Empirical Software Engineering (2023) 28:47
https://doi.org/10.1007/s10664-022-10267-7

Vulnerability management in Linux distributions

An empirical study on Debian and Fedora

Jiahuei Lin[1] · **Haoxiang Zhang**[1] · **Bram Adams**[2] · **Ahmed E. Hassan**[3]

Accepted: 25 November 2022 / Published online: 16 February 2023

Abstract

Vulnerabilities in software systems not only lead to loss of revenue, but also to loss of reputation and trust. To avoid this, software providers strive to remedy vulnerabilities rapidly for their customers. However, in open-source development, the providers do not always control the distribution of their software themselves, but instead typically rely on Linux distributions to integrate and distribute upstream projects to millions of end users, which increases the difficulty of vulnerability management. In addition, an upstream project is usually packaged into several Linux distributions so that a vulnerability can propagate across multiple distributions via the upstream project. In this work, we empirically investigate a large number of vulnerabilities registered with the Common Vulnerabilities and Exposures (CVE) program in two popular Linux distributions, i.e., Debian (21,752 CVE-IDs) and Fedora (17,434 CVE-IDs), to study the practices of vulnerability management in such ecosystems. We investigate the lifecycle of fixing vulnerabilities, analyze how fast it takes for a vulnerability to go through each phase of its lifecycle, characterize the commonly occurring vulnerabilities that affect both distributions, and identify the practices that developers use to fix vulnerabilities. Our results suggest that the vulnerability testing period (i.e., the period from when the vulnerability fix is committed for testing to when the vulnerability fix is released) accounts for the largest number of days (median of 15 days) in Fedora. 74% (i.e., 16,070) and 92% (i.e., 16,070) of the vulnerabilities in Debian and Fedora, respectively, occur in both Linux distributions, which we refer to as common security vulnerabilities (CSVs). This result is impacted by the package selection and customization of the distributions. Finally, on a representative sample of 345 fixed CSVs, we find that upstream projects were responsible for fixing 303 (85%) and 267 (76%) out of the 345 CSVs in Debian and Fedora, respectively, with distribution maintainers integrating those fixes. Our work aims to gain a deeper understanding of the current practices in the vulnerability management of Linux distributions, and propose suggestions to distribution maintainers for better mitigation of the risks of vulnerabilities.

Communicated by: Jacques Klein

✉ Haoxiang Zhang
haoxiang.zhang@acm.org

Extended author information available on the last page of the article.

Keywords Open-source software ecosystems · Vulnerability management · Developer coordination · Linux distributions

1 Introduction

Attackers seek flaws or weaknesses in software systems as a weapon, also known as security vulnerabilities, and exploit them to attack or steal information from the victim systems. Given the increase of malicious attacks on software systems, developers' main strategy to reduce the exploit of vulnerable code is to try to update their software as quickly as possible with patches (fixes) for these vulnerabilities. In particular, each vulnerability has its lifecycle that consists of several phases defined by the events of its discovery, disclosure, exploitation, and patching (Shahzad et al. 2012), as shown in Fig. 1. The highest risk in a vulnerability lifecycle happens in the phase after its disclosure (Shahzad et al. 2012) as miscreants are actively trying to make *zero-day* attacks (Anderson 2002). Sometimes the exploitation occurs even prior to the disclosure, when a vulnerability is discovered by a Black-Hat hacker who does not report the vulnerability to the affected vendors. The lifecycle of the vulnerability ends when all victim systems have adopted the fix patch (Shahzad et al. 2012).

Ironically, in today's open-source software (OSS) ecosystems, the patch adoption phase is not as trivial as it sounds, since a software project can be deployed and used directly or be integrated into a larger system, creating a system of systems, which forms a software supply chain (Ellison et al. 2010; Al Sabbagh and Kowalski 2015). In other words, a software product is assembled by integrating smaller software components that are developed either by developers inside or outside of the product team. Therefore, the vulnerability could propagate through OSS projects across OSS ecosystems (Ohm et al. 2020), and so must any patch for it. To this end, once a vulnerability is identified, miscreants can exploit it to attack several victim software systems at the same time.

In open-source software ecosystems, due to the emergence of software supply chains and bug bounty programs, fixing a vulnerability may require the involvement of multiple parties associated with the vulnerability, rather than just the entity who detected a vulnerability and the impacted provider. The involvement of multiple parties complicates the process of fixing vulnerabilities and increases coordination and disclosure challenges. First.org is one of the leading organizations in incident response and includes a variety of computer security incident teams from governments, corporations, and research organizations. To deal with the challenges of patching vulnerabilities in an ecosystem, first.org suggests practices for multi-party vulnerability coordination and disclosure (Guidelines and practices for multi-party vulnerability coordination and disclosure online). These practices indicate that a vulnerability should be reported to the upstream project responsible for developing the

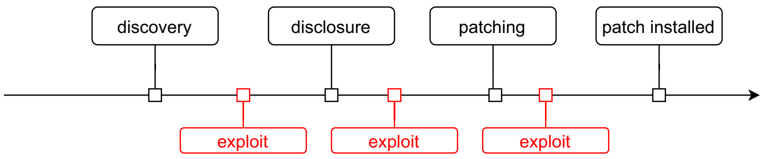


Fig. 1 An illustration of events in a vulnerability lifecycle started by the initial discovery of a vulnerability. Note that exploits can happen at any time during the lifecycle

vulnerability fix, which should then propagate the fix to downstream ecosystems. The reason for this is that developers in the upstream projects are familiar with their own projects and are experts of the vulnerable source code (Ma et al. 2017), hence they should be able to fix the vulnerability in a timely manner to reduce the exposure time of the vulnerability.

While a vulnerability fix is more likely to happen upstream, downstream ecosystems or their users could also come up with fixes themselves. For example, vulnerability CVE-2014-3207 was fixed by an upstream developer who sent a fix to the oss-security community by email.[1] In order to streamline such early detection and distribution of available fixes and workarounds downstream, several distributions in the Linux ecosystem, including Debian and Fedora, share a mailing list for handling security issues (e.g., reports, discussions, notifications, *etc*.) (Operating system distribution security contact lists online).

Prior work investigated the lifecycle of a vulnerability in software projects (Shahzad et al. 2012; Frei et al. 2006; Frei et al. 2008). For example, (Shahzad et al. 2012) leveraged public documentation from vulnerability databases to analyze the evolution of disclosure, exploitation, and patching behaviors. Prior studies also surveyed the patch deployment process, such as the timelines of vulnerability fix patches (Li and Paxson 2017), the adoption speed of patches from users (Nappa et al. 2015), and the reliability of vulnerability fix patches (Huang et al. 2016). However, prior work did not investigate how developers work together to fix vulnerabilities across OSS ecosystems with multi-party coordination.

In this paper, we identify the vulnerability fixing process and divide it into three phases from the maintainer perspective (i.e., awareness, vulnerability fix integration, and vulnerability fix testing) and two phases from the user perspective (i.e., user unawareness and users waiting for a fix) for two popular Linux distributions (i.e., Debian and Fedora). In particular, we study 21,752 and 17,434 security vulnerabilities, i.e., unique CVE-IDs, in Debian and Fedora, respectively, to compare the duration of each phase in the process from both perspectives across the studied distributions. We also investigate the strategies that maintainers use to fix the common vulnerabilities across the studied distributions. In the following, we list our research questions along with the key results:

RQ1: ***How fast are security vulnerabilities fixed within the studied distributions?***
This first RQ analyzes to what extent distribution maintainers react to vulnerabilities and how fast users get the information (e.g., fixes, advisories) related to vulnerabilities in the vulnerability fixing process in Linux distributions. In general, maintainers in Debian and Fedora take a median of 20 and 27 days to fix vulnerabilities, respectively, from when the security team creates a vulnerability report to when the fix is released. This time includes the time taken by Debian and Fedora maintainers to integrate vulnerability fixes, which happens within a median of 1 week. From the user perspective, users are unaware of any existing vulnerability for a median of 17 days, then wait for a median of 0.2 days to get vulnerability fixes.

RQ2: ***What are the characteristics of common security vulnerabilities (CSVs) across the studied distributions?***
Since a vulnerability can affect several distributions at the same time, the second RQ aims to characterize security vulnerabilities that have been reported in both studied distributions (CSVs). CSVs make up 74% of Debian vulnerability reports, and 92% of Fedora's. We find that the CSVs are easy to exploit (median exploitability score of 8.6), though their severity level is medium (median base score of 5.0), based

[1] https://seclists.org/oss-sec/2014/q2/225

on the Common Vulnerability Scoring System CVSS 2.0. Testing fixes for CSVs requires a median of 4.3 days longer than integrating fixes for CSVs.

RQ3: How do developers fix common security vulnerabilities (CSVs) across the studied distributions?

Given the prevalence of CSVs, this RQ investigates the collaborative strategies of fixing CSVs across distributions by a qualitative study on a representative random sample of 345 fixed CSVs (confidence level = 95%, confidence interval = 5%) and several quantitative analyses. From the qualitative study, we found that distribution maintainers have five main types of sources from which they become aware of vulnerabilities, i.e., upstream, Debian, Fedora, upstream projects, or others, and seven main types of strategies for fixing the identified vulnerabilities, such as in-house vs. external fix development or the usage of embargoes. Furthermore, upstream projects were found to develop vulnerability fixes for the majority of CSVs in Debian (58%±5%) and Fedora (59%±5%), and to distribute the fixes to downstream distributions. Ironically, from the quantitative analyses, the availability of a peer fix before maintainers start to analyze the vulnerability does not correlate with faster integration of the fix.

Maintainers in the two studied distributions dedicate substantial effort to securing their distributions, such as integrating vulnerability fixes in a median of one week after vulnerabilities are reported. Our results suggest that the studied distributions collaborate with each other and with upstream after the discovery of vulnerabilities, while they integrate and test vulnerability fixes in parallel.

Paper organization Section 2 provides the background of vulnerability management process in Linux. Section 3 describes prior related work to our study. Section 4 describes how we design our study and obtain our studied dataset. Section 5 answers our research questions. Section 6 discusses our findings and their implications. Section 7 discusses the threats to the validity of our study. Finally, Section 8 concludes the paper.

2 Background

In this section, we briefly introduce the Common Vulnerabilities and Exposures (CVE) system and Common Vulnerability Scoring System (CVSS) as well as the vulnerability management process in Linux distributions.

2.1 Common Vulnerabilities and Exposures (CVE) and Common Vulnerability Scoring System (CVSS)

The Common Vulnerabilities and Exposures (CVE) (CVE online) platform is one of the most popular platforms for vulnerability registration. The CVE system stores a large number of common software and hardware vulnerabilities and reports. More than 100 major software providers, security companies and research organizations form the CVE Numbering Authorities (CNAs), which are responsible for the vulnerability disclosure process. When a vulnerability is discovered, a unique identifier, i.e., CVE-ID, is issued by one of the CNAs for the vulnerability. Information about the vulnerability, such as the description, symptoms and references, are then posted on the CVE website as a CVE report. In general, a CVE-ID is usually assigned several days up to months before its mitigation (e.g., fix, security advisories) is made public.

The Common Vulnerability Scoring System (CVSS) (US national institute of standards and technology online) is an open standard for assigning scores to a vulnerability to indicate its threat level. CVSS scores are widely used by many prior studies (Joh and Malaiya 2011; Fruhwirth and Mannisto 2009; Shahzad et al. 2012; Zhang et al. 2021) and large organizations and companies (e.g., Computer Emergency Response Team (CERT), Cisco) to communicate the severity of the vulnerabilities found in their products. A CVSS score consists of scores in three groups: base, temporal and environmental, each group consisting of a set of metrics. The base score represents the innate characteristics (e.g., access complexity, exploitability) of a vulnerability to indicate its severity. For instance, a base score in the range of 4.0 to 6.9 indicates medium severity. The base score of a vulnerability is computed by six metrics grouped into two aspects: exploitability, i.e., how the vulnerability is exploited, and impact, i.e., the extent of victim system losses if the vulnerability is exploited.

The temporal and environmental scores capture dynamics and subjective information. The temporal score represents the characteristics of a vulnerability that change over time, such as the availability of exploit code and techniques, disclosed remediation plans of a vulnerability, and confirmation of a vulnerability. The environmental score represents the characteristics of a vulnerability that are relevant to a particular environment, such as damage or potential losses to productivity, the proportion of vulnerable systems, and the extent of impact. Note that we use CVSS 2.0 as CVSS 3.0 (released in 2015) and 3.1 (released in 2019) are not available for the majority of our data. Our dataset consists of vulnerabilities whose CVE-ID was issued between Jan. 2007 and Dec. 2019 (see Section 4).

2.2 Vulnerability Management Process in Linux Distributions

Vulnerability management plays an important role in Linux distributions because they have a large user base, including individuals who use them on desktops and companies who use them on servers. In a Linux distribution, security vulnerabilities are addressed by the collaboration between the security team, maintainers who integrate and maintain packages from third-party providers, i.e., upstream projects, and upstream developers. The role of the security team is to track vulnerabilities that affect the packages in their distribution.

Figure 2 represents the typical process of vulnerability tracing in the studied Linux distributions. We derive the process from the official documentation in Debian (Debian security team online) and Fedora (Fedora - security basics online) by an open-coding approach. The 1st author derived the initial version of the process, discussed this version with the 2nd and 3rd authors, then iteratively resolved any disagreements. We identified the process from two perspectives, i.e., **maintainers** and **users**, and separated the process into three and two phases, respectively. Distribution maintainers are developers who integrate upstream projects into the distribution following the distribution's policy, and maintain the integrated upstream projects (e.g., synchronize fixes from upstream, forward bugs to upstream) for end users. Users are stakeholders interested in the management of security vulnerabilities by a distribution. They might be technical experts or system administrators in companies, security specialists, developers in peer distributions, end users of the distributions, etc.

From the maintainer perspective, the three phases consist of awareness of vulnerabilities, integration of vulnerability fixes and testing of vulnerability fixes. The awareness phase

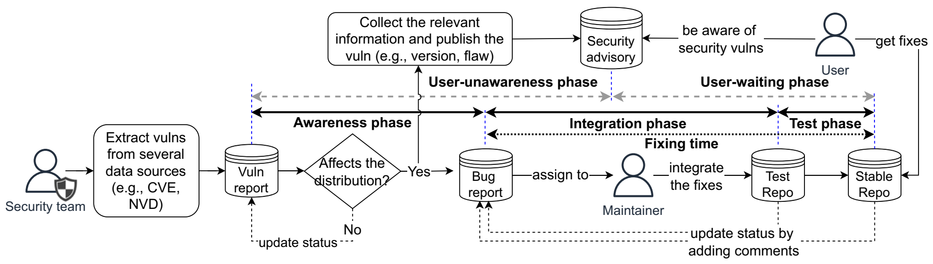


Fig. 2 Overview of vulnerability management in a given distribution. The black solid and grey dashed arrow lines represent the periods of the five phases of tracking a vulnerability from the maintainer and user perspectives, respectively. To fix a vulnerability in the distribution, the required time (i.e., the fixing time) is the sum of the integration and test periods (the black dotted arrow line). Note that the disclosure time of security advisories for end users (i.e., when the user unawareness phase ends) could happen at any time during the maintainer awareness/integration/test periods

starts when a distribution is made aware of a vulnerability through a report in the vulnerability repository, after which the security team creates corresponding bug reports for the vulnerability, one per affected package. Note that a vulnerability either can be found by the security team itself or the security team gets a notification of the vulnerability from somewhere else. Since the awareness phase represents the time taken by the security team to evaluate the extent to which a vulnerability affects the team's distribution, a smaller awareness period implies that the information regarding the vulnerability is sufficient for the security team to more quickly investigate the packages affected by the vulnerability, or the security team gets access more quickly to an existing investigation report, streamlining the start of vulnerability analysis.

During the awareness phase, when a vulnerability is discovered without a fix, the security team can choose to embargo the vulnerability to reduce its impact on the distribution. Embargo of a vulnerability means that the vulnerability will not be disclosed for a period of time to allow the affected software providers to develop a fix. When a vulnerability is under embargo, a small group of people (e.g., the reporter, the developers of the affected software) coordinate to develop and test a fix together. In our studied distributions, Debian avoids embargoes "*since coordination in private tends to cause a lot of friction and makes it difficult to involve the right subject matter experts,*" (Debian vulnerability disclosure policy online) while Fedora applies the embargo policy of Red Hat that keeps vulnerabilities private for a certain period (Fedora - security bugs online).

The integration phase is calculated from the creation time of a bug report for the vulnerability to when a vulnerability fix has been integrated and pushed into the test repository. Since the vulnerability may affect several distributions or other software systems, the vulnerability fix can be developed by a wide range of involved people, e.g., upstream developers, and maintainers, security experts. The integration period also allows maintainers to integrate a vulnerability fix that is compatible with the other packages in their distribution (Adams et al. 2016; Foundjem and Adams 2021).

Lastly, the vulnerability fix needs to be tested before it is released into the stable repository, which we refer to as the vulnerability test phase.

The two phases from the user perspective are unawareness of vulnerabilities and waiting for vulnerability fixes. The security team in a distribution publishes a security advisory that describes the related information (e.g., the names of the affected packages, how to fix them) for a particular vulnerability to users. In our studied distributions, security advisories are usually posted in a particular mailing list where users get proactive notifications by subscription (Debian security faq onlinea; Fedora - security basics online). We define the period before a security advisory is published as the user-unawareness period and the period after that as the user-waiting period. The user-waiting period represents the time for users to get a vulnerability fix, once they know that the vulnerability exists.

3 Related Work

We discuss prior empirical studies on (1) vulnerability life cycles and (2) vulnerability fix development.

3.1 Vulnerability Lifecycles

The vulnerability lifecycle describes the events that characterize the period of a vulnerability from its discovery until all victim systems have adopted a fix patch (Joh and Malaiya 2011). Most of the existing research has focused on one or several particular events of the vulnerability lifecycles. For example, *zero-day* attacks, the damage caused by exploiting unpatched vulnerabilities, are believed to represent a large amount of targeted attacks (Anderson 2002; Bilge and Dumitraş 2012; Wang et al. 2019). Algarni and Malaiya (2014) identified several vulnerability markets where discoverers offer the undisclosed vulnerabilities to buyers for a monetary reward. Joh and Malaiya (2011) formalize the risk measurement in each stage of a vulnerability lifecycle by a stochastic model based on CVSS scores to help project managers make decisions about whether to apply the patches. Different from prior studies that focus on particular events in the vulnerability lifecycle of individual projects, we focus our study on the whole fixing process from the user and maintainer perspectives in the Linux ecosystem.

Prior studies also investigated the evolution of the events (e.g., how long after its discovery is a vulnerability disclosed) in a vulnerability lifecycle. Frei et al. (2006) aggregated data from several public data sources (e.g., the National Vulnerability Database (NVD) (National vulnerability database online)) to analyze the distributions of the discovery, exploit, and patch-availability date with respect to the disclosure date of a vulnerability. For example, they found that the trend of zero-day attacks has increased. Shahzad et al. (2012) extended Frei et al.'s (2006) work and investigated the evolution of events (e.g., disclosure trend, exploitation behaviors) in the vulnerability life cycle from three public data sources (i.e., NVD (National vulnerability database online), open-source vulnerability database (OSVDB) and the data source from (Frei et al. 2006)).

However, the public data sources have been reported to have an undetectable bias due to the various disclosure methods from different software providers (Schryen 2009; Christey and Martin 2013). For example, the two studied distributions disclose vulnerabilities by publishing security advisories (Debian vulnerability disclosure policy online; Fedora - security bugs online). To avoid this problem, our work extracts multiple types of data (i.e., vulnerability reports, bug reports and security advisories) from a particular software ecosystem and aggregates these data by CVE identifiers. In addition, we derive the lifecycle of a vulnerability in a particular software ecosystem instead of individual software projects.

3.2 Vulnerability Fix Development

Adopting security patches is a common practice to prevent the present vulnerabilities from being exploited. This practice involves a number of processes, such as impact analysis of the patch, integration of the patch, and resolving any potential side-effects caused by the patch. Some researchers analyzed the timelines of security patches by software providers (Frei et al. 2006; Shahzad et al. 2012). For example, Shahzad et al. (2012) observed that the trend of patch availability before or on the disclosure of a vulnerability had decreased before 2005 but improved after 2008 (until 2011). The authors also showed that closed-source providers release at least 70% of vulnerabilities on or before disclosure dates.

Another major line of research has focused on the dynamics of patch development (Zaman et al. 2011; Huang et al. 2016; Ozment and Schechter 2006). (Ozment and Schechter 2006) observed that the density of vulnerabilities is in the range of 0 to 0.033 vulnerabilities per thousand lines of code in the OpenBSD system. (Huang et al. 2016) proposed a model to generate security workarounds rapidly for vulnerabilities in five projects. The model generated workarounds for remediating 75% of vulnerabilities but possibly at the expense of loss of functionality. Different from the aforementioned studies that only focus on the stage of patch development in a few projects, our study investigates the entire fixing process of a vulnerability in software ecosystems (i.e., Linux distributions). For example, our findings suggest that Debian and Fedora integrate vulnerability fixes of the majority of CSVs that are from vulnerable upstream projects.

Li and Paxson (2017) conducted a large-scale study of security patches on 4,080 commits across 682 open-source projects for 3,094 vulnerabilities. They identified that the changes for security patches in source code are more likely to be made by the project's own contributors, compared to ordinary (non-security) bug fixes made by external contributors. They also showed that vulnerabilities reside in source code for years prior to remediation. Li and Paxson (2017) studied vulnerability fixes in a large number of OSS projects that may/may not have relations with each other, while in our work we study OSS projects that form a self-sustainable software ecosystem. In addition, since collaboration is the nature of OSS development for successful software (Kakimoto et al. 2006; Duc et al. 2011), our study investigates the collaboration of patch development across maintainers in Linux distributions.

4 Study Design

This section presents the approach for our empirical study to understand how security vulnerabilities are managed in Linux distributions. RQ1 studies the duration of the five phases for fixing vulnerabilities according to the two perspectives and compares them. RQ2 investigates the characteristics of vulnerabilities that affect the two studied distributions. Finally, RQ3 studies the strategies that maintainers use to fix vulnerabilities that are shared by both analyzed distributions.

4.1 Subject System Selection

We select Debian and Fedora for several reasons. First, Linux distributions form families based on their package file formats. Deb-based and Rpm-based families are two notable Linux distribution families, with Debian and Fedora each belonging to a different family.[2] Second, within a family, ancestor relations exist based on the provenance of a distribution's packages. For example, Debian is the root of the Deb-based family of distributions, with 44 "child" distributions extending and customizing Debian's packages, each of which can have its own child distributions. Fedora is the root distribution of the Rpm-based distribution family, with 17 children. Despite being root, both Debian and Fedora are popular Linux distributions by themselves, according to the well-known distrowatch.com portal and considered amongst the most popular end-user Linux distributions,[3] and best multi-purpose Linux distributions.[4] Prior work leveraged Debian's and Fedora's data on studying security vulnerabilities, such as training vulnerability detection models (Russell et al. 2018; Harer et al. 2018), building vulnerability discovery tools (Alhazmi and Malaiya 2008), security vulnerability assessment (Ristov et al. 2013) and studying the impact of vulnerabilities (Yilek et al. 2009).

Furthermore, Fedora is a not-for-profit version of Linux that is primarily sponsored by Red Hat.[5] For example, Fedora follows the "upstream first" policy[6] from Red Hat to do software development (e.g., forwarding bugs to upstream, fixing bugs) side-by-side with upstream developers in the upstream projects. On the other hand, Debian, while still trying to get bug fixes merged upstream, is a purely open-design Linux distribution and has its own community of contributors (Reasons to use debian online). As such, there are differences between the strategies of upstream package management in Debian and Fedora in terms of where bug fixes are developed. For these reasons, we are interested in comparing the strategies of vulnerability management between the two distributions.

4.2 Data Collection

Figure 3 presents an overview of our study approach. To extract the phases stated in Section 2.2, we leverage vulnerability reports, bug reports and security advisories to measure the duration of each lifecycle phase for each vulnerability. We archived our scripts and uploaded them to GitHub.[7] For a given vulnerability, we use its unique CVE-ID to identify its corresponding reports and advisories. A vulnerability report contains the CVE-ID, the creation date, description, affected packages and package versions, links to its bug reports, and references to other information (e.g., the fix patch), for example, the vulnerability report of CVE-2018-7225.[8] A security advisory is issued for one or several CVE-IDs and consists of the affected packages, remediation plans (e.g., security updates), and reference reports (e.g., National Vulnerabilities Database (NVD) reports).

Table 1 presents the software systems and mailing lists for which we obtain the three types of data sources in each studied distribution. We extract data from Bugzilla (Fedora) and GitLab (Debian) using their relevant APIs, access security advisories by the mailbox package[9] in Python, and customize crawlers for Debian's Debbugs repository. We only

[2] https://en.wikipedia.org/wiki/List_of_Linux_distributions

[3] https://www.zdnet.com/article/the-five-most-popular-end-user-linux-distributions/

[4] https://itsfoss.com/best-linux-distributions/

[5] https://getfedora.org/sponsors/

[6] https://www.redhat.com/en/blog/what-open-source-upstream

[7] https://github.com/SAILResearch/suppmaterial-22-justina-vulnerability_management_in_linux_distributions

[8] https://security-tracker.debian.org/tracker/CVE-2018-7225

[9] https://docs.python.org/3/library/mailbox.html

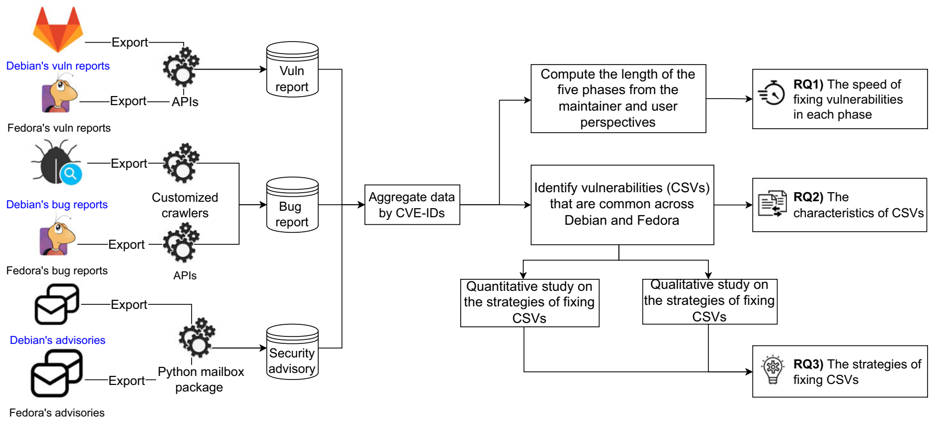

Export
Debian's vuln reports
Export
APIs
Fedora's vuln reports
Vuln report
Export
Debian's bug reports
Customized crawlers
Export
Fedora's bug reports
APIs
Bug report
Export
Debian's advisories
Export
Python mailbox package
Fedora's advisories
Security advisory
Aggregate data by CVE-IDs
Compute the length of the five phases from the maintainer and user perspectives
Identify vulnerabilities (CSVs) that are common across Debian and Fedora
Quantitative study on the strategies of fixing CSVs
Qualitative study on the strategies of fixing CSVs
RQ1) The speed of fixing vulnerabilities in each phase
RQ2) The characteristics of CSVs
RQ3) The strategies of fixing CSVs

Fig. 3 Overview of our study approach

consider security vulnerabilities with a valid CVE-ID according to the CVE system, as discussed in Section 2.1. Therefore, we filter out 5,138 security vulnerabilities with an invalid CVE-ID. For example, some CVE-IDs should not have been assigned (e.g., not a real vulnerability or should remain private), some CVE-IDs refer to the same vulnerability, or one CVE-ID actually refers to two different vulnerabilities, etc.[10] This leaves us with a total of 21,752 and 17,434 security vulnerabilities, i.e., unique CVE-IDs, in Debian and Fedora, respectively, whose CVE-ID was issued between Jan. 2007 and Dec. 2019.

From the maintainer perspective, the duration of the awareness phase represents the time difference between the creation time of a vulnerability report and the creation time of its associated bug report. The duration of the integration phase represents the time difference between the creation time of the bug report and the time when the vulnerability fix is ready for testing. To identify when a vulnerability fix is ready for testing, i.e., in the test repository, or available to users, i.e., in the stable repository, we extract the corresponding automated messages in the comments of a bug report. For example, Fig. 4 indicates that the vulnerability fix for CVE-2018-7889 has been pushed into the test repository on March 11th, 2018.[11] To validate the existence of the automated messages, we randomly select a set of 50 fixed vulnerabilities in each of the studied distributions and study their respective bug reports, i.e., a total of 100 bug reports. We observe that all these bug reports contain such automated messages. The duration of the test phase represents the time difference between when the vulnerability fix is ready for testing and when it is available to users.

From the user perspective, the duration of the user-unawareness phase reflects the time difference between the creation time of a vulnerability report and when its security advisory is published. The duration of the user-waiting phase reflects the time difference between when the security advisory is published and when the vulnerability fix is available to users.

4.3 Methodology

In this section, we present why and how we leverage the vulnerability data for our RQs and how we perform the manual study (RQ3).

[10] https://cve.mitre.org/cve/list_rules_and_guidance/correcting_counting_issues.html

[11] https://bugzilla.redhat.com/show_bug.cgi?id=1553919

Table 1 The software systems and mailing lists where we extract the three kinds of repositories in the studied distributions

Distribution	Vulnerability reports	Bug reports	Security advisories
Debian	Gitlab[1]	Debbugs[3]	Mailing list[4]
Fedora	Bugzilla[2]	Bugzilla[2]	Mailing list[5]

[1] https://salsa.debian.org/security-tracker-team/security-tracker

[2] https://bugzilla.redhat.com/

[3] https://www.debian.org/Bugs/

[4] debian-security-announce@lists.debian.org

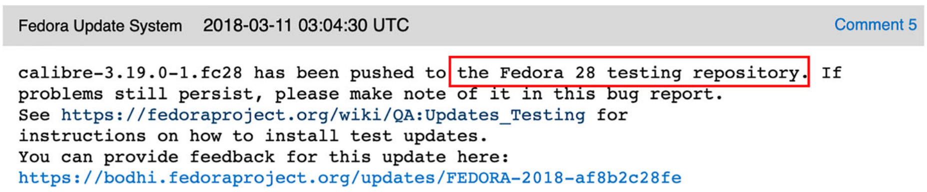
[5] package-announce@fedoraproject.org

4.3.1 RQ1: How fast are security vulnerabilities fixed within the studied distributions?

Motivation The goal of this research question is to measure an upper bound of how much time maintainers spend in each phase while fixing security vulnerabilities in a given distribution (Fig. 2). Such an investigation will give insights into the maximum cost these phases require, both from the maintainer and user perspectives. Comparing the time taken in each phase can provide useful indications of the practices that maintainers use to fix vulnerabilities.

Approach We study the duration of the five phases from the **maintainer** and **user** perspectives, as discussed in Section 2.2. From the maintainer perspective, we compute the number of days in each phase for each vulnerability and classify vulnerabilities into 6 classes based on the duration (i.e., <0, 0-1, 1-7, 7-30, 30-90 and 90+ days). These 6 classes correspond to instantaneous, same-day, same-week, same-month, same-quarter, and more-than-one-quarter, respectively. To compare the speed of fixing vulnerabilities in the studied distributions, we analyze a cumulative density plot to show the proportion of vulnerabilities over time. For example, we study how fast maintainers integrate vulnerability fixes in the integration period. Since Debian is a purely community-driven distribution and Fedora is a not-for-profit distribution backed by Red Hat, we compare the speed in each period to understand whether they follow different strategies for vulnerability management.

From the user perspective, similar to the maintainer perspective, we compute the number of days for the two phases, i.e., the user-unawareness and user-waiting phases, in the studied distributions. We classify the phases into the same 6 classes to understand how long users are unaware of a vulnerability and how much time it takes to get a fix. Since vulnerability exploits usually have a large impact (e.g., data leakage, service temporarily unavailable)


Fedora Update System 2018-03-11 03:04:30 UTC Comment 5

```
calibre-3.19.0-1.fc28 has been pushed to the Fedora 28 testing repository. If
problems still persist, please make note of it in this bug report.
See https://fedoraproject.org/wiki/QA:Updates_Testing for
instructions on how to install test updates.
You can provide feedback for this update here:
https://bodhi.fedoraproject.org/updates/FEDORA-2018-af8b2c28fe
```

Fig. 4 An example of a comment in a Fedora bug report that mentions that a vulnerability fix has been pushed into the test repository

on victim software systems (Klinke and Renn 2002), we compare the duration of the two phases between the studied distributions to understand whether they have different practices of vulnerability management.

4.3.2 RQ2: What are the characteristics of common security vulnerabilities (CSVs) across the studied distributions?

Motivation Developers may take more than two years to discover and fix vulnerabilities in software projects (Ozment and Schechter 2006). During this period, the vulnerabilities might propagate from one software ecosystem that has packaged the affected software project to another (Li and Paxson 2017). In this case, several software ecosystems (i.e., Linux distributions) are affected by a given set of vulnerabilities. As such, given the reliance of any open (and even closed) source project on a supply chain of software dependencies, such vulnerabilities risk impacting not only direct end users of a distribution package, but large cross-sections of the software world.

On the plus side, such spreading of vulnerabilities also increases the chance of vulnerability detection and opportunities for collaboration on fixing vulnerabilities (which is studied in RQ3), *cf.* Linux's Law ("*given enough eyeballs, all bugs are shallow*") (Raymond 1999). Given that such collaboration transcends project, ecosystem or even company borders, it is important to understand the dynamics of such collaboration, as well as when such collaboration happens (e.g., prioritization of vulnerabilities to fix collaboratively). For these reasons, we are interested in the characteristics of the security vulnerabilities that affect several distributions, and we investigate whether maintainers follow different strategies to fix these security vulnerabilities.

Approach We analyze the characteristics of common security vulnerabilities (CSVs). To study CSVs, we first identify security vulnerabilities whose CVE-ID exists across the two studied distributions. For example, CVE-2014-0160 is an example CSV that has affected both Debian[12] and Fedora[13] via the *OpenSSL* package.

We study the prevalence of CSVs and their threat level across the studied distributions. To obtain the prevalence, we calculate the number of CSVs that have been reported and fixed, and the number of their corresponding security advisories. To evaluate the impact, we compare the threat level of CSVs based on their CVSS scores, as discussed in Section 2.1. Similar to RQ1, we investigate the vulnerability fixing process of CSVs w.r.t. the five phases from the two perspectives in Fig. 2. We calculate the number of days in each phase and compare these numbers of CSVs between the studied distributions.

4.3.3 RQ3: How do developers fix common security vulnerabilities (CSVs) across the studied distributions?

Motivation In a software project, developers often collaborate for security vulnerabilities instead of relying on hero-centric contribution, which is different from ordinary bugs (Wang and Nagappan 2019). Given the large number (i.e., 16,070) of CSVs across distributions and the observations in RQ2 that maintainers take a longer time to test vulnerability fixes of CSVs than to integrate fixes, we are interested in how maintainers across distributions

[12]https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=743883

[13]https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2014-0160

collaborate when integrating and testing vulnerability fixes in their distributions. Therefore, understanding the collaboration patterns across distributions for fixing CSVs can provide insights for improving current practices and sharing with other ecosystems.

Approach To understand how maintainers collaborate on fixing security vulnerabilities, we focus on CSVs that are marked as fixed by both Debian and Fedora. We select a set of 3,336 **fixed** CSVs in both studied distributions before we conduct a manual study on a representative random sample (confidence level = 95%, confidence interval = 5%) of 345 **fixed** CSVs across the studied distributions. The goal of this qualitative analysis is to identify the flow of strategies, i.e., the actions for fixing vulnerabilities, as well as the prevalence of each strategy. In particular, we leverage the comments in the vulnerability reports, their corresponding bug reports in the distribution, the related bug reports in upstream projects where vulnerabilities could be discussed and fixed, and links to reference web pages in these reports. Based on these comments and links, we compile a catalog of strategies for fixing vulnerabilities, which we have grouped into three types according to their purpose (i.e., awareness, embargo, vulnerability fix integration).

For a given CSV, we first study the strategies of fixing vulnerabilities in the distribution. The first author sorts the post time of each comment in the vulnerability and bug reports to investigate the timeline of strategies through developer discussions, then labels strategies (e.g., report upstream) that are used for fixing the vulnerability. After that, the first author checks the links in the comments to reference web pages and the corresponding upstream reports. These reference links indicate the information relevant to the development of a vulnerability fix, such as a patch, a notification email for the disclosure of the vulnerability, and the reference reports of the vulnerability. Similarly, the first author labels strategies (e.g., propose a patch to upstream) along with (if this data is available) when they happened relative to each other, before structuring them into a timeline. Finally, the first author discusses the strategies with the 2nd and 3rd authors and addresses disagreement. We performed this process iteratively for each CSV and arrived at the strategies summarized in Fig. 5, along with the numbers (proportions) of the analyzed CSVs using a particular strategy.

We also perform several quantitative analyses to evaluate the three types of strategies of Fig. 5 (the boxes with grey background) to gain comprehensive insights on potential collaboration across distributions to fix CSVs. We describe the details of approaches for each type below:

(A) Awareness Since the security team is responsible for discovering vulnerabilities for their distribution, we analyze how fast the security team becomes aware of a vulnerability. We compare the creation time of vulnerability reports for fixed CSVs between Debian and Fedora to measure the speed of awareness. We leverage the number of bug reports related to vulnerabilities that were created by the security team (i.e., internal developers) to measure the extent of responsibility for the security team.

(B) Embargo When a vulnerability is first reported to a distribution, the security team could forward the vulnerability to upstream after it assesses the impact of the vulnerability. In such a case, the security team could embargo the vulnerability (i.e., only certain people with permission can read the vulnerability report) to reduce its impact according to the distribution's policy. For severe vulnerabilities, vulnerability fixes are usually discussed privately under embargoes to prevent early disclosure (Ramsauer et al. 2020). To understand how embargoes work in practice, we investigate the embargo policy for CSVs in Fedora (Red Hat), since Debian encourages public disclosure of vulnerabilities even before a fix has been developed (Debian vulnerability disclosure policy online).

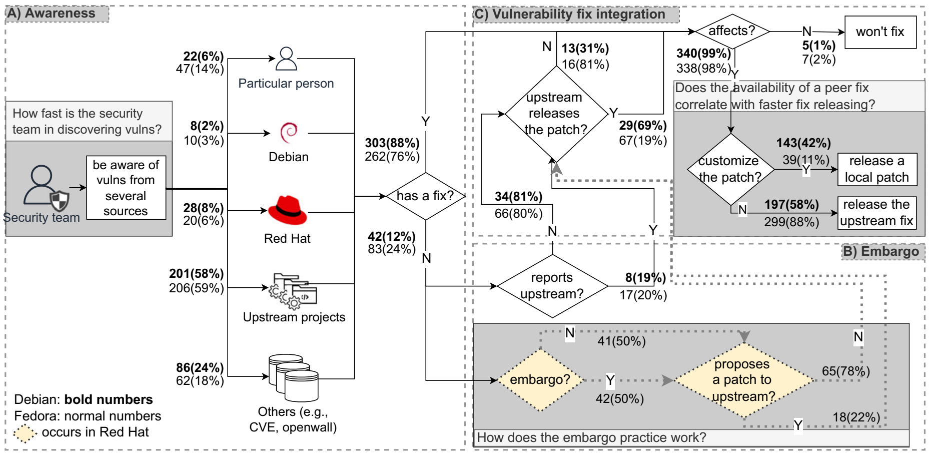


Fig. 5 The flow chart of the potential strategies that are used in Debian and Fedora for dealing with vulnerabilities. The bold numbers represent the number and proportion of CSVs in each strategy in Debian and the normal numbers are for Fedora. The two grey dotted diamonds and dotted lines represent the strategies related to Red Hat's embargo policy. The three grey dashed boxes group the strategies based on their purpose. The three boxes with grey background represent the quantitative analyses to measure the degree of potential collaboration in the three groups

From the history of a Fedora vulnerability report, we observe that the Fedora security team members add the *"EMBARGOED"* keyword as a prefix in the title of a vulnerability report to indicate that the vulnerability is under embargo. The keyword is removed once the embargo is lifted and this change is recorded in the history of the vulnerability report. For example, CVE-2015-8382 was embargoed for 5 days in Red Hat.[14] Therefore, we study the embargo policy by studying the characteristics of embargoed CSVs and the time taken by the five phases of the maintainer and user perspectives (Fig. 2).

(C) Vulnerability fix integration As discussed in Section 2.2, vulnerability fixes can be developed by anyone before being integrated into the distributions that contain the affected packages. We investigate the extent to which maintainers have access to a vulnerability fix generated by the peer distribution (the other studied distribution), which we refer to as *"peer (vulnerability) fixes"*. For a CSV, we identify when such a peer fix is available, i.e., before or during integration or not at all, by comparing the creation and fixing time of a bug report to the release time of the peer fix. To measure whether maintainers across distributions collaborate to integrate vulnerability fixes into their own packages, we hypothesize that distribution maintainers become aware of a peer fix once it is available. For example, Debian released a fix for CVE-2014-8145 on January 3rd, 2015,[15] while the bug for CVE-2014-8145 was reported in Fedora on January 20th, 2015.[16] We conclude that Fedora had access to a peer fix (in Debian) for CVE-2014-8145 before integration.

[14]https://bugzilla.redhat.com/show_activity.cgi?id=1187225
[15]https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=773720
[16]https://bugzilla.redhat.com/show_bug.cgi?id=1184079

5 Results

RQ1: How fast are security vulnerabilities fixed within the studied distributions?

Results: From the maintainer perspective, Debian and Fedora take a median of 20 and 27 days, respectively, from when the security team creates a vulnerability report to when the fix is released. Figure 6 indicates the duration of each phase. In particular, the awareness period accounts for a median of 2 and 0.003 days (4.3 minutes), which indicates a significant difference (Wilcoxon test: p-value $< 2.2e^{-16}$) in Debian and Fedora, respectively, although with negligible effect size (0.14). We will discuss the awareness period in the next finding.

One possible reason for the significant difference between Debian and Fedora is that Fedora is a not-for-profit Linux distribution that is primarily sponsored by Red Hat (as discussed in Section 4.1). Furthermore, Red Hat also has a dedicated security team working on vulnerabilities in Fedora. The integration period accounts for a median of 6.1 and 5.3 days (Wilcoxon test: p-value = 0.0001) and the test period accounts for a median of 0 and 15 days (Wilcoxon test: p-value $< 2.2e^{-16}$) in Debian and Fedora, respectively. The differences between Debian and Fedora in both the integration and test periods are significant. The effect size of the integration period is negligible (0.0006), and the effect size of the test period is small (0.24).

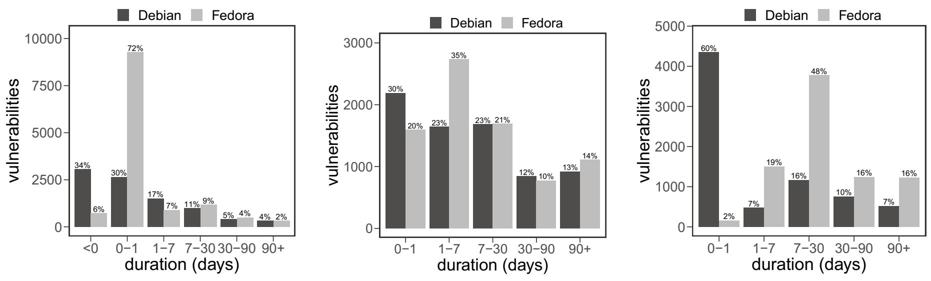
The security team in Fedora notifies its maintainers of at least 78% of vulnerabilities within one day after the vulnerability reports were created (i.e., a median of 4.3 minutes), while the security team in Debian takes one week for 81% of vulnerabilities (i.e., a median of 2 days), as shown in Fig. 6a. The awareness of a vulnerability is the first step of tracking a vulnerability that the security team of a distribution faces (cf. Fig. 2). The security team can discover a new vulnerability by itself or via other channels, e.g., open-source security communities. Once the security team is aware of the new vulnerability, the team evaluates, assesses and validates whether the packages in the distribution are affected before assigning the vulnerability to maintainers.

Meanwhile, users report 34% and 6% of vulnerabilities as ordinary bugs to Debian and Fedora, respectively, before their security teams become aware of these vulnerabilities, as shown in Fig. 6a. For example, a Debian bug, CVE-2017-12424,[17] was reported 3 years before the CVE-ID was assigned, i.e., when the security team became aware of the vulnerability.[18] In a given distribution, a vulnerability report can be created earlier or later than its related bug report, suggesting two fixing processes of vulnerabilities. If the vulnerability report is created earlier, the respective bug report is created for tracking the progress of fixing the vulnerability. By contrast, when a bug report is created earlier, maintainers and the community collaborate and discuss how to fix the bug before the bug is identified as a vulnerability, suggesting the vulnerability is publicly disclosed without a fix. Researchers should take into account how a vulnerability was discovered to study the impact of such vulnerabilities.

Both Debian and Fedora maintainers integrate the vulnerability fixes for the majority (50%) of security vulnerabilities within 1 week of the creation of their corresponding bug reports, and integrate 75% of security vulnerabilities within 1 month. Figure 7 shows that Debian maintainers integrate more vulnerability fixes within 3 days

[17] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=756630

[18] https://salsa.debian.org/security-tracker-team/security-tracker/-/commit/1d9590647d



(a) Awareness (b) Integration (c) Testing

Fig. 6 The time taken in the three phases from the maintainers' perspective

compared to Fedora maintainers. After that, Fedora catches up and both Debian and Fedora maintainers integrate vulnerability fixes at the same speed. Since both studied distributions have a large number of packages with complex relations among them, this result indicates that the speed of integrating vulnerability fixes into a distribution is fast, compared to the time of integrating patches into one project, i.e., the Linux Kernel project, which had an increasing trend in the integration time of patches from 2006 to 2012, from instantly to a month (Jiang et al. 2013).

Vulnerability fixes for 75% of vulnerabilities in Debian pass testing in 17 days, compared to 44 days in Fedora. Figure 8 shows the cumulative distribution of the test period. Vulnerability fixes pass testing in a median of 10 days across Debian and Fedora. The delay in the number of days for a fixed vulnerability to become available for users is shorter than that of an ordinary fixed bug (da Costa et al. 2014). For example, (da Costa et al. 2014) observed that the median delay time for fixed bugs was 42 days in the Firefox project (the minimum median delay time among three projects that they studied).

Figures 8 and 6c indicate that the test period for 60% of vulnerabilities is 0 days in Debian. One possible reason is that Debian does not integrate the results of its continuous integration (CI) system into bug reports. The continuous integration system in Debian automatically coordinates the automated test against packages in the Debian archive when the packages and/or their dependent packages have been updated (Debian continuous integration online). However, since the test logs in the CI system do not include whether the test was triggered by which bug, we are not able to link the testing logs to the corresponding bug. In addition, 53% of these vulnerabilities only affect the development (i.e., unstable)

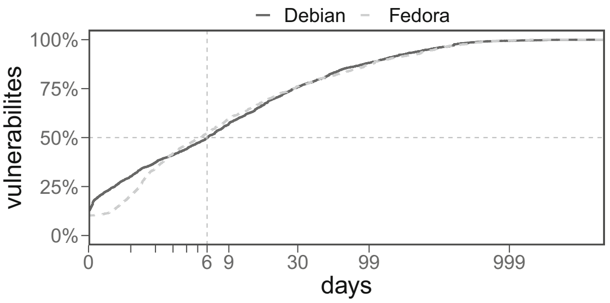


Fig. 7 Cumulative density plot of the duration of the integration phase of a security vulnerability. Debian and Fedora maintainers take a median of 6.1 and 5.3 days, respectively, to integrate fixes for security vulnerabilities after their bug reports were created

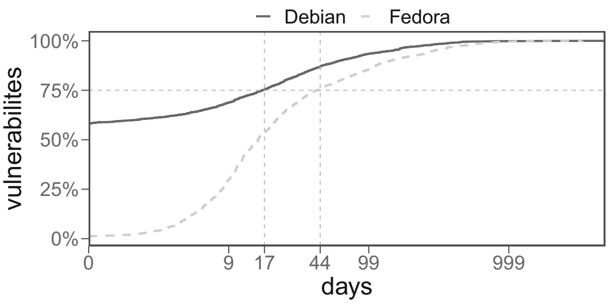


Fig. 8 Cumulative density plot for the testing phase of a security vulnerability. Debian maintainers test the fixes of 75% of security vulnerabilities in 17 days, while Fedora maintainers take 44 days for testing. 60% of vulnerabilities in Debian have a 0-day test phase

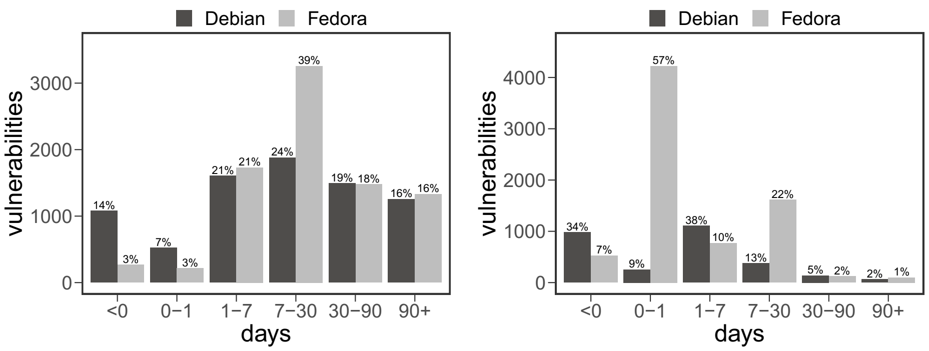
version of Debian, which are less likely to impact users. For example, a denial-of-service flaw in the *Quassel* IRC, i.e., CVE-2015-2779, only affected the development version.[19] Furthermore, 11% out of the 53% vulnerabilities with 0 days of testing only affect the long-term support (LTS) releases that are maintained by the Debian LTS team (Debian long term support online), which has a different development policy for security vulnerabilities (LTS development online).

The integration phase accounts for the largest proportion of time (i.e., median of 6.1 days, 40%) for tracking a security vulnerability in Debian, while Fedora spends the largest proportion of time (i.e., median of 15.7 days, 58%) on vulnerability fix testing. The number of days in the integration phase in Debian (median of 6.1 days) is similar to the number in Fedora (median of 5.8 days). Figure 6b indicates that Debian integrates more vulnerability fixes within one day than Fedora.

Concerning the testing phase, Fedora spends a higher proportion of time (median of 15.7 days) for testing vulnerability fixes, compared to Debian. Figure 6c indicates that Fedora takes 7 and up to 30 days to test 48% of vulnerabilities (the largest proportion among the 6 classes). One possible explanation is that the test logs in Debian's continuous integration (CI) system do not link back to the respective bug reports, making it impossible for us to trace CI test results back to possible vulnerability fixes being tested. The CI system executes automated test suites against packages when they and/or their dependent packages have been updated (Debian continuous integration online). The test logs only include packages that have been tested and their testing status (e.g., pass or fail), but identifiers of bug reports are not included, since a test can be triggered for a variety of reasons (e.g., a feature change, bug fixes in the dependent packages). As a rough estimation, Debian states that it usually takes 10 days to test a security update of a package (Securing debian manual - before the compromise online), though it might take longer due to the dependent packages needing testing, or shorter due to the emergency level of releasing a new version of the package, suggesting that Debian's test phase might be longer than the integration phase. In comparison with Debian, Fedora integrates its CI and issue report system, and thereby the testing results are recorded back into the bug reports (Fedora - update policy online).

From the user perspective, in general, Debian and Fedora publish security advisories, i.e., end the user-unawareness period, in a median of 17 days. Figure 9a indicates that Debian's users are aware of the majority (>50%) of vulnerabilities 30 days after the

[19]https://security-tracker.debian.org/tracker/CVE-2015-2779



(a) User unawareness (b) User waiting

Fig. 9 The number of vulnerabilities during the two phases from the user perspective

creation of vulnerability reports. Fedora's users are aware of the largest proportion (39%) of vulnerabilities within 7 to 30 days after the creation of vulnerability reports. In order to assess how fast Debian and Fedora publish security advisories, we compare them with GitHub security advisories, since GitHub is one of the most popular OSS code repositories. The speed of publishing security advisories in Debian and Fedora is faster than the speed of publishing security advisories in open-source projects on GitHub, based on the 2020 GitHub security report (Github - securing the world's software online). That report stated that users usually receive the alert of a security advisory more than 10 weeks after a vulnerability becomes known across GitHub projects. In particular, the speed of publishing security advisories provided by GitHub directly is 1 week, while the speed of publishing security advisories that are imported from other sources into GitHub projects is 20 weeks. A potential explanation could be that the two Linux distributions are self-sustainable with organized efforts, while GitHub projects are loosely coupled in nature, unless they are part of other ecosystems (e.g., npm). As such, our results could be explained by a community-driven effort to manage vulnerabilities in a Linux distribution ecosystem.

Users wait for a median of 0.2 days to get vulnerability fixes across Debian and Fedora, which is 85 times shorter than the user-unawareness period. Figure 9b shows that users wait for less than 7 days to get fixes for 81% and 74% of vulnerabilities in Debian and Fedora, respectively. Fedora releases the largest proportion (57%) of vulnerability fixes within 1 day after the end of the user-unawareness period, dropping to 10% within 7 days and increasing to 22% within 30 days. The priority of fixing vulnerabilities is high due to their potential impact (e.g., market value loss (Telang and Wattal 2005)) on a large number of users. Figure 9b indicates that the fixes for 34% (i.e., 980) of vulnerabilities in Debian were available before Debian published their security advisories, suggesting insufficient awareness about those security vulnerabilities by the Debian's security team. For example, for a Debian bug representing a vulnerability that was reported for the *vlc* package (i.e., CVE-2008-2147),[20] its fix was available for users on May 17, 2008, while its security advisory was published on June 18, 2009.[21]

One possible explanation is that vulnerabilities might be assumed to be ordinary bugs when they were reported. As a consequence, their fixes might be ignored until they are

[20] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=480724

[21] https://www.debian.org/security/2009/dsa-1819

identified as a vulnerability due to various reasons. For example, bug fixes often are developed relative to a new version of a package that the package maintainers cannot or do not plan to upgrade to. Kula et al. (2018) identified that 81.5% of the analyzed software systems stick to outdated dependencies. Second, a bug fix could introduce a new bug impacting the software system (Yin et al. 2011), which is challenging for the maintainers to assess before the adoption of bug fixes. Even if the maintainers would like to integrate the bug fix, they usually update the software system periodically for a batch of bug fixes. This is because the integration could require changes to their package's code, and because they need to organize (manual) testing activities. Depending on the workload of the maintainers and the testers, the integration of a batch of bug fixes might be skipped until more fixes are available, at which time all test activities are performed once.

Another explanation is Debian's policy regarding publishing security advisories based on the impact of vulnerabilities to Debian (Debian security faq onlineb). To avoid cognitive overload, Debian typically includes the security advisory for low-impact vulnerabilities of a particular package into the security advisory of a high-impact vulnerability in that package. As such, the advisory of low-impact vulnerabilities might be delayed until the next high-impact advisory is issued. Moreover, out of the 34% vulnerabilities whose fixes were available before their security advisory in Debian, 55% did not affect any packages in Fedora, since Fedora does not contain those packages for a variety of reasons (e.g., license, patents) (Fedora - packagekit items not found online). Only 1% already had a fix prior to the disclosure of their security advisories in Fedora.

Summary of RQ1

In terms of phases from the maintainers' perspective, both Debian and Fedora maintainers integrate vulnerability fixes at a fast pace, *i.e.*, within a median of 1 week, after they are assigned the task. Fedora maintainers spend the longest time to test vulnerability fixes in the vulnerability fixing process. In terms of phases from the user perspective, the user-unawareness period accounts for a median of 17 days and the user-waiting period accounts for a median of 0.2 days.

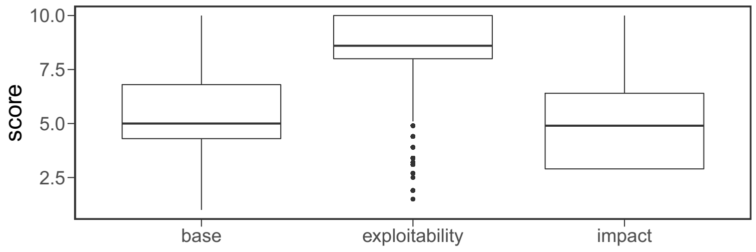
RQ2: What are the characteristics of common security vulnerabilities (CSVs) across the studied distributions?

Results: There are 16,070 CSVs across Debian (74%) and Fedora (92%). Table 2 shows the statistics of the number of reported CSVs along with their related information for the two studied distributions. One possible reason for the higher proportion (92%) of CSVs in Fedora is that Debian includes over 89,000 packages (Debian packages online), while Fedora only includes over 33,000 packages (Fedora package sources online). Another possible reason is that Debian offers its users 3 years of support for each release and 2 years of extra long-term support (LTS) (Debian releases online), while Fedora supports its releases for approximately 13 months (Fedora release life cycle online). The longer support period increases the possibility of identifying and having to fix security vulnerabilities specific to Debian in the packages of old releases.

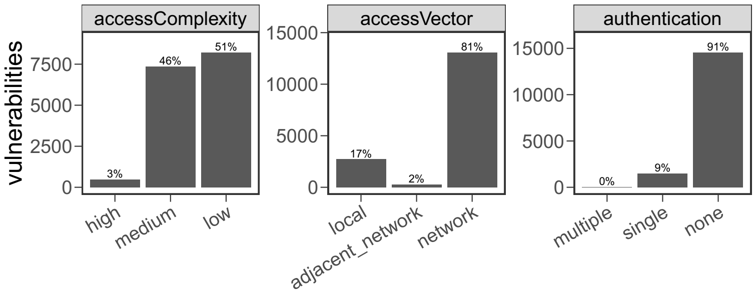
Based on CVSS 2.0, the CSVs can be exploited easily (i.e., median exploitability score of 8.6), though their severity level and impact is medium (i.e., median of both base and impact score of 5.0), as shown in Fig. 10a. A high exploitability score indicates

Table 2 The descriptive statistics of common security vulnerabilities (CSVs) along with their related information. Note that a vulnerability report may map to several bug reports, and vice versa

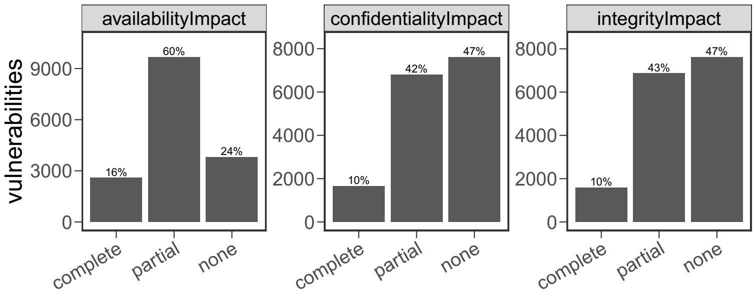
Feature	Debian	Fedora
# reported vulns	16,070	16,070
# vulns with bug reports	7,265	12,034
# vulns with advisories	6,760	8,710
# vulns with fixes	5,951	7,310



(a) The majority of CSVs are medium-severity vulnerabilities (median base score of 5), but they are easy to be exploited (median exploitability score of 8.6).



(b) The majority of CSVs are easy to be exploited, due to low complexity, remote exploits and no authentication.



(c) The majority of CSVs that have been exploited have partial or no impact.

Fig. 10 The (a) high-level CVSS 2.0 scores for CSVs, and a breakdown for (b) exploitability and (c) impact scores

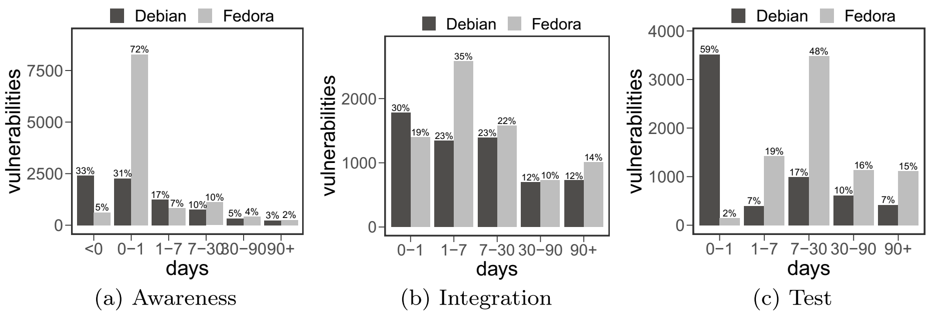


Fig. 11 The duration of the three phases for CSVs from the maintainers' perspective

that attackers can easily exploit these CSVs to infiltrate a victim system, potentially on a large scale. Figure 10b indicates that the complexity to exploit 51% of CSVs is low and that of 46% of CSVs is medium. Furthermore, attackers can exploit the majority of CSVs using networks and without any authentication. In terms of the impact of CSVs, Figure 10c indicates that the majority of CSVs have partial or no impact on victim systems based on the three metrics, i.e., availability, confidentiality and integrity.

From the maintainer perspective, Debian and Fedora take a median of 20 and 27.1 days, respectively, to fix CSVs, from when the security team creates a vulnerability report to when the fix is released. In particular, the security teams in Debian and Fedora notify their maintainers in a median of 3.6 hours and 2.9 minutes, respectively, after the creation of a vulnerability report. Maintainers in Debian and Fedora take a median of 6.1 and 5.5 days to integrate fixes, respectively, and a median of 0 and 15 days to test fixes, respectively. These results are in line with the result in RQ1 (Fig. 6).

Figure 11 indicates the distribution of CSVs in each of the three phases. The security teams in Debian and Fedora notify their maintainers of 81% and 84% of CSVs, respectively, within 7 days after the creation of a vulnerability report. Maintainers integrate fixes for the majority of CSVs within one week. For testing fixes, Fedora takes up to 30 days for 59% of CSVs, while Debian takes one day for the same proportion. As discussed in RQ1, Debian performs part of tests in its CI system but there is no link between bug reports and the test logs in the CI system.

Testing fixes for CSVs requires a median of 4.3 days longer than integrating fixes for CSVs across Debian and Fedora, as shown in Fig. 12. The difference between the

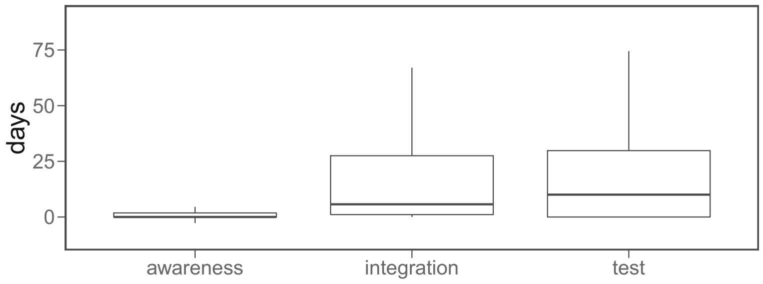


Fig. 12 The comparison of the length of each phase from the maintainers' perspective for CSVs. The awareness period is shorter than the integration and test phases

integration and test phases is significantly different, according to the Wilcoxon test, with a p-value of 0.001 ($\alpha = 0.01$) and a negligible effect size of 0.03.

From the user perspective, users wait for a median of 0.2 days to get fixes for CSVs while they take a median of 17 days to be aware of CSVs. The Wilcoxon test indicates that there is a significant difference between the user-unawareness and user-waiting period, with a p-value $< 2.2e^{-16}$ and a medium effect size of 0.47. These median values of the two phases for CSVs are the same as the median values of the two phases in RQ1 (Fig. 9).

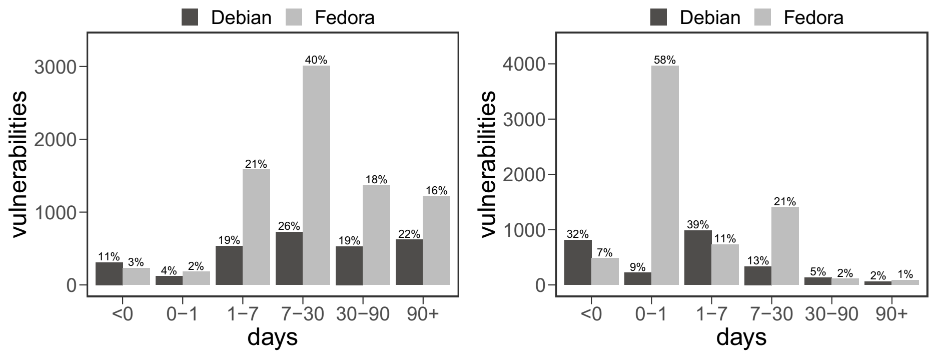
In addition, similar to the results from the maintainer perspective, the user-unawareness and user-waiting periods have different distributions in Debian and Fedora. Figure 13a and b show the distribution of CSVs in the two user perspective phases in Debian and Fedora. These differences might be due to the different strategies of vulnerability management in Debian and Fedora. Future work could replicate our work on other distributions to generalize our findings.

Summary of RQ2

The majority of reported vulnerabilities in Debian (74%) and Fedora (92%) are security vulnerabilities common to both distributions (CSVs). The CSVs are easy to exploit, though with a medium impact on victim systems. Maintainers take less time to integrate vulnerability fixes for CSVs than testing fixes. Fedora tests the fixes of 59% of CSVs in 30 days, compared to one day in Debian. Furthermore, Fedora users get fixes for 65% of CSVs within one day, compared to within one week in Debian.

RQ3: How do developers fix common security vulnerabilities (CSVs) across the studied distributions?

Results (awareness): For the majority of fixed CSVs, distributions are notified about vulnerabilities by the upstream projects. Figure 5 presents the 5 identified sources through which the security team becomes aware of CSVs. The security teams in Debian



(a) User unawareness (b) User waiting

Fig. 13 The duration of the two phases from the user perspective for CSVs

and Fedora are aware of 58% (201 out of 345) and 59% (206 out of 345) of the fixed CSVs from the affected upstream projects directly. Our results show that the awareness of distributions about vulnerabilities comes mainly from upstream projects, along with a fix. This indicates that the collaboration across distributions and upstream projects on vulnerability fix propagation is in line with the guidelines for multi-party coordination from first.org (as mentioned in Section 1).

For the majority of vulnerabilities, the security teams in the studied distributions are also made aware of an available fix. Figure 5 shows that the security team provides the information (e.g., links) to a fix patch for the majority of the fixed CSVs in Debian (303 out of 345, 88%) and Fedora (262 out of 345, 76%). Maintainers in Debian and Fedora need to fix vulnerabilities for their users even if they did not introduce the vulnerabilities. However, due to their lack of detailed knowledge about the upstream projects' internals (Ma et al. 2017), adopting a provided fix for a vulnerability is the common way for distribution maintainers to remediate the vulnerability and quickly release a new version to users, compared to investigating and addressing the root causes by themselves.

To this end, our results indicate that the security team usually offers maintainers sufficient information (e.g., patch, upstream reports that they require) to achieve the goal of fast remediation for vulnerabilities. For example, for the RTP resource exhaustion (CVE-2016-7551) in the *asterisk* package, the security team posted the upstream bug report along with the reference link to a fix patch when the security teams created the corresponding bug reports in both Debian[22] and Fedora.[23]

For Fedora, the security team of Red Hat is almost exclusively responsible for discovering the CSVs, while, in Debian, 30% of fixed CSVs are reported by users. More specifically, the security team of Red Hat creates bug reports for 99% of fixed CSVs, while the security team in Debian "only" reports 70% of fixed CSVs, although with an increasing trend. As mentioned in Section 4.1, since Fedora is primarily sponsored by Red Hat,[24] the security team of Red Hat handles vulnerabilities in Fedora (Fedora - security bugs online). This result is similar to the result we found in RQ1 that users report 34% and 6% of vulnerabilities to Debian and Fedora, respectively.

The security teams in Debian and Fedora are aware of fixed CSVs at a similar speed. The security team in Debian is aware of 41% of fixed CSVs faster than Fedora and vice versa, while the security teams are aware of 18% of fixed CSVs on the same day. Figure 14 shows that the awareness speed for the 41% of fixed CSVs in Debian is a median of 5.5 days faster than in Fedora. For the 41% of fixed CSVs that Fedora became aware of earlier, the awareness speed is a median of 7.5 days faster than Debian.

Results (embargo): Red Hat becomes aware of 24% (i.e., 83 out of 243) of CSVs without a fix and embargoes half (42 out of 83) of these CSVs to prevent early disclosure. Since Red Hat plays the role of a trusted partner for many upstream open-source projects to assist them with security issues,[25] upstream projects reach out to Red Hat for rapidly developing a fix when they became aware of an embargoed vulnerability. In addition, the Red Hat security team contributes patches for 13 (31%) out of the 42 embargoed CSVs, suggesting a close collaboration between Red Hat and upstream projects on fixing vulnerabilities.

[22]https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=838832

[23]https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2016-7551

[24]https://getfedora.org/sponsors/

[25]https://www.redhat.com/en/resources/managing-vulnerabilities-FAQ

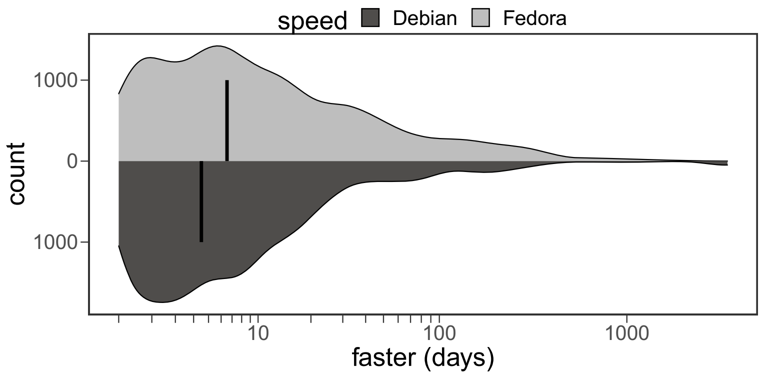


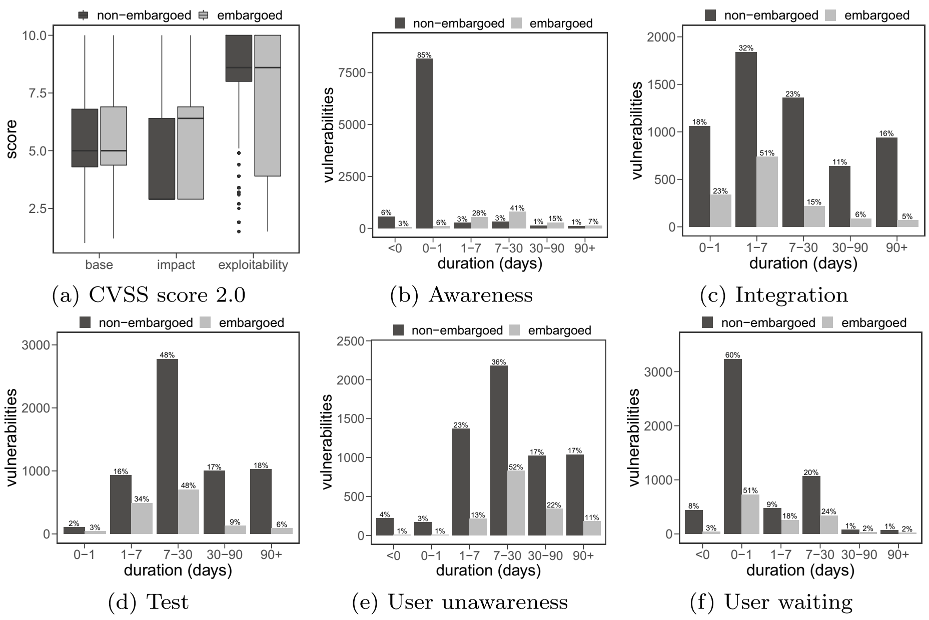
Fig. 14 The distributions of the difference (in log-scale days) between the creation times of fixed CSV reports with the same CVE-ID in Debian and Fedora. The black lines indicate the median values of the two distributions

On the other hand, Debian does not embargo vulnerabilities (Debian vulnerability disclosure policy online), which increases the chance of attackers exploiting vulnerabilities after their disclosure, if the fixing process takes longer than foreseen. For example, the bug report for a CSV (CVE-2013-6456) was publicly disclosed in Debian[26] when Debian maintainers were not yet able to fix it. One Red Hat developer was involved in fixing the vulnerability by evaluating a patch and providing suggestions. In the end, the upstream project accepted the patch that was submitted by the Red Hat developer.[27]

Although Debian does not embargo any vulnerabilities, Debian is potentially involved in vulnerability fix development during the embargo period. Out of the 42 embargoed CSVs, Debian referred to Red Hat's vulnerability reports and/or security advisories for 12 (29%) embargoed CSVs and attached the same references that were attached to Red Hat's vulnerability reports for 6 (14%) embargoed CSVs. For example, CVE-2017-1000409[28] was a buffer overflow in the *glibc* 2.5 package and can be triggered by a specific environment variable. The fix was developed by the members of the open-source security group[29] of which both Debian and Fedora are a member. Another embargoed CSV, CVE-2012-2737,[30] was found and embargoed by Red Hat[31] and also affected the *accountsservice* package in Debian. The respective bug report in Debian was created within one hour after the embargo was lifted and one Debian maintainer posted the link to the Red Hat's report,[32] with the report hinting at the corresponding maintainer being aware of the embargoed vulnerability during the embargo period.

Embargoed CSVs are more risky than non-embargoed CSVs since they have a higher impact score, based on CVSS score 2.0. Figure 15 shows the comparison between embargoed and non-embargoed CSVs. Embargoes can reduce the risk of a vulnerability being exploited by attackers before having a mitigation (The hidden costs of embargoes

[26]https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=732394
[27]https://libvirt.org/git/?p=libvirt.git;a=commit;h=5fc590ad9f4
[28]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000409
[29]https://seclists.org/oss-sec/2017/q4/385
[30]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-2737
[31]https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2012-2737
[32]https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=679429



(a) CVSS score 2.0 (b) Awareness (c) Integration

(d) Test (e) User unawareness (f) User waiting

Fig. 15 The comparison between the embargoed and non-embargoed CSVs in Fedora. (a) shows the distribution of the base, impact and exploitability scores. (b), (c), and (d) reflect the three phases from the maintainer perspective. (e) and (f) reflect the two phases from the user perspective

online). However, embargoing a vulnerability is challenging in an open source development setting, since the code repositories (e.g., GitHub) of software projects are publicly available. Hence, the embargoed vulnerability can be found by anyone during the embargo period, such as the *Heartbleed* vulnerability (CVE-2014-0160), which was initially found by Codenomicon, before being independently found by the Google security team.[33] The history of the vulnerability report of *Heartbleed* indicates that it was embargoed for a short period (i.e., about 12 hours) after the security team created the vulnerability report.[34]

Embargoed CSVs have a slower speed of awareness for maintainers but a faster speed of integrating and testing vulnerability fixes, compared to non-embargoed CSVs. Figure 15b - f show the five phases for the embargoed and non-embargoed CSVs from the maintainer and user perspectives, respectively. For the maintainer perspective, the awareness period of embargoed CSVs accounts for a median of 43% of the total time for tracking a CSV in Fedora, while that of non-embargoed CSVs accounts for a median of 0%, though those non-embargoed CSVs might have been embargoed somewhere else (without Fedora being aware of this). However, the integration and test phases for embargoed CSVs are shorter than non-embargoed CSVs (Wilcoxon test: p-value $< 2.2e^{-16}$, $\alpha = 0.01$). Both the integration and test phases have a small effect size of 0.29 and 0.3, respectively. By contrast, from the user perspective, both the user-unawareness and user-waiting phases for the embargoed CSVs are longer than non-embargoed bugs, by the Wilcoxon test (user-unawareness: p-value = $5e^{-10}$, user-waiting: p-value $< 2.2e^{-16}$, $\alpha = 0.01$). Both the user-unawareness and user-waiting phases have a negligible effect size of 0.13 and 0.002, respectively.

[33]https://www.zdnet.com/article/heartbleed-serious-openssl-zero-day-vulnerability-revealed/
[34]https://bugzilla.redhat.com/show_activity.cgi?id=1084875

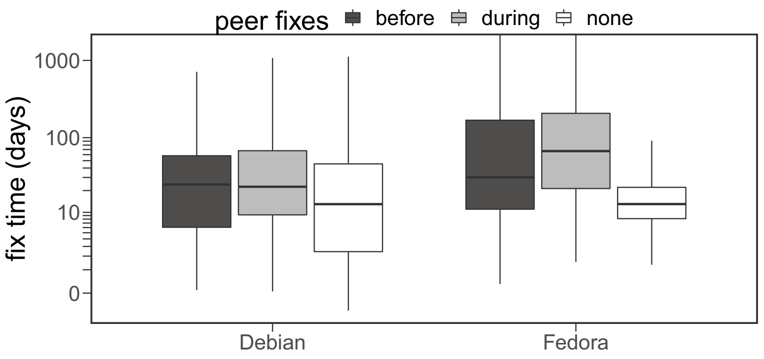


Fig. 16 Debian and Fedora maintainers do not speed up the integration of vulnerability fixes (in log-scale days) when peer fixes are available before the integration

Results (vulnerability fix integration): Debian maintainers fix 42% (143 out of 345) of the CSVs by applying customized patches, compared to 11% (39 out of 345) of the CSVs in Fedora. Once the upstream project has a patch to fix a vulnerability, distribution maintainers either leverage the fix patch and integrate it into their package manually or wait for the upstream release that incorporates the patch. The former case is the fastest way for distribution maintainers to release remediation of vulnerabilities to their users. However, distribution maintainers need to maintain these patches until the upstream has a release (Ma et al. 2017). By contrast, adopting the upstream release represents the simplest way for distribution maintainers to fix vulnerabilities though users might need to wait a longer time for the vulnerability fix. For example, Debian maintainers fixed the CVE-2017-16808 vulnerability[35] by producing a patch, 54 days faster than Fedora maintainers who applied the upstream release.[36]

In addition, distribution maintainers might not be available to rapidly produce a new version for the affected packages or there might not even be a maintainer available for it. Therefore, the security team would take care of these affected packages and release new versions for users. For example, CVE-2010-3695 in Debian[37] was fixed by the security team, as stated in the release message *"Non-maintainer upload by the security team"*. In fact, the security team in Debian has fixed 1,707 (8%) vulnerabilities for the affected packages that did not have maintainers when the vulnerabilities were reported.

In a distribution, the availability of a peer fix for a CSV before maintainers start to integrate the CSV fix does not correlate with faster integration and testing of the vulnerability fix. Figure 16 shows boxplots of the fixing time, i.e., the sum of the duration of the integration and test phases from the maintainer perspective, for CSVs based on when a peer fix is available in the two studied distributions. In particular, Fedora maintainers fix CSVs without a peer fix (the white box) faster than those that have a peer fix before integration (the dark grey box) (Wilcoxon test: p-value = $2.2e^{-16}$, $\alpha = 0.01$), with a medium effect size of 0.71. In Debian, we observe a similar phenomenon where maintainers take more time to fix CSVs when a peer fix is available before or during development, compared to those without a peer fix (Wilcoxon test: p-value = $2.6e^{-6}$, $\alpha = 0.01$). The effect size

[35] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=881862

[36] https://bugzilla.redhat.com/show_bug.cgi?id=1516995

[37] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=698916

is small (0.15). Our results suggest that collaboration across distributions is less likely to happen after maintainers start to integrate a vulnerability fix, since maintainers integrate packages into their distributions following their own packaging policy.

Summary of RQ3

Our qualitative analysis showed five sources used by the security teams in Debian and Fedora to become aware of CSVs and seven strategies used by distribution maintainers in the fixing process of vulnerabilities. In the awareness phase, the security team becomes aware of the majority of CSVs from upstream along with the respective fixes. When a CSV fix has not yet been developed, Red Hat (Fedora) embargoes half of the CSVs to prevent early disclosure due to their threat level. In the phase of vulnerability fix integration, having a peer fix available does not reduce the needed time for maintainers to integrate vulnerability fixes.

6 Discussion

In this section, we discuss the findings presented in Section 5.

Distribution maintainers collaborate closely across distribution boundaries before vulnerability fixes are available, but less afterwards, i.e., in the integration and test phases. Even though the majority of CSVs in both Debian (51%) and Fedora (65%) have a peer fix before the integration of CSV fixes, we observe in RQ3 that maintainers do not release CSV fixes faster in such cases. In addition, Debian (58%) and Fedora (59%) are notified about the majority of fixed CSVs by upstream projects, even though they share a mailing list with other Linux distributions (Operating system distribution security contact lists online), indicating that Debian and Fedora are collaborating on developing fixes rather than on the discovery of a vulnerability.

Hence, since the two studied distributions are major software ecosystems in the larger Linux ecosystem, there is still room for more collaboration on fixing vulnerabilities at the scale of the whole Linux ecosystem. This should lead to more benefits in terms of reduced fixing time, especially since the majority of vulnerabilities of both Debian and Fedora are CSVs (RQ2).

In addition, there is **a need for better tracking of the collaborative fixing process of vulnerabilities** to enable the ability of evaluating and continuously improving the collaboration model. For example, in the studied distributions, some of the collaborative work occurs via email (both public and private mailing lists) or other channels, including reporting a vulnerability to upstream, discussions before a fix is developed, and notifications of a vulnerability (with its fix). Some parts of the process happen under embargo, with some of the work being opened up afterwards. Similarly, some vulnerability fixes are applied locally in the distribution's code base, while others come in through new version updates of packages.

Hence, there is a lack of traceability from a particular vulnerability fix to all the phases of work involved. Since vulnerabilities have a large impact on software systems, having effective collaboration is necessary to develop vulnerability fixes quickly for vulnerable software systems and to minimize the losses of vulnerability exploits. Adding labels and corresponding times for the collaborative behaviors (e.g., receiving the notification of the discovery of a vulnerability) in issue tracking systems might be a first step towards improved traceability. Researchers could leverage this data to improve the collaboration model across distributions in the Linux ecosystem.

Embargoed high-severity vulnerabilities have quicker fixes for users than non-embargoed ones. Embargoes can reduce the risk of a vulnerability being exploited by attackers before the fix has been developed. Red Hat states the expected benefit of embargoes as *"we'll fix these issues in private and release the update in a coordinated fashion in order to minimize the time an attacker knows about the issue and an update is not available"* (The hidden costs of embargoes online). We found in Fig. 15a that the embargoed vulnerabilities have a higher impact on victim software systems. Figure 15c indicates that 74% of embargoed CSVs have been integrated into the distribution within 7 days after maintainers have been assigned the task, compared to 50% for non-embargoed CSVs. For the testing phase, Fig. 15d indicates that 37% of embargoed CSVs passed testing within 7 days, compared to 18% of non-embargoed CSVs.

However, this success comes at a cost, since fixing an embargoed vulnerability in secret requires additional costs (The hidden costs of embargoes online). For this reason, Red Hat has chosen to only embargo vulnerabilities having a high risk with respect to its products (The hidden costs of embargoes online).

7 Threats to Validity

7.1 Internal Validity

We study vulnerability management based on vulnerability reports, bug reports and security advisories in a given distribution. These three sources might not capture all activities related to fixing vulnerabilities in a distribution. For example, developers have other channels (e.g., emails) where they discuss vulnerabilities. Our dataset does not include vulnerabilities in the Long-Term-Support (LTS) versions of Debian releases since those are not recorded in Debian's issue tracker. In addition, the LTS versions are maintained by a group of volunteers and companies, instead of by Debian itself (Debian long term support online). Nevertheless, to the best of our knowledge, no prior work leverages the three data sources in a given distribution to draw insights into how its security team, maintainers and upstream projects collaborate to fix vulnerabilities. We encourage future work to add additional data sources to study developer communication on vulnerability management.

Our study focuses on vulnerabilities with a valid CVE-ID and ruled out those with an invalid CVE-ID.[38] These invalid CVE-IDs may still correspond to vulnerabilities to a certain extent but they were invalid due to a variety of reasons. For example, the CVE website stated that a CVE-ID could be rejected when further research determines the issue not to be a vulnerability.

In RQ2, we only study CSVs across the studied distributions. We did not consider vulnerabilities that are found only in one of the studied distributions, since our focus on two distributions only is not sufficient to conclude that such vulnerabilities would only impact one distribution. On the other hand, the studied distributions reflect the root distribution of two notable Linux distribution families (i.e., RPM-based and Deb-based) and both of them

[38] https://cve.mitre.org/cve/list_rules_and_guidance/correcting_counting_issues.html

have existed over a decade. Many prior works leveraged the data in these studied distributions to analyze security vulnerabilities (Russell et al. 2018; Harer et al. 2018). Future studies could consider adding other distributions to identify vulnerabilities that are specific to one distribution (as opposed to being common to other distributions) and why they are specific.

Our study involved a qualitative study performed by humans based on their experience and knowledge as well as the available data sources, (e.g., vulnerability reports, bug reports). To reduce the bias of our analysis, each vulnerability is labeled by two of the authors individually and discrepancies are discussed until a consensus is reached. The data sources might not capture all activities related to fixing vulnerabilities in the distribution. For example, Debian has its continuous integration (CI) system[39] perform automated tests against each package when it and/or its dependencies have been updated. However, we cannot link the logs in the CI system to vulnerability fixes because of the lack of explicit information on how/why the test was triggered. For example, a package can be tested because of a bug fix, a new feature, or its dependent packages that are updated.

7.2 External Validity

A threat to the external validity of our study is that we only studied vulnerability fixes and propagation in Debian and Fedora. However, Debian and Fedora are two popular Linux distributions that have existed for more than 10 years, as discussed in Section 4. In addition, there are Linux distributions (e.g., Arch Linux) using another type of release model, i.e., rolling release. Such distributions release a new version once their code has been updated, and might follow different practices to manage vulnerabilities. Thus, we cannot claim that our results generalize to other Linux distributions that are not studied in this work. Hence, future work should replicate this work to reach more general conclusions.

8 Conclusion

Our study focuses on 21,752 and 17,434 vulnerabilities in Debian and Fedora, respectively, to study the practices of vulnerability management within a distribution and across the distributions. Our analysis shows that Debian and Fedora maintainers integrate vulnerability fixes for the majority (50%) of vulnerabilities within 1 week. Fedora maintainers take a longer time to test vulnerability fixes in the vulnerability fixing process. From the user perspective, the time during which users are not aware of vulnerabilities is 85 times longer than the time during which users wait for vulnerability fixes after becoming aware, across Debian and Fedora. Users wait for less than 2 weeks for the fixes of 75% vulnerabilities after becoming aware.

Common security vulnerabilities (CSVs) are dominant across Debian (74%) and Fedora (94%). These CSVs are easy to exploit, i.e., they have a median exploitability score of 8.6, though with a medium impact (median of 5), according to CVSS score 2.0. Maintainers spend more time testing vulnerability fixes for CSVs than integrating vulnerability fixes by a median of 4.3 days across Debian and Fedora. Users wait for a median of 0.2 days to get fixes of CSVs after the releases of their respective security advisories.

[39] https://ci.debian.net/doc/

Finally, we conduct a qualitative study that investigates the sources through which maintainers become aware of CSVs and the practices with which maintainers manage CSVs. In general, upstream projects provide notifications and fixes for the majority of fixed CSVs to Debian (58%) and Fedora (59%). For the CSVs without an available fix, Fedora embargoes half of them to prevent early disclosure since embargoed CSVs have a higher impact score, according to CVSS score 2.0. Although the embargo policy makes maintainers become aware of vulnerabilities later than non-embargoed vulnerabilities, maintainers integrate and test fixes of the embargoed vulnerabilities faster than non-embargoed vulnerabilities. Additionally, Debian maintainers are more likely to integrate their own patches for CSVs, while Fedora maintainers tend to integrate new upstream releases.

Our study is a first step to aggregate and empirically study several data sources in Linux distributions to understand vulnerability management in practice. Our results suggest that distribution maintainers collaborate to develop vulnerabilities together before public disclosure, while they integrate fixes into their distributions in parallel. Future work could investigate additional practices by exploring other datasets (e.g., emails, developer communication channels, vulnerability fix releases) and interviewing maintainers. Future work could also investigate the practices of vulnerability management in upstream projects to have comprehensive insights of the collaboration model in the Linux ecosystem.

Acknowledgements We would like to thank the anonymous reviewers for their insightful comments.

Data Availability Statement The datasets generated during and/or analysed during the current study are available online.[40]

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

Adams B, Kavanagh R, Hassan AE, German DM (2016) An empirical study of integration activities in distributions of open source software. Empirical Software Engineering (EMSE'16) 21(3):960–1001

Al Sabbagh B, Kowalski S (2015) A socio-technical framework for threat modeling a software supply chain. IEEE Security & Privacy 13(4):30–39

Algarni A, Malaiya Y (2014) Software vulnerability markets: Discoverers and buyers. Int J Comput Inf Sci Eng 8(3):71–81

Alhazmi OH, Malaiya YK (2008) Application of vulnerability discovery models to major operating systems. IEEE Trans Reliab 57(1):14–22

Anderson R (2002) Security in open versus closed systems—the dance of boltzmann, coase and moore. Tech. rep., Technical report. Cambridge University, England

Bilge L, Dumitraş T (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 (CCS'12), pp 833–844

Christey S, Martin B (2013) Buying into the bias: Why vulnerability statistics suck. Blackhat, Las Vegas, USA, Tech, Rep 1

CVE (online) https://cve.mitre.org/. Last accessed: 2021-06-02

da Costa DA, Abebe SL, McIntosh S, Kulesza U, Hassan AE (2014) An empirical study of delays in the integration of addressed issues. In: Proc. of the 30th int'l conf. on software maintenance and evolution (ICSME'14), pp 281–290

[40]https://github.com/SAILResearch/suppmaterial-22-justina-vulnerability_management_in_linux_distributions

Debian continuous integration (online) https://ci.debian.net/doc/. Last accessed: 2021-06-02
Debian long term support (online) https://wiki.debian.org/LTS. Last accessed: 2021-06-02
Debian packages (online) https://packages.debian.org/stable/. Last accessed: 2021-06-02
Debian releases (online) https://www.debian.org/releases/. Last accessed: 2021-06-02
Debian security faq (online) https://www.debian.org/security. Last accessed: 2021-06-02
Debian security faq (online) https://www.debian.org/security/faq. Last accessed: 2021-06-02
Debian security team (online) https://security-team.debian.org/security_tracker.html. Last accessed: 2021-06-02
Debian vulnerability disclosure policy (online) https://www.debian.org/security/disclosure-policy. Last accessed: 2021-06-02
Duc AN, Cruzes DS, Ayala C, Conradi R (2011) Impact of stakeholder type and collaboration on issue resolution time in OSS projects. In: IFIP International conference on open source systems, pp 1–16. Springer
Ellison RJ, Goodenough JB, Weinstock CB, Woody C (2010) Evaluating and mitigating software supply chain security risks. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst
Fedora - packagekit items not found (online) https://docs.fedoraproject.org/en-US/quick-docs/packagekit-not-found/. Last accessed: 2021-06-02
Fedora - security basics (online) https://fedoraproject.org/wiki/SecurityBasics#Subscribing_to_Security_Announcement_Services. Last accessed: 2021-06-02
Fedora - security bugs (online) https://fedoraproject.org/wiki/Security_Bugs. Last accessed: 2021-06-02
Fedora - update policy (online) https://docs.fedoraproject.org/en-US/fesco/Updates_Policy/. Last accessed: 2021-06-02
Fedora package sources (online) https://src.fedoraproject.org/?page=1&sorting=None. Last accessed: 2021-06-02
Fedora release life cycle (online) https://fedoraproject.org/wiki/Fedora_Release_Life_Cycle. Last accessed: 2021-06-02
Foundjem A, Adams B (2021) Release synchronization in software ecosystems. Empir Softw Eng 26(3):1–50
Frei S, May M, Fiedler U, Plattner B (2006) Large-scale vulnerability analysis. In: Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense (LSAD'06), pp 131–138
Frei S, Tellenbach B, Plattner B (2008) 0-day patch exposing vendors (in) security performance. BlackHat Europe
Fruhwirth C, Mannisto T (2009) Improving CVSS-based vulnerability prioritization and response with context information. In: 2009 3Rd international symposium on empirical software engineering and measurement, pp 535–544. IEEE
Github - securing the world's software (online) https://octoverse.github.com/static/github-octoverse-2020-security-report.pdf. Last accessed: 2021-06-02
Guidelines and practices for multi-party vulnerability coordination and disclosure (online) https://www.first.org/global/sigs/vulnerability-coordination/multiparty/guidelines-v1.1. Last accessed: 2021-06-02
Harer JA, Kim LY, Russell RL, Ozdemir O, Kosta LR, Rangamani A, Hamilton LH, Centeno GI, Key JR, Ellingwood PM et al (2018) Automated software vulnerability detection with machine learning. arXiv:1803.04497
Huang Z, DAngelo M, Miyani D, Lie D (2016) Talos: Neutralizing vulnerabilities with security workarounds for rapid response. In: 2016 IEEE Symposium on security and privacy (SP'16), pp 618–635. IEEE
Jiang Y, Adams B, German DM (2013) Will my patch make it? and how fast? case study on the linux kernel 2013 10Th working conference on mining software repositories (MSR'13), pp 101–110. IEEE
Joh H, Malaiya YK (2011) Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics. In: Proceedings of the 2011 international conference on security and management (SAM'11), vol 1, pp 10–16
Kakimoto T, Kamei Y, Ohira M, Matsumoto K (2006) Social network analysis on communications for knowledge collaboration in OSS communities. In: Proceedings of the international workshop on supporting knowledge collaboration in software development (KCSD'06), pp 35–41. Citeseer
Klinke A, Renn O (2002) A new approach to risk evaluation and management: Risk-based, precaution-based, and discourse-based strategies 1. Risk Analysis: An Int J 22(6):1071–1094
Kula RG, German DM, Ouni A, Ishio T, Inoue K (2018) Do developers update their library dependencies? Empir Softw Eng 23(1):384–417
Li F, Paxson V (2017) A large-scale empirical study of security patches. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17), pp 2201–2215
LTS development (online) https://wiki.debian.org/LTS/Development#Prepare_security_updates_for_LTS. Last accessed: 2021-06-02

Ma W, Chen L, Zhang X, Zhou Y, Xu B (2017) How do developers fix cross-project correlated bugs? a case study on the github scientific python ecosystem. In: 2017 IEEE/ACM 39Th international conference on software engineering (ICSE'17), pp 381–392. IEEE

Nappa A, Johnson R, Bilge L, Caballero J, Dumitras T (2015) The attack of the clones: a study of the impact of shared code on vulnerability patching. In: 2015 IEEE Symposium on security and privacy, pp 692–708. IEEE

National vulnerability database (online) https://nvd.nist.gov/. Last accessed: 2021-06-02

Ohm M, Plate H, Sykosch A, Meier M (2020) Backstabber's knife collection: a review of open source software supply chain attacks. In: International conference on detection of intrusions and malware, and vulnerability assessment (DIMVA'20), pp 23–43. Springer

Operating system distribution security contact lists (online) https://oss-security.openwall.org/wiki/mailing-lists/distros. Last accessed: 2021-06-02

Ozment A, Schechter SE (2006) Milk or wine: does software security improve with age? In: USENIX Security symposium, vol 6

Ramsauer R, Bulwahn L, Lohmann D, Mauerer W (2020) The sound of silence: Mining security vulnerabilities from secret integration channels in open-source projects. In: Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop (CCSW'20), pp 147–157

Raymond E (1999) The cathedral and the bazaar. Knowledge, Technology & Policy 12(3):23–49

Reasons to use debian (online) https://www.debian.org/intro/why_debian. Last accessed: 2021-06-02

Ristov S, Gusev M, Donevski A (2013) Openstack cloud security vulnerabilities from inside and outside. Cloud Computing, 101–107

Russell R, Kim L, Hamilton L, Lazovich T, Harer J, Ozdemir O, Ellingwood P, McConley M (2018) Automated vulnerability detection in source code using deep representation learning. In: 2018 17Th IEEE international conference on machine learning and applications (ICMLA'18), pp 757–762. IEEE

Schryen G (2009) A comprehensive and comparative analysis of the patching behavior of open source and closed source software vendors. In: 2009 Fifth international conference on IT security incident management and IT forensics (IMF'09), pp 153–168. IEEE

Securing debian manual - before the compromise (online) https://www.debian.org/doc/manuals/securing-debian-manual/ch10.en.html#security-support-testing. Last accessed: 2021-06-02

Shahzad M, Shafiq MZ, Liu AX (2012) A large scale exploratory analysis of software vulnerability life cycles. In: 2012 34Th international conference on software engineering (ICSE'12), pp 771–781. IEEE

The hidden costs of embargoes (online) https://access.redhat.com/blogs/766093/posts/1976653. Last accessed: 2021-06-02

Telang R, Wattal S (2005) Impact of software vulnerability announcements on the market value of software vendors-an empirical investigation. Available at SSRN 677427

US national institute of standards and technology (online) CVSS information. https://nvd.nist.gov/vuln-metrics/cvss. Last accessed: 2021-06-02

Wang S, Nagappan N (2019) Characterizing and understanding software developer networks in security development. arXiv:1907.12141

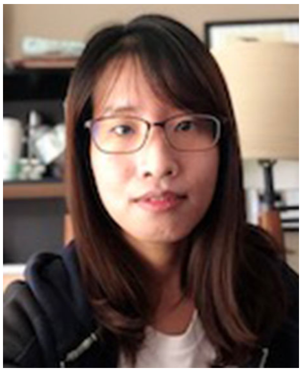
Wang X, Sun K, Batcheller A, Jajodia S (2019) Detecting "0-day" vulnerability: an empirical study of secret security patch in OSS. In: 2019 49Th annual IEEE/IFIP international conference on dependable systems and networks (DSN'19), pp 485–492. IEEE

Yilek S, Rescorla E, Shacham H, Enright B, Savage S (2009) When private keys are public: Results from the 2008 Debian openSSL vulnerability. In: Proceedings of the 9th ACM Conference on Internet Measurement (IMC'09), pp 15–27

Yin Z, Yuan D, Zhou Y, Pasupathy S, Bairavasundaram L (2011) How do fixes become bugs? In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (FSE'11), pp 26–36

Zaman S, Adams B, Hassan AE (2011) Security versus performance bugs: a case study on firefox. In: Proceedings of the 8th working conference on mining software repositories (MSR'11), pp 93–102

Zhang H, Wang S, Li H, Chen THP, Hassan AE (2021) A study of c/C++ code weaknesses on stack overflow. IEEE Transactions on Software Engineering (TSE'21)



Jiahuei Lin

Haoxiang Zhang

Bram Adams

Ahmed E. Hassan

Affiliations

Jiahuei Lin[1] · **Haoxiang Zhang**[1] · **Bram Adams**[2] · **Ahmed E. Hassan**[3]

Jiahuei Lin
jhlin@cs.queensu.ca

Bram Adams
bram.adams@queensu.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

1 Software Analysis and Intelligence Lab (SAIL), Queen's University Kingston, Kingston, Ontario, Canada

2 Lab on Maintenance, Construction, and Intelligence of Software (MCIS), Queen's University Kingston, Kingston, Ontario, Canada

3 Software Analysis and Intelligence Lab (SAIL), Queen's University Kingston, Kingston, Ontario, Canada