A Mixed-Methods Study of Open-Source Software Maintainers On Vulnerability Management and Platform Security Features

Jessy Ayala, Yu-Jye Tung, Joshua Garcia
University of California, Irvine

Abstract

In open-source software (OSS), software vulnerabilities have significantly increased. Although researchers have investigated the perspectives of vulnerability reporters and OSS contributor security practices, understanding the perspectives of OSS maintainers on vulnerability management and platform security features is currently understudied. In this paper, we investigate the perspectives of OSS maintainers who maintain projects listed in the GitHub Advisory Database. We explore this area by conducting two studies: identifying aspects through a listing survey ($n_1 = 80$) and gathering insights from semi-structured interviews ($n_2 = 22$). Of the 37 identified aspects, we find that supply chain mistrust and lack of automation for vulnerability management are the most challenging, and barriers to adopting platform security features include a lack of awareness and the perception that they are not necessary. Surprisingly, we find that despite being previously vulnerable, some maintainers still allow public vulnerability reporting, or ignore reports altogether. Based on our findings, we discuss implications for OSS platforms and how the research community can better support OSS vulnerability management efforts.

1 Introduction

The open-source software (OSS) ecosystem invites different kinds of reporters, contributors, and maintainers, creating a deeply integrated ecosystem crucial to the modern software supply chain. OSS projects range from software on which many others depend on to software that depends on many others. The pervasiveness of OSS is evident in a recent report by Synopsys [94], where they found that 96% of 1,067 codebases they scanned across 17 industries contained use of OSS.

Given the pervasiveness of OSS, OSS software vulnerabilities deep within the software supply chain can have a crippling effect on society [33, 37, 42]. At the forefront of defending against these vulnerabilities are OSS maintainers who oversee those projects, where many may not have experience handling a security incident [101]. OSS platforms such as GitHub offer various platform security features (e.g., dependency management [52]), but these features have been underutilized [11, 12, 14, 15]. With the many responsibilities of being a maintainer, some have felt burned out [67] or left their project completely [63]. The recent `xz-utils` incident [80] in which an XZ Utils maintainer, who joined the project two years earlier, maliciously introduced a vulnerability in the project further highlights the challenges existing maintainers face in vulnerability management and mistrust in the software supply chain. While several studies focus on the perspectives of vulnerability reporters [4, 6, 39, 60, 71, 75, 108] and the security practices of OSS contributors [57, 101], there exist knowledge gaps concerning the perspectives of OSS maintainers, especially those who own previously vulnerable projects.

In this paper, we conduct a mixed-methods study that systematically identifies and quantifies the factors that affect OSS project maintainers' involvement in conducting vulnerability management. Unlike prior work, we focus on recruiting OSS maintainers who maintain previously vulnerable projects, sourced from the GitHub Advisory Database and focusing on underused platform security features, and share experiences to allow researchers and OSS platforms to prioritize exploring efforts for improving the vulnerability management lifecycle, i.e., for those with little or no security background. To do so, we explore the following research questions:

RQ1: How do OSS maintainers with previously vulnerable OSS projects currently conduct vulnerability management? What are the challenges they face?

RQ2: Why are platform security features underutilized in previously vulnerable OSS projects? What are the challenges and barriers to adopting such features?

We conduct two studies, described in Section 3, to address our RQs: a survey ($n_1 = 80$) to list current practices, general vulnerability management challenges, platform security feature challenges, platform security feature barriers, and wanted

features or improvements, and conduct semi-structured interviews ($n_2 = 22$) study to contextualize overall survey results. Our primary contributions are as follows:

- We are the first to investigate vulnerability management challenges that OSS maintainers, whose projects have a history of patched vulnerabilities, face regarding platform security features using GitHub Advisory Database.
- We confirm previously discovered challenges, and further unveil novel challenges and barriers, OSS maintainers face when using current platform security features. Further, we systematically determine the breadth of security tooling and practices OSS maintainers use.
- We provide an in-depth discussion, including suggestions and actionable items, which have corresponding insights to improve the OSS maintainer security processes, and have made our questionnaires for both studies publicly available [2] to facilitate future research.

Section 4 describes participant demographics and their backgrounds. We present results in Section 5. In summary, supply chain trust and a lack of understanding are the top general challenges. Limited automation, vulnerability scoring, and missing CI processes or features for vulnerability management, are particularly challenging and understudied for platform security features. The most prominent barriers to adopting platform security features are a lack of awareness, poor usability of such features, the complexity of their setup and usage, and the perception that they are unnecessary. We also report wants from OSS maintainers to improve vulnerability management efforts, the most listed being assisted analysis and triaging, e.g., to automatically triage false positives, assisted platform security feature setup, e.g., setting up a security policy, and funding to be specifically used for security efforts, e.g., a bounty pool.

In Section 6, we discuss implications, i.e., based on significant challenges and barriers, for stakeholders involved in the vulnerability management lifecycle and future work that encourages researchers to develop tooling that minimizes regressions in the security context, advances current approaches working towards automated vulnerability scoring, and improving the usability of platform security features so that OSS maintainers have purpose to adopt them. Finally, we conclude in Section 7.

2 Related Work

The openness of OSS invites various areas of studies, including automatic commit message generation [30, 70, 73], the pull-request development model [43, 55, 98, 107], and historical analysis of vulnerabilities [38, 59]. To position our work in the body of literature on OSS, we focus our discussion of prior work through the following lenses: (1) vulnerability management in the OSS ecosystem; (2) the role that platform security features (PSFs) play in vulnerability management of the OSS ecosystem, including their benefits and challenges in adoption; and (3) challenges and support for OSS maintainers.

Vulnerability management in the OSS ecosystem Our study centers around vulnerability management since it plays an important role in OSS. Lack of effective vulnerability management can affect trust [9, 45], causing maintainers and users to abandon a project. Prior work had found current OSS vulnerability management to be insufficient by mining repositories [11, 12, 14, 17, 69] and analyzing commits [17] and analyzing security patches [25, 69]. Ayala et al. [11] found a lack of security policy for many GitHub repositories. Bandara et al. [17] and Li et al. [69] found that initial security fixes are often insufficient, requiring multiple fixes. Li et al. [69] further found that a third of the security issues remain in repositories for three years before remediation, indicating a potential lack of effective vulnerability management practices.

Prior work had also studied vulnerability management by interviewing security practitioners [9] and OSS maintainers [101]. Alomar et al. [9] studied vulnerability discovery and management processes from the perspective of organizations. They found that organizations may take on vulnerability management for compliance reasons rather than improving the security of their products, with some organizations even asking pen-testers (external security experts) to lower the severity rating of vulnerabilities they discovered. Wermke et al. [101] interviewed 27 OSS maintainers to understand their behind-the-scene processes (e.g., vulnerability management, processes for security and trust). Wermke found that most maintainers include a disclosure policy or contact for security issues in their projects. However, security plays a minor role in the many hats that maintainers wear: Only a small number of maintainers (5/27) knew of a security role in their projects, four maintainers did not have a disclosure policy or contract for security issues, and few maintainers had dealt with security incidents. Our work focuses on maintainers who have experienced security incidents and aims to understand how platforms like GitHub can better support vulnerability management and secure the OSS ecosystem.

The role of OSS platform security features (PSFs) in vulnerability management PSFs offered directly on the OSS platforms provide easy access for OSS maintainers to use for vulnerability management. PSFs such as dependency management tools (e.g., Dependabot [52], Renovate Bot [76], Greenkeeper [24]) notify maintainers of dependency updates (e.g., vulnerability fixes, version updates). Prior work [7, 40, 81] found Dependabot to be helpful in vulnerability management, where most vulnerable dependencies identified by Dependabot were addressed within several days [81] or a day [7]. GitHub also found that 60% more vulnerability-related, automated pull requests were merged in 2023 than in 2022 [53]. Mirhosseini et al. [79] found that maintainers update dependencies 1.6x more frequently with Greenkeeper than without because of the automated pull requests feature. However, they also found that the automated pull request feature can create too many pull requests, generating significant noise and causing notification fatigue. Others also found that the noise

from dependency management tools to be a significant pain point for OSS maintainers despite their benefits [22, 56, 103]. Wessel et al. [102] introduced the idea of a meta-bot to mitigate noise. He et al. [56] found that Renovate Bot is a popular migration target from Dependabot. One reason for this is Renovate Bot's auto-merge feature, which automatically merges pull requests for minor or patch dependency updates.

Aside from dependency management tools that retrospectively fix security issues, PSFs such as static code analysis tools can be used to proactively detect security issues; however, Ayala et al. [11] found that they are underutilized by OSS maintainers. One plausible explanation is the noise or false positives that static code analysis tools generate [64, 100, 105]. Our work confirms the aforementioned challenges of using PSFs, unveils further challenges, and provides suggestions to improve PSF adoption.

Two popular forms of support for OSS maintainers prior work studied are gamification [31, 82, 97] and donations [87, 92]. Dabbish [31] found that maintainers appreciate *"signals of attention,"* e.g., visible cues letting maintainers know that someone found their projects interesting. Trockman et al. [97] also found the use of gamification to be positive. In particular, badges such as quality assurance badges positively correlate with developers improving their test suites. They also speculate the usefulness of badges for bug fixing, stating, *"One can imagine other badges with gamification value [e.g., around bug fixing] being used in the future to encourage desirable practices."* However, Molden et al. [82] cautioned the use of gamification on GitHub, such as the daily activity streak features. They found that gamification may elicit unwanted behaviors, e.g., making contributions only to maintain activity streak. Besides gamification, another popular form of support is monetary incentives [87, 92]. Prior work found that GitHub Sponsors [92], a service provided by GitHub for maintainers to accept donations, are more effective than GitHub maintainers asking for donations using other donation platforms (e.g., PayPal, Patreon, Flattr) [87]. Our work provides guidance on how gamification and funding can be used to promote PSF adoption and improve vulnerability management practices.

Challenges and support for OSS maintainers Besides the challenges that afflict PSF usage, OSS maintainers face other challenges: toxicity [13, 63, 67, 77, 88] and limited personpower [67, 101]. Similar to other platforms such as Reddit, OSS platforms are also plagued by toxicity [77]. Toxic security reporters, even with good intentions, may jeopardize the timeliness of maintainers fixing reported issues [34]. Toxicity can strain maintainers emotionally [77], causing maintainers to burn out [67] or even leave the project [63]. In the context of dealing with bug bounty reports, the authors of [13] uncover that OSS maintainers find the diverted focus on money or CVEs from bug hunters and pressure to review reports the most challenging to deal with, especially with the threat of early vulnerability disclosures prior to patching. In addition to dealing with toxic reporters, which includes other maintainers [77], OSS projects, particularly smaller ones, are generally limited in personpower. On how to attract new contributors to increase personpower, Balali et al. [16] detailed 13 strategies. However, most of the one-time contributors have no intention to become a long-time contributor [68]. Unlike the aforementioned studies, our mixed-methods study focuses on the challenges arising for OSS maintainers with vulnerability management, especially platform security features involving the GitHub Advisory Database.

3 Methodology

In this section, we provide an overview of our study approach and the outline of the semi-structured interviews. We also discuss ethics, the qualitative coding process, report on our data collection, and discuss study limitations. We designed and conducted two studies to investigate our research questions. A listing survey study to determine factors (collected from 04/2024 to 07/2024) and an interview study (conducted from 05/2024 to 08/2024). The listing study allows us to understand current practices, challenges associated with, and needs of OSS project maintainers. We conduct a follow-up interview study to contextualize results.

Ethics To comply with GitHub's terms of service [50], we only reached out to maintainers who have publicly available contact information advertised as reachable to the general public, e.g., in text as a part of their profile introduction markdown, or through an external website, e.g., a personal homepage. Our institution's ethics review board approved both studies. Participants signed consent forms detailing study plans and participant rights before data collection. Further, our study is GDPR-compliant. We discuss ethics in recruiting specialized OSS maintainer populations in Section 8.

3.1 Listing survey study

To identify factors, e.g., challenges, that impact OSS vulnerability management practices, we conducted an online survey on OSS maintainers who own previously vulnerable projects using entries from the GitHub Advisory Database ($n_1 = 80$).

3.1.1 Participant recruitment and piloting

Before reaching out to OSS project maintainers who maintain projects with reviewed GitHub security advisories, we scrape the most recent 1,450 advisories per severity category, i.e., Low, Medium, High, and Critical, resulting in 5,096 advisories since there were under 1,450 Low reviewed advisories listed.

We filtered the advisories that were tied to projects hosted on GitHub using advisory metadata, resulting in just over 2,000 unique GitHub projects. Two researchers then manually examined project organizations and profiles for metadata, discarding projects that do not fall under our IRB-approved protocol, which excludes subjects who reside in OFAC-sanctioned countries and regions, e.g., the `qq.com` email suffix. Thus, leaving us with 1,920 unique GitHub projects to potentially

recruit from. To further comply with GitHub's terms of service [50], two researchers then manually examined GitHub profiles, discarding subjects who do not have publicly available contact information advertised as reachable to the general public, through an external website, or social media.

We piloted the survey with ten respondents and reviewed the quality of responses, i.e., making sure the instructions were clear to create a list for each free-response question, for the listing survey study described in Section 3.1.2. We then made wording updates to gather as many listing responses as possible to generate codes for categories shown in Table 2.

3.1.2 Survey details

For each question, we informed that study participants should list all tooling and factors they use when conducting vulnerability management. We use an open-ended listing approach for free-response questions called free-listing [18], common in research when a domain is understudied, to elicit a full breadth of responses. Next, we asked participants to self-report OSS maintenance and industry experience, if they have a security background, project funding, and how often their vulnerability management process is reviewed. We ended with demographic questions to understand our population. The full outline and questions asked can be found in our artifact [2].

Prior work has not systematically determined the breadth of security tooling and practices OSS maintainers use. Instead, it has studied specific tooling in specific contexts, e.g., dependency tooling [8, 56, 79], static code analysis [65, 100, 105], and what OSS stakeholders implement for incident-handling [11, 101], while we focus on OSS maintainers since they are the primary point-of-contact for handling vulnerabilities. Prior work informs our survey design, which explores the breadth of approaches, challenges, and tooling GitHub OSS maintainers use to improve their projects' security, which is currently understudied; our survey additionally draws from the *Getting started* GitHub security features guide [49] and established initiatives that inform OSS maintainers on how to approach vulnerability management, e.g., guides from OpenSSF [91].

3.1.3 Data analysis

We analyzed listing survey responses with exploratory open-coding [90]. Two researchers independently coded batches of ten responses at a time; further, resolving differences and updating the codebook after each batch. Because the survey lent itself to multiple listings per category, we were able to create an initial codebook from 27/80 (33.8%) respondents, resulting in 23/37 (62.2%) factors in Table 2. In particular, two researchers analyzed the same 34 participant responses by engaging in open coding [27] and discussing the initial emerging themes, meeting five times. Both researchers then independently coded the remaining 46 responses in batches of 6 and met frequently to discuss findings and reach a consensus, resulting in 14 emerging codes after the initial analysis. The final 11 responses were received after the final codebook was established, i.e., coders met synchronously to fit them within the appropriate existing codes,containing 37 codes in Table 2.

3.2 Interview study

We asked consenting and interested listing study participants to participate in remote semi-structured interviews to learn more about why the identified factors and tooling listed are especially important to them and how such listings fit in current vulnerability management practices ($n_2 = 22$).

3.2.1 Participant recruitment and piloting

Interviewees were a subset of the initial Listing survey study described in Section 3.1. If respondents were interested in participating in the interview study, they were directed to a Calendly [1] space where they could join a publicly available Zoom link during specified time ranges and, if so, provide the GitHub project they oversee. Of the 35/80 who responded *Yes*, 19/35 scheduled a Calendly meeting. Using emails for recruitment, we contacted the other 16/35: 6 scheduled a meeting, and 10 did not respond. Of the 25 who scheduled a meeting, 3 were no-shows, resulting in 22 interviews.

We conducted pilots with three OSS project maintainers to test our semi-structured interview protocol, which can be found in our artifact [2]. We revised our interview questions based on the interviewees' quality of responses to narrow our research focus. Details about participants regarding project information and interviews can also be found in our artifact [2].

3.2.2 Interview details and structure

We conducted semi-structured interviews with 22 OSS project maintainers, averaging 51 minutes and 59 seconds. Although survey participants were not compensated, interviewees were thanked with a virtual Visa $20 gift card.

For reporting, we group the interview into six sections, each consisting of 1-2 opening questions, corresponding follow-up questions, and circling back to earlier ideas when applicable. Before the main interview portion, we introduced our institution affiliations, an overall project overview, and our motivations. We went over how we only intended to keep audio, gathered consent for recording, and enabled closed captions and transcription. We proceeded as follows:

0. Project maintainer duties This section is intended to ease nervous participants into the interview and establish initial context to later combine with actual repository data.

1. Current vulnerability management practices This section explores structures that are not easily visible, such as who is involved in handling vulnerabilities, how vulnerabilities are *actually* handled, and participant perceptions of why approaches are appropriate for them (Section 5.1).

2. Challenges with vulnerability management This section explores past vulnerability management challenges encountered by our participants, e.g., through experiences, and how they have dealt with such challenges (Section 5.2).

3. Challenges with platform security features This section explores what overhead participants face with the PSFs they use and how such challenges affect their overall vulnerability management practices (Section 5.3).
4. Barriers to adopting platform security features This section explores why participants refrain from using PSFs and their perceptions of what such features offer (Section 5.4).
5. Opportunities for improvement and support This section explores participants' wants and needs for improving current PSFs and vulnerability management practices in general (Section 5.5).

3.2.3 Data analysis

All 22 audio-recorded interviews were transcribed and were checked for quality and accuracy by the researchers. Data was analyzed using thematic analysis [19], starting with open coding [27] using each transcript, and developing thematic codes with axial coding [27] to describe common arising themes, e.g., *Lack of awareness*. Two researchers then engaged in memo writing and constant comparison [54], and inductive analysis based on grounded theory [27]. Both researchers analyzed the same 6 transcripts by engaging in open coding [27] and discussing the initial emerging themes, meeting four times. During the analysis and iteratively discussing emerging themes across the 6 transcripts, both researchers engaged in axial coding [27], analyzing the remaining 16 transcripts in parallel, and met frequently to discuss findings and reach consensus. Researchers were open to emerging themes; if a theme was adopted after discussion, researchers returned to previous interviews and re-coded accordingly.

3.3 Limitations

Our methodology relies on self-reported data, which often entails substantial noise. To mitigate this, we investigate our research questions using two studies, consisting of one survey study and one interview study; meanwhile, we maximize face validity by piloting each stage of the study, and revising procedures with feedback. Further, not all of our sample participants likely have extensive experience with OSS vulnerability management; however, we expect some participants with less OSS vulnerability management participation will have similar experiences or various perspectives.

While we had a sufficient number of participants to conduct our studies, i.e., 80 survey respondents and 22 interviewees, we cannot claim our results can be necessarily generalized to the OSS project maintainer demographic. To mitigate this, we conduct two studies to ensure a holistic perspective, and our sample population varies in range regarding education level and years of involvement in managing OSS projects, as shown in Table 1. Further, although the GitHub Advisory Database is our primary source of recruitment for the listing and interview studies, 1,920 unique GitHub projects are sampled and range in popularity, some with as many as 185 thousand stars and 400 thousand listed dependent GitHub projects.

4 Participants

Table 1 summarizes participants' self-reported demographics and experiences. We had 80 participants in the listing study and 22 interviewees. Participants were mainly from Europe and North America and were overwhelmingly male. This is consistent with other qualitative studies with OSS stakeholders as participants, e.g., [61, 72]. Further, security experts from industry recommend organizations revisit their security policies and processes at least once a year [26, 95], which is consistent with half of our study participants.

In the listing survey and interview studies, our participants are OSS project maintainers who have received and patched vulnerabilities as seen in the GitHub Advisory Database [51]. We are especially curious about this subset of OSS project maintainers because although they have experience triaging vulnerabilities for projects they own, easily configurable platform security features are generally underused [12].

5 Results

In the following section, we report and discuss the results for 22 semi-structured interviews with open-source maintainers. In our reporting, we adhere to the structure of the interview guide described in Section 3.2. We report quotes as transcribed with minor grammatical corrections and omissions. e.g., using *"[...]"*, for readability.

We identified 37 factors from OSS project maintainers, whose projects have a history of vulnerabilities, regarding their vulnerability management efforts using five categories: current practices, general challenges, platform security feature challenges, platform security feature barriers, and platform security feature wants. The curated codes can be found in Table 2 and list relevant prior work for each code, if applicable.

Our study is closely related to a recent paper that investigated OSS security and trust practices of contributors, owners, and maintainers [101], i.e., general OSS stakeholders, regardless of vulnerability history or contributions; however, **we are the first to investigate vulnerability management challenges that OSS maintainers, whose projects have a history of patched vulnerabilities, face regarding platform security features involving the GitHub Advisory Database**, i.e., by recruiting maintainers whose projects are linked from GitHub security advisories. This particular perspective is important as OSS project maintainers typically do not have a security background, so learning from maintainers who have a history of vulnerability triaging provides a glimpse of progress towards security-awareness trickling throughout the OSS ecosystem. Further, using recommended platform security features and practices are key components of transparency and vulnerability awareness, e.g., by publishing security advisories after patching vulnerabilities, for OSS stakeholders. We hope to learn more about *how* and *why* such adoption is present, or not, in previously vulnerable projects.

		L	I
Gender	Man	71	19
	Woman	6	1
	Non-binary	3	2
Age	18-29	12	3
	30-39	28	8
	40-49	32	6
	50+	8	5
Residence	Australia	0	0
	Africa	1	0
	Europe	39	12
	Middle East	3	1
	North America	27	7
	South America	2	1
	South Asia	1	0
	Southeast Asia	1	0
	Other	6	1
Education	$\leq$ Completed high school	3	1
	Trade/technical/vocational	1	1
	College, no degree	8	2
	Associate's degree	1	1
	Professional degree	5	0
	Bachelor's degree	29	8
	Graduate degree	33	9
Years working in industry	< 1 year	4	1
	1-5 years	10	2
	5-10 years	4	1
	10+ years	62	18
Years in OSS maintainence	< 1 year	1	0
	1-5 years	16	4
	5-10 years	18	5
	10+ years	45	13
Has a security background	Yes	29	9
	No	51	13
Vulnerabilty management process review or update frequency	Every 1-3 months	10	1
	Every 6 months	11	3
	Every year	19	4
	Every 2+ years	8	1
	Never	32	13
OSS project has funding	Yes	31	7
	No	49	15
# of participants		**80**	**22**

Table 1: The number of participants across the two studies along with their respective demographics, various backgrounds, and project details. Acronyms listed represent, respectively: listing (**L**) survey study and interview (**I**) study.

5.1 Current practices used for vulnerability management

Both of our studies reflect that OSS project maintainers use a variety of tooling both in and out of the GitHub platform. Most encourage using a private avenue for reporting vulnerabilities while some are okay with using public channels, e.g., GitHub issues, for security bugs. Two listing study respondents indicated that they ignore vulnerabilties altogether.

Emails and external tooling or reports (F1 & F7) Using a form of a security contact email or mailing list consisting of project maintainers was the most listed form of current vulnerability management practices ($L = 50$, $I = 11$). This is consistent with a finding from Wermke et al., where authors identified that having a security contact point was the commonly mentioned aspect of a security policy [101].

Participants indicated that using email for security reports is sufficient because it discourages reporters from creating a public issue and acts as a private avenue for report resolution ($L = 11$, $I = 4$), e.g., *"we'll take time to really have a discussion and not just here's my report, okay, I dismiss it or I accept it. It's more like a conversation"* (P4). Others mention having security experts on their maintenance team ($L = 3$).

> *"With the projects I've been a part of, we just use one email address everybody can log into, which I suppose is just not a super secure way of managing that point of contact, but we all trust each other as maintainers. We have like a shared password on that email account, and that's just kind of how we do things."* (P12).

Listing participants and interviewees also mentioned using tooling outside of GitHub, including internal tools for static analysis ($L = 2, I = 3$), e.g., Coverity [93], and receiving external reports from companies ($L = 1, I = 2$). One interviewee described their relationship with a company that sends bundled vulnerability reports for their OSS project semi-annually:

> *"They send me a lot of bugs I never knew about and they give me time to fix it. Once fixed, they publish, and then they come back next year [...] I'm surprised with what they can find, I would like to know why they do that but I never complain. Be grateful, you know?"* (P17).

Proactive disclosure process (F2) Having an established disclosure process after patching vulnerabilities consisting of project maintainers was the second-highest form of current vulnerability management practices ($L = 46$, $I = 7$). This demonstrates a sense of responsibility within the OSS community since it ensures affected dependent client projects within reach are notified to upgrade [81].

Listing participants use different mediums to let the community know about the need to upgrade, which include sending messages to mailing lists and backchannels ($L = 7$), creating GitHub security advisories ($L = 6$), and in some cases, quickly requesting a CVE ($L = 5$). Interviewee participants were also wary of making sure users are aware of vulnerabilities ($I = 7$), two of which felt that creating a CVE is enough to ensure visibility to users and dependents. One interviewee

	F#	Factors and tooling	Description	RW	L	I
Current practices	F1	**Email or mailing list**	Uses a private email or mailing list for vulnerabilities.	[68, 101]	50	11
	F2	**Proactive disclosure**	Established disclosure process after a vulnerability patch.	[101]	46	7
	F3	**Security policy**	Instructs users how to report vulnerabilities.	[101]	37	10
	F4	**GitHub private reporting**	A built-in feature for privately reporting vulnerabilities.		20	9
	F5	**Automation tooling**	E.g., Dependabot, code scanning, secret scanning.	[7,35,56,79,81,86]	16	12
	F6	**GitHub Issues**	Vulnerabilities are reported publicly with GitHub Issues.	[29]	9	6
	F7	**External tooling**	Utilizes tooling or reports outside of GitHub.	[40, 56, 101]	3	2
	F8	**Ignores vulnerabilties**	Ignores vulnerability reports or does nothing in response.		2	–
General challenges	F9	**Supply chain trust**	E.g., waiting for upstream dependencies to deploy a fix.	[9, 31, 45, 85, 101]	36	6
	F10	**Lack of understanding**	E.g., knowledge gaps, how to start developing a patch.	[9, 13]	35	12
	F11	**Lack of time**	Balancing vulnerability priority with other OSS tasks.	[13, 67, 101]	25	9
	F12	**Lack of resources**	E.g., *"I do almost everything myself"* (P4).	[25, 101]	23	11
	F13	**CVE relationships**	Negative experiences with CVE-assigned vulnerabilities.		8	9
	F14	**Lack of procedure**	Maintainers are not sure how to deal with vulnerabilities.	[101]	7	5
	F15	**Disclosure coordination**	How to reach out to all users and what to disclose.	[9]	7	4
	F16	**Negative attitudes**	Past negative interactions with reporters or companies.	[34, 36, 63, 67, 77, 88]	6	9
PSF challenges	F17	**Not enough automation**	More automation needed in PSF functionality.	[23, 66]	39	11
	F18	**Too much noise**	E.g., too many false positives.	[22, 56, 79, 100–103, 105]	24	10
	F19	**Vulnerability scoring**	E.g., having a hard time calculating CVSS scores.		21	9
	F20	**CI processes are missing**	GitHub Actions & tests are not available in private forks.		8	7
	F21	**Broken tests & builds**	Failing builds & tests after vulnerability patch is applied.	[22, 79, 89, 105]	6	7
	F22	**Cluttered notifications**	E.g., vulnerable dependencies presentation is cluttered.		4	4
	F23	**Too much manual setup**	E.g., manually enabling PSFs on all projects.		3	3
PSF barriers	F24	**Lack of awareness**	E.g., not knowing about private vulnerability reporting.	[12]	33	12
	F25	**Complex to set up or use**	PSFs are too complex for maintainers with limited time.		26	7
	F26	**They are unnecessary**	Perception that using PSFs is not needed.		23	6
	F27	**Reputation concerns**	Past vulnerabilities reflect a negative project reputation.		11	4
	F28	**Bad UI presentation**	Hard to find or hidden, *"second-class features"* (P10).	[32, 40, 64]	8	6
	F29	**Lack of motivation**	Feeling burned out from other OSS tasks.	[10, 44, 67, 74, 88]	5	4
	F30	**Not sure what they do**	Unsure of PSFs' functionality, benefits, etc.		4	2
Maintainer wants	F31	**Assisted triaging**	Automation for impact analysis, patch development, etc.	[23, 66, 102]	39	9
	F32	**Assisted PSF setup**	More guidance for the PSF setup process.		37	10
	F33	**Security-specific funding**	Opportunities for funding OSS security efforts.	[101]	32	7
	F34	**User-friendly resources**	Tailored for those without a security background.		17	8
	F35	**Gamification for projects**	E.g., a green shield for projects with proper PSFs set up.	[31, 82, 97]	13	6
	F36	**PSF checklist**	To-do list for enabling easily configurable PSFs.		15	3
	F37	**Nudges or reminders**	Nudges reminding maintainers to consider using PSFs.		12	5

Table 2: 37 factors and tooling codes, shortened to preserve space, identified from the listing survey study and interview study divided into five categories as a result of software vulnerability management efforts. Acronyms listed represent, respectively: platform security features (**PSF**), existing related work (**RW**), the number of listing (**L**) survey study mentions ($n_1 = 80$), and the number of interview (**I**) study mentions ($n_2 = 22$). The extended table without omitted codes can be found in our artifact [2].

felt that disclosure is only necessary when a vulnerability is particularly important because there is *"a lot of bandwidth on security disclosures to begin with."* (P6).

Security policies (F3) The next current practice most listed is having a project security policy ($L = 37, I = 10$). OSS security policies have mostly been used to inform reporters how to properly communicate vulnerabilities to maintainers via security contact [101], but they are generally underused [11].

Although most listing participants mention using security policies to simply provide a security contact ($L = 11$), in some cases, maintainers link an organization-specific security policy published outside of GitHub ($L = 3$). One interviewee mentioned how they use security policies to explicitly discourage contributors from reporting vulnerabilities publicly

and provide multiple methods of private communication:

> *"We have a security policy in place where we say please do not report it publicly but try to contact me personally via email or send a mail to our security mailing list or create a security advisory on GitHub."* (P13).

Private vulnerability reporting (F4) Another GitHub platform security feature listing participants mentioned frequently is private vulnerability reporting ($L = 20, I = 9$). This particular feature allows contributors to privately report vulnerabilities within the GitHub platform. OSS maintainers can then review a report with functionalities to update its severity, invite others to develop a fix, and decide whether or not to request a CVE.

Listing survey participants and interviewees mention using the private forks from private vulnerability reporting to quietly develop fixes ($L = 6, I = 7$). Four interviewees mentioned how they like this feature because it provides an easy method of reporting security bugs and is quick to set up, i.e., by clicking enable. One interviewee describes private vulnerability reporting as *"very comfortable and very easy to use."* (P9). Others expressed satisfaction with private vulnerability reporting because everything is in a central space ($I = 2$), i.e., submitted reports are contained in GitHub.

Automation tooling (F5) Another method of current practices listing and interviewees mention is the usage of automated dependency analysis tooling and code scanning ($L = 16, I = 12$). Dependency analysis provides alerts of library upgrades, e.g., patched versions, while code scanning can reveal vulnerabilities, e.g., secrets, as a result of running static analysis.

Listing participants and interviewees mentioned using Dependabot as a primary source of upgrading dependencies ($L = 12, I = 6$). Others mentioned using the Renovate bot that can allow dependencies to be auto-merged if there are no code conflicts ($L = 4, I = 3$), reducing manual overhead:

> *"The first one that I use quite often is Renovate, that is a tool in Github easily available where you can configure: I want this and this upgraded like that, and you can have all kinds of settings and then it automatically gives you a pull request [...] with a dependency update and automatically, the test pipeline fires."* (P1).

However, participants describe how platform security features, which are recommended for effective vulnerability management, pose still too much manual effort (Section 5.3).

GitHub Issues (F6) Some listing participants and interviewees indicate using GitHub issues for reporting vulnerabilities ($L = 9, I = 6$). When a GitHub Issue is submitted, it is publicly accessible in the Issues tab located at the top of the project landing page.

Two listing participants in particular do not mind having vulnerabilities being reported publicly because of visibility, while one listing participant described how they use *"the same public PR process I use for any other issue."* One interviewee described how they do not mind using GitHub issues when a security bug is not determined to be severe to get help from the public, while another interviewee explained their rationale:

> *"If somebody reports a security issue [publicly], I don't really see a problem with it. Obviously it can be problematic, but I don't really care that much about how it is reported as long as I fix it in a short time."* (P19).

Ignoring vulnerabilties (F8) Two listing participants mention how they ignore vulnerabilities. One attributed this behavior to a lack of motivation for setting up any security features or tooling and indicated that *"financial support or sponsorship"* would be most beneficial for them to improve their vulnerability management practices. The other respondent listed *"unless the project is extremely sensitive - avoid implementing any policy"* and expressed the perception that having security features enabled for projects is unnecessary. Though this respondent listed that they were interested in participating in our interview study, they did not respond with any contact information or reach out to us directly.

Takeaway: Most OSS maintainers in our study take vulnerability management seriously, citing dedicated emails for private vulnerability reporting and established disclosure processes for informing affected projects after patching. However, built-in PSFs are not primarily used, indicating a need for usability improvement and further awareness.

5.2 Challenges with vulnerability management

All interviewees described wearing many hats when it comes to OSS maintenance, some of who are lone maintainers ($I = 11$). General vulnerability management challenges OSS maintainers face range from trusting the software supply chain to lack of time and resources to issues with CVEs. Participants also describe how a lack of standardized vulnerability handling procedures and implementing proper coordinated disclosure are prominent challenges. Some bring up the idea of having *"imposter syndrome"* when receiving vulnerability reports, while others share negative reporter experiences.

Supply chain trustworthiness (F9) Supply chain trust was the most listed challenge with vulnerability management ($L = 36, I = 6$). Use cases and duties related to the software supply chain in OSS include adopting upstream dependencies, tracking the status of dependencies, and updating dependencies in a timely manner, e.g., when a vulnerability is patched.

Specific challenges that the participants mentioned include the burden of keeping updated with dependencies and the latest vulnerabilities ($L = 20$) and dealing with unmaintained dependencies or delays in pushing a vulnerability fix ($L = 12$), which can be time-consuming and resource-intensive. One interviewee was particularly concerned about scenarios *"where people purposefully like to put security vulnerabilities into [OSS projects]"* (P6), highlighting the risk of malicious actors deliberately introducing vulnerabilities into upstream dependencies, e.g., the `xz-utils` incident (CVE-2024-3094) [80].

Lack of understanding (F10) The second most listed challenge for conducting vulnerability management is a lack of

understanding ($L = 35, I = 12$). Not being able to effectively understand reported vulnerabilities can lead to delays in patching and disclosure to affected dependent client projects.

Factors reported affecting a lack of understanding include complexity ($L = 19, I = 4$), developing a patch ($L = 7, I = 7$), testing ($L = 6, I = 6$), and knowledge gaps ($L = 5, I = 8$). One participant listed that *"some vulnerabilities can be complex and require extensive investigation and testing to ensure they are fully resolved without introducing new issues,"* reflecting platform security feature challenges we explore in Section 5.3, while one interviewee said *"my brain is far too small to understand [vulnerabilities]"* (P9). We explore potential avenues maintainers believe OSS platforms can take to minimize such knowledge gaps and awareness in Section 5.5.

Lack of time and resources (F11 & F12) The next most listed software vulnerability management challenges are the lack of time ($L = 25, I = 9$) and resources ($L = 23, I = 11$). The time OSS maintainers spend on developing and managing projects is limited, e.g., hobbyists. Further, resources are scarce, e.g., large-scale software ecosystems like Python Package Index and NPM are primarily made up of projects with a single maintainer [20, 21].

Challenges associated with a lack of time and resources include balancing prioritizing vulnerabilities with ongoing development tasks ($L = 21, I = 16$), the ability to triage reports promptly ($L = 15, I = 11$), and getting help from others ($L = 8, I = 5$). One interview participant summarizes:

> *"While I prioritize addressing vulnerabilities immediately, balancing this with ongoing development tasks can sometimes be challenging [...] There are limited resources and time available to address every reported vulnerability quickly."* (P17).

Negative relationships with CVEs (F13) Participants also expressed bad relationships with CVEs and the overall CVE process as a challenge when conducting vulnerability management ($L = 8, I = 9$). CVEs play an important role in the overall disclosure process and help promote transparency in the OSS ecosystem, e.g., upgrade notifications.

One listing participant explains their stance on CVEs:

> *"The CVE program suffers from many single points of failures: managed by the USA (not 24/7) hence a CVE ID cannot be delivered fast. CISA analysts backlog and [don't] have enough time and understanding of the system's complexity to properly analyze reports; thus, publish poor quality content. The whole process is outdated and must be reformed to a distributed approach to enable international sources of trust to work."*

Others described experiences where fixes were put on hold because of pending CVEs ($L = 3$) or feel intimidated by the nature of CVEs ($L = 2, I = 2$). In particular, one interviewee described their negative perception of CVEs and observation of other project maintainers' handling of CVEs:

> *"I'm incentivized to lie to make the CVE [severity] lower because it makes my project look bad, you have to be really, really honest [...] I noticed a lot of people like downgrade their CVEs."* (P15).

This quote highlights potential ethical dilemmas and pressures faced by OSS maintainers when dealing with CVEs, which can negatively affect the accuracy and reliability of severity metrics throughout the OSS ecosystem.

Lack of procedures and coordinated disclosure (F14 & F15) Though established procedures can help alleviate the receive-to-resolve timeline, the lack of procedures is a challenge OSS maintainers also face ($L = 7, I = 5$). Further, participants also express challenges with being able to effectively reach affected dependent projects and users ($L = 7, I = 4$).

OSS maintainers express a lack of recommended standardized processes for handling vulnerabilities ($L = 5, I = 1$) due to limited experience ($L = 2$), some even feeling intimidated ($L = 2$). Two interviewees described a sense of imposter syndrome when receiving vulnerabilities because *"there are so many things we need to watch out for, it's hard to stay confident that you're doing the right thing."* (P13). Further, participants expressed concerns about effectively disclosing the nature of the vulnerabilities ($L = 4, I = 7$) and a sense of mistrust with upstream projects not disclosing vulnerabilities properly ($L = 2, I = 2$). These perspectives suggest needs for clearer guidelines and better support for OSS maintainers in handling vulnerabilities, reported in Section 6, as well as further research on the adoption of and improving attestation practices, e.g., software-bill-of-materials, to foster upstream trust within the software supply chain [83, 84, 104].

Negative attitudes (F16) The least listed challenge of vulnerability management is negative attitudes from reporters ($L = 6, I = 9$). The OSS ecosystem is human-centered, e.g., via community and collaboration, and is not new to toxic interactions [77, 88]. One interviewee described experiences with companies demanding pentests. Participants mention the argumentative nature with vulnerability reporters ($L = 3, I = 7$) and those with sole intentions to make money ($L = 3, I = 5$).

> *"They give me this list of vulnerabilities in an email and then say, we're going to make it public in a week. And I think it's not so bad to make it public; in fact, both times I said go ahead and make it public right now. But the way people give you this problem and say fix it or else, it's not a very conducive environment."* (P2).

Takeaway: Aside from knowledge gaps and lack of time/resources, most OSS maintainers in our study are wary of adopting dependencies due to supply chain risks. Ethical dilemmas around disclosure and reporters/companies demanding fixes further complicate OSS responsibilities.

5.3 Challenges with platform security features

The adoption of platform security features (PSFs) promotes a robust vulnerability management process, especially if they are usable for OSS maintainers without a security background. In this section, we report various challenges and first-hand

experiences as a result of adopting such features, revealing why some maintainers may choose to disable them as a result.

Not enough automation (F17) The most listed PSF challenge described is that there is not enough automation ($L = 39, I = 11$). As described earlier in Section 5.2, OSS maintainers have limited time and resources to address every reported vulnerability quickly. Automation for security-oriented tasks helps promote easy security without additional overhead.

Participants express challenges regarding automation to update dependencies ($L = 24, I = 7$). In particular, five interviewees are okay with auto-merging dependency upgrades as much as possible, while others would prefer to do so on a library-by-library basis ($I = 2$). One interview participant mentioned auto-merging private vulnerability report fixes if the build does not break; however, private forks do not have CI integration and participants describe experiences with broken builds and tests, which we report later in this section.

One interviewee describes how they are okay with merging automated pull requests regardless of testing: *"Go ahead and merge the PR. I don't even really test it."* (P5). This perspective reflects a desire for streamlined processes, even if it comes at the cost of rigorous testing, underscoring the tension between automation convenience and careful oversight needs.

Too much noise and vulnerability scoring issues (F18 & F19) Listing participants mentioned that too much noise from PSFs ($L = 24, I = 10$) and issues from vulnerability scores ($L = 21, I = 9$), e.g., inaccuracy, are particularly challenging to deal with. Participants describe noise as many false alarms, different from spam or low-quality reports, and how noise can take time away from general OSS tasks and in some cases, overwhelm or scare OSS maintainers ($L = 4, I = 6$).

It is common knowledge that noises from dependency management [56, 79] and static analysis [64, 100] tools are problematic. A majority of noise mentioned is from dependency false positives ($L = 8, I = 6$), while others said it is from static analysis tooling ($L = 4, I = 2$), e.g., code scanning, and a general sense of annoyance from notifications ($L = 2, I = 2$). One interviewee mentioned how even proprietary scanners produce too much noise – we report remediations to help improve PSF setup and reduce noise in Section 5.5. Two interviewees mentioned how noise negatively impacts their motivation to continue maintaining OSS projects, one described:

> *"I left this project, I'm not doing this anymore [...] I think there's a lot of noise [...] and then the dedication, the love and the passion, the patience, going over it, and taking care of it, it's not easy at all. So security is kind of a second thought to most of us."* (P8).

In addition to noise, participants describe issues with vulnerability scores from reported vulnerabilities, e.g., CVE scores are inflated ($L = 5, I = 4$) and unsure how to correctly calculate or adjust CVSS scores ($L = 3, I = 8$). One interview participant summarizes their past experiences with attempting to properly calculate security metrics:

> *"I always have issues with calculating CVSS [scores]. All the documentation I found seems to be like for security researchers, or people who have experience with security incidents. There's nothing that just describes it in everyday terms that I can really wrap my head around, so that's one of the biggest difficulties."* (P3).

Absent CI processes in private forks, failing tests and broken builds in the public eye (F20 & F21) The next most listed challenge with PSFs is the lack of CI processes when developing fixes on a private fork ($L = 8, I = 7$), i.e., as a result of adopting the built-in private vulnerability reporting feature. Although private forks are recommended for working on fixes with reporters, not being able to run CI processes on such fixes in the same environment as merging pull requests on the main branch can be problematic, resulting in overhead.

To get around the lack of CI processes in private forks, participants have resorted to hosting additional private repositories outside of GitHub that mimic their public presence ($L = 2, I = 1$) or running potential fixes through build processes on personal machines ($I = 4$). Both of these solutions, while functional, introduce additional challenges and complexities, e.g., maintaining parallel environments and ensuring consistency across different setups. One interview participant described an elaborate approach:

> *"We have [someone], our traffic controller, and their role is to check that the incoming security reports [...] If they consider it something we need to be concerned with, they create an issue in a private project [...] It really slows the process down, because we have to merge one patch, then we have to go to our repo, pull in commits to the new one we need to release."* (P14).

Participants report experiences of failed tests or broken builds after patches are merged to the main public repository ($L = 6, I = 7$), so OSS maintainers must thoroughly ensure fixes do not introduce new issues. *"The biggest problem for us is the fact we can't run our automated tests on the fixes that we make on those forks."* (P14). This emphasizes a development workflow issue caused by the lack of automated testing capabilities in private forks, which is crucial for program repair and preventing regressions – a current multi-dimensional and challenging research area [23, 66, 78].

Cluttered notifications, setup and usage is too manual (F22 & F23) The least listed PSF challenges are cluttered notifications ($L = 4, I = 4$) and too much manual effort ($L = 3, I = 3$), i.e., usability challenges. Interviewees describe that the best security is *"easy security"* ($I = 6$), including the ease of use for PSFs. When security features are user-friendly and require minimal effort to manage, they can be effectively utilized by OSS maintainers without additional overhead.

Participants describe how dependency notifications are too cluttered ($L = 2, I = 3$), and how there should be a merge-all button, i.e., to reduce manual effort, for dependency upgrades that do not cause merge code conflicts ($L = 1, I = 3$).

> *"They are all grouped together at the bottom of your notification panel and they appear only if all the others are marked as done [...] just terrible to deal with [...]*

There is like a cron job that runs then delivers you all of those once a week, they're all bundled." (P10).

Other tasks described as too manual by interviewees include having to individually enable PSFs for each project ($I = 4$), e.g., as opposed to a centralized approach where all projects are presented on a single screen, and having to manually add the same collaborators to a private fork every time a vulnerability is privately reported ($I = 3$).

Takeaway: PSFs often rely on manual intervention and are noisy, requiring maintainers to sift through and triage security risks without sufficient automated support. Further, some PSFs are missing core CI processes, cause broken tests/builds, and are not UI-friendly, resulting in overhead.

5.4 Barriers hindering adoption of platform security features

Despite the importance of actively practicing secure coding practices and being knowledgeable about general security concerns, OSS developers typically do not have a security background. A majority of our participants also lack such background ($L = 50, I = 15$). In this section, we report possible barriers that hinder PSF adoption, including lack of awareness, complex setup procedures, concerns about project reputation, and perceptions that they are unnecessary.

Lack of awareness and bad UI presentation (F24 & F28) The most listed PSF barrier is lack of awareness ($L = 33, I = 12$). PSFs are available for usage so that OSS projects can have robust vulnerability management processes and proper disclosure upon vulnerability patching. This lack of awareness implies that although tools and processes are available to help promote a thriving secure OSS ecosystem, they are underutilized as indicated in prior work [11].

Participants mentioned how they were not aware of security policies ($L = 43, I = 8$), private vulnerability reporting ($L = 43, I = 10$), and public security advisories ($L = 34, I = 4$) or the GitHub advisory database ($L = 13, I = 6$). Further, 26.7% (21/80) of listing participants indicated that they have none of the three previously mentioned PSFs enabled. One interviewee expresses how *"the main problem is just that many people don't know about it"* (P18), and another interviewee, who has had eight CVE-assigned vulnerabilities in a project they manage, expressed curiosity after learning about what private vulnerability reporting has to offer:

"It's on my list to actually research now. Yeah, it looks like they've got a pathway to disclosure in CVEs that's integrated and seems like a good tool for us." (P20).

Related to lack of awareness, bad UI presentation is another PSF barrier participants mentioned ($L = 8, I = 6$). In particular, participants feel that PSFs are too hidden or hard to find ($L = 5, I = 6$) and that there are just too many features to potentially configure ($L = 2, I = 3$). One interviewee described how PSFs feel like *"second-class features"* (P10). This perception can discourage maintainers from exploring PSFs' capabilities, further contributing to their underuse.

"I mean, I'm just looking at how to or for a way to roll stuff out, but it's pretty hidden. I can roll out a policy that allows private vulnerability and a lot of other stuff, but you have to really click through and find it." (P11).

Complex to set up or use, not sure what they do (F25 & F30) Listing participants mentioned that PSFs are too complex to set up or use ($L = 26, I = 7$). Though not nearly listed as much, participants are *"not sure what they do in the first place"* ($L = 4, I = 2$). These perceptions reflect a barrier of adoption since OSS maintainers question their overhead and purpose. Interviewees express a sense of excessive overhead with setting up PSFs ($I = 4$). Others cite general ignorance as to why they are not aware of what particular PSFs do ($L = 3, I = 7$). As a result, they did not bother using PSFs.If users are unclear about what these features do or how they can benefit their projects, they are less likely to invest effort in setting them up, especially if they are perceived as complex.

"The biggest reason I never used them is they've never been pushed or the benefits of them sold to me [...] If it's really easy and simple to use, it'd be nice if that is kind of turned on by default on all projects." (P12).

Participants also reference previous experiences with trying out PSFs but refuse to look into additional offered tooling because of usage complexity ($L = 11, I = 4$), e.g., with dependency graphs ($L = 2$). These insights suggest simplifying the setup process and providing clearer information about the purpose and use of PSFs for OSS maintainers looking to adopt them to manage vulnerabilities.

Perception that they are unnecessary (F26) The next most listed PSF adoption barrier is the perception that they are unnecessary ($L = 23, I = 6$). When OSS maintainers perceive PSFs as not adding value to their current processes, they will likely refrain from investing additional time and resources into learning about or implementing these tools; thus, undermining recommended efforts to promote a secure OSS ecosystem.

Participants mention how PSFs are unnecessary for a variety of reasons, including the perception that projects lack importance ($L = 16, I = 6$), adding PSFs is overkill ($L = 5, I = 3$), or that maintainers do not value security ($I = 2$). These perceptions highlight a need for more documentation on the value of security measures ($L = 17, I = 8$) and mechanisms for PSFs to be tailored to fit specific project needs and scale.

"I haven't really needed anything more involved than GitHub issues [...] Security isn't something that we worry too much about. We're not ready to hear that message, even if GitHub does push me, I'll probably just skim over them, because I'm not ready to actually to, you know, get that message [...] We worry, we kind of have it in mind, but it's not our main goal." (P8).

Project reputation and a lack of motivation (F27 & F29) The remaining barriers to PSF adoption OSS maintainers mention are concerns with project reputation ($L = 11, I = 4$) and a lack of motivation ($L = 5, I = 4$). Concerns about maintaining

a positive public perception can deter OSS maintainers from implementing PSFs, that would otherwise enhance security, while those without motivation to adopt PSFs may deprioritize or ignore the use of PSFs. The attitude concerning reputation is not unique to maintainers. Alomar et al. [9] observed this attitude at the organizational level, where organizations try to lower the severity ratings of vulnerabilities in their products.

Participants point out fear of negative project reputation as a reason to not adopt PSFs ($L = 3, I = 3$), e.g., owning a project with previous high or critical CVE-assigned vulnerabilities ($I = 1$). On the other hand, some interviewees consider a project without vulnerabilities as suspicious ($I = 2$), i.e., patched vulnerabilities are a good sign of a "healthy project" (P14). One interviewee reflected on such a dilemma:

> *"It's always in the back of my mind when looking at an issue and seeing, should this go through the security advisory process? Or should it just be a normal PR and fix? That's the end of it. But that's wrong, and I know it. It still feels like, you know, hurting the reputation of my project. But it's wrong, I know it."* (P3).

When it comes to lack of motivation, participants cite "maintainer burnout" as the primary reason for not wanting to deal with vulnerabilities ($L = 3, I = 4$). This burnout can lead to a reduced willingness to engage with vulnerabilities, as maintainers may already be overwhelmed by other OSS tasks, e.g., high-stress levels from frequent demands for features and bug fixes [88]. Further, others see no benefit for improving project reputation through a security lens ($L = 1, I = 2$), leading them to deprioritize or disregard the adoption of PSFs.

Takeaway: Most OSS maintainers are not aware of PSFs; there are too many to look through and understand their relevance. Further, project reputation concerns, e.g., past vulnerabilities make projects look bad, and lack of motivation to manage vulnerabilities also hinder PSF adoption.

5.5 Opportunities for improvement & support

Lastly, we asked participants about success stories or positive outcomes as a result of their continuous vulnerability management efforts, if applicable, as well as supplemental features or improvements to current PSFs that would benefit their current vulnerability management approaches the most.

Assisted vulnerability analysis and triaging (F31) Participants expressed the most need for assisted vulnerability analysis and triaging ($L = 39, I = 9$), i.e., via automation mechanisms. In particular, participants felt that this would be useful for understanding vulnerabilities and their actual impact on OSS projects ($L = 34, I = 7$) to reduce false positives and noise described in Section 5.3, e.g., *"I hope a tool can take a context into consideration and it should help tooling avoid the false positives, and to trigger only this stuff when [vulnerable]"* (P7). Further, participants express a need for assistance with developing fixes and generating tests ($L = 21, I = 6$), which could streamline the remediation process and ensure more effective vulnerability management, e.g., *"generate, you know, tests with test cases or something like that, maybe as a way to lighten the burden of actually developing tests"* (P5).

PSF setup assistance, checklists, and nudges (F32, F36, & F37) Participants also showed interest in having assisted PSF setup ($L = 37, I = 10$). In particular, interviewees mention needs for assistance generating security policy content using project scope ($I = 3$), security tooling recommendations based on project contents ($I = 6$), and interactive guidance throughout the setup process ($I = 7$), e.g., *"it's a little bit not easy to use [...] I need to figure out, okay, what is the proper setup for my project [since] I'm of the mindset that brevity is king."* (P9). Further, participants mention wanting checklists and suggestions of things to do for recommended PSFs ($L = 15, I = 3$), alongside nudges or reminders and "easy" configurations as methods of encouraging others to strengthen their projects' security posture ($L = 16, I = 5$). One interview elaborates on their view of overall PSFs: *"It's pretty overwhelming, and you might just be like, why do I need any of these? Like, why is this important? Having a single button that just says, enable best practices, would go a long way."* (P16).

Security funding and cyber defense gamification (F33 & F35) The next most-listed want for participants is security-specific funding ($L = 32, I = 7$), i.e., funding to be used for security-related tasks and efforts. In particular, participants mention using funds for maintainers and reporters to get paid ($L = 18, I = 6$), funding a dedicated bounty pool for valid vulnerabilities ($I = 3$), or requesting annual reviews from security experts ($I = 2$). There was also interest in cyber defense gamification ($L = 13, I = 6$), i.e., to recognize projects that adopt vulnerability management processes. Four interviewees mention how they use reporter reputation to determine vulnerability report legitimacy. Such participants and others expressed interest in project reputation icons, e.g., a *"green shield"* ($I = 3$), for projects that adopt recommended PSFs and tooling ($I = 7$); thus, recognizing and rewarding projects that exhibit a proper security posture. *"Everyone wants to have the green shield [...] This community health status people see, stuff like that, like you'll be surprised by how many people want to feel those bars."* (P10).

User-friendly resources and documentation (F34) Lastly, participants expressed interest in having more user-friendly documentation and resources ($L = 17, I = 8$), some citing general *"ignorance"* of proper OSS security ($I = 6$). In particular, participants mention OSS recommended training material for *"normie developers"* ($L = 3, I = 4$), e.g., free webinars ($I = 2$), redirection to related projects with implemented security practices ($L = 7, I = 3$), and *"easier directions"* for setting up and what to expect from PSFs ($I = 6$), e.g., not being *"scared"* of false positives ($I = 3$). *"Security best practices and tooling and all of that is tribal knowledge, and it shouldn't be."* (P12).

Takeaway: Most OSS maintainers want automated vulnerability triaging and PSF support for less overhead when managing vulnerabilities. Further, gamification and context-aware checklists can help simplify security efforts and encourage proactive engagement with security best practices.

6 Discussion

In this section, we present additional analyses of the challenges identified and presented in Section 5,implications for OSS platforms and OSS maintainers to alleviate prominent challenges we discovered, explore areas for reducing PSF barriers, and expand the benefits participants expressed as helpful for vulnerability management. Further, we present potential research directions aimed at supporting OSS project maintainers during the vulnerability management lifecycle.

6.1 Bridging factors to reveal underlying issues

OSS maintainers described challenges from vulnerability management and their experiences with PSFs. Uncovering underlying issues can help identify areas of need, why particular PSFs may not be adopted, and inform future solutions.

OSS maintainers described challenges that hinder their ability to take necessary actions from PSF output rooted in overwhelming security duties and missing PSF support that would help vulnerability triaging. Dependency trust issues (F9), e.g., too many false positives from upstream dependencies (F18) and cluttered update notifications (F22 & F28), cause fatigue (F29) and a disinclination to continue PSF adoption. Further, maintainers expressed struggle with developing patches (F10, F14, & F31) since core CI/CD processes are missing from some PSFs (F20) that cause broken tests/builds (F21); thus, leading maintainers to deem PSFs as unnecessary (F26), deploy custom behind-the-scenes processes that can cause burnout (F29), and become overwhelmed in the CVE process (F13, F15, & F16). Without any clear prioritization or impact analysis, some PSFs may seem more like distractions, and the lack of trust demonstrated by maintainers is a reinforcing factor in the perception that PSFs are unnecessary, especially if they seem to add little value to the project. Addressing this concern means incorporating features that would be perceived to provide real value, e.g., timely patching, and the use of gamification or funding (F33 & F35) and checklists (F36) can also be used to reframe these tasks as rewarding and meaningful, promoting a positive narrative for security.

OSS maintainers also described challenges that hinder their ability to adopt PSFs rooted in misunderstanding and misinterpretation of PSFs' configurations, use cases, and purposes. Knowledge gaps (F10), e.g., around vulnerability scoring procedures (F19), hinder effective prioritization and remediation efforts. Resource constraints (F12), including limited time (F11) and lack of automation (F17), exacerbate these difficulties, often leading to delays in addressing vulnerabilities or adopting PSFs; further, the complexity (F25) and lack of awareness (F24) surrounding PSFs create additional barriers. If OSS maintainers do not fully understand the benefits or use cases for PSFs (F30), or find them confusing, they are unlikely to explore or enable them (F37). Training resources that simplify security concepts and integrate clear documentation (F32 & F34) for PSFs could bridge this gap, empowering maintainers to overcome these hurdles and integrate PSFs effectively into their workflow.

6.2 Implications & future directions based on collective challenges

OSS maintainers mentioned supply chain trust and lack of understanding as the most generally challenging in our listing study, while not enough automation and too much noise were the most listed PSF-specific challenges.

When maintainers wait for upstream dependencies to fix a vulnerability, there is a delay in addressing security issues within their projects, thus exposing OSS projects to potential exploits. Tools that automate identifying and prioritizing vulnerabilities can help maintainers quickly determine which issues require immediate attention and which can be deferred. To support assisted vulnerability analysis and triaging, tooling should be explored that implements mechanisms for automatically identifying the impact of vulnerabilities using source code as context, i.e., to reduce noise from security tooling. To that end, future research can leverage LLMs to (1) help OSS maintainers with interpreting reported vulnerabilities, e.g., by leveraging OSS security datasets like those curated by OpenSSF [44], and (2) help OSS maintainers generate patches for reported vulnerabilities with minimized regression tests by using the contents of disclosed reports. Although LLMs are useful for tasks like code summarization and generation [3, 106], uses of LLMs pose challenges. For instance, LLMs may misinterpret security reports or generate incomplete/inaccurate patches, leading to regressions; further, there may be hesitance to trust AI-generated suggestions because LLMs' decision-making processes are uninterpretable [58].OSS platforms should consider providing CI feature capabilities in private forks so OSS maintainers can expedite fixing vulnerabilities and reduce costs of regression tests, which can be very high [28, 62], indicating a need for research focused on regression testing for security. GitHub disallows private forks from using CI features for security purposes [48], but OSS maintainers desire such features in our studies, especially for fixing vulnerabilities, suggesting the need for further research on secure CI features for vulnerability management. This might involve sandboxing or access controls, e.g., where private forks can use CI features under controlled conditions.

Other prominent vulnerability management challenges OSS project maintainers face are negative CVE relationships and vulnerability scoring. Collectively, these aspects may lead to undermining or misreporting critical vulnerabilities; as a consequence, polluting the software supply chain with inconsistencies. Tooling that can take context into account when a

vulnerability exists, including the deployment environment, the project's purpose, and how the vulnerability interacts with other components, would be especially beneficial for working toward problems in automated vulnerability scoring, which is particularly understudied, as far as we are aware.

6.3 Implications & future directions based on PSF adoption barriers

The most listed PSF barriers by OSS maintainers were lack of awareness, complexity, and perception of being unnecessary were the most prominent barriers; thereby, preventing PSF adoption as a part of vulnerability management processes. Notably, of the twelve interview participants who described a lack of PSF awareness, five said they would enable them as a result of just learning about their existence.

We encourage OSS maintainers to enable PSFs, e.g., especially those that are togglable by nature, such as private vulnerability reporting, and recommend OSS platforms provide an *Enable best practices* button for PSFs that require no-to-minimal setup and include links to respective PSFs that are user-friendly for OSS maintainers without security backgrounds. Research challenges for enabling such a button include automatically analyzing the context of the OSS project, including its application-specific code and its dependencies. Another research challenge of such a button would be conducting a developer study to determine the best manner in which to incorporate developer preferences into the automation of best PSF practices provided by the button.

Our study further demonstrates a wide variety of OSS maintainers' views as to the extent to which vulnerabilities should be publicized, especially before a fix is provided. That spectrum includes a preference for using email due to the more inherent privacy compared to public GitHub issues or having to deal with the usability issues involving PSFs. Although GitHub discourages public vulnerability reporting [47] and, hence, encourages using the private vulnerability reporting PSF [96], our study provides little evidence to back up such a recommendation from OSS maintainers' perspectives.

Nevertheless, preventing premature disclosure of a potentially severe and easily exploitable vulnerability, which may even be particularly difficult to fix, is a potentially strong argument backing GitHub's recommendation. To that end, feature research can help automatically identify opportunities for beneficial PSF usage. For example, when a user tries to submit a public issue mentioning keywords like "vulnerability" or "security", OSS platforms could display a warning and automatically nudge maintainers to encourage enabling lightweight PSFs that require minimal effort to implement.

Other future research opportunities encouraging the use of private vulnerability reporting include producing a convincing means or incentive of recommending using PSF features (e.g., by gamifying the PSFs with badges or achievements) or even enforcing the use of PSFs that OSS platform providers recommend. As an example of such enforcement, future research can automatically convert issues submitted to the public that are likely vulnerabilities or may simply be bugs that, as identified through static reachability analysis, are reachable on the project's attack surface. Such research would go beyond the coarse-grained identification of vulnerabilities in a GitHub project's dependencies by helping maintainers triage potential vulnerabilities as early as possible with higher confidence (e.g., as provided through a static or dynamic analysis during report submission). Such a future approach would help alleviate the steep learning curve of understanding software security while promoting the adoption of PSF features that can reduce exploitation of publicly-reported OSS vulnerabilities.

OSS platforms and researchers should also consider working towards functionalities that (1) improve the quality and presentation of security notifications; (2) automatically inform contributors about if their issue should be filed as a vulnerability report based on past OSS vulnerabilities; (3) provide PSF setup assistance, e.g., generating a context-aware security policy;and (4) gamification features to reward projects with recommended security posture and be designed in such a way that creates a positive reputation. Research in such directions can focus on creating better-designed PSFs to provide more value to OSS maintainers, such that PSFs are easier to use, provide guidance to reporters, and promote positivity with their adoption. Future research can also investigate how to effectively use gamification to disseminate user-friendly resources on proper OSS security, allowing any maintainers to effortlessly learn security best practices. For example, a project reputation icon, e.g., "green shield", for enabling best-practices PSFs can come with documentation on why each PSF should be enabled. Further, PSFs can have a leaderboard of many kinds: maintainers who followed security best practices most, reporters who provide the best-rated (e.g., by maintainers) vulnerability-oriented issues, reporters who provide the easiest-to-understand and most convincing fixes (e.g., avoiding regressions while ensuring security), and even OSS projects with the best security-focused documentation. This can further motivate maintainers to engage with security-related gamification features and available security resources.

7 Conclusion

In this paper, we conduct a mixed-methods study, i.e., one listing survey and semi-structured interviews, to investigate challenges and barriers by systematically identifying and quantifying the factors that affect how OSS maintainers, who manage previously vulnerable projects, conduct vulnerability management. OSS project maintainers find trusting the supply chain and lack of automation to be the most challenging aspects of vulnerability management. Further, a lack of awareness, complexity, and perception of being unnecessary are the most prominent barriers to setting up platform security features. Based on our findings, future work should investigate secure CI features for vulnerability management, improving PSF usability, and automated vulnerability management tooling.

8 Ethics considerations

Our institution's ethics review board approved both studies. Participants signed consent forms detailing study plans and participant rights before data collection. Further, our study is GDPR-compliant. We exclude subjects who reside in an OFAC-sanctioned region and/or are affiliated with an OFAC-sanction entity, as required by our institution's ethics review board for the nature of an international, online study.

Participants were asked to familiarize themselves with consent and data handling information on a study information sheet before agreeing to participate in our studies. Considering the nature of our questions regarding vulnerability-related incidents, in particular to interviewee participants, we state upfront how participants skip any questions, e.g., that they may not be comfortable with answering, or end the interview anytime. Lastly, we shared a preprint of our study to ensure they were okay with our use of their quotes and referencing respective conversations.

We did not monetarily compensate survey participants because of (1) our funding limits as academic researchers, and (2) similar to other qualitative studies, compensating international participants is logistically difficult. We focused on creating a study that provided intrinsic value to participants through meaningful engagement and insights. Further, (3) we reduced the time cost of completing our survey by allowing participants to skip any free-response questions and encouraged them not to spend more than 15 minutes filling out our survey, and (4) survey participation is voluntary, OSS participants often support academic research out of intrinsic motivation [5, 41, 46] to help improve the security posture of the OSS ecosystem through our study's results. This approach ensured that participation gives a sense of purpose and contributing to the greater good, i.e., for further securing the OSS ecosystem, despite the lack of monetary compensation for survey participants. However, each of our 22 interviewees was monetarily compensated $20.

The most recent human-centered security paper that recruited GitHub subjects, published at PETS, mined commit information for maintainer emails [99]. After consulting with ethics reviewers, they concluded that moving forward, researchers should *"only use contact information that has visibly been made public by the individuals themselves with the intention of allowing the general public to contact them [since] GitHub's email address mechanics and users' lack of knowledge about them had neither been mentioned nor addressed by previous work that used public GitHub repositories for recruitment"* [99]. To comply with GitHub's terms of service [50] and following the PETS paper statement, we only reached out to maintainers who have publicly available contact information advertised as reachable to the general public, e.g., in text as a part of their profile introduction markdown, or through a self-hosted website, e.g., a personal homepage.

9 Open science

We make our listing study survey questions, semi-structured interview guide, detailed interview participants' information, and an extended table of codes available in our artifact [2].

References

[1] Calendly. https://calendly.com/.

[2] Repository for open science. https://zenodo.org/records/14721125, 2024.

[3] Toufique Ahmed and Premkumar Devanbu. Few-shot training llms for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Omer Akgul, Taha Eghtesad, Amit Elazari, Omprakash Gnawali, Jens Grossklags, Michelle L. Mazurek, Daniel Votipka, and Aron Laszka. Bug hunters' perspectives on the challenges and benefits of the bug bounty ecosystem. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA, 2023. USENIX Association.

[5] Shaosong Ou Alexander Hars. Working for free? motivations for participating in open-source projects. *International Journal of Electronic Commerce*, 6(3):25–39, 2002.

[6] Nikolaos Alexopoulos, Andrew Meneely, Dorian Arnouts, and Max Mühlhäuser. Who are vulnerability reporters?: A large-scale empirical study on floss. pages 1–12, 10 2021.

[7] Mahmoud Alfadel, Diego Elias Costa, Emad Shihab, and Mouafak Mkhallalati. On the use of dependabot security pull requests. In *2021 IEEE/ACM 18th International conference on mining software repositories (MSR)*, pages 254–265. IEEE, 2021.

[8] Mahmoud Alfadel, Diego Elias Costa, Emad Shihab, and Mouafak Mkhallalati. On the use of dependabot security pull requests. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 254–265, 2021.

[9] Noura Alomar, Primal Wijesekera, Edward Qiu, and Serge Egelman. " you've got your nice list of bugs, now what?" vulnerability discovery and management processes in the wild. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 319–339, 2020.

[10] Idan Amit and Dror G. Feitelson. A large scale survey of motivation in software development and analysis of its validity, 2024.

[11] Jessy Ayala and Joshua Garcia. An empirical study on workflows and security policies in popular github repositories. In *2023 IEEE/ACM 1st International Workshop on Software Vulnerability (SVM)*, pages 6–9, 2023.

[12] Jessy Ayala, Steven Ngo, and Joshua Garcia. Poster: Towards understanding the underuse of security features in open-source repositories. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, 2024.

[13] Jessy Ayala, Steven Ngo, and Joshua Garcia. A deep dive into how open-source project maintainers review and resolve bug bounty reports. In *2025 IEEE Symposium on Security and Privacy (SP)*, 2025.

[14] Jessy Ayala, Yu-Jye Tung, and Joshua Garcia. Poster: A glimpse of vulnerability disclosure behaviors and practices using github projects. In *Proceedings of the 45th IEEE Symposium on Security and Privacy (S&P)*, 2024.

[15] Jessy Ayala, Yu-Jye Tung, and Joshua Garcia. Investigating vulnerability disclosures in open-source software using bug bounty reports and security advisories, 2025.

[16] Sogol Balali, Umayal Annamalai, Hema Susmita Padala, Bianca Trinkenreich, Marco A Gerosa, Igor Steinmacher, and Anita Sarma. Recommending tasks to newcomers in oss projects: How do mentors handle it? In *Proceedings of the 16th International Symposium on Open Collaboration*, pages 1–14, 2020.

[17] Vinuri Bandara, Thisura Rathnayake, Nipuna Weerasekara, Charitha Elvitigala, Kenneth Thilakarathna, Primal Wijesekera, and Chamath Keppitiyagama. Fix that fix commit: A real-world remediation analysis of javascript projects. In *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 198–202. IEEE, 2020.

[18] H.R. Bernard. *Research Methods in Anthropology: Qualitative and Quantitative Approaches*. Rowman & Littlefield Publishers, 2017.

[19] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[20] Josh Bressers. npm maintainers bar chart of maintainers. https://user-images.githubusercontent.com/1692786/169064720-aeb04238-c618-4d5e-a964-0f563b87210e.png, 2022.

[21] Josh Bressers. Pypi maintainers bar chart of maintainers. https://user-images.githubusercontent.com/1692786/169667411-0107650f-deb2-4a4a-ae32-5f5f785cc3d1.png, 2022.

[22] Chris Brown and Chris Parnin. Sorry to bother you: Designing bots for effective recommendations. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 54–58. IEEE, 2019.

[23] Quang-Cuong Bui, Ranindya Paramitha, Duc-Ly Vu, Fabio Massacci, and Riccardo Scandariato. Apr4vul: an empirical study of automatic program repair techniques on real-world java vulnerabilities. *Empirical Software Engineering*, 29(18), 2024.

[24] Stephan Bönnemann and Christoph Witzko. Greenkeeper. https://github.com/greenkeeperio/greenkeeper, 2015.

[25] Gerardo Canfora, Andrea Di Sorbo, Sara Forootani, Antonio Pirozzi, and Corrado Aaron Visaggio. Investigating the vulnerability fixing process in oss projects: Peculiarities and challenges. *Computers & Security*, 99:102067, 2020.

[26] Foster Charles. Knowing when your security policies need updating. https://blog.charlesit.com/knowing-when-your-security-policies-need-updating, 2022.

[27] Kathy Charmaz. Constructing grounded theory. 2014.

[28] Pavan Kumar Chittimalli and Mary Jean Harrold. Recomputing coverage information to assist regression testing. *IEEE Transactions on Software Engineering*, 35(4):452–469, 2009.

[29] Kattiana Constantino, Mauricio Souza, Shurui Zhou, Eduardo Figueiredo, and Christian Kästner. Perceptions of open-source software developers on collaborations: An interview and survey study. In *Journal of Software: Evolution and Process*, 2021.

[30] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. On automatically generating commit messages via summarization of source code changes. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, pages 275–284. IEEE, 2014.

[31] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*, pages 1277–1286, 2012.

[32] Anastasia Danilova, Alena Naiakshina, and Matthew Smith. One size does not fit all: a grounded theory and online survey study of developer preferences for security warning types. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 136–148, 2020.

[33] Baden Delamore and Ryan KL Ko. A global, empirical analysis of the shellshock vulnerability in web applications. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1129–1135. IEEE, 2015.

[34] Giuseppe Destefanis, Marco Ortu, Steve Counsell, Stephen Swift, Michele Marchesi, and Roberto Tonelli. Software development: do good manners matter? *PeerJ Computer Science*, 2:e73, 2016.

[35] Jens Dietrich, Shawn Rasheed, Alexander Jordan, and Tim White. On the security blind spots of software composition analysis. *arXiv preprint arXiv:2306.05534*, 2023.

[36] Alan Diggs. 'windows is sh*t:' linux users and the technical superiority problem. https://medium.com/linuxforeveryone/windows-is-sh-t-linux-users-and-the-technical-superiority-problem-196a597aa860, 2021.

[37] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488, 2014.

[38] Nigel Edwards and Liqun Chen. An historical examination of open source releases and their vulnerabilities. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 183–194, 2012.

[39] Ryan Ellis and Yuan Stevens. Bounty everything: Hackers and the making of the global bug marketplace. *SSRN Electronic Journal*, 01 2022.

[40] Felix Fischer, Jonas Höbenreich, and Jens Grossklags. The effectiveness of security interventions on github. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2426–2440, 2023.

[41] Konstantin Fischer, Ivana Trummová, Phillip Gajland, Yasemin Acar, Sascha Fahl, and Angela Sasse. The challenges of bringing cryptography from research papers to products: Results from an interview study with experts. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 7213–7230, Philadelphia, PA, 2024. USENIX Association.

[42] Benjamin Fogel, Shane Farmer, Hamza Alkofahi, Anthony Skjellum, and Munawar Hafiz. Poodles, more poodles, freak attacks too: how server administrators responded to three serious web vulnerabilities. In *Engineering Secure Software and Systems: 8th International Symposium, ESSoS 2016, London, UK, April 6–8, 2016. Proceedings 8*, pages 122–137. Springer, 2016.

[43] Denae Ford, Mahnaz Behroozi, Alexander Serebrenik, and Chris Parnin. Beyond the code itself: How programmers really look at pull requests. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 51–60. IEEE, 2019.

[44] The Linux Foundation. Open secure software foundation (openssf) working groups. https://openssf.org/community/openssf-working-groups/, 202.

[45] Marcel Fourné, Dominik Wermke, William Enck, Sascha Fahl, and Yasemin Acar. It's like flossing your teeth: On the importance and challenges of reproducible builds for software supply chain security. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1527–1544. IEEE, 2023.

[46] Marco Aurélio Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. The shifting sands of motivation: Revisiting what drives contributors in open source. *CoRR*, abs/2101.10291, 2021.

[47] GitHub. About coordinated disclosure of security vulnerabilities. https://docs.github.com/en/code-security/security-advisories/guidance-on-reporting-and-writing-information-about-vulnerabilities/about-coordinated-disclosure-of-security-vulnerabilities#standard-process.

[48] GitHub. Collaborating in a temporary private fork to resolve a repository security vulnerability. https://docs.github.com/en/code-security/security-advisories/working-with-repository-security-advisories/collaborating-in-a-temporary-private-fork-to-resolve-a-repository-security-vulnerability.

[49] GitHub. Github security features. https://docs.github.com/en/code-security/getting-started/github-security-features.

[50] GitHub. Github terms of service. https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies.

[51] GitHub. Github security advisory database. https://github.com/advisories, 2017.

[52] GitHub. Dependabot: Automated dependency updates built into github. https://github.com/dependabot, 2019.

[53] GitHub. The state of the octoverse. https://octoverse.github.com/, 2023.

[54] Barney G Glaser. The constant comparative method of qualitative analysis. *Social problems*, 12(4):436–445, 1965.

[55] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*, pages 345–355, 2014.

[56] Runzhi He, Hao He, Yuxia Zhang, and Minghui Zhou. Automating dependency updates in practice: An exploratory study on github dependabot. *IEEE Transactions on Software Engineering*, 49(8):4004–4022, 2023.

[57] Stephen Hendrick and Ashwin Ramaswami. Maintainer perspectives on open source software security. https://www.linuxfoundation.org/hubfs/LF

[58] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Trans. Softw. Eng. Methodol.*, 33(8), December 2024.

[59] Allen D Householder, Jeff Chrabaszcz, Trent Novelly, David Warren, and Jonathan M Spring. Historical analysis of exploit availability timelines. In *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, 2020.

[60] Keman Huang, Michael D. Siegel, Stuart E. Madnick, Xiaohong Li, and Zhiyong Feng. Poster: Diversity or concentration? hackers' strategy for working across multiple bug bounty programs. In *37th IEEE Symposium on Security and Privacy (S&P)*, 2016.

[61] Yu Huang, Denae Ford, and Thomas Zimmermann. Leaving my fingerprints: Motivations and challenges of contributing to oss for social good. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1020–1032, 2021.

[62] Yu-Chi Huang, Kuan-Li Peng, and Chin-Yu Huang. A history-based cost-cognizant test case prioritization technique in regression testing. *Journal of Systems and Software*, 85(3):626–637, 2012.

[63] Bryan Hughes. Why i'm leaving the node.js project. https://medium.com/@nebrius/why-im-leaving-the-node-js-project-bff946845a77, 2017.

[64] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *2013 35th International Conference on Software Engineering (ICSE)*, pages 672–681. IEEE, 2013.

[65] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *2013 35th International Conference on Software Engineering (ICSE)*, pages 672–681, 2013.

[66] Vinay Kabadi, Dezhen Kong, Siyu Xie, Lingfeng Bao, Gede Artha Azriadi Prana, Tien-Duy B. Le, Xuan-Bach D. Le, and David Lo. The future can't help fix the past: Assessing program repair in the wild. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 50–61, 2023.

[67] Nolan Lawson. What it feels like to be an open-source maintainer, 2017.

[68] Amanda Lee, Jeffrey C Carver, and Amiangshu Bosu. Understanding the impressions, motivations, and barriers of one time code contributors to floss projects: a survey. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 187–197. IEEE, 2017.

[69] Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2201–2215, 2017.

[70] Jiawei Li, David Faragó, Christian Petrov, and Iftekhar Ahmed. Only diff is not enough: Generating commit messages leveraging reasoning and action of large language model. *Proceedings of the ACM on Software Engineering*, 1(FSE):745–766, 2024.

[71] Yuni Li and Ling Zhao. Collaborating with bounty hunters: How to encourage white hat hackers' participation in vulnerability crowdsourcing programs through formal and relational governance. *Information & Management*, 59(4):103648, 2022.

[72] Jenny T. Liang, Thomas Zimmermann, and Denae Ford. Understanding skills for oss communities on github. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 170–182, New York, NY, USA, 2022. Association for Computing Machinery.

[73] Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. Changescribe: A tool for automatically generating commit messages. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 709–712. IEEE, 2015.

[74] Johan Linåker, Georg J. P. Link, and Kevin Lumbard. Sustaining maintenance labor for healthy open source software projects through human infrastructure: A maintainer perspective, 2024.

[75] Thomas Maillart, Mingyi Zhao, Jens Grossklags, and John Chuang. Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs. *Journal of Cybersecurity*, 3(2):81–90, 10 2017.

[76] Mend.io. Renovate bot. https://github.com/renovatebot/renovate, 2017.

[77] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian KaUstner. " did you miss my comment or what?" understanding toxicity in open source discussions. In *Proceedings of the 44th International Conference on Software Engineering*, pages 710–722, 2022.

[78] Nasir Mehmood Minhas, Thejendar Reddy Koppula, Kai Petersen, and Jürgen Börstler. Using goal–question–metric to compare research and practice perspectives on regression testing. *Journal of Software: Evolution and Process*, 35(2), 2023.

[79] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *2017 32nd IEEE/ACM international conference on automated software engineering (ASE)*, pages 84–94. IEEE, 2017.

[80] MITRE. Xz utils vulnerability. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-3094, 2024.

[81] Hamid Mohayeji, Andrei Agaronian, Eleni Constantinou, Nicola Zannone, and Alexander Serebrenik. Investigating the resolution of vulnerable dependencies with dependabot security updates. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 234–246. IEEE, 2023.

[82] Lukas Moldon, Markus Strohmaier, and Johannes Wachs. How gamification affects software developers: Cautionary evidence from a natural experiment on github. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 549–561. IEEE, 2021.

[83] Sabato Nocera, Simone Romano, Massimiliano Di Penta, Rita Francese, and Giuseppe Scanniello. Software bill of materials adoption: A mining study from github. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 39–49, 2023.

[84] Eric O'Donoghue, Ann Marie Reinhold, and Clemente Izurieta. Assessing security risks of software supply chains using software bill of materials. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C)*, pages 134–140, 2024.

[85] Chinenye Okafor, Taylor R. Schorlemmer, Santiago Torres-Arias, and James C. Davis. Sok: Analysis of software supply chain security by establishing secure design properties. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, SCORED'22, page 15–24, New York, NY, USA, 2022. Association for Computing Machinery.

[86] Hassan Onsori Delicheh, Alexandre Decan, and Tom Mens. Quantifying security issues in reusable javascript actions in github workflows. In *Proceedings of the 21st International Conference on Mining Software Repositories*, pages 692–703, 2024.

[87] Cassandra Overney, Jens Meinicke, Christian Kästner, and Bogdan Vasilescu. How to not get rich: An empirical study of donations in open source. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, pages 1209–1221, 2020.

[88] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. Stress and burnout in open source: toward finding, understanding, and mitigating unhealthy interactions. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '20, page 57–60, New York, NY, USA, 2020. Association for Computing Machinery.

[89] Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 345–355. IEEE, 2017.

[90] Johnny Saldaña. The coding manual for qualitative researchers (4th edition). pages 1–440, 2021.

[91] Open secure software foundation (OpenSSF). Guide to implementing a coordinated vulnerability disclosure process for open source projects. https://github.com/ossf/oss-vulnerability-guide/blob/main/maintainer-guide.md, 2022.

[92] Naomichi Shimada, Tao Xiao, Hideaki Hata, Christoph Treude, and Kenichi Matsumoto. Github sponsors: exploring a new way to contribute to open source. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1058–1069, 2022.

[93] Synopsys. Coverity scan static analysis. https://scan.coverity.com/, 2006.

[94] Synopsys. 2024 open source security and risk analysis report. https://www.synopsys.com/software-integrity/engage/ossra/ossra-report, 2023.

[95] Sagacent Technologies. How often should you update your cyber security policies. https://sagacent.com/blog/how-often-should-you-update-your-cyber-security-policies, 2021.

[96] Eric Tooley and Kate Catlin. Private vulnerability reporting now generally available. https://github.blog/2023-04-19-private-vulnerability-reporting-now-generally-available/, 2023.

[97] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th international conference on software engineering*, pages 511–522, 2018.

[98] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th international conference on Software engineering*, pages 356–366, 2014.

[99] Christine Utz, Sabrina Amft, Martin Degeling, Thorsten Holz, Sascha Fahl, and Florian Schaub. Privacy rarely considered: Exploring considerations in the adoption of third-party services by websites. *arXiv preprint arXiv:2203.11387*, 2022.

[100] Fadi Wedyan, Dalal Alrmuny, and James M Bieman. The effectiveness of automated static analysis tools for fault detection and refactoring prediction. In *2009 International Conference on Software Testing Verification and Validation*, pages 141–150. IEEE, 2009.

[101] Dominik Wermke, Noah Wöhler, Jan H. Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. Committed to trust: A qualitative study on security & trust in open source software projects. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1880–1896, 2022.

[102] Mairieli Wessel, Ahmad Abdellatif, Igor Wiese, Tayana Conte, Emad Shihab, Marco A Gerosa, and Igor Steinmacher. Bots for pull requests: The good, the bad, and the promising. In *Proceedings of the 44th International Conference on Software Engineering*, pages 274–286, 2022.

[103] Mairieli Wessel, Igor Wiese, Igor Steinmacher, and Marco Aurelio Gerosa. Don't disturb me: Challenges of interacting with software bots on open source software projects. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2):1–21, 2021.

[104] Boming Xia, Tingting Bi, Zhenchang Xing, Qinghua Lu, and Liming Zhu. An empirical study on software bill of materials: Where we stand and the road ahead. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2630–2642, 2023.

[105] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. How open source projects use static code analysis tools in continuous integration pipelines. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 334–344. IEEE, 2017.

[106] Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. Large language models meet nl2code: A survey. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 7443–7464, 2023.

[107] Xunhui Zhang, Yue Yu, Georgios Gousios, and Ayushi Rastogi. Pull request decisions explained: An empirical overview. *IEEE Transactions on Software Engineering*, 49(2):849–871, 2022.

[108] Ling Zhao and Yun Li. Collaborating with white hat hackers: A study of vulnerability crowdsourcing program from control perspective. In *Pacific Asia Conference on Information Systems*, 2020.