approaches for privately contacting OSS maintainers.

74% (5,984 out of 8,073) of studied repositories lack security policies. Among the 8,073 analyzed repositories, 1,749 (22%) contain an explicit security policy module, and 624 (8%) repositories include security policies in their `Readme` files.

68% (5,489 out of 8,073) of the studied repositories lack private contact information, and 60% (4,805 out of 8,073) lack both security policies and contact information. Out of the 8,073 analyzed repositories, 32% repositories offer private contact information for vulnerabilities, in which 2,061 (26%) contain contact information in their `Readme` files or security policy files and 905 (11%) repositories provide a private report button for vulnerability finders. Among the 5,984 repositories without explicit security policies, 1,179 (14.6%) provide private contact information. Consequently, 60% of the repositories lack both clearly defined security policies and private contact information.

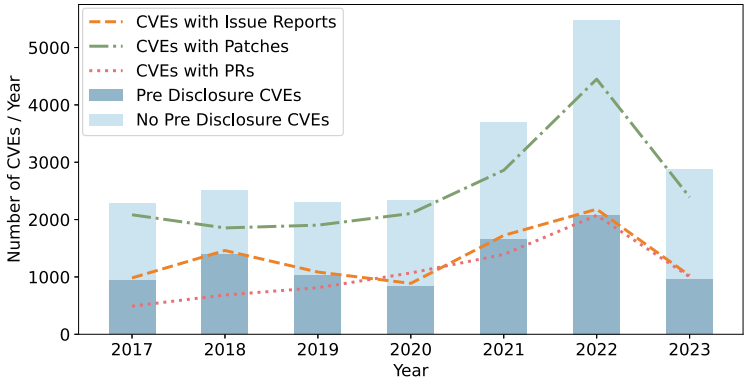

Fig. 2. Trends in vulnerabilities and vulnerability responses.

Summary. A significant portion (60%) of OSS lack both security policies and private contact approaches, resulting in vulnerability discoverers being unable to follow established procedures for reporting vulnerabilities and lacking channels for private vulnerability disclosure.

4.3.2 Vulnerability Responses. This section focuses on discussing vulnerability responses with an analysis of each type of response.

Figure 2 illustrates the trends in the number of vulnerabilities and vulnerability responses in our dataset from 2017 to 2023. Overall, the total number of CVEs increased significantly from 2017 to 2023, peaking in 2021 and 2022. This indicates a notable rise in vulnerability disclosure activities during these years, likely due to increased emphasis on vulnerability disclosure and management by the security community and enterprises. Although the total number of CVEs with pre-disclosure behaviors (i.e., having issue reports before official disclosure) increased, their proportion relative to the total CVEs decreased, reflecting a growing awareness of the risks associated with premature vulnerability handling. The number of CVEs publicly discussed in issue reports steadily increased throughout the entire period, with a sharp rise after 2020. This trend suggests that an increasing number of CVEs are being thoroughly documented and discussed during the vulnerability management process, reflecting the heightened emphasis on transparency in open-source projects.

Issue Report: Among the 21,501 studied CVEs, 43.47% have related issue reports, indicating public discussions on GitHub. Although CVD recommends private reports before disclosure, many vulnerabilities on GitHub are still undergoing discussions in issue reports before their official disclosure.

41.66% (8,958 out of 21,501) of studied CVEs have related issue reports created before their disclosure (39 days in median). Only 3.91% (840 out of 21,501) of CVEs are fixed before the issue reports and the pre-disclosure discussions expose vulnerabilities at an early stage, making them vulnerable to exploitation by attackers. Significantly, 60.16% (5,389 out of 8,958) of pre-disclosure discussion CVEs lack a security policy in their respective OSS projects. Consequently, the vulnerability discoverers have to decide how to report the vulnerability by themselves, which potentially contributes to the prevalence of pre-disclosure discussions. Additionally, certain projects adopt a full disclosure security policy. For example, the GPAC project encourages people to report vulnerabilities through the public GitHub issue tracker, resulting in 99% of the CVEs in GPAC having corresponding issue reports before disclosure. Unless the reporters consider public disclosure to be inappropriate, they can contact the organization via e-mail to report the vulnerability [23].

16.94% (3,642 out of 21,501) of CVEs have related issue reports but remain unresolved to date. This suggests that some vulnerabilities are poorly managed, highlighting potential challenges in

the current vulnerability management process. Vulnerability discoverers might resort to public discussions in issue reports to inform the maintainers and alert users about the vulnerabilities.

1.50% (322 out of 21,501) of CVEs are publicly discussed after official disclosure, with 1.04% (224 out of 21,501) of CVEs already fixed before the issue report. Issue reports for these CVEs are created after the official disclosure. Among them, 1.04% (224 out of 21,501) of the CVEs have already been fixed before the issue report. These issue reports serve as supplementary means of disclosure. They reiterate the existence of the vulnerabilities in the OSS and remind users of their presence. However, 0.46% (98 out of 21,501) of the CVEs either remain unfixed or are fixed after the issue report. We sampled and examined specific issue reports created after disclosure, finding that some reports serve to remind maintainers of the vulnerability's existence and encourage them to fix vulnerabilities (e.g., CVE-2021-32819[2] and CVE-2018-17317[3]).

Summary. Discussions in issue reports serve various purposes; 41.66% of CVEs are publicly discussed before disclosure which poses a risk of leakage; 1.50% of CVEs have discussions after disclosure to serve issue reports as a disclosure channel or remind maintainers the existence of vulnerabilities.

Patch: Patch is the most prevalent form of disclosed information; 64% of the CVEs with GitHub links in external references directly disclose commit information. Through indirect associations, we supplemented the patch information. Among the 21,501 CVEs related to GitHub OSS, we collected patch information for 17,859 CVEs, accounting for 83%.

The patches of 78% (16,714 out of 21,501) CVEs are publicly available before the official disclosure, with 38 days in the median from fix to disclosure. Our dataset not only includes publicly disclosed commit information but also utilizes pull and issue report data to extract commit details. Consequently, our dataset provides a more comprehensive understanding; 78% (16,714 out of 21,501) of the studied CVEs are fixed before disclosure, 5% (1,076 out of 21,501) are fixed after disclosure with 24 days in median, and the remaining 17% (3,711 out of 21,501) of CVEs are discussed in issue reports but lack any fixes. This is consistent with the observation made by Li and Paxson [36], who found that 21.2% of the vulnerabilities are unpatched at disclosure and 26.4% of them remain so for over 30 days.

34% (5,697 out of 16,714) of the successfully fixed before disclosure CVEs are fixed directly and explicitly, indicating potentially dangerous behavior. A `successfully fixed vulnerability` is defined as one with an associated commit or PR indicating that the vulnerability has been addressed. The absence of issue reports before the fix for these CVEs implies that discoverers might have privately notified the OSS owners, prompting them to initiate the fixes. Among the 16,714 CVEs successfully fixed before disclosure within GitHub projects, 12,987 (78%) do not have an issue report before the fix, which means they may report vulnerability privately. This reporting strategy adheres to the CVD process and helps to preserve confidentiality until vulnerabilities are fixed. However, among the successfully fixed CVEs, 14,724 (88%) are fixed directly. Direct fixes involve committing directly to the main branch or merging fixes through PRs. On the other hand, 1,990 vulnerabilities (12%) are not fixed directly. Among them, 1,697 (10%) originate from forked repositories, while 293 (2%) are fixed by initially addressing them in separate branches and subsequently merging them into the main branch. Direct fixes are easier for attackers to discover, while indirect fixes are more hidden. Furthermore, in terms of whether vulnerability information is explicitly mentioned in the commit messages, 6,351 (38%) of CVEs explicitly indicate the fixes that could potentially expose the vulnerability to attackers in advance. In a word, 34% of the successfully fixed CVEs exhibit the risk behavior of addressing the fixes explicitly and directly, accounting for 26% of all CVEs.

[2]https://github.com/squirrellyjs/squirrelly/issues/238.
[3]https://github.com/xtr4nge/FruityWifi/issues/276.

Summary. 78% of studied CVEs are fixed before disclosure with 38 days in median from fix to disclosure, and 78% of the successfully fixed CVEs are reported privately. However, 34% are fixed directly and explicitly which may make vulnerabilities be discovered and exploited by attackers.

PR: During the process of fixing vulnerabilities, OSS maintainers frequently use PRs to integrate code changes into the primary repository. PRs provide a transparent and manageable way for collaboration and visibility. Among the 21,501 studied CVEs, 7,526 CVEs (35%) are associated with PRs, emphasizing the important role they play in the vulnerability fixing process.

31% (6,637 out of 21,501) of the studied CVEs have their PR initiated before the disclosure, with a median time interval of 39 days; 10% (2,147 out of 21,501) of CVEs have a time interval of 100 days or more between the initial PR submission and the disclosure. PR before disclosure leads to premature disclosure of the vulnerability, as it is shown publicly in GitHub. Analyzing the time interval between the initial PR submission and the disclosure, we observe that 8% (1,761 out of 21,501) of CVEs have a time interval of less than 10 days, and 11% (2,439 out of 21,501) have a time interval of less than 20 days, indicating good practices in terms of prompt PR initiation. However, for 10% (2,147 out of 21,501) of the CVEs, the time interval exceeds 100 days, which is relatively long. This delay could be attributed to various factors such as the complexity of the vulnerability, resource constraints within the team, technical challenges, and more. Nevertheless, such prolonged time intervals increase the exposure of systems to potential risks and may provide attackers with sufficient time to discover and exploit the vulnerabilities.

The decision to use PRs or not to merge fixes is a project-specific convention, and the tendency to use PRs may decrease in cases where the vulnerability is severe. A significant portion of the repositories in our dataset (67%) have only one CVE. The way these isolated CVEs are managed may not provide a reliable representation of the repository's overall approach to fixing vulnerabilities. For instance, a repository with only one or two CVEs might show unusual or inconsistent PR usage that does not reflect its typical practices. Therefore, repositories with fewer than 10 CVEs are excluded. Figure 3 illustrates the distribution of CVE adoption with PRs in different repositories. The graph shows that repositories with either very high or very low PR adoption proportions (i.e., the ratio of CVEs fixed by PRs to the total number of CVEs in the repository) are more frequently represented in the dataset. Specifically, repositories with an adoption ratio of PRs exceeding 80% account for 23.1% of the total repositories, while repositories with an adoption ratio below 20% represent 47.8%. These findings suggest that the adoption of PRs in vulnerability management processes is influenced by repository-specific conventions. Projects that habitually incline toward using PRs rely on them consistently, while those who oppose adopting PR tend to completely avoid them. Furthermore, we focus on projects where PR adoption for CVEs exceeds 90% but does not reach 100%. We speculate that certain CVEs with high severity are not fixed using PRs to prevent possible attacker exploitation. We calculate the median CVSS scores separately for CVEs fixed with and without PR in these projects. Subsequently, using these medians, we compute the overall median CVSS scores for all projects concerning PR-adoption and non-PR-adoption CVEs. The analysis reveals that the median CVSS score for CVEs fixed using PR in the projects is 7.04, while for CVEs not using PR, the median score is 7.48. These results indicate that CVEs without PR adoption tend to have a relatively higher severity level, although the difference is not prominently significant.

High severity issues tend to receive faster fixes through PRs. We filter the CVEs in which the vulnerabilities are publicly reported in issue reports before the PRs, resulting in a total of 2,153 CVEs, representing 10% of all studied CVEs. This subset of CVEs represents vulnerabilities that are initially publicly discussed on GitHub, thereby posing a risk of exploitation by attackers. These CVEs have a median time interval of 23 days from the issue to the PR. Furthermore, we isolated 396 CVEs

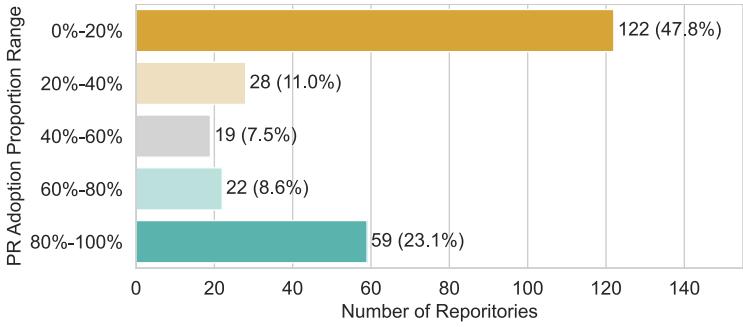

Fig. 3. Repositories' utilization of PRs for CVE.

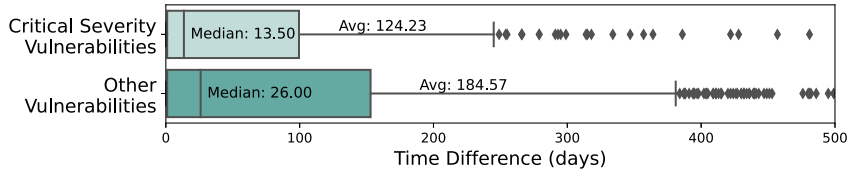

Fig. 4. Time difference from issue report to PR for different CVSS severities.

classified as "CRITICAL" severity, which is 2% of all CVEs. Figure 4 illustrates the time differences between the issue report and PR for critical severity vulnerabilities and other vulnerabilities. Critical vulnerabilities have a median resolution time of 13.5 days and a mean of 124.23 days, while other vulnerabilities have a median of 26.0 days and a mean of 184.57 days. Both the median and mean resolution times for critical vulnerabilities are lower than those for other vulnerabilities. However, there is substantial overlap in the interquartile ranges of the two groups, indicating variability in response times. The presence of many outliers in both groups further highlights this variability. To determine the statistical significance of this finding, we performed a Mann-Whitney U test, which resulted in a p-value of 0.015. This indicates a notable distinction in the median time intervals between critical CVEs and other CVEs. However, the effect size measured by Cliff's delta was −0.077, showing a negligible practical difference. This suggests that, while higher severity issues tend to receive faster PR-based fixes, the size of the overall difference in response times is negligible.

Summary. There is often a substantial time gap between the PRs and disclosure, which can result in premature disclosure. The decision to use PRs to merge fixes depends on the conventions of the repositories and is influenced by the severity of vulnerabilities. In cases where the vulnerability is severe, the tendency to use PRs may diminish, and the vulnerability will receive faster fixes if PR is employed.

Distribution of the Disclosed Responses: From 2017 to 2023, 160,594 CVEs are reported in the NVD and the GHSA. More precisely, the NVD reports 160,317 CVEs, while GHSA contributes 11,952. Among these, 21,501 CVEs include GitHub links in their external links, representing approximately 14% of the total.

On the external GitHub link types in CVE disclosures, the patch is the most commonly provided information. Within this subset of CVEs with GitHub links, 64% of the CVEs contain commit links, 42% contain issue links, and 24% contain PR links. Since both PR and commit links indicate patching,

Table 1. Distribution of CVSS Severity Levels Based on the Presence of Patches When Disclosing CVEs

	Low	Medium	High	Critical	Total
CVEs with Patch	261	6,985	6,163	2,840	16,249
CVEs without Patch	18	2,441	1,703	951	5,331
ALL CVEs	279	9,426	7,866	3,791	21,362

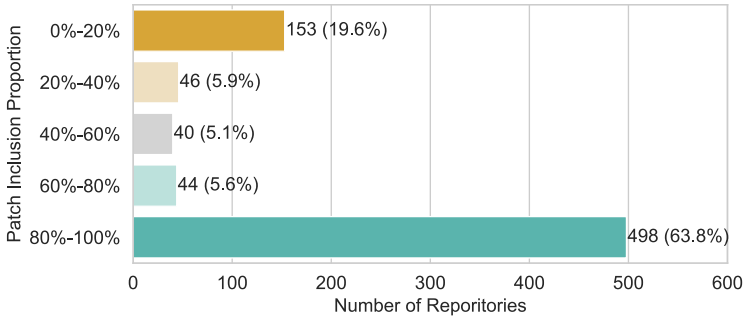

Fig. 5. Distribution of repositories based on CVE patch ratio.

we consider a CVE to provide a patch if it includes either PR or commit links. Consequently, 76% of the CVEs in our dataset provide patches when disclosure.

The presence or absence of patches during the disclosure of CVEs exhibits a weak correlation with CVSS scores. Table 1 shows the CVSS severity distribution of vulnerabilities with or without patches at the time of disclosure. We can observe that almost all (93.5%) CVEs with a CVSS severity level of "LOW" include patches when disclosed. This may be attributed to the fact that low-severity vulnerabilities have a smaller impact when disclosing the details of the fixes. Their influence on overall security posture is relatively minimal. Additionally, we used the point-biserial correlation coefficient [49] to examine the correlation between CVSS scores for CVE and the presence of patches when disclosed. The calculated correlation coefficient is −0.022, with a p-value less than 0.001. The correlation coefficient indicates a weak negative correlation between the CVSS scores and the inclusion of patches when disclosing CVEs. This suggests that while the statistical analysis shows a significant result, the actual impact of CVSS scores on the presence of patches is minimal.

The inclusion of patches when disclosing CVEs is more related to the disclosure conventions of the OSS itself. A significant portion of repositories in our dataset had only one or two CVEs. Including these repositories could have introduced bias, as their practices may not represent typical behavior. Therefore, repositories with fewer than five CVEs are excluded to ensure the reliability of the data. Figure 5 illustrates the distribution of repositories based on the CVE patch ratio; 63.8% of repositories disclose patches for more than 80% of their CVEs, while 19.6% disclose patches for fewer than 20% of CVEs. The graph indicates that the number of repositories decreases as the patch inclusion proportion approaches 50%. This indicates the choice to include patches or not when disclosing a vulnerability may be driven by the preferences and practices of the developers or maintainers. As an example, the Spring framework strictly adheres to the CVD standards [55]. The project encourages people who discover vulnerabilities to report them privately to the organization by e-mail, and the public release of the fix occurs only on the day of disclosure. The official team does not proactively provide fix information. Brian Clozel, one of the core developers of the Spring framework, has expressed reservations about explicitly disclosing patch information, stating that

Table 2. Timeline Table for Response Combination of Vulnerability-Related Behaviors

Type	First	Second	Third	Count	Proportion	Total
Disclosure after Fix	Fix	Disclosure	None	11,893	55.31%	56.49%
Without Public Report	Fix	Disclosure	Report	254	1.18%	
Public Reporting Vulnerability	Report	Fix	Disclosure	3,727	17.33%	37.76%
	Report	Disclosure	None	3,581	16.66%	
	Report	Disclosure	Fix	810	3.77%	
Fix before Public Report	Fix	Report	Disclosure	840	3.91%	3.91%
Disclosure First	Disclosure	Fix	None	196	0.91%	1.52%
	Disclosure	Report	None	63	0.29%	
	Disclosure	Report	Fix	50	0.23%	
	Disclosure	Fix	Report	20	0.09%	

"Disclosing patch information explicitly would also make the exploit ability much simpler [9]." This disclosure approach reduces the likelihood of attackers exploiting the vulnerabilities. However, there are also opposing perspectives suggesting that *"publicly disclosing this information does not make the attack easier because threat actors will be able to exploit it regardless of whether you share the commit or not. On the other hand, not sharing will harm all those who can benefit from knowing the fix and could use it to help protect against this vulnerability."*

Summary. Commit links are the most frequently observed type among the external GitHub link types in CVE disclosure. The decision to include patches in CVE disclosures is driven more by the practices and preferences of the OSS rather than being solely determined by the severity of the vulnerability.

4.3.3 Response Sequence. In this section, we studied the sequence of vulnerability responses in each phase of the lifecycle to summarize different types of vulnerability disclosure in practice. Table 2 displays the sequential order of various responses associated with a CVE and the distribution of specific response combinations. We further discuss the categorized combinations of responses below.

— *Disclosure after Fix without Public Report*: This type emerges as the predominant mode, comprising 56% (12,147 out of 21,501) of all vulnerabilities. Following the fix, disclosure takes place without a reporting phase. It demonstrates a behavior similar to CVD, where vulnerabilities are initially reported confidentially and subsequently publicly disclosed following their fix by the organization.
41% (4,950 out of 12,147) of commit messages contain keywords related to vulnerabilities, indicating the potential for attackers to exploit vulnerabilities before their official disclosure. The median duration from the first patch submission to the formal disclosure is 36 days. This gap offers enough time for attackers to discover the vulnerability. Studies [4, 13] indicate that 23% of exploits are available within the first week after a patch release 50% are published within a month, highlighting the critical window in which attackers can reverse-engineer patches to discover and exploit vulnerabilities before they are formally disclosed. This highlights the importance of careful handling of vulnerability information, as attackers may gain insight from these messages if they contain sensitive keywords.
Repositories employing this combination over an extended period are more likely to have a security policy. Our findings indicate that 62% of the studied projects have used this particular combination of responses before. Among repositories with more than 50% CVEs employing

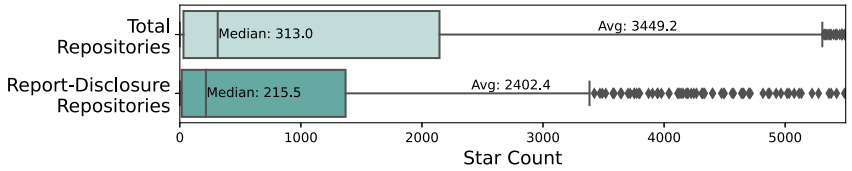

Fig. 6. Comparison of star counts between total repositories and report-disclosure repositories.

this approach (repositories with fewer than 10 CVEs are excluded), 87% (90 out of 104) include a security policy, while only 26% of the repositories we investigate have a security policy. Upon further manual inspection, we found that among the 14 repositories lacking a security policy, six had the "Issues" feature disabled on GitHub. This limitation prevents vulnerability discoverers from publicly discussing vulnerabilities through issue reports, thereby encouraging the use of this specific vulnerability response combination.

— *Public Reporting Vulnerability*: This combination of responses involves the initial public reporting of vulnerabilities on OSS platforms, accounting for 38% (8,118 out of 21,501) of all vulnerabilities. This constitutes full disclosure, where vulnerability information is initially publicly discussed, potentially exposing vulnerabilities to exploitation by attackers.
Repositories that use "report-fix-disclosure" for a long time have a higher probability of having a security policy. In this combination, we discover that 23% of projects have employed this disclosure approach. Among the repositories with 50% or more of their CVEs using this approach (repositories with fewer than 10 CVEs are excluded), 68% (27 out of 40) have implemented a security policy. This is significantly higher than the overall proportion of projects with a security policy (26%), suggesting that repositories with a preference for this disclosure method are more likely to have formal security policies. These policies often encourage vulnerability reporters to initially use the GitHub issue tracking system for reporting, which is consistent with the "report-fix-disclosure" approach.
Vulnerabilities publicly reported but left unfixed are associated with small-scale and low-active projects. We observe the "report-disclose" combination, the vulnerabilities in this combination remain unfixed until now, accounting for 16.7% of the total vulnerabilities; 19% of projects have employed this combination of responses. As Figure 6 shows, the median number of stars for this subset of repositories is 215, while the overall median number of stars for all repositories is 313. Furthermore, the Mann-Whitney U test reveals a difference between the two groups (Mann-Whitney U statistic: 5,043,708, p-value: 8.8e-08). However, the effect size calculated using Cliff's delta was 0.09, indicating a practically insignificant disparity. This indicates that projects associated with unfixed vulnerabilities tend to have a relatively smaller scale, although this difference is not significant in practice. We hypothesize that this trend could be due to the prevalence of small-scale projects across the entirety of GitHub repositories, resulting in an increased occurrence of unfixed vulnerabilities in such projects. Additionally, we manually examine a random sample of 100 unfixed vulnerabilities and discover that most of the projects to which they belong have stopped maintenance before official disclosure of vulnerabilities, without any new commits.

— *Fix before Public Report*: This combination of responses involves fixing vulnerabilities first and subsequently reporting them on public platforms. Issue reports serve as a supplement approach to disclose the vulnerabilities. These combinations account for 4% of all vulnerabilities.

— *Disclosure First*: These responses combination involves the initial official disclosure of vulnerabilities, constituting 1.52% of all vulnerabilities. This approach is used to publicly announce the existence of vulnerabilities.

Summary. 56% of the analyzed vulnerabilities are disclosed in CVD manner, however, 41% of them actually leak the vulnerability information, which may result in premature disclosure; 38% of the analyzed vulnerabilities are disclosed in a full disclosure manner and most of the unfixed vulnerabilities happen in low-active and small-scale projects.

5 RQ2: What Are the Preferences of Practitioners in the OSS Community Regarding Vulnerability Disclosure Management?

5.1 Motivation

Despite CVD being commonly adopted, our findings from RQ1 reveal that numerous disclosures deviate from CVD guidelines in practice. For example, a significant number (42%) of vulnerability discoverers engage in public discussions about vulnerabilities before their official disclosure or the release of a CVE patch. Also, a large number (38%) of vulnerabilities follow a full disclosure approach. The results of RQ1 (Section 4.3) lead us to question whether practitioners' motivations and preferences match the practices we observe in the wild. Therefore, in RQ2, we aim to study practitioners' preferences to better understand the phenomena observed in our empirical study. This study provides us with insights into practitioners' perspectives on open source vulnerability disclosure, aiding in identifying challenges within the current vulnerability disclosure practices compared to the observations from RQ1.

5.2 Approach

To understand practitioners' preferences regarding OSS vulnerability disclosure management, we designed a questionnaire[4] based on a pilot survey. Then, we sampled a set of the collected OSS projects (Section 3) and collected publicly available contact information of all contributors associated with these projects. Finally, we conducted an online survey of 27,279 identified contributors and received 770 valid responses.

5.2.1 Design. We designed our questionnaire based on the four sections described below.

- *Demographics*: The survey first asks for demographic information about the respondents. It contains the primary job role and years of experience.
- *Practitioners' Experiences*: This section inquires respondents about their practical experience in disclosing OSS vulnerabilities. Specifically, it includes experience in handling security-related issue reports and requesting a CVE for a vulnerability.
- *Practitioner's Preferences*: This section aims to investigate practitioners' preferences across different dimensions of vulnerability disclosure consisting of disclosure model, discussion timing, preferred reporting channels, and the roles of individuals who request a CVE. This section also includes open-ended questions to gain insights into the preferred vulnerability disclosure model and the discussion timing.
- *Best Practices*: This section focuses on exploring the best practices from the respondents' perspective. It includes questions related to the content of security issue reports and suggestions to improve the vulnerability management process overall.

We conducted a preliminary survey pilot with a small group of practitioners different from our survey respondents. We obtained feedback on (1) whether the length of the survey was appropriate, and (2) the clarity and understandability of the terms. Based on the feedback received, we made minor modifications to the preliminary survey and finalized the survey version. It is important to

[4]https://forms.gle/jmxmKLkorwjzK5DN9.

Table 3. Respondent Roles and Programming Experience

Role	Population	<1 y	2–3 y	4–5 y	6–9 y	≥10 y
Development	673	11	69	93	147	353
Security Management	27	3	2	4	7	11
Testing	11	0	3	1	4	3
Project Management	28	0	2	6	4	16
Others	31	1	2	4	7	17
Total	770	15	78	108	169	400

note that the responses collected during the pilot survey were not included in the presented results in this article.

5.2.2 Respondent Recruitment. We followed two steps to invite respondents:

— *Contact Information of Contributors*: Based on the 8,073 GitHub projects related to vulnerabilities collected in Section 3.3, we randomly sampled 2,000 GitHub projects. This approach aimed to ensure a diverse sample, enabling a comprehensive understanding of contributors' perspectives on vulnerability disclosure management across different kinds of OSS, while also simplifying the survey process. In total, we obtained 2,000 repositories owned by 1,889 different organizations or individuals. To locate survey candidates, we found developers who were contributors to these repositories. We retrieved their e-mail address from git commits and excluded invalid e-mails. After manually merging their identities, we got 27,279 contributors.
— *Questionnaire Distribution*: We sent out questionnaire invitation e-mails to the 27,279 GitHub contributors we collected in the previous step. In the e-mails, we provided a detailed introduction to the purpose and significance of our research, explained the anonymity of the survey and invited them to participate by completing the questionnaire. Additionally, we included a questionnaire completion guide to ensure the quality of the responses.

In total, we received 793 responses, resulting in a response rate of 2.91%. This response rate was comparable to previous SE surveys [31]. Out of these, 23 respondents declined to participate, leaving us with 770 valid responses. An overview of the surveyed respondents and their experience is shown in Table 3; 87% of respondents were involved in development roles within OSS and 52% had more than 10 years of professional experience. Additionally, further statistics indicate that 61% of the respondents have contributed to fewer than OSS repositories, while 12% have contributed to more than 50 repositories.

5.2.3 Analysis. We started our analysis by excluding all the answers classified as "Not sure" to ensure the reliability of the data. We then calculated the percentage of responses for each option in the analysis of the second and third sections of the questionnaire (practitioners' experiences and preferences). We manually examined and extracted information from respondents who selected the option "Other" and provided their perspectives. We utilized an open-coding approach [56] for an in-depth analysis of open-ended questions about respondents' rationale for their choices. Three authors of this article conducted a comprehensive analysis of the response to the question regarding selecting a specific vulnerability disclosure model, examining it at the level of individual lines and paragraphs to extract key concepts, themes, and perspectives. Initially, we collaboratively created a set of codes to describe the reasons underlying respondents' choices of a specific vulnerability disclosure model. Following this, the three authors independently coded the remaining responses. When an author identified new key points or themes not covered by the initial codes, these findings

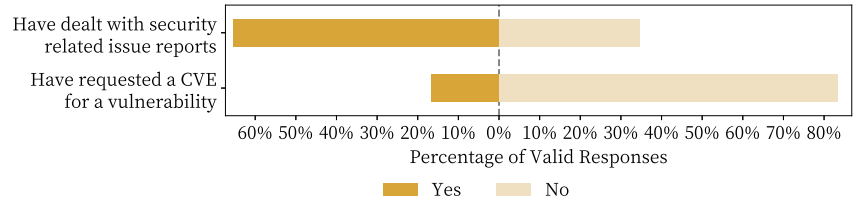

Fig. 7. Experience in vulnerability disclosure experiences.

were discussed with the other authors. We re-examined previous responses each time new codes were added to ensure consistent application of the updated codebook. By continuously comparing and revising, which involved merging similar codes, redefining code meanings, and creating new codes to encompass emerging themes, we progressively developed a codebook that served as a reliable reference for the analysis, ensuring both consistency and accuracy. Finally, we obtained 21 unique codes that encompass the reasons behind the choices of a vulnerability disclosure model, considering perspectives from the respondents and their respective open source communities. We discussed each aspect of our open-coding in the results of RQ2 (Section 5.3). Please refer to the replication package for detailed information regarding the open-coding results.

To analyze the answers for the fourth section of our survey (i.e., best practices), we calculate the percentage of responses for each option. For respondents who selected the "Others" option and responded to the open-ended question, we manually reviewed all provided opinions and included those we found valuable in the article. These findings will be discussed in Section 6.2.

Due to space constraints, the results and analysis of the sections on the disclosure role and enhancing vulnerability management processes are included in the replication package.

5.3 Result

5.3.1 Experiences of Practitioner. Figure 7 illustrates the experience of the respondents in vulnerability disclosure practice. Specifically, we investigate whether the respondents have dealt with security-related issue reports and whether they have requested a CVE for a vulnerability.

65% of the respondents have experience in handling an issue report, but only 16.6% have experience in requesting a CVE. This indicates the significance of effectively managing and addressing reported issues, as a majority of the respondents have encountered this aspect of software development. However, there may be a lack of awareness or understanding of the CVE request process among the respondents. This could be attributed to various factors, such as a lack of knowledge about the CVE process, limited exposure to vulnerability management practices, or a perception that requesting a CVE is less relevant or necessary in the vulnerability process.

Summary. While many practitioners have experience in dealing with vulnerabilities, there is a lack of experience in standardizing the vulnerability handling process.

5.3.2 Practitioner Preference. In this section, we present the preferences of the respondents surveyed regarding the disclosure models, discussion timing, disclosure roles, and vulnerability report channels.

Disclosure Model: As Figure 8 shows, more than 81% of the respondents show a preference for CVD, while 10% of the respondents prefer private disclosure and 7% of the respondents prefer full disclosure. Furthermore, some respondents provide alternative perspectives beyond the three disclosure models. Five respondents suggest that the choice of disclosure method should be determined on a *case-by-case* basis, considering the nature and resources of the project itself, as well as the

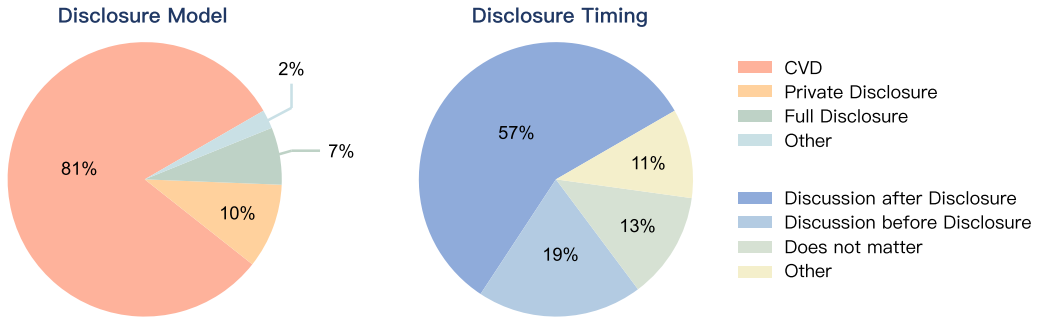

Fig. 8. Respondent preferences for disclosure model and disclosure timing.

severity of the vulnerability. For widely used OSS, they propose giving users a brief notification before disclosure. In the case of highly critical vulnerabilities, they emphasize the importance of confidentiality through private disclosure to prevent exploitation by others. Five respondents suggest that CVD should be prioritized but with a *time limit*. If developers are unable to provide a viable patch within the specified time limit, full disclosure of the vulnerability should be initiated.

Table 4 shows the reasons for respondents' choice of vulnerability disclosure models and the corresponding number of individuals for each reason. Supporters of full disclosure believe that it aligns with the open principles of OSS and enables faster vulnerability fixes while making OSS users aware of the risks they face. Advocates of private disclosure argue that it mitigates the impact of vulnerabilities and provides developers with sufficient time to develop patches. However, it is important to note that some respondents express concerns regarding the possibility of OSS projects being abandoned. Consequently, they deem private disclosure to be ineffective in fixing vulnerabilities in such cases.

Respondents who favor CVD perceive it as a relatively balanced model. This model considers the perspectives of different stakeholders involved in the disclosure of vulnerability. From the developer's perspective, CVD allows them adequate time to develop patches while also imposing a sense of urgency to fix vulnerabilities promptly. It reinforces the developers' responsibility to fix vulnerabilities in their repositories and provides recognition for their efforts. For users, this disclosure model ensures they have sufficient time to apply patches and take the necessary actions to mitigate risks. It promotes user awareness and encourages timely software upgrades. For reporters, it recognizes and respects their contributions. CVD also upholds the principles of OSS by promoting openness and transparency. It facilitates better practices within the open source community and allows for knowledge sharing about disclosed vulnerabilities. It also facilitates monitoring of patch quality and encourages prompt remediation efforts.

Summary. Most of the respondents (81%) prefer CVD due to its balance between human, open source nature, public factors, and vulnerability. Certain respondents recommend implementing a time limit for CVD.

Discussion Timing: As Figure 8 shows, only 57% of the respondents believe that public discussion of vulnerabilities should take place after the CVE publication. Most (81%) of the respondents prefer CVD. However, this preference remains an ideal scenario, since only 57% of the respondents indicate a tendency to engage in public discussions after the disclosure of CVE. Furthermore, 11% of the respondents believe that deciding whether or not to discuss vulnerabilities in advance should be a multifaceted process, taking into account various factors. These factors include the severity of the

Table 4. Reasons for the Choice of Respondents of Vulnerability Disclosure Models

Type	Reason	F[a]	P[a]	C[a]
Human Perspective	Developer:			
	Leave time for developing patches	-	9	131
	Push developers to fix the vuln.	4	-	50
	Lack of human resources	1	1	-
	Responsible	-	3	19
	Provides recognition for researchers	-	-	6
	Related to the interests of the company	-	-	1
	User:			
	Leave users enough time to apply the patch	-	1	15
	Users of vulnerable versions can act accordingly to either upgrade or mitigate	10	-	-
	Making bugs public encourages people to upgrade their software	-	-	14
	Reporter:			
	Better for reporters (protection, give credit to them, respect their effort)	-	-	9
Open Source	Openness	13	-	87
	For better security OSS practice, help improve other projects	1	-	71
	The OSS may be abandoned	1	-	2
Public	Avoid panic	-	2	-
	Improvements over the chain of trust	-	-	1
Vulnerability	Reduce the impact of expoitation	9	23	171
	Monitor patch quality	-	-	4
	Quicker	4	-	2
	Less work/easier	1	3	-
	Easier to discuss and test	-	2	-
Total		52	77	624

F, Full Disclosure; P, Private Disclosure; C, Coordinated Vulnerability Disclosure.
[a]F, P, and C represent the number of respondents with the reason why this disclosure model is chosen.

vulnerability, the availability of published patches, the use of the CVE system, and the potential risks and impacts on affected individuals, as well as the response of organizations or vendors involved.

Figure 9 displays the perspective of the respondents who choose to discuss publicly before the CVE publication which contradicts the principles of CVD. The figure reveals that 75% of the respondents consider that such disclosure accelerates the process of fixing vulnerabilities, as respondents stated: "*After immediate public disclosure, other users can suggest a better fix or find similar vulnerabilities in other projects*"; 61% of respondents believe it enables timely notification to users about the existence of vulnerability, and 32% find the release of CVEs time-consuming, leading them to prefer not to disclose vulnerabilities after CVE publication. In addition, some respondents expressed: "*Vulnerabilities can be disclosed without stating the full effect and security implications.*" This approach can inform users about the existence of vulnerabilities and give developers time to develop patches.

Summary. Only 57% of the respondents believe that public discussions should occur after the publication of the CVE, which contradicts the proportion of supporters of CVD.

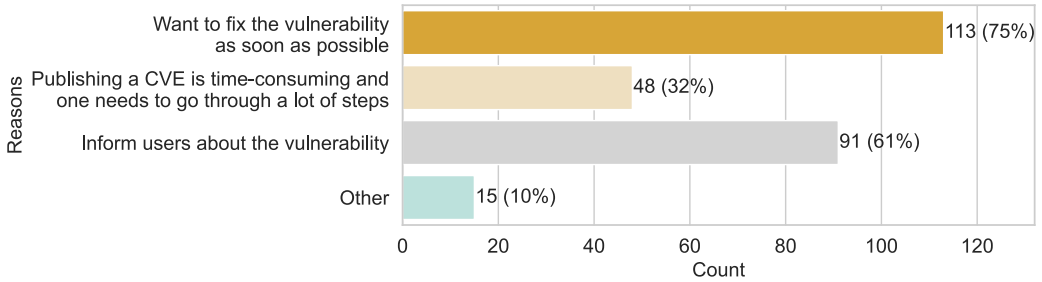

Fig. 9. Reasons for pre-CVE discussion.

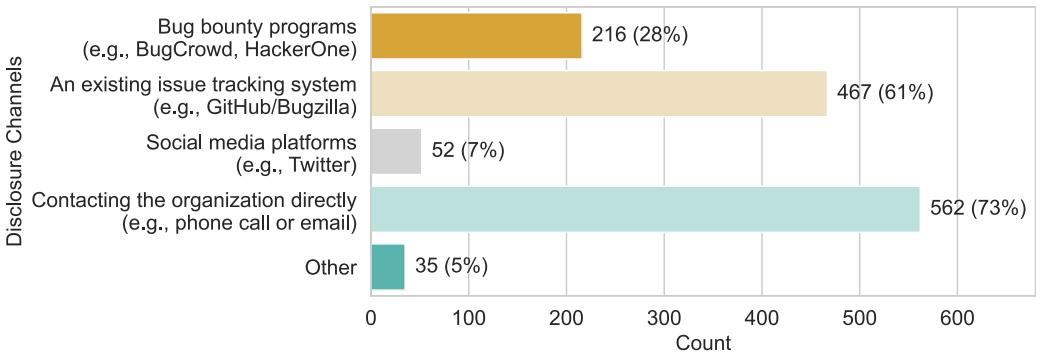

Fig. 10. Vulnerability disclosure channel tendency.

Report Channel: When OSS does not explicitly recommend a vulnerability report channel, vulnerability discoverers have the flexibility to choose their preferred channels (e.g., bug bounty programs and an existing issue tracking system). Figure 10 illustrates the vulnerability report channel preferred by the respondents. Note that the percentages do not add up to 100% since a respondent could indicate more than one preferred vulnerability report channel. Most of the respondents (73%) indicate a preference to report vulnerabilities by contacting organizations directly. The second largest group (61%) prefers to use an existing issue tracking system. Furthermore, 28% of the respondents adopt bug bounty programs as their preferred vulnerability report method, while only 7% select social media platforms. Some individuals express reservations about reporting vulnerabilities due to potential legal implications. On the other hand, certain respondents recommend a proactive approach by advocating the use of e-mail communication with developers or maintainers, emphasizing the importance of employing PGP encryption [65] whenever possible to ensure CVD. Another group of respondents propose an alternative strategy by suggesting the initiation of private messaging to engage with the most active contributors to the project. Additionally, several respondents highlight the usefulness of vulnerability databases, such as CNVD and CNNVD.

Summary. The majority of respondents prefer to report vulnerabilities by directly contacting the organization (73%) or using an existing issue tracking system (61%).

6 Discussion

This section of the article examines the misalignment between practitioners' expectations and the actual practices in vulnerability disclosure management. Subsequently, we present several implications for practitioners in the OSS and research community and explore best practices in vulnerability management.

6.1 Misalignment between Disclosure Practice and Practitioner Preference

We compare the vulnerability disclosure practice (Section 4.3) with the preference of practitioners (Section 5.3) and identify misalignment between them.

Comparing with the preference of the practitioners, there is a higher prevalence of full disclosure and a lower adoption of CVD in practice. According to the reply to our survey, over 80% of the respondents support CVD and only 7% tend to full disclosure. However, 56% of the studied vulnerabilities conform to CVD in practice and 38% of the vulnerabilities employed a disclosure approach similar to full disclosure.

Only 20% of the surveyed respondents express support for discussion before disclosure, but 42% of the CVEs are publicly discussed in practice, potentially due to the absence of security policies and direct communication channels. Only 20% of the surveyed practitioners support discussion before disclosure, while 58% explicitly state that vulnerabilities should not be discussed prematurely. Respondents who favor pre-disclosure emphasize its benefits in expediting vulnerability fixes and promptly notifying users of potential risks. However, 42% of the vulnerabilities are discussed in the issue report on GitHub before official disclosure in practice, deviating from the expectations of the respondents. The prevalence of public discussion also leads to the prevalence of full disclosure. In the analysis of the practitioners' preferences regarding the channels of reporting vulnerabilities in Section 5.3.2, it is found that practitioners are more inclined to directly contact organizations or report vulnerabilities through an issue tracking system. Direct contact with organizations involves private reporting vulnerabilities without public discussion. However, as highlighted in Section 4.3.1, only 26% of repositories have explicit security policies. In the absence of explicit security policies, vulnerability discoverers might prefer to directly contact organizations. However, only 32% of the repositories offer private contact information. As a result, the discover tend to report vulnerabilities through an issue tracking system, leading to premature disclosure of vulnerabilities.

Some OSS projects may be abandoned or lack the necessary maintenance, leading to unresolved vulnerabilities. In Section 4.3.3, it can be observed that 17% of CVEs are discussed in the issue report and disclosed but remain unfixed. For example, CVE-2021-39503 [45] has an external link to an issue report [33] at the time of CVE disclosure. However, the issue report remains open, indicating that it has not been fixed. On August 22, 2021, KietNA opened the issue and urged repository owners to address the vulnerability in a comment on August 27, 2021. As of today (September 28, 2023), the vulnerability remains unresolved, leaving the repository susceptible to potential attacks. Combining the findings of Section 5.3.2, some survey respondents who support CVD propose the inclusion of a time limit. If developers do not provide a viable patch within the specified timeframe, full disclosure of the vulnerability should be enforced to apply pressure and supervision on the developers. Additionally, some respondents who advocate full disclosure believe that certain OSS projects may be abandoned and that developers may not prioritize fixing reported vulnerabilities. Therefore, full disclosure is necessary to inform software users about the risks they face. These practical findings corroborate the perspectives of the aforementioned respondents, as some vulnerabilities are left unfixed after being reported.

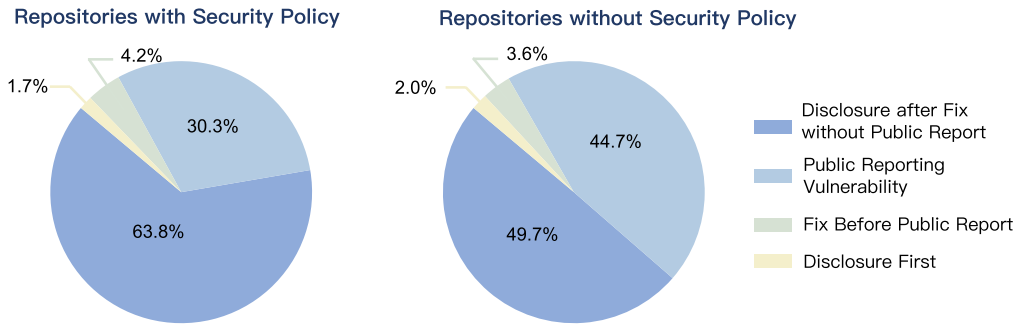

Fig. 11. Comparison of CVE action classes in repositories with and without security policies.

6.2 Implications

Our results offer several suggestions for vulnerability management and implications for the research community.

Enhance Vulnerability Handling Guidance: We divided all CVEs into two groups: those associated with projects having a security policy and those with projects lacking a security policy. Following the classification outlined in Section 4.3.3, we analyzed the responses for these two groups into four types. As shown in Figure 11, the proportion of CVEs publicly discussed before disclosure and fix is higher in projects without a security policy (44.7%) compared to those with a security policy (30.3%). This indicates that security policies help reduce the risk of vulnerabilities being publicly discussed before official disclosure. Additionally, 63.8% of CVEs in projects with security policies employed a method similar to CVD, compared to 49.7% in projects without security policies. This suggests that security policies may encourage and facilitate more coordinated disclosure and fix processes. Therefore, we recommend OSS projects establish explicit security policies to enhance the effectiveness of vulnerability management. However, our study find that 60% of the examined OSS projects lack information on security policies or private contact channels. This highlights the crucial need to improve the availability and clarity of vulnerability handling guidance. OSS projects should establish explicit security policies to ensure that discoverers of vulnerabilities can follow a recommended reporting procedure. To facilitate private reporting of vulnerabilities, OSS projects can consider implementing a private report button [24] or providing contact e-mails in their documentation.

Preventing Premature Disclosure: Preventing premature disclosure of vulnerabilities is crucial to mitigate potential attacks. Several identified dangerous practices can lead to premature disclosure of vulnerabilities. During the fix process, the OSS should pay attention to avoiding vulnerability-related keywords in the commit message and be careful to choose a public fix method such as PR. Furthermore, OSS projects should be careful of public discussions about vulnerabilities before disclosure, as premature disclosure of sensitive details or exploit techniques can provide adversaries with valuable information.

Implement Time Limits for CVD: To address the issue of persistent unpatched vulnerabilities in repositories with inadequate maintenance, the implementation of time limits for CVD can be beneficial. If a vulnerability remains unpatched beyond the specified time limit, a process for initiating full disclosure of the vulnerability should be established. This can incentivize timely vulnerability remediation and inform the users of the OSS about the existence of the vulnerability. Mend.io[5] is the only application security platform in the world that provides automated remediation

[5]https://www.mend.io/.

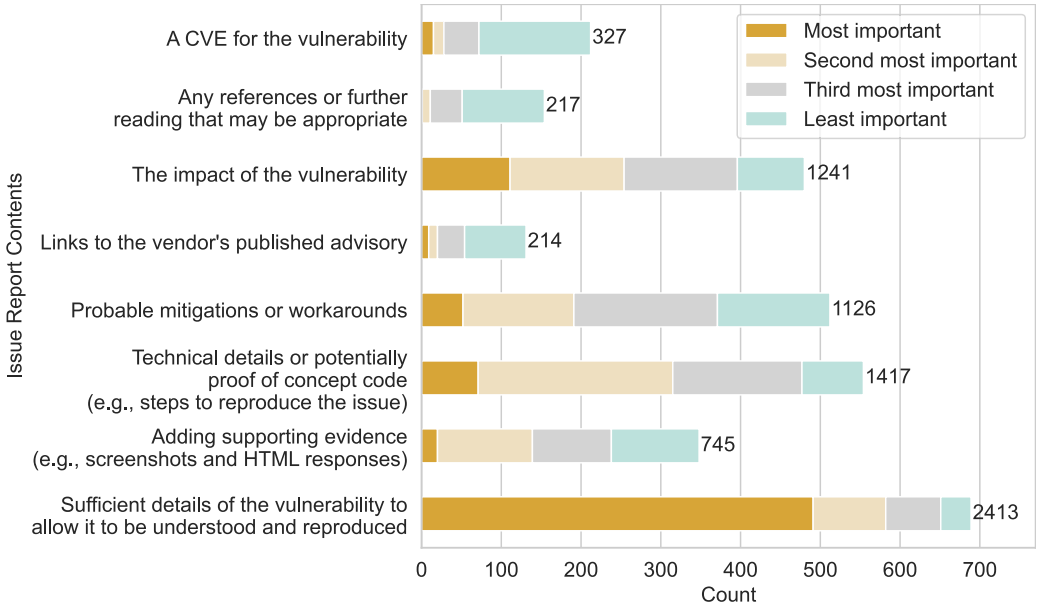

Fig. 12. Importance of security issue report content among surveyed respondents.

workflows for both open source vulnerabilities and custom code. Mend.io adopts a vulnerability disclosure policy with time limits for CVD [38]. If the maintainer does not reply to the initial disclosure e-mail within 15 days, Mend.io will send a second notification. Mend.io provides the maintainer with an additional 15 days to respond once the second notification is sent, providing a total of 30 days to respond from the initial disclosure. If the maintainer does not respond to the two notifications, Mend.io will issue a public advisory with no further collaboration.

Issue Report Content: An issue report includes various aspects of information about a vulnerability. It is crucial to identify the importance of different parts in issue report so that vulnerability discoverers can submit a clarified and appropriate issue report. Figure 12 presents the perspective of the respondents on the importance levels assigned to different content in the issue report. To quantitatively measure the importance attributed by respondents to each content, we devise a formula to calculate an importance value[6] and present it on the right side of each entry. The findings indicate that OSS practitioners place greater emphasis on the intrinsic aspects of vulnerabilities and their associated impact, such as sufficient details of the vulnerability, the technical details or potential proof of concept code, and the impact of the vulnerability. They demonstrate relatively lower attention to some information outside the details of the vulnerability, such as assigning a CVE for the vulnerability, providing references or further reading materials, and including links to the vendor's published advisory. Therefore, vulnerability discoverers can optimize their reporting practices by prioritizing the essential aspects of vulnerabilities and streamlining the inclusion of non-essential information. Building upon this discovery, the development of automated tools for generating comprehensive vulnerability reports can alleviate the workload of report authors while enhancing the quality of the reports. Additionally, the creation of assessment tools specifically designed to evaluate the quality of security issue reports can assist OSS owners in effectively managing and reviewing incoming issue reports.

[6]Importance Value = (Most important) * 4 + (Second most important) * 3 + (Third most important) * 2 + (Least important).

Table 5. Impact of Vulnerability Management Experience on Survey Responses

Question	Response Option	Non-Experienced[a]	Experienced[b]	CVE[c]
Preferred Vulnerability Disclosure Model	CVD	214 (83.3%)	408 (79.5%)	101 (78.9%)
	Private disclosure	21 (8.2%)	56 (10.9%)	16 (12.5%)
	Full disclosure	17 (6.6%)	35 (6.8%)	8 (6.3%)
Public Disclosure before CVE Publication	Yes	50 (19.5%)	100 (19.5%)	19 (14.8%)
	No	156 (60.7%)	286 (55.8%)	75 (58.6%)
	Does not matter	30 (11.7%)	67 (13.1%)	18 (14.1%)
Preferred Role to Open the CVE	Security expert	124 (48.2%)	249 (48.5%)	56 (43.8%)
	Developer	46 (17.9%)	80 (15.6%)	20 (15.6%)
	Project manager	38 (14.8%)	75 (14.6%)	24 (18.8%)
	Issue reporter	29 (11.3%)	61 (11.9%)	14 (10.9%)
Total		257	513	128

[a]"Non-Experienced" refers to respondents without experience managing security issues or CVE.
[b]"Experienced" refers to respondents who have managed security issues or requested a CVE.
[c]"CVE" refers to respondents who have requested a CVE for a vulnerability.

6.3 Differences between Respondents

Our questionnaire includes several questions related to vulnerability management, such as preferred vulnerability disclosure models, attitudes toward premature disclosure, and roles involved in the disclosure process. However, not all respondents have experience in vulnerability management. In this section, we discuss in detail whether the experience of the respondents influences their views on certain aspects of the vulnerability management process.

As shown in Table 5, we examined the choices of respondents with different levels of experience regarding three questions related to vulnerability management. The people who have requested a CVE are more professional in vulnerability management. We observed that the majority of respondents, regardless of their level of experience, preferred CVD as their vulnerability disclosure model. This consistency suggested a broad consensus on the importance of coordinated disclosure practices within the OSS community. Additionally, it is noteworthy that the preference for private disclosure increases with respondents' experience in vulnerability management. To further analyze the relationship between experience levels and disclosure model preferences, we conducted a Chi-square test [50]. The test yielded a Chi-square statistic of 1.582 with a p-value of 0.453, indicating no statistically significant association between experience and the choice of disclosure model. These results further supported the finding that, across both experienced and non-experienced respondents, there was a general agreement on the preferred disclosure model.

When asked whether it was a good practice to publicly discuss the vulnerability before the CVE publication, experienced respondents in requesting CVEs were less likely to answer "Yes." This may be due to their experience with the consequences of premature disclosure. To assess whether experience significantly affects responses, we also conducted a Chi-square test [50]. The test returned a Chi-square statistic of 0.745 with a p-value of 0.689, indicating no statistically significant relationship between experience levels and opinions on premature disclosure. This result suggests that, while experienced respondents may have slightly different preferences, the overall distribution of responses does not vary significantly between experienced and non-experienced participants.

Regarding the preferred role in requesting the CVE, project managers are more favored among those with experience in requesting CVEs (18.8%) compared to other groups, which may reflect their experience in coordinating the overall security response.

Overall, these findings highlight that while there are minor differences in responses based on experience levels, the overall trends and preferences remain consistent across all groups.

7 Threats to Validity

In this section, we discuss the threats to the validity of our findings.

To conduct the vulnerability practices survey, GitHub was selected as the representative open source platform for data collection due to its prominence in the open source community. However, it is imperative to investigate whether the insights gained from this platform can be extrapolated to other open-source platforms. Future research endeavors should encompass a broader range of platforms to enhance the external validity of our findings. While our analysis focused exclusively on issue reports on GitHub, it is worth considering that vulnerability discoverers may also report vulnerabilities through alternative public channels, such as Twitter or bug bounty platforms. Exploring the landscape of vulnerability reports across various platforms would contribute to a more comprehensive dataset and improve our understanding of the disclosure process.

Additionally, using the earliest commit and PR timestamp as the fix time may introduce certain risks. While this method marks the initiation of the remediation process, it may overlook subsequent important commits that finalize the fix. Consequently, this approach can lead to an overall early assessment of vulnerability fix times and an incomplete representation of the entire remediation process. Although this method provides a consistent and straightforward measure for the start of the fix, future research should consider more nuanced methods to capture the full scope of the fix process.

Regarding the analysis of security policies, we employed an iterative keyword approach to determine the presence of security policies within `Readme` of a GitHub project. Nevertheless, it is crucial to acknowledge that this method is not infallible, and there remains a possibility of inadvertently overlooking certain security policies. To improve the accuracy of policy identification, future research could explore alternative approaches, such as training a large language model specifically designed to detect the presence of security policies in project documentation.

Furthermore, in the practitioner preferences survey, it is essential to recognize that some respondents may possess limited understanding of the vulnerability disclosure process or the intricacies of our survey questions. This potential knowledge gap introduces the risk of bias into the collected data. Although we have taken measures to mitigate this issue, it is impossible to ascertain with certainty whether respondents' responses accurately reflect their true beliefs and preferences. It is worth noting that this concern is not unique to our study, but is a prevalent and acknowledged threat to validity in numerous studies exploring practitioners' perspectives and expectations. For example, Kim et al. [34] also encountered this challenge and made the assumption that most responses faithfully represent the respondents' genuine beliefs.

8 Related Work

In this section, we present a comprehensive review of prior research on vulnerability practices, practitioners' preferences, and vulnerability management in OSS systems.

8.1 Vulnerability Practices

The first subsection examines the practices surrounding vulnerability handling and disclosure.

Li and Paxson [36] conducted a large-scale empirical study of security patches, discovering that more than 20% of CVEs were unpatched when they were first announced. This gap provided attackers with the knowledge and time to strike. In our research, we discovered the commit link was the most commonly observed link when disclosure and patches of 78% CVEs were publicly available before the official disclosure which was consistent with the prior study. Regarding the

vulnerabilities that had been patched, the median time window from fix to disclosure was 32 days. This duration was longer than the study conducted by Li and Paxson, who reported a median time of 7 days [36]. The disparity arose for Li and Paxson considered public discussion as the disclosure time, but we considered the CVE publication date as the disclosure time.

Bilge and Dumitras [8] conducted a study where they monitored the downloads of benign and malicious binaries on 11 million real hosts globally and analyzed them to identify files that exploited known vulnerabilities. Through this analysis, they discovered 18 vulnerabilities that were exploited before their disclosure and revealed that the average duration of a typical zero-day attack is 312 days. These findings provide empirical evidence supporting our earlier discussion on the importance of reducing premature disclosure, which can potentially contribute to the occurrence of dangerous zero-day attacks. In our work, we have identified several behaviors that may lead to premature disclosure, such as including vulnerability keywords in commit messages.

8.2 Practitioners' Preferences

In this subsection, we delve into the preferences and knowledge needs of practitioners regarding vulnerability discovery and disclosure. Many previous research have been based on surveys of practitioners. Rahman and Williams [52] studied the knowledge needs of inexperienced security testers and observed a limited availability of resources to discover software vulnerabilities. Bhuiyan et al. [7] surveyed 51 practitioners to assess four vulnerability discovery strategies and they observed 88% of the surveyed practitioners to agree that diagnostics could be used to discover vulnerabilities. Votipka et al. [61] reported on a semi-structured interview study (n = 25) with both testers and hackers, focusing on how each group finds vulnerabilities, how they develop their skills, and the challenges they face. The results suggested that hackers and testers follow similar processes, but get different results due largely to differing experiences and therefore different underlying knowledge of security concepts.

The work by Fang and Hafiz [17, 28] is highly relevant to our research. They conducted a survey by distributing questionnaires and obtained 58 responses from reporters. Their findings indicated that the majority of reporters, particularly experienced ones, prefer full disclosure and do not cooperate with vulnerable software vendors. This contrasts with our research findings. Our questionnaire survey revealed that more than 80% of the respondents preferred CVD, while only 7% showed a tendency for full disclosure. It may be due to the prevalence of CVD and the improvement of regulations in recent years.

8.3 Vulnerability Management

This subsection explores the field of vulnerability management, including the vulnerability lifecycles and suggestions for better vulnerability management.

Frei et al. [20] examined vulnerability management in large-scale networks and systems, proposed a definition for the date of vulnerability disclosure, and determined the dates of discovery, disclosure, exploit, and patching of these vulnerabilities using over 80,000 security advisories. In our investigation, we focused on CVE data, collecting 21,501 OSS-related CVEs over the past 5 years. We determined the report, patching, and disclosure dates of these vulnerabilities to investigate recent trends in vulnerability disclosure practices specific to OSS.

Moura and Heidemann [42] shared their personal experience of accidentally discovering a DNS vulnerability and navigating the vulnerability disclosure process for the first time. They suggested vendors' roles and timeframes should be clarified for vendors may refuse to fix the issue or indefinitely delay its resolution. We also highlighted this aspect in our work, we suggested implementing a time limit for CVD. If the vulnerability remains unpatched beyond the specified time limit, it should be fully disclosed.

Arora et al. [6] examined the influence of vulnerability disclosure and patch availability on attackers' inclination to exploit vulnerabilities and vendors' likelihood of releasing patches. Their findings indicate that vendors promptly respond to immediate disclosure. However, such disclosure also leads to a higher frequency of attacks, and the frequency of attacks diminishes over time. These research findings align with certain perspectives expressed by respondents in our survey, suggesting that full disclosure can push vendors to patch vulnerabilities but may also facilitate the exploitation of vulnerabilities by attackers.

Carlson et al. [10] conducted an empirical study on open source vulnerability notification, focusing on the presence of vulnerable dependencies and the existence of security notification policies in open source projects. They found that 96.6% of the projects with vulnerabilities did not have a security notification process for private disclosure, and 19.8% had no publicly available contact information. We also investigated the presence of security policies and private contact channels in OSS. Our findings reveal that 60% of the repositories lack both security policies and private contact information.

Furthermore, researchers have explored the ideal timing for patch releases. Okamura et al. [47], building on the research of Cavusoglu et al. [11, 12], introduced a patch management model incorporating non-homogeneous vulnerability discovery processes to identify the optimal security patch release time. They employed cost analysis to establish the most suitable schedule for releasing security patches during the entire software lifecycle.

9 Conclusion

Vulnerability disclosure management is a crucial aspect of OSS. In this study, we examined 21,501 CVEs associated with GitHub and studied vulnerability handling guidance of 8,073 Github projects to investigate the practices of vulnerability disclosure; 60% of the studied OSS lack security policies or private report channels for vulnerabilities. Several dangerous practices are observed in OSS, including discussions before disclosure and explicit mention of vulnerability keywords in commit messages. Furthermore, we conducted a survey involving 770 practitioners to explore their preferences regarding vulnerability disclosure. The majority of respondents expressed a preference for CVD as the preferred model for vulnerability disclosure. Additionally, they tended to opt for direct contact with the organization (73%) or the use of an existing issue tracking system (61%) when reporting vulnerabilities. We then compared the observed practices with the preferences of the practitioners to identify discrepancies and proposed additional implications. We suggest completing vulnerability handling guidance in OSS, preventing premature disclosure, and implementing time limits for CVD. In conclusion, this study highlights the importance of vulnerability disclosure management in OSS. Based on this study, the OSS community can strengthen its vulnerability management processes and ultimately enhance the overall security of OSS.

References

[1] Biton A. 2020. *Responsible Disclosure: The Impact of Vulnerability Disclosure on Open Source Security*. Retrieved from https://snyk.io/blog/responsible-disclosure/

[2] Rasheed Ahmad, Izzat Alsmadi, Wasim Alhamdani, and Lo'ai Tawalbeh. 2023. Zero-day attack detection: A systematic literature review. *Artificial Intelligence Review* 56, 10 (2023), 10733–10811. DOI: https://doi.org/10.1007/s10462-023-10437-z

[3] Nikolaos Alexopoulos, Sheikh Mahbub Habib, Steffen Schulz, and Max Mühlhäuser. 2020. The tip of the iceberg: On the merits of finding security bugs. *ACM Transactions on Privacy and Security* 24, 1 (2020), 1–33.

[4] Mohammed Almukaynizi, Eric Nunes, Krishna Dharaiya, Manoj Senguttuvan, Jana Shakarian, and Paulo Shakarian. 2019. Patch before exploited: An approach to identify targeted software vulnerabilities. *AI in Cybersecurity* 151 (2019), 81–113.

[5] Apache. 2023. Apache HTTP Server Security Report. Retrieved from https://httpd.apache.org/security_report.html

[6] Ashish Arora, Ramayya Krishnan, Anand Nandkumar, Rahul Telang, and Yubao Yang. 2004. Impact of vulnerability disclosure and patch availability-an empirical analysis. In *Proceedings of the 3rd Workshop on the Economics of Information Security*, Vol. 24, Citeseer, 1268–1287.

[7] Farzana Ahamed Bhuiyan, Justin Murphy, Patrick Morrison, and Akond Rahman. 2021. Practitioner perception of vulnerability discovery strategies. In *Proceedings of the 2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*. IEEE, 41–44.

[8] Leyla Bilge and Tudor Dumitraş. 2012. Before we knew it: An empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 833–844.

[9] Mário Leitão-Teixeira and Brian Clozel. 2023. *Comment on Spring Framework Security Disclosure*. Retrieved from http://disq.us/p/2tlnizx

[10] Brandon Carlson, Kevin Leach, Darko Marinov, Meiyappan Nagappan, and Atul Prakash. 2019. Open source vulnerability notification. In *Open Source Systems*. Francis Bordeleau, Alberto Sillitti, Paulo Meirelles, and Valentina Lenarduzzi (Eds.), Springer International Publishing, Cham, 12–23.

[11] Huseyin Cavusoglu, Hasan Cavusoglu, and Jun Zhang. 2006. Economics of security patch management. In *Workshop on the Economics of Information Security*.

[12] Hasan Cavusoglu, Huseyin Cavusoglu, and Jun Zhang. 2008. Security patch management: Share the burden or share the damage? *Management Science* 54, 4 (2008), 657–670.

[13] Jay Chen. 2020. *The State of Exploit Development: 80% of Exploits Publish Faster than CVEs. Paloalto Networks's Web Page*.

[14] Steve Christey and Chris Wysopal. 2002. *Responsible Vulnerability Disclosure Process*. Technical Report. Retrieved from https://tools.ietf.org/html/draft-christey-wysopal-vuln-disclosure-00

[15] Alexandre Decan, Tom Mens, and Eleni Constantinou. 2018. On the impact of security vulnerabilities in the NPM package dependency network. In *Proceedings of the 15th International Conference on Mining Software Repositories*, 181–191.

[16] Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, and Wenke Lee. 2017. Identifying open-source license violation and 1-day security risk at large scale. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2169–2185.

[17] Ming Fang and Munawar Hafiz. 2014. Discovering buffer overflow vulnerabilities in the wild: An empirical study. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–10.

[18] Zihan Fang, Madeline Endres, Thomas Zimmermann, Denae Ford, Westley Weimer, Kevin Leach, and Yu Huang. 2023. A four-year study of student contributions to OSS vs. OSS4SG with a lightweight intervention. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23)*. ACM, New York, NY, 3–15. DOI: https://doi.org/10.1145/3611643.3616250

[19] FIRST. 2023. *Common Vulnerability Scoring System (CVSS)*. Retrieved from https://www.first.org/cvss/

[20] Stefan Frei, Martin May, Ulrich Fiedler, and Bernhard Plattner. 2006. Large-scale vulnerability analysis. In *Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense*, 131–138.

[21] Michael Gegick, Pete Rotella, and Tao Xie. 2010. Identifying security bug reports via text mining: An industrial case study. In *Proceedings of the 2010 7th IEEE Working Conference on Mining Software Repositories (MSR '10)*. IEEE, 11–20.

[22] Github. 2023. About coordinated disclosure of security vulnerabilities. GitHub. Retrieved from https://docs.github.com/en/code-security/security-advisories/guidance-on-reporting-and-writing-information-about-vulnerabilities/about-coordinated-disclosure-of-security-vulnerabilities

[23] GitHub. 2023. GPAC security policy. Retrieved from https://github.com/gpac/gpac/security

[24] GitHub. Accessed 2023. Privately reporting a security vulnerability - GitHub Docs. Retrieved from https://docs.github.com/en/code-security/security-advisories/guidance-on-reporting-and-writing-information-about-vulnerabilities/privately-reporting-a-security-vulnerability

[25] GitHub Inc. 2023. GitHub security advisory. Retrieved from https://github.com/advisories

[26] GitHub Inc. 2023. GitHub advisory *database*. Retrieved from https://github.com/github/advisory-database

[27] Google. 2023. How Google handles security vulnerabilities. Retrieved September 25, 2023 from https://about.google/appsecurity/

[28] Munawar Hafiz and Ming Fang. 2016. Game of detections: How are security vulnerabilities discovered in the wild? *Empirical Software Engineering* 21 (2016), 1920–1959.

[29] Allen D. Householder. 2017. *The CERT Guide to Coordinated Vulnerability Disclosure*. Carnegie Mellon Univ, Pittsburgh, PA.

[30] Allen D. Householder and Jonathan Spring. 2022. Are we skillful or just lucky? Interpreting the possible histories of vulnerability disclosures. *Digital Threats: Research and Practice* 3, 4 (2022), 1–28.

[31] Xing Hu, Xin Xia, David Lo, Zhiyuan Wan, Qiuyuan Chen, and Thomas Zimmermann. 2022. Practitioners' expectations on automated code comment generation. In *Proceedings of the 44th International Conference on Software Engineering*, 1693–1705.

[32] ISO. 2018. Information technology – Security techniques – Vulnerability disclosure. *Standard ISO 29147:2018*, Geneva, CH.
[33] KietNA-68. 2021. *PHP* Code Execution via WriteConfig() function. GitHub. Retrieved September 24, 2023 from https://github.com/gaozhifeng/PHPMyWind/issues/15
[34] Miryung Kim, Thomas Zimmermann, and Nachiappan Nagappan. 2014. An empirical study of refactoring challenges and benefits at Microsoft. *IEEE Transactions on Software Engineering* 40, 7 (2014), 633–649.
[35] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue. 2018. Do developers update their library dependencies? *Empirical Software Engineering* 23, 1 (2018), 384–417.
[36] Frank Li and Vern Paxson. 2017. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2201–2215.
[37] Jiahuei Lin, Bram Adams, and Ahmed E Hassan. 2023. On the coordination of vulnerability fixes: An empirical study of practices from 13 CVE numbering authorities. *Empirical Software Engineering* 28, 6 (2023), 151.
[38] Mend.io. 2023. Mend.io Vulnerability Disclosure Policy. Retrieved from https://www.mend.io/vulnerability-database/disclosure-policy/
[39] Microsoft. 2023. Report an issue and submission guidelines. Retrieved from https://httpd.apache.org/security_report.html
[40] MITRE Corporation. 2023. About CWE. Retrieved September 25, 2023 from https://cwe.mitre.org/about/index.html
[41] MITRE Corporation. 2023. Common vulnerabilities and exposures. Retrieved from https://cve.mitre.org/
[42] Giovane C. M Moura, and John Heidemann. 2023. Vulnerability disclosure considered stressful. *ACM SIGCOMM Computer Communication Review* 53, 2 (2023), 2–10. DOI: https://doi.org/10.1145/3610381.3610383
[43] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. 2015. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. IEEE, 692–708.
[44] National Cyber Security Centre. 2018. *Coordinated Vulnerability Disclosure: The Guideline*. Technical Report. National Cyber Security Centre, Netherlands.
[45] NIST. 2021. CVE-2021- 39503. Retrieved from https://nvd.nist.gov/vuln/detail/CVE-2021-39503
[46] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's knife collection: A review of open source software supply chain attacks. In *Proceedings of the 17th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 23–43.
[47] Hiroyuki Okamura, Masataka Tokuzane, and Tadashi Dohi. 2009. Optimal security patch release timing under non-homogeneous vulnerability-discovery processes. In *Proceedings of the 2009 20th International Symposium on Software Reliability Engineering*. IEEE, 120–128.
[48] S. Pan, J. Zhou, F. R. Cogo, X. Xiaÿuto, L. Bao, X. Hu, S. Li, and A. E. Hassan. 2022. Automated unearthing of dangerous issue reports. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 834–846.
[49] Karl Pearson. 1895. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London* 58, (1895), 240–242.
[50] Karl Pearson. 1900. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine*. 50, 302 (1900), 157–175. DOI: https://doi.org/10.1080/14786440009463897
[51] Qwaz. 2021. Buffer overflow in insert_many(). GitHub issue report. Retrieved from https://github.com/servo/rust-smallvec/issues/252
[52] Akond Rahman and Laurie Williams. 2019. A bird's eye view of knowledge needs related to penetration testing. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, 1–2.
[53] C. Ruffin and Christof Ebert. 2004. Using open source software in product development: A primer. *IEEE Software* 21, 1 (2004), 82–86.
[54] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu. 2012. A large scale exploratory analysis of software vulnerability life cycles. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 771–781.
[55] Spring Framework. 2023. Spring Security Policy. Retrieved November 22, 2023 from https://spring.io/security-policy
[56] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded theory in software engineering research: A critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering*, 120–131.
[57] The Black Duck by Synopsys. 2018. *Open Source Security and Risk Analysis*. Technical Report.
[58] U.S. National Institute of Standards and Technology. 2023. National Vulnerability Database. Retrieved from https://nvd.nist.gov/home.cfm
[59] U.S. National Institute of Standards and Technology. 2023. NVD Data Feed. Retrieved from https://nvd.nist.gov/vuln/data-feeds

[60] Georg Von Krogh, and Eric Von Hippel. 2003. Special issue on open source software development. *Research Policy* 32, 7 (2003), 1149–1157. DOI: https://doi.org/10.1016/S0048-7333(03)00054-4

[61] Daniel Votipka, Rock Stevens, Elissa Redmiles, Jeremy Hu, and Michelle Mazurek. 2018. Hackers vs. testers: A comparison of software vulnerability discovery processes. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 374–391.

[62] Xinda Wang, Kun Sun, Archer Batcheller, and Sushil Jajodia. 2019. Detecting "0-day" vulnerability: An empirical study of secret security patch in OSS. In *Proceedings of the 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 485–492.

[63] Dumidu Wijayasekara, Milos Manic, Jason L. Wright, and Miles McQueen. 2012. Mining bug databases for unidentified software vulnerabilities. In *Proceedings of the 2012 5th International Conference on Human System Interactions*. IEEE, 89–96.

[64] Yaqin Zhou and Asankhaya Sharma. 2017. Automated identification of security issues from commit messages and bug reports. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 914–919.

[65] Philip R. Zimmerman and Peter Ludlow. 1996. *How PGP Works/Why Do You Need PGP*. The MIT Press Cambridge.

Received 26 December 2023; revised 24 October 2024; accepted 29 January 2025