Unveil the Mystery of Critical Software Vulnerabilities

Shengyi Pan
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, Zhejiang, China
shengyi.pan@zju.edu.cn

Lingfeng Bao*†
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, Zhejiang, China
lingfengbao@zju.edu.cn

Jiayuan Zhou
Waterloo Research Center, Huawei
Waterloo, Ontario, Canada
jiayuan.zhou1@huawei.com

Xing Hu
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Ningbo, Zhejiang, China
xinghu@zju.edu.cn

Xin Xia
Huawei
Hangzhou, Zhejiang, China
xin.xia@acm.org

Shanping Li
The State Key Laboratory of Blockchain and Data Security, Zhejiang University
Hangzhou, Zhejiang, China
shan@zju.edu.cn

ABSTRACT

Today's software industry heavily relies on open source software (OSS). However, the rapidly increasing number of OSS software vulnerabilities (SVs) poses huge security risks to the software supply chain. Managing the SVs in the relied OSS components has become a critical concern for software vendors. Due to the limited resources in practice, an essential focus for the vendors is to locate and prioritize the remediation of critical SVs (CSVs), i.e., those tend to cause huge losses. Particularly, in the software industry, vendors are obliged to comply with the security service level agreement (SLA), which mandates the fix of CSVs within a short time frame (e.g., 15 days). However, to the best of our knowledge, there is no empirical study that specifically investigates CSVs. The existing works only target at general SVs, missing a view of the unique characteristics of CSVs.

In this paper, we investigate the distributions (from temporal, type, and repository dimension) and the current remediation practice of CSVs in the OSS ecosystem, especially their differences compared with non-critical SVs (NCSVs). We adopt the industry standard to refer SVs with a 9+ Common Vulnerability Scoring System (CVSS) score as CSVs and others as NCSVs. We collect a large-scale dataset containing 14,867 SVs and artifacts associated with their remediation (e.g., issue report, commit) across 4,462 GitHub repositories. Our findings regarding CSV distributions can help practitioners better locate these *hot spots*. Regarding the remediation practice, we observe that though CSVs receive higher priorities, some practices (e.g., complicated review and testing process) may unintentionally cause the delay to their fixes. We also point out the risks of SV information leakage during remediation process, which could leave a *window-of-opportunity* of over 30 days on median for zero-day attacks. Based on our findings, we provide implications to improve the current CSV remediation practice.

CCS CONCEPTS

• **Security and privacy → Software security engineering**; **Vulnerability management**.

KEYWORDS

Empirical Study, Critical Software Vulnerability, CVSS

ACM Reference Format:
Shengyi Pan, Lingfeng Bao, Jiayuan Zhou, Xing Hu, Xin Xia, and Shanping Li. 2024. Unveil the Mystery of Critical Software Vulnerabilities. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3663529.3663835

1 INTRODUCTION

Nowadays, software industry heavily relies on open source software (OSS), which serves as the foundation of the majority commercial codebases [3, 45]. Synopsys reported that 96% of scanned codebases (over 1,700 commercial codebases across 17 industries) contained open source and 76% of code in codebases was open source [45]. However, the increasing attacks targeting OSS software vulnerabilities (SV) in the software supply chain pose huge security risks to the software industry and may lead to significant consequences [44, 68, 76]. For example, Equafix suffered from a data breach due to a late fix of SV in Apache Structs (CVE-2017-5638 [9]), compromising the personal information of 143 million consumers and company loss exceeding $650 million [1]. Managing the SVs in the relied OSS components has become a critical concern for OSS users, especially for the commercial companies [2].

An essential focus for software vendors in managing the OSS SVs is to locate and remediate the critical SVs (CSVs) [4, 17, 36], i.e., those tend to be exploited and cause huge losses (e.g., the log4j vulnerability [12]). ISO/IEC 30111 [36], the industrial standard for SV handling processes, requires vendors to promptly address CSVs. A large IT company usually relies on hundreds of OSS components

*Also with Hangzhou High-Tech Zone (Binjiang) Blockchain and Data Security Research Institute
†Corresponding Author

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663835

in their codebases and products. Also, there is a rapid increase in the number of newly disclosed OSS SVs [45]. Facing a large number of OSS SVs, the vendor has to prioritize the remediation of CSVs due to the limited resource available and the potential significant consequences that these *hot spots* can cause. In practice, a typical organization only has the capacity to remediate one out of every ten SVs [35, 40]. Particularly, in the software industry, customers expect a short time frame on CSVs' remediation Service Level Agreements (SLA) [35, 46, 48, 50], which mandate the vendor to mitigate CSVs in a timely manner. European Cybersecurity Certification Scheme [17] requires cloud service vendors to handle CSVs within a few hours.

Though OSS SVs have attracted significant recent research attentions [51, 58, 59, 63, 65], the existing works only target at general SVs, missing a view of the unique characteristics of CSVs, e.g., highlighting the differences between CSVs and non-critical SVs (NCSVs). Considering that CSVs are of the greatest concern and the highest priority in practice, the goal of this study is to help practitioners understand the distributions and the current remediation practice of CSV in open source ecosystem though a comprehensive empirical study. Specifically, we follow the Common Vulnerability Scoring System (CVSS), the industry open standard for SV assessment [7, 60], to refer SVs with a 9+ CVSS score (under version 3) as CSVs and others as NCSVs in our work. We investigate the following research questions:

RQ1: What is the the spatio-temporal distribution of critical software vulnerabilities (CSVs)?

There is a lack of illustration of CSVs, which is essential to provide practitioners with the exact insights of their targets, and further help to locate CSVs and devise better remediation strategies. Thus, we first provide a high-level impression of CSVs by revealing its distribution from multiple perspectives, particularly, the temporal distribution, type distribution, and repository distribution. These perspectives are similar to those investigated in the existing studies targeting at general SVs [51, 65]. During the analysis, we pay special interests to unveiling the distribution differences of CSV against NCSV, which highlight its unique characteristics.

Besides, CVSS has finished evolving from version 2 to 3 with the new version designed to overcome several known issues of the old one [14, 39]. However, most of the existing industry practices [35, 46, 48] (e.g., the scope of CSV) and studies [60, 62] (e.g., severity prediction) are still based on the deprecated CVSSv2 standard. It is important to unveil the potential changes, especially regarding the scope of CSV (e.g., proportion, composition), when transition from CVSSv2 to CVSSv3 for both practitioners and researchers. Thus, we investigate the differences in severity distribution between v2 and v3, and how does the evolution of CVSS affect the scope of CSV.

RQ2: How CSVs are remediated in the OSS ecosystem?

Knowing the SV remediation practice in OSS is important for practitioners to understand the risks in the current practice and identify areas for improvement. Given the transparency of OSS, development activities during the remediation process, e.g., issue report (IR) and commit, are visible to attackers. This brings huge security risks, as attackers can take advantage of the SV-relevant information leaked in these mitigation activities to conduct zero-day attacks. Existing works [51, 58, 63] only investigate the general SV life span (i.e., the gap between SV being introduced and removed in the codebase) without a focus on the remediation process, or only discuss the gap between SV fix and disclosure [59]. We comprehensively review the entire SV remediation process (i.e., from report to disclosure) and identify the potential risks. Figure 1 shows a comprehensive timeline on how SVs are *typically* remediated in the OSS and the possible windows of delays. Note that there could be some special cases, e.g., the patch has not been developed when the SV is disclosed on the National Vulnerability Database (NVD) [37].

Besides, one might expect practitioners to attach higher priorities to its remediation over NCSVs, since CSVs are believed to have less exploitation complexity and larger security impacts [14]. A closer examination specifically targeting at the remediation practice of CSVs are needed to investigate 1) if CSVs are treated differently with the higher priority and shorter delay, 2) to what extent the aforementioned risks affect CSV. Thus, we further take a closer look to the CSV remediation practice, and highlight the differences in the remediation timeliness against NCSVs.

In summary, our paper makes the following contributions:

- Different from prior works that target at general SV, we are the first to investigate the characteristics of CSV and compare its differences against NCSV, We aim to provide insights for CSV characteristics, which are of the greatest concern and the highest priority in practice. We also investigate the evolution of the CVSS schema.
- We conduct a deep analysis of the comprehensive timeline of the current SV remediation practice in the OSS community, reviewing the possible risks and pointing out suggestions for both OSS maintainers and users (e.g., software vendors).
- We collect a large-scale dataset that ties the SV metadata with the extensive software artifacts (e.g., IR, PR, and commit) of the remediation process. We open this dataset along with the scripts used in collection to facilitate future researches [49].

2 BACKGROUND

Common Vulnerabilities and Exposure (CVE) provides a standardized method to identify, define, and catalog publicly disclosed SVs [5]. Each CVE record has a CVE ID, a brief description, and the related references.

Common Weakness Enumeration (CWE) serves as a common language for discussing and describing weaknesses [8]. Each CWE entry represents a single SV type, providing detailed information including the common causes, behaviors, and consequences.

Common Vulnerability Scoring System (CVSS) is the *de facto* standard for assessing the severity of SVs [6]. CVSS consists of three metric groups [14], Base, Temporal, and Environmental. Public assessments of SV severity (e.g., NVD [37], Synk [47]) typically refer to Base metrics, which represent the intrinsic properties of SVs that are constant over time and across specific user environments [14]. We also focus on the Base metrics in our study. The Base metrics characterize SV from two aspects (i.e., exploitability and impact), and produce a severity score ranging from 0 to 10, which can be further mapped into four ratings (see Table 1). CVSS has finished evolving from v2 to v3 [39]. CVSSv3 addresses several known issues of v2 and introduces scoring changes that more accurately reflected the reality of SV encountered in the wild [14]. We adopt the CVSSv3 to determine the scope of CSV and NCSV in this work. We also discuss the changes when transition from v2 to v3.

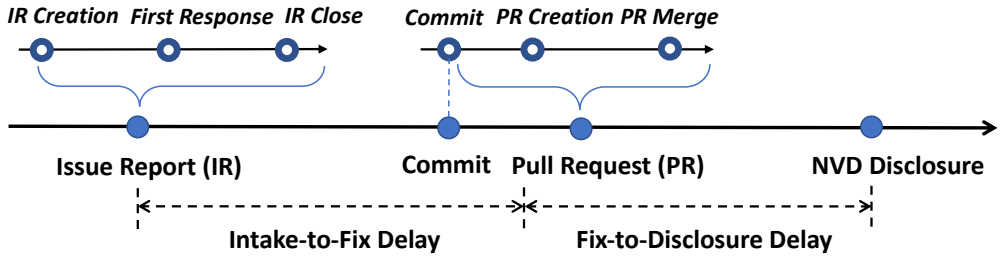


Figure 1: A typical remediation timeline of OSS SVs

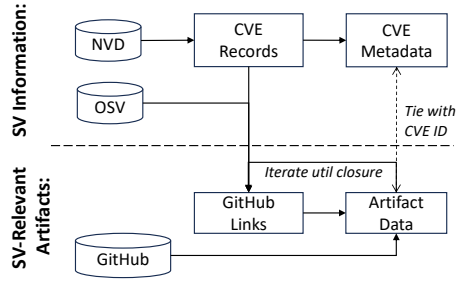


Figure 2: Overview of data collection process

3 DATA COLLECTION METHODOLOGY

In this section, we describe the data collection process and results. Figure 2 presents the overview of the data collection process. First, we collect CVE records associated with OSS projects hosted on GitHub from NVD. Second, we use the software artifacts listed in the external references of the CVE record as seeds to retrieve as much relevant artifacts as possible. Finally, we crawl all valid artifacts and tie them with the associated SV metadata. The collected dataset, along with the scripts used in collection are provided at [49].

3.1 Collecting SV Information

We first collect all CVE records associated with OSS projects hosted on GitHub from NVD (on Sept. 15, 2022). Specifically, we specify six types of software artifacts (i.e., commit, issue report, pull request, GitHub security advisory, release tag, and change log), and collect CVE records with external references containing at least one of the specified artifact link. We match each external reference with predefined regex expressions. These six types of artifacts are the most commonly listed references by NVD for CVEs relevant to GitHub projects according to our observations. For GitHub links apart from the specified six types, we find they are likely to be *out-of-scope* (not the software artifacts related to SV remediation in the vulnerable GitHub repository), e.g., artifacts from repositories owned by bug hunters for documenting their discovered SVs (see the reference of CVE-2021-33823 [10] for an example). Furthermore, we observe that it is also possible for the six specified types of artifacts to be *out-of-scope*. We further filter out invalid links by heuristics refined according to our observations, e.g., matching the repository name extracted from the link with the affected software listed by NVD.

Besides, we further complete the relevant GitHub links from OSV [41], a recently popular SV database targeted for OSS, using CVE ID as the unified SV identifier. By incorporating OSV as an additional source, we further retrieve 2,943 GitHub links.

For each valid CVE record (i.e., with at least one of the valid GitHub link), we further extract relevant information (i.e., CVSS scores, CWE ID, publish date, and description) from NVD. Note that OSV provides its own SV metadata (e.g., CVSS, CWE), which can be different from those provided by NVD. Thus, we only use OSV as an additional source for GitHub links, while extract all other SV metadata from NVD to ensure consistency.

Finally, we clean each collected GitHub link with the following three steps: ❶ remove extra suffixes of the URL link, ❷ update the URL link to the standard form, e.g., update the outdated repo/owner name, complete the prefix of commit hash, ❸ remove duplicate links, In total, we collect 18,970 CVEs and 28,106 associated Github links.

3.2 Completing SV-Relevant Artifacts

The RQ2 of our work investigates the detailed timeline of SV remediation in OSS (see Figure 1). We focus on three specific software artifacts in the remediation process, i.e., issue report (IR), commit, and pull request (PR). The external references provided by NVD are not intended to be complete. Also, once the CVE is published, NVD is unlikely to continue tracking the SV and updating the reference list unless updates are reported to them. Thus, we attempt to further complete the associated artifacts using the existing ones as seeds.

Generally, we build custom crawlers and parsers (based on the GitHub REST API [32] and GraphQL API [19]) for each artifact type to mine potential linkages to the relevant artifacts. Then, we clean the retrieved URLs as described in Section 3.1, and repeat this mining-cleaning process until the closure. We discard the release tag and change log from the mining process (accounting for 9.3% of the collected GitHub links), as these two types of artifacts typically consist of free-form text and it is hard to precisely identify the links to the SV-relevant artifacts. The brief mining strategies of the other four artifact types are:

- **Issue Report (IR):** We crawl the IR timeline, which records every related GitHub events in a chronological order [33]. We search for events of selected types to retrieve relevant artifacts (e.g., event *referenced* for a commit that references the IR).
- **Commit:** We apply regex expressions to extract the IR/PR id, commit hash, and GHSA/CVE id mentioned in the commit message. GitHub automatically converts the references to artifacts into shortened links [18]. We define the regex according to the autolink formats listed in the GitHub document.
- **Pull Request (PR):** Similar to commit, we extract the relevant artifact ids from the PR title and body. Besides, using GitHub API, we can directly list the commits associated with the PR.
- **GitHub Security Advisory (GHSA):** Given a GHSA/CVE id, we retrieve the advisory using GraphQL API [20]. The advisory is structured, with related links listed in the *references* field.

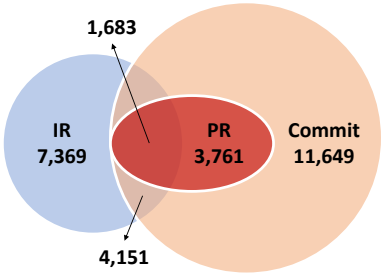


Figure 3: Venn plot of the collected dataset

Table 1: Distribution of severity ratings

Severity Ratings	CVSS-3	CVSS-2
CRITICAL (9.0-10.0)	2676 (18.0%)	316 (2.1%)*
HIGH (7.0-8.9)	5437 (36.6%)	2653 (17.8%)*
MEDIUM (4.0-6.9)	6591 (44.3%)	10164 (68.4%)
LOW (0.1-3.9)	163 (1.1%)	1734 (11.7%)

CVSSv2 does not have the rating CRITICAL. There are 2969 (20.0%) SVs with HIGH (7.0-10.0) severity.

We valid each retrieved artifact with pre-defined heuristics, e.g., ensuring it belongs to the same repository, the relative chronological order is logical. After several iterations, we finally retrieve 36,242 relevant GitHub links, which is nearly 30% more than the number of links listed in the public SV databases (i.e., 28,106), validating the value of our proposed extensive completing process.

3.3 Crawling Software Artifacts

Based on the GitHub links collected in the last step, we crawl the information of these SV-relevant software artifacts and further integrate the artifact data and the SV data (collected in Section 3.1) to generate the final dataset. We implement custom crawlers for repository (e.g., stars, last commit date), IR (e.g., creation date, close date, title), commit (e.g., commit date, diff), and PR (e.g., merge date, whether from a fork project).

After excluding the broken links (e.g., the repository is deleted), there are 17,306 CVEs left, each with at least one of the valid GitHub IR, commit, or PR. We then further remove CVEs (2,439 in total) with either invalid CVSSv2 or CVSSv3 scores. NVD does not give CVSSv3 scores for CVEs that were analyzed before Dec. 2015 [38], and CVSSv2 scores for those after July 2022 [39].

Finally, we collect 14,867 CVE records spanning across 4,462 GitHub repositories, among which 7,369, 11,649, and 3,761 have associated IRs, commits, and PRs, respectively (see Figure 3). Note that each RQ has additional requirements (e.g., investigating the IR-to-PR delay requires the SV to have both associated IRs and PRs), which would further restrict the dataset. Compared with the datasets used in the existing empirical studies [51, 58, 59, 63] (ranging from 550 to 4,377 SVs), our dataset for analysis is much larger and includes extensive relevant software artifacts to facilitate researches regarding SV remediation practice.

4 RQ1: SPATIO-TEMPORAL DISTRIBUTION

In this section, we present our findings for RQ1. Specifically, we investigate the distribution of CSVs from three dimensions, i.e., over time, across CWEs, and across repositories. Besides, we analyze the differences in severity distributions between CVSSv2 and CVSSv3 to unveil the potential changes brought by the evolution of CVSS.

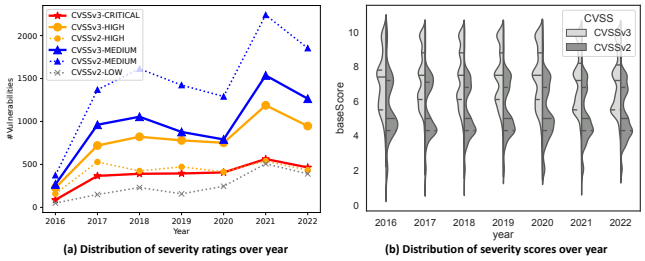


Figure 4: Distribution of severity ratings and scores over years

4.1 Trend over Time

Table 1 presents the severity rating distribution of the 14,867 collected SVs (see Section 3.3) under CVSSv2 and CVSSv3, respectively. Figure 4 further presents the distribution of severity ratings and scores over the years. Note that since there are only 1.1% of the SVs in the RQ1 dataset are of LOW severity under CVSSv3 (see Table 1), we merge the LOW SVs into the MEDIUM ones in Figure 4(a). We observe a stable score distribution (so as the proportions of severity ratings) over years (see Figure 4). The proportion of CSVs (i.e., CRITICAL rating under CVSSv3) is roughly 20%.

Compared with CVSSv2, ❶ CVSSv3 is less biased to the MEDIUM rating as shown in Figure 4(a), and thus has larger information entropy [57] (i.e., 1.57 *vs.* 1.30). ❷ CVSSv3 scores SVs much higher. As presented in Figure 4(b), the median/bottom-quartile score of CVSSv3 is above the top-quartile/median score of CVSSv2 in every year. Moreover, CVSSv3 is more likely to rate SVs with extremely high scores (i.e., close to 10.0). These changes are also reflected in severity ratings (see Table 1). The proportion of SVs of CRITICAL and HIGH SVs increase by 757% and 105%, respectively.

We further quantify the score changes when switching from CVSSv2 to CVSSv3. 94.4% (14,041 out of 14,867) SVs in the RQ1 dataset increase the severity score. The median and average scores increase is 2.2 and 1.95, respectively. The three largest types of rating changes are MEDIUM→HIGH (4,783), HIGH→CRITICAL (2,110), and LOW→MEDIUM (1,541).

However, although CVSSv3 scores are generally higher than CVSSv2 scores, we observe that the proportions of the CRITICAL rating (i.e., 9+ score) under CVSSv3 and HIGH rating (i.e., 7+ score) under CVSSv2 remain consistently close (roughly 20%) over the years (see Figure 4(a)).

The severity distributions are stable over years, and the proportion of CSVs is roughly 20%. Compared with CVSSv2, CVSSv3 scores SVs with higher severity scores (94.4% SVs observed with an increase in severity score). The proportions of SVs with 9+ CVSSv3 score and 7+ CVSSv2 score are close to.

4.2 Distribution across CWE Categories

The entire SVs (14,867 in total) in the RQ1 dataset correspond to 225 CWE categories, and 80% of which fall into the top 33 CWE categories. The distribution of CSVs is more concentrated towards specific CWE categories. 2,676 CSVs correspond to 132 CWE categories and 80% of which are concentrated in the top 22 CWE categories. Table 2 presents the five most common CWE categories among the total and CRITICAL SVs, respectively. We observe that CWEs with more SVs do not necessarily correspond to more CSVs.

Table 2: Five most common CWEs of total and Critical SVs

Rank	Total		
	CWE ID	CWE Name	Proportion
1	CWE-79	Cross-site Scripting	15.3%
2	CWE-787	Out-of-bounds Write	6.1%
3	CWE-125	Out-of-bounds Read	6.0%
4	CWE-476	NULL Pointer Dereference	4.3%
5	CWE-89	SQL Injection	3.9%
	Critical		
1	CWE-89	SQL Injection	12.8%
2	CWE-787	Out-of-bounds Write	8.1%
3	CWE-125	Out-of-bounds Read	7.5%
4	CWE-119	Buffer Overflow	4.8%
5	CWE-78	OS Command Injection	3.8%

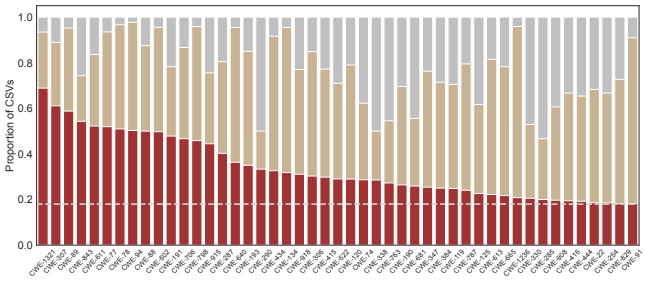


Figure 5: CSV proportion of each CWE category. The red, yellow, and grey bar represents the CRITICAL, HIGH, and MEDIUM SVs, respectively. The gray dotted line represents the average proportion of CSV in the entire dataset.

Since CWE illustrates the nature of SVs (e.g., root cause and impact) [60], we explore its relation with the severity rating by comparing the CSV proportion of each CWE category. We remove CWE categories (113 out of 255) with fewer than 10 SVs, which correspond to 2% SVs in the RQ1 dataset. We observe that certain CWE categories have a higher proportion of CSVs. Figure 5 presents the proportion of different severity ratings for 47 CWE categories whose CSV proportion is above the average (i.e., 0.18). SVs belonging to CWE-1321 [15] (*Prototype Pollution*) has the highest CSV proportion (i.e., 0.69). We further compare the CWE Top 25 Most Dangerous Software Weakness list [16], a commonly adopted reference for *hot spot* SVs in practice, with the top 25 CWEs with the highest proportion of CSVs. Surprisingly, there are six differences. For example, CWE-1321, the one with the highest CSV proportion, is not in the list, possibly due to a relatively small population of the corresponding SVs since the CWE Top25 list is ranked by considering both the number and the average severity of the corresponding SVs [16]. We argue that CWEs with high CSV proportions are also worth extra attention from practitioners. Since once SVs of these types are detected, they are likely to be critical.

We further compare the CWE distributions between SVs of CRITICAL rating (i.e., 9+ score) under CVSSv3 and HIGH rating (i.e., 7+ score) under CVSSv2. For the top 20 CWEs with the largest population of each group (corresponding to 77.9% and 75.2% of SVs of CRITICAL severity under CVSSv3 and HIGH severity under CVSSv2, respectively), except for one different CWE, the two groups of SVs share the left 19 CWEs. Though CVSSv3 tends to give higher severity scores, it largely follows the same criterion (e.g., proportion, composition) of the most severe SVs from CVSSv2.

CWEs with more SVs do not necessarily correspond to more CSVs. Certain CWEs have a higher proportion of CSVs, and they worth more attention from practitioners. SVs of 9+ CVSSv3 score and 7+ CVSSv2 score correspond to similar CWE categories.

4.3 Distribution across Repositories

Similar to the distribution across CWEs, we find that repositories with more SVs do not necessarily have more CSVs, e.g. there are seven differences between the top 10 repositories with the most SVs and CSVs. Besides, the entire SVs in the RQ1 dataset span across 4,462 GitHub repositories, while the CVSs only span across 1,543 repositories, meaning that nearly two third of the repositories do not have any CSVs. However, among the repositories with at least one CSV, over half only report CSVs. We further investigate the details of these two contrasting cases.

First, 2,919 (65.4%) repositories do not have any CSV, which corresponds to 5,599 (37.7%) SVs. These repositories generally have fewer SVs compared to the left ones. We find that the difference is statistically significant ($p < .0001$) by applying Mann-Whitney U statistical test [67]. Based on manual analysis, we observe reasons including: ❶ the repository is not widely used. 836 (28.6%) repositories have less than 100 Stars. ❷ the repository has not been maintained. The date of the last commit of 706 (24.2%) repositories is one year ago. ❸ topics of the repository are less security sensitive or the development of the repository follows good security practices. For example, `tesseract-ocr/tesseract` [31] is a widely-used (48.5k Star) and actively-developed OCR engine, which is not a critical application and unlikely to touch the sensitive data. It has only one published SV [11] so far. Besides, there are still 56 repositories that have more than 10 SVs (within the top 5% repositories with the most SVs). For these repositories, a possible explanation for the absence of associated public CSVs is that the maintainers may intentionally hide CSVs due to certain concerns (e.g., reputation) [73].

Among the remaining 1,543 (34.6%) repositories that have at least one CSV, we observe that over half (832) only report CSVs. We observe a statistically significant difference ($p < .0001$) for the number of stars between repositories with only CSVs and the left ones, as these repositories generally have fewer stars. We suspect that small-scale projects tend to favor simplified software management (i.e., disclosing the documenting only severe SVs) due to the lack of manpower and security regulations. Thus, these repositories only disclose CSVs that have higher priorities. However, there are still 208 repositories with only CSVs (i.e., the fourth quartile) that have more than 1255 Stars. We conduct a manual analysis of such repositories and conclude the following findings: ❶ Some repositories are no longer maintained. However, due to the widespread use of these repositories, there are still new SVs being disclosed recently. Similar to the small-scale projects, they only disclose CVSs due to limited management resources. Considering that such SVs may still affect a considerable amount of downstream projects/users, we suggest that the community should pay more attention to such popular-yet-unmaintained repositories and take the responsibility of maintaining the most essential SV management (e.g., patch development, notifications). ❷ For repositories that under active

development, we find that they have strict standards for determining SVs, and their security practices are to disclose CSVs only. For example, `facebook/folly` [30], a popular (24.5k Stars) and actively-developed open source C++ library, clearly illustrates in its security policy [43] that users should privately report security issues via a specified channel and the reported issue will be further examined and triaged by the repository security team.

Repositories have varying CSV proportions, relating to the development topics, scale and popularity, and disclosure practices. We observe repositories avoiding disclosing CSVs, as well as those only disclosing CSVs. For unmaintained-yet-popular projects, there might still have new CSVs being disclosed.

5 RQ2: SV REMEDIATION TIMELINE

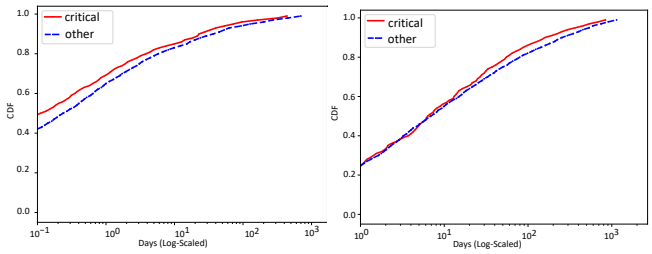
In this section, we present our findings regarding the timeline of SV remediation. Figure 1 shows a comprehensive timeline on how SVs are *typically* remediated in the OSS and the possible windows of delays. We aim to help practitioners better understand the details of the SV remediation process and the differences between CSVs and NCSVs, unveiling the potential risks in the current practice.

5.1 Issue Report Timeline

Issue Report (IR) signals the start of the SV remediation, where for the first time the project team is informed of the existence of a potential SV. When a project user discovers a potential SV, it is common for he/she to report it to the project team using the GitHub issue tracking system (ITS) [71]. However, the report also marks the start of the dangerous *window of opportunity*, since the attackers can leverage the leaked SV information to conduct zero-day attacks before disclosure (when the public becomes aware of the SV and start remediation), by monitoring the public development activities of OSS (e.g., newly posted IRs in the ITS). Moreover, the SV details reported in the IR (e.g., SV behaviour, reproduction steps) provide attackers with advantages in creating a functional exploit (see IR [21] of CVE-2022-31836 [13] for an example). Specifically, we observe that at least 41.7% (3,072/7,369) of the SV-reporting IRs in our dataset contain attack steps. We use a set of keywords (i.e., "steps to reproduce", "steps to replicate", "proof-of-concept", "proof of concept"", and "poc") to identify IRs with attack steps follow Pan *et al.* [71]. Note that this method provides lower-bound estimation, since some attack steps can be described in other ways (e.g., "vulnerability recurrence").

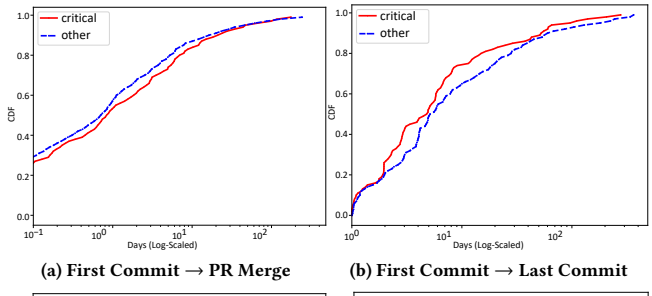
IR Creation → First Response. We use a cumulative distribution function (CDF) plot (see Figure 6(a)) to show the time delay of the first response relative to the IR creation for CSVs and NCSVs, respectively. The delay for 69.1% CSVs and 65.0% NCSVs are within one day, respectively. As shown in the figure, IRs of CSVs generally have a quicker response. Also, we find the difference to be statistically significant ($p < 0.0001$).

IR Creation → IR Close. Figure 6(b) presents the time delay of the IR close after its creation. There are nearly 20% SVs whose IRs closed within one day. The median lifespan for IRs of CSVs and NCSVs are 6.3 and 6.9 days, respectively. We find no statistically significant difference ($p > 0.19$) for the creation-to-close gap between CSVs and NCSVs. However, from the CDF plot, we observe that for IRs closed with a relatively large delay (e.g., more than one week), CSVs

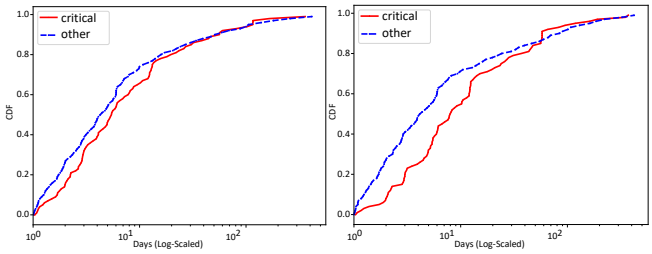


(a) IR Creation → First Response (b) IR Creation → IR Close

Figure 6: CDFs of the delay of the SV IR timeline



(a) First Commit → PR Merge (b) First Commit → Last Commit



(c) Last Commit → PR Merge (d) PR Creation → PR Merge

Figure 7: CDFs of the delay of the SV PR timeline

are less likely to face extremely long delays. We are more concerned with IRs closed with long gaps, since a larger delay brings much more risks given that the SV may have been exposed to malicious actors once the IR is publicly reported. For IRs with a creation-to-close delay larger than one week (corresponding to 48.5% CSVs and 49.8% NCSVs), we observe a statistically significant difference ($p < 0.01$) between CSVs and NCSVs.

Attackers can leverage the leaked SV information in IR to conduct exploits. Practitioners attach higher priorities to CSV reports, i.e., IRs of CSVs are responded and addressed in a more timely manner.

5.2 Pull Request Timeline

After the project maintainers intake the SV information through IR, the next critical development activity is patching the SV (see Figure 1), by either directly pushing the commits into the codebase or using a PR. PR documents the changes to be pushed to fix the SV and allows practitioners to discuss, review, and test the fix before it is merged into the branch.

First Commit → PR Merge The date of the first associated commit of PR signals the start of the patch development. However, the patch is not available to project users until the PR is merged, since users typically update the project by pulling from the base branch (e.g., `master`) or just wait for the next release [59]. We investigate the delay of patch availability in the base branch (i.e., the time lag

between the first commit and the PR merge) of CSVs and NCSVs. As shown in Figure 7(a), the time lag of CSVs is longer than that of NCSVs. The difference is significant ($p < 0.03$), indicating that CSVs face larger delay compared to NCSVs though they should have higher priorities. This observation is contrary to the one from IRs (see Section 5.1), where IRs of CSVs are addressed in a more timely manner. We further break down the time gap between the first commit and the PR merge into two pieces, i.e., the development process of the patch (i.e., the time gap between the first and the last commit) and the time lag between the last commit and PR merge.

First Commit → Last Commit For the patch development process, the majority (i.e., 81.5% of total SVs) have all commits committed within one day. Among the left SVs (i.e., whose patches have a relatively long development process), we observe that patches of CSVs are developed more quickly (see Figure 7(b)) and the difference is significant ($p < 0.03$).

Last Commit → PR Merge However, after the patch is ready, we find that the patches of CSVs require a larger delay to be finally merged (see Figure 7(c)). The difference in the time lag between the last commit and the PR merge of CSVs and NCSVs is significant ($p < 0.01$). We conduct a manual analysis to explore the reasons behind such a delay. We find that CSVs generally lead to more reviews and discussions. Thus, although the patch of CSV is developed more quickly given its priority, its availability in the codebase suffers a larger delay (i.e., 5.4 days on median) due to more complicated discussions and review processes.

PR Creation → PR Merge Generally, contributors first develop the patch in a dev branch or a forked project and later create the PR when they think the patch is ready or close to. The majority of PRs (76.0%) in our dataset are created with associated commits. Once the PR is created, it makes the un-patched SV more obvious to potential attackers as the PR explicitly shows the intent of SV fixing and SV-relevant information (e.g., the vulnerable code lines). Thus, the larger the delay of PR merge after its creation, the higher the risk of SV being exploited. We observe that severity affects whether a patch is developed before or after PR creation. PRs of CSVs are less likely to be created without any commits compared with those of NCSVs (20.1% vs 24.9%), which is considered to be less dangerous since having patch developed after PR creation is likely to delay the merge and the PR also makes the patch development process more visible. However, for PRs with the associated commits ready before the creation, we find that CSVs suffer a larger delay to be merged (considering those with a delay over one day) as shown in Figure 7(d). The difference is significant ($p < 0.001$). This observation verifies our previous finding that CSV fixes require more discussions and reviews, which unintentionally (as CSV patches are developed more quickly) leads to larger merge delays.

Unmerged PRs We also observe that there are 514 (13.7%) SVs whose associated PRs are unmerged, Among which, 395 are closed. To investigate the possible reasons behind these closed yet unmerged PRs, We manually examine 50 randomly sampled PRs and have the following observations: ❶ The PR is closed due to security concerns (e.g., switch into a private channel to hide the details). For example, in PR#2924 of project `pjsip/pjproject` [28], one of the team members reviewed the PR and suggested using a private PR instead. The PR was soon closed to avoid further information leakage. ❷ The fix is merged into the codebase through another PR (e.g., a better way to patch the SV). For example, PR#113 of project `actix/actix-net` [25] was initially proposed to patch the SV. However, it was stuck for more than 6 months due to a lengthy discussion and development to optimize the patch. Finally, one of the team members suggest proposing to apply a simple patch first due to the urgency of the SV [26]. ❸ The fixing commits are directly pushed to the codebase by the team member of the project instead (e.g., for convenience) [27, 29].

Although CSV patches are developed more quickly given their priorities, they may not appear in the codebase timely due to the complicated review and testing processes. Possible reasons behind unmerged security PRs including, using a private PR instead due to security concerns, associated changes being directly pushed by team members, etc.

5.3 Report-to-Fix Delay

We investigate how long the project team starts to patch the SV after the issue is reported, and whether CSVs are patched with higher priorities compared with NCSVs.

IR Creation → PR Creation The median time lag between IR creation and PR creation for CSVs and NCSVs are 3.0 and 7.8 days, respectively. We find CSVs are patched much more quickly compared with NCSVs. The difference is significant ($p < 0.01$), indicating that developers attach higher priorities to the remediation of CSVs. Besides, we observe that for 32.1% of SVs in our data, the associated PRs are created over 30 days after the IR creation. Such long report-to-fix delays provide malicious actors sufficient time to develop and conduct functioning exploits.

IR Creation → First Commit We do not retrieve any PRs for 2,468 SVs in our dataset. Some patches are directly pushed into the codebase by the project maintainers without a corresponding PR due to security concerns or easy development (see Section 5.2). We observe that patches pushed without PRs are associated with more small-scale (i.e., fewer Stars) projects, and the difference is significant ($p < 0.01$). Besides, it is also possible that some PRs are missed during our data collection (see discussions in Section 7).

For the time lag between IR creation and the first commit, we do not observe a significant difference ($p > 0.1$) between CSVs and NCSVs. The first commits of 42.9% and 71.3% SVs come within 1 and 7 days after the IR creation, respectively, which is much faster than those using a PR. However, there is still a considerable amount (24.9%) of SVs whose fixes are committed over 30 days after the creation of the associated IR.

Generally, CSVs are patched in a more timely manner compared with NCSVs. Besides, a considerable amount (over 25%) of SVs suffer a report-to-fix delay over 30 days, leaving a window wide open to for potential attackers to conduct exploitations.

5.4 Fix-to-Disclose Delay

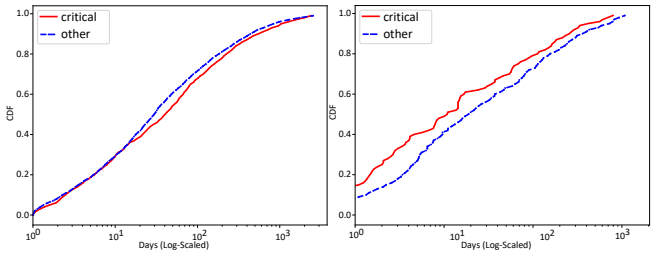
IR Creation → NVD Publish Most IRs (98.8%) are created before the NVD disclosure of the associated SVs, and over half are posted in public ITS 30 days earlier than the NVD disclosure. This leaves an exploit window wide open, as we show in Section 5.1 that attackers can leverage the leaked SV information in these IRs to conduct

zero-day attacks. We further observe that 10.5% SVs are disclosed on NVD within one day after the creation of the associated IR. One possible explanation is that the reporters of these SVs *irresponsibly* disclose the SV without giving project maintainers sufficient time or even a chance to take remediation. Such disclosure behaviours could bother the project maintainers [42]. For SVs with a publishing delay over one day, we observe that CSVs generally face a larger delay. The median delay for CSVs and NCSVs are 45.5 and 37.6 days, respectively. This observation motivates us to further investigate the time lag between the fix and NVD publish, as we discussed in Section 5.3 that developers are more urgently to patch CSVs.

Fix → NVD Publish While the patch is available in the codebase, the majority of OSS users still remain unaware of the patch or even the SV since they typically rely on public SV databases (e.g., NVD) for SV-relevant information (e.g., using SCA) [59, 69, 75]. Thus, we further investigate the delay between patch commit date (or PR creation date) and the NVD disclosure date for SVs in our dataset. We observe that 10.6% SVs are disclosed on NVD within one day after the patch is committed. Disclosing SVs immediately after the fix will force developers to quickly react as the advisory circulates the SV information. According to Bilge *et al.* [52], the exploits of the SV increase as high as five orders of magnitude at the disclosure. For the rest SVs, we find the fix-to-disclosure delay of CSVs is longer (Figure 8(a)), and the difference is significant ($p < 0.01$). The median disclosure delay for CSVs and NCSVs are 40.2 and 29.7 days, respectively. One possible reason is that project maintainers are more cautious about the disclosure of CSVs, i.e., leaving more time to allow the security release to be sufficiently installed by the client users [34]. However, there is also a risk with the relatively long disclosure delay, as the security release may unlikely to be widely installed in the wild before the public disclosure [63], which in turn provides potential attackers a window wide open for exploitation.

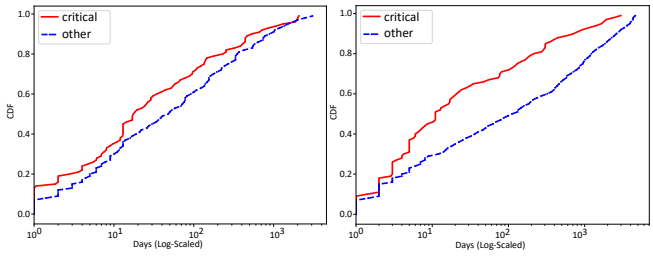
We also observe that 7.7% SVs are disclosed on NVD even before a fix has been made. Among these SVs (patched after disclosure), we find the patch to come within 10.8 and 18.8 days (on median) after the disclosure for CSVs and NCSVs, respectively. 36.2% CSVs and 43.8% NCSVs have no patch available until 30 days after the disclosure. One possible explanation is that the discoverer of the SV (e.g., bug hunter) discloses it without any prior notice to the project maintainers [59, 63]. Disclosing SVs without an available patch is extremely dangerous, which directly exposes the unpatched SVs to attackers and put the practitioners at a great disadvantage as they have to take mitigation in a hurry. As shown in Figure 8(b), we find that CSVs are patched more quickly after the disclosure. The difference is significant ($p < 0.05$), indicating that developers attach much higher priorities to CSVs under such a scenario.

The report-to-disclosure delay for over half of the SVs exceeds 30 days, leaving an exploit window wide open for attackers. CSVs face a longer fix-to-disclosure delay (40.2 days on median), as project maintainers may intentionally delay the CSV disclosure to allow security releases to be sufficiently installed. 7.7% SVs are patched after disclosure, bringing huge security risks. Among them, 36.2% CSVs have patch available for more than 30 days.



(a) SV Fix → NVD Publish (b) NVD Publish → SV Fix

Figure 8: CDFs of delta days between SV disclosure and fix.



(a) First PR → Last PR (b) First Commit → Last Commit

Figure 9: CDFs of the survive time of problematic patches

5.5 Patch Reliability

End users generally trust in applying the released patch to fix the SV. However, the patch developed to address an SV is not always reliable [63]. We also observe SVs that suffer from problematic patches in our dataset, i.e., those associated with multiple PRs/commits. For example, the IR [22] reporting CVE-2017-5617 was first closed on 2017-05-26 as addressed by PR#12 [24]. However, one year later, on 2018-10-15, one of the project's core developers commented that the original fix is incomplete and further addressed in PR#33 [23].

We show in the previous Sections that practitioners generally pay more attention to the remediation of CSVs compared with NCSVs. Thus, one might expect that patches of CSVs have better reliability. In this section, we further investigate whether there is a difference in patch reliability between CSVs and NCSVs.

Multiple PRs. 13.7% CSVs and 19.6% NCSVs in our dataset have multiple associated PRs. The security fixes for CSVs are less likely to be problematic compared to NCSVs. We further investigate the timespan of problematic patches remaining unresolved. The top quartile of days required to complete the original broken fixes is 136, suggesting that problematic fixes can remain unresolved for a considerably long time. We further compare the lifespan of problematic patches (i.e., the time gap between the creation of the first and the last PR) between CSVs and NCSVs. We find the difference to be statistically significant ($p < 0.05$), as problematic patches for CSVs are discovered and fixed in a more timely manner (see Figure 9(a)). The median time lag for problematic patches to be resolved for CSVs and NCSVs are 17.5 and 48 days, respectively.

Multiple Commits. We also investigate the SVs associated with multiple commits to explore the patch reliability. Since developers may split the patch development into successive commits [63], we regard fixes with all commits committed within one day (54% of all fixes with multiple commits) as *piece-wise* fixes and non-problematic. Generally, our observations are inline with those towards multiple PRs. 7.0% CSVs and 10.3% NCSVs in our dataset

have multiple associated commits, suggesting that security fixes for CSVs are less likely to be problematic. When exploring the lifespan of problematic patches, we also have the same observations that problematic patches of CSVs are discovered and addressed more quickly (see Figure 9(b)) and the difference is significant ($p < 0.001$). The median time lag of problematic patches remain unsolved for CSVs and NCSVs are 11 days and 111 days, respectively.

Compared with NCSVs, CSVs have better patch reliability and have problematic patches resolved in a much more urgent manner. However, even for CSVs, nearly 10% patches are problematic when first introduced and requires over 11 days on median to be resolved.

6 DISCUSSION

In this section, we discuss the practical implications of our findings for both OSS maintainers and users (e.g., the software vendor) to better understand and remediate the CSVs.

Transition from CVSSv2 to CVSSv3. The transition from CVSSv2 to CVSSv3 in NVD is completed in July, 2022 [39]. However, most of the existing industry practices [35, 46] and research studies [60] (e.g., SV assessment techniques) are based on the deprecated CVSSv2 standard. It is important to unveil the differences in severity distributions between CVSSv2 and CVSSv3, which help practitioners transit their existing practices (e.g., the scope of CSV). We show in Section 4.1 that CVSSv3 generally assigns higher severity scores/ratings to SVs compared with CVSSv2. The severity scores of 94.4% of SVs increase from CVSSv2 to CVSSv3 with a median increase of 2.2. Under CVSSv2, a common practice of security SLA adopted by the software industry is to fix SVs of 7+ score within a reasonable time frame [35, 46]. When transitioning to CVSSv3, a similar goal would be fixing SVs of 9+ score. We have shown that these two CSV proxies are generally equivalent regarding both the proportions and the CWE categories (see Section 4.2).

Pay special attention to certain CWEs with high proportions of CSVs. We show in Section 4.2 that certain CWE categories have a higher proportion of CSVs (e.g., 69%), and the top 22 categories (less than 10% of the total categories) cover over 80% of CSVs. The practitioners should pay special attention to these CWE categories due to their tight connections with CSVs, e.g., building automatic techniques dedicated to these SV types. In recent years, automatic techniques (e.g., SV detection [53, 56, 71, 74], repair [64, 66]) have been proposed to promote the SV remediation practice. However, their performance is not yet satisfied in a practical scenario [53, 72]. One of the challenges faced by these data-driven models is the scattered and insufficient SV knowledge (i.e., the SV data is limited while spanning across various types) [71]. We argue that it is worth trying to limit the scope of automatic techniques to certain key SV types. Moreover, we show that practitioners should not only focus on common and obvious SV types, as CWE categories with more SVs are not necessarily those have more CSVs. For example, the MITRE CWE Top 25 list [16], a commonly adopted reference for *hot spot* SVs in practice, misses certain CWE categories with a high proportion of CSVs (e.g., CWE-1321). We argue that SVs of these CWE categories are also worth the attention from practitioners since once these SVs are detected, they are likely to be critical.

Different disclosure practices regarding CSVs among the repositories. In Section 4.3, we find that a considerable amount of repositories adopt the practice of only disclosing CSVs. Thus, practitioners should attach higher priorities to newly disclosed SVs associated with these repositories, as such SVs are likely to be critical. Besides, there are also some repositories that intentionally avoid disclosing any CSVs (e.g., due to reputation concerns [73]). For these repositories, end users may need to keep an eye on the new releases of these repositories and update the package in time (since CSVs may be secretly patched in the new releases).

Better prioritizing CSV remediation in practice. Given the importance of CSV, it naturally attracts more attention from practitioners. We show in Section 5 that CSVs generally gain higher priorities than NCSVs (e.g., more timely response). However, the extra attention also leads to a more complicated discussion, review, and testing process. This could unintentionally increase security risks given the transparency of OSS, i.e., delay in remediation expands the *window-of-opportunity* as attackers can leverage the SV-relevant information from the development activities to conduct zero-day attacks. For example, we discuss in Section 5.2 that although the patches of CSVs are developed in a more timely manner, the discussions and review processes of the corresponding pull request (PR) may unintentionally cause the delay to its merge. We suggest maintainers to balance the remediation quality with speed, especially for CSVs. A temporary or intermediary mitigation can be necessary to ensure a quick fix of CSV [36]. Another aspect to reduce the risk is to hide the relevant development activities from the public channel (e.g., coordinate with a private PR).

Another observation that raised our attention is that CSVs have a longer fix-to-disclosure delay compared to NCSVs (see Section 5.4). One possible reason is that maintainers may intentionally delay the CSV disclosure to allow more time for security releases to be sufficiently installed [34]. However, some works suggest that the security release is unlikely to be widely installed before public disclosure since developers typically do not proactively update the software unless they are aware of a critical change [63]. Thus, delaying the disclosure may actually turn out to be risk-increasing, as it expands the exploit window for attackers. We encourage future works to explore the best practice for promptly notifying valued project users of security releases while avoiding direct public disclosure of SV details.

Risks in the current remediation practice. Given the public nature of OSS, development activities related to the remediation are visible to attackers. This creates a *window of opportunity* for attackers, as they may take advantage of the leaked SV information to conduct zero-day attacks. We comprehensively review the possible risks throughout the remediation process in Section 5. We show that the *window of opportunity* for attackers (i.e., the risk of SV information leakage) can start at the very beginning of the remediation (i.e., IR reporting the SV), and lasts over 30 days for over half of the SVs. As the remediation process progresses, more SV information is leaked (e.g., commits tell the exact vulnerable code lines), which further eases the difficulties for attackers to develop a functional exploit. Our findings suggest a general lack of awareness regarding risks of SV information leakage during the remediation process in the OSS community. OSS maintainers should

better manage the remediation activities to avoid the leakage of SV information before public disclosure, e.g., conduct coordination within a private channel. Besides, our findings also suggest a need for OSS users, especially the software vendors, to timely sense the SV information by monitoring the OSS development activities (e.g., IR reporting the SV, patch committed to the codebase) instead of waiting for public disclosure. So that the OSS users can start the remediation earlier and thus avoid falling into a disadvantage in the race against potential attackers.

More regulations are required to avoid disclosing SVs without noticing or leaving time for maintainers to remediate. 7.7% of SVs are patched after disclosure. Among them, 36.2% CSVs have no patch available until 30 days later. Besides, we also call for practitioners' attentions on patch reliability, as even for CSVs, there are nearly 10% of patches are problematic when first introduced and require over 11 days on median to be resolved (see Section 5.5).

7 THREATS TO VALIDITY

Threats to internal validity relate to the experiment bias and errors. One threat is the completeness of the dataset collected in our study. The collected SV-associated development artifacts (e.g., IRs, PRs, and commits) might be incomplete, since the external references listed by NVD are not meant to be complete. To enhance the completeness, we leverage another popular SV database (i.e., OSV) to complete GitHub links. We further conduct an extensive crawling process to complete the SV-associated artifacts by using the listed GitHub artifacts as seeds and retrieve relevant development artifacts (see Section 3.2). Another threat is that some collected development artifacts may not relate to the SV remediation process, e.g., artifacts from the bug hunter's own repository for documenting the discovered SV. We check each retrieved artifacts with several heuristics (e.g., a blacklist of common repositories specially used to document the discovered SVs, ensuring the artifact belong to the same repository with the reported SV) to filter out noises. Therefore, we believe the quality of our dataset is reliable.

Threats to external validity relate to the generalizability of our study. The main threat is that we only consider OSS projects hosted on GitHub. OSS projects hosted on other platforms might have different workflows for SV remediation (e.g., do not use PR-based development model in collaboration). So, the analysis approach and findings in our study might not be applicable to them. However, since GitHub is the most popular OSS platform and the SV remediation process analyzed in our study is representative, we believe our findings can provide useful implications for SV remediation.

8 RELATED WORK

Empirical Studies on SV. Many empirical studies have analyzed OSS SVs from different aspects [51, 58, 59, 63, 65]. However, they typically target at general SVs, missing a view of the unique characteristics of CSVs. Alfadel *et al.* [51] investigate the distribution and life span of 550 SVs within the Python ecosystems. Liu *et al.* [65] analyze the SV distribution within five representative OSS projects over four dimensions (e.g., files, SV types). Li and Paxson [63] analyze the patch development process and characterize security patches by comparing with non-security bug fixes using patches of over 3,000 SVs affecting 682 OSS projects. Imtiaz *et al.* [59] investigate the practice regarding security release of OSS packages using 4,377 SVs across seven package ecosystems. Iannone *et al.* [58] analyze the contributing and fixing commits of 3,663 SVs from 1,096 GitHub projects to understand how (e.g., commit goal, developer experience and workloads) SVs are introduced and removed in OSS codebases. Different from previous works, 1) We are the first to specifically investigate the characteristics of CSV and highlight its differences against NCSV. 2) We comprehensively investigate the timeline of SV remediation (see Figure 1) and identify potential shortcomings. Existing works [51, 58, 63] only investigate the general SV life span (i.e., the gap between SV being introduced and removed in the codebase) without a focus on the remediation process, or only discuss the gap between SV fix and disclosure [59]. 3) Our collected dataset (14,867 SVs across 4,462 GitHub projects) is much larger than those used in the existing studies (ranging from 550 to 4,377 SVs), and contains more comprehensive relevant artifacts (e.g., IR, PR) to facilitate researches regarding SV remediation practice.

SV Assessment and Prioritization. Our focus regarding CSV is closely related to the SV assessment [54, 55, 60], an important phase in the SV management life cycle. Most of the existing studies focus on the technical aspect to automate SV assessment and prioritization by predicting CVSS metrics using data-driven approaches [60–62, 70]. However, according to the literature review conducted by Le *et al.* [60], the performance of these approaches are not yet satisfying. We take a first step to empirically investigate CSVs, which provides insights of CSV characteristics and helps to guide the automation of SV assessment. Besides, while SV assessment focuses on the intrinsic characteristics (e.g., exploitability, impact) of specific SVs [14], we provide a high-level illustration of CSVs, i.e., their general distribution and remediation practices, which is essential for practitioners to devise mitigation strategies.

9 CONCLUSION AND FUTURE WORK

In this work, we take the first look at the distribution and remediation practice of CSVs, and highlight their differences against NCSVs. We build a dataset containing 14,867 SVs from NVD and their associated software artifacts (i.e., IR, PR, and commit) across 4,462 GitHub repositories. We find that CSV distribution varies across CWE categories and GitHub repositories. Regarding the remediation practice, we point out that the leakage of SV information and delay in disclosure can pose huge risks. Moreover, we find that though CSVs receive higher remediation priorities (i.e., responded and patched more quickly), their patches may not appear in the codebase timely due to complicated review or testing process. Also, maintainers tend to delay the disclosure of CSVs, which may turn out to be risk-increasing. Besides, the irresponsible disclosure and patch reliability are also of serious concerns. In the future, we plan to leverage the insights from our findings to build more accurate automatic techniques for locating and patching the CSVs.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments to improve the paper. This research/project is supported by the National Key Research and Development Program of China (No. 2021YFB2701102), the National Natural Science Foundation of China (No.62372398, No.7234202, and U20A20173), the Ningbo Natural Science Foundation (No. 2023J292), and the Fundamental Research Funds for the Central Universities (No. 226-2022-00064).

REFERENCES

[1] 2019. Equifax to Pay at Least $650 Million in Largest-Ever Data Breach Settlement. https://www.nytimes.com/2019/07/22/business/equifax-settlement.html.
[2] 2022. GitHub | Open Source Supply Chain Security. https://github.blog/2022-11-09-improving-open-source-supply-chain-security/.
[3] 2022. GitHub Octoverse 2022: 10 years of tracking open source. https://github.blog/2022-11-17-octoverse-2022-10-years-of-tracking-open-source/.
[4] 2023. BOD 19-02: Vulnerability Remediation Requirements for Internet-Accessible Systems | CISA. https://www.cisa.gov/news-events/directives/bod-19-02-vulnerability-remediation-requirements-internet-accessible-systems.
[5] 2023. Common Vulnerabilities and Exposure. https://www.cve.org/.
[6] 2023. Common Vulnerability Scoring System. https://www.first.org/cvss/.
[7] 2023. Common Vulnerability Scoring System - Wikipedia. https://en.wikipedia.org/wiki/Common_Vulnerability_Scoring_System.
[8] 2023. Common Weakness Enumeration. https://cwe.mitre.org/index.html.
[9] 2023. CVE-2017-5638. https://nvd.nist.gov/vuln/detail/CVE-2017-5638.
[10] 2023. CVE-2021-33823. https://nvd.nist.gov/vuln/detail/CVE-2021-33823.
[11] 2023. CVE-2021-36081. https://nvd.nist.gov/vuln/detail/CVE-2021-36081.
[12] 2023. CVE-2021-442288. https://nvd.nist.gov/vuln/detail/CVE-2021-442288.
[13] 2023. CVE-2022-31836. https://nvd.nist.gov/vuln/detail/CVE-2022-31836.
[14] 2023. CVSS 3.1 Specification Document. https://www.first.org/cvss/v3.1/specification-document.
[15] 2023. CWE-1321: Improperly Controlled Modification of Object Prototype Attributes (Prototype Pollution). https://cwe.mitre.org/data/definitions/1321.html.
[16] 2023. CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html.
[17] 2023. EU Cloud Certification Scheme. https://ec.europa.eu/newsroom/cipr/items/713799/en.
[18] 2023. GitHub Docs - Autolinked references and URLs. https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/autolinked-references-and-urls.
[19] 2023. GitHub GraphQL API. https://docs.github.com/en/graphql.
[20] 2023. GitHub GraphQL API - securityadvisory. https://docs.github.com/en/graphql/reference/objects#securityadvisory.
[21] 2023. GitHub Issue Report: beego/beego issue #4961. https://github.com/beego/beego/issues/4961.
[22] 2023. GitHub Issue Report: blackears/svgSalamander issue #11. https://github.com/blackears/svgSalamander/issues/11.
[23] 2023. GitHub Issue Report: blackears/svgSalamander issue #33. https://github.com/blackears/svgSalamander/pull/33.
[24] 2023. GitHub Issue Report: blackears/svgSalamander pull #12. https://github.com/blackears/svgSalamander/pull/12.
[25] 2023. GitHub Pull Request: actix/actix-net pull #113. https://github.com/actix/actix-net/pull/113.
[26] 2023. GitHub Pull Request: actix/actix-net pull #158. https://github.com/actix/actix-net/pull/158.
[27] 2023. GitHub Pull Request: ckeditor/ckeditor4 pull #49. https://github.com/ckeditor/ckeditor4/pull/49.
[28] 2023. GitHub Pull Request: libreoffice/core pull #2924. https://github.com/pjsip/pjproject/pull/2924.
[29] 2023. GitHub Pull Request: OpenRC/openrc pull #462. https://github.com/OpenRC/openrc/pull/462.
[30] 2023. GitHub Repository: facebook/folly. https://github.com/facebook/folly.
[31] 2023. GitHub Repository: tesseract-ocr/tesseract. https://github.com/tesseract-ocr/tesseract.
[32] 2023. GitHub REST API. https://docs.github.com/en/rest?apiVersion=2022-11-28.
[33] 2023. GitHub REST API - IR timeline. https://docs.github.com/en/rest/issues/timeline?apiVersion=2022-11-28.
[34] 2023. Google Project Zero. https://googleprojectzero.blogspot.com/2021/04/policy-and-disclosure-2021-edition.html.
[35] 2023. Introducing SLAs for Vulnerability Management. https://web.archive.org/web/20230327180000/https:/www.kennasecurity.com/blog/vulnerability-management-sla/.
[36] 2023. ISO/IEC 30111:2019 Information technology Security techniques Vulnerability handling processes. https://www.iso.org/standard/69725.html.
[37] 2023. National Vulnerability Database. https://nvd.nist.gov/.
[38] 2023. NVD - CVSS v3.1 Official Support. https://nvd.nist.gov/general/News/CVSS-v3-1-Official-Support.
[39] 2023. NVD - Retirement of CVSS v2. https://nvd.nist.gov/general/news/retire-cvss-v2.
[40] 2023. Organizations Fix Only 1 in 10 Vulnerabilities Monthly. https://www.msspalert.com/news/organizations-fix-only-1-in-10-vulnerabilities-monthly.
[41] 2023. OSV Vulnerability Database. https://osv.dev/.
[42] 2023. Riot/Issue-10753. https://github.com/riot-os/riot/issues/10753.
[43] 2023. Security Policy: facebook/folly. https://github.com/facebook/folly/security/policy.
[44] 2023. Synk | 2023 Software Supply Chain Attack Report. https://go.snyk.io/2023-supply-chain-attacks-report.html.
[45] 2023. Synopsys 2023 Open Source Security and Risk Analysis Report. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html.
[46] 2023. Using SLAs for Better Vulnerability Management & Remediation: Improving Developer's Workflow - Phoenix Security. https://phoenix.security/using-slas-for-better-vulnerability-management-remediation-improving-developers-workflow/.
[47] 2023. Vulnerability DB | Synk. https://security.snyk.io/vuln.
[48] 2023. Vulnerability Management Overview. https://handbook.gitlab.com/handbook/security/threat-management/vulnerability-management/.
[49] 2024. The Collected Dataset. https://doi.org/10.5281/zenodo.11179523.
[50] 2024. Vulnerability Management SLAs Guide. https://hostedscan.com/blog/vulnerability-management-slas-guide.
[51] Mahmoud Alfadel, Diego Elias Costa, and Emad Shihab. 2021. Empirical analysis of security vulnerabilities in python packages. In *2021 IEEE international conference on software analysis, Evolution and Reengineering (SANER)*. IEEE, 446–457.
[52] 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.
[53] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2021. Deep learning based vulnerability detection: Are we there yet. *IEEE Transactions on Software Engineering* (2021).
[54] Nesara Dissanayake, Asangi Jayatilaka, Mansooreh Zahedi, and Muhammad Ali Babar. 2022. An Empirical Study of Automation in Software Security Patch Management. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.
[55] Park Foreman. 2019. *Vulnerability management*. CRC Press.
[56] Michael Fu and Chakkrit Tantithamthavorn. 2022. LineVul: a transformer-based line-level vulnerability prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 608–620.
[57] Ahmed E Hassan. 2009. Predicting faults using the complexity of code changes. In *2009 IEEE 31st international conference on software engineering*. IEEE, 78–88.
[58] Emanuele Iannone, Roberta Guadagni, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba. 2022. The secret life of software vulnerabilities: A large-scale empirical study. *IEEE Transactions on Software Engineering* 49, 1 (2022), 44–63.
[59] Nasif Imtiaz, Aniqa Khanom, and Laurie Williams. 2022. Open or Sneaky? Fast or Slow? Light or Heavy?: Investigating Security Releases of Open Source Packages. *IEEE Transactions on Software Engineering* (2022).
[60] Triet HM Le, Huaming Chen, and M Ali Babar. 2022. A survey on data-driven software vulnerability assessment and prioritization. *Comput. Surveys* 55, 5 (2022), 1–39.
[61] Triet Huynh Minh Le and M Ali Babar. 2022. On the use of fine-grained vulnerable code statements for software vulnerability assessment models. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 621–633.
[62] Triet Huynh Minh Le, David Hin, Roland Croft, and M Ali Babar. 2021. Deepcva: Automated commit-level vulnerability assessment with deep multi-task learning. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 717–729.
[63] 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.
[64] Yi Li, Shaohua Wang, and Tien N Nguyen. 2022. Dear: A novel deep learning-based approach for automated program repair. In *Proceedings of the 44th International Conference on Software Engineering*. 511–523.
[65] Bingchang Liu, Guozhu Meng, Wei Zou, Qi Gong, Feng Li, Min Lin, Dandan Sun, Wei Huo, and Chao Zhang. 2020. A large-scale empirical study on vulnerability distribution within projects and the lessons learned. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1547–1559.
[66] Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F Bissyandé. 2019. TBar: Revisiting template-based automated program repair. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 31–42.
[67] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.
[68] 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 (DIMVA)*. Springer, 23–43.
[69] Shengyi Pan, Lingfeng Bao, Xin Xia, David Lo, and Shanping Li. 2023. Fine-grained commit-level vulnerability type prediction by CWE tree structure. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 957–969.
[70] Shengyi Pan, Lingfeng Bao, Jiayuan Zhou, Xing Hu, Xin Xia, and Shanping Li. 2024. Towards More Practical Automation of Vulnerability Assessment. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.
[71] Shengyi Pan, Jiayuan Zhou, Filipe Roseiro Cogo, Xin Xia, Lingfeng Bao, Xing Hu, Shanping Li, and Ahmed 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.

[72] Arthur D Sawadogo, Quentin Guimard, Tegawendé F Bissyandé, Abdoul Kader Kaboré, Jacques Klein, and Naouel Moha. 2021. Early Detection of Security-Relevant Bug Reports using Machine Learning: How Far Are We? *arXiv preprint arXiv:2112.10123* (2021).

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

[74] Yueming Wu, Deqing Zou, Shihan Dou, Wei Yang, Duo Xu, and Hai Jin. 2022. VulCNN: an image-inspired scalable vulnerability detection system. In *Proceedings of the 44th International Conference on Software Engineering*. 2365–2376.

[75] Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E. Hassan. 2021. Finding A Needle in a Haystack: Automated Mining of Silent Vulnerability Fixes. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 705–716. https://doi.org/10.1109/ASE51524.2021.9678720

[76] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small world with high risks: A study of security threats in the npm ecosystem. In *28th USENIX Security Symposium (USENIX Security 19)*. 995–1010.

Received 2024-02-08; accepted 2024-04-18