Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects

(Preprint)

Dominik Wermke*, Noah Wöhler*, Jan H. Klemmer†, Marcel Fourné‡, Yasemin Acar§, and Sascha Fahl*
*CISPA Helmholtz Center for Information Security, Germany, {dominik.wermke,noah.woehler,fahl}@cispa.de
†Leibniz University Hannover, Germany, klemmer@sec.uni-hannover.de
‡Max Planck Institute for Security and Privacy, Germany, marcel.fourne@mpi-sp.org
§George Washington University, United States, acar@gwu.edu

***Abstract*—Open Source Software plays an important role in many software ecosystems. Whether in operating systems, network stacks, or as low-level system drivers, software we encounter daily is permeated with code contributions from open source projects. Decentralized development and open collaboration in open source projects introduce unique challenges: code submissions from unknown entities, limited personpower for commit or dependency reviews, and bringing new contributors up-to-date in projects' best practices & processes.**

In 27 in-depth, semi-structured interviews with owners, maintainers, and contributors from a diverse set of open source projects, we investigate their security and trust practices. For this, we explore projects' behind-the-scene processes, provided guidance & policies, as well as incident handling & encountered challenges. We find that our participants' projects are highly diverse both in deployed security measures and trust processes, as well as their underlying motivations. Based on our findings, we discuss implications for the open source software ecosystem and how the research community can better support open source projects in trust and security considerations. Overall, we argue for supporting open source projects in ways that consider their individual strengths and limitations, especially in the case of smaller projects with low contributor numbers and limited access to resources.

I. Introduction

Open Source Software (OSS) is an unavoidable component in many of today's software ecosystems. Whether as low-level system drivers in operating systems, as tooling in daily jobs, or simply as dependencies of hobby projects, OSS is an important building block in our everyday software interactions.

In a 2020 report covering 45,000 repositories, GitHub found that most projects on their platform rely on some form of OSS [1]. In recent years, collaborative version control platforms such as GitHub [2] and GitLab [3] introduced a wide field of developers to open source projects. As the complexity of modern software development increased, so did the number of dependencies and involved contributors. Decentralized development and open collaboration of open source projects introduce unique challenges: code submissions from unknown entities, limited personpower for reviewing commits and dependencies, and bringing new contributors up-to-speed in projects' best practices and processes.

Assessing vulnerabilities in components is a difficult task, as the large number of dependencies required by today's software result in a complex software supply chain, including software repositories, package managers, and package registries. The median number of transitive dependencies in the npm ecosystem was reported as 683 in a 2020 GitHub report [1]. In addition to vulnerabilities in components, dependency sources often lack basic security and trust controls due to historical and economic reasons. Recent incidents in the npm ecosystem highlight the large attack surface provided by such registries: in late October 2021, versions of the npm package *ua-parser-JS* with 7 million weekly downloads included malicious code [4]. An attacker gained access to the maintainer's account and released three manipulated versions executing a Monero cryptocurrency miner and password-stealing trojans [5]. Less than a month later, GitHub reported an authorization vulnerability in npm, allowing attackers to publish manipulated, authorized versions of their packages, which could actually be applied to any npm package without authorization [6]. While GitHub stated with "*high confidence*" that the vulnerability had not been exploited maliciously, telemetry data was only available from September 2020 onwards [7]. Analogous to a 2020 report from The Linux Foundation [8], we consider the software supply chain in this work to include technical features such as how the software is stored, how it can be retrieved, and how it is analyzed during these processes.

The same holds true for commercial software: by building their software as a wrapper or glue around open source components, companies can leverage OSS as building blocks in their processes and products, allowing them to focus their efforts on features and faster delivery. In 2020, 95% of IT departments and companies considered OSS as strategically important to their organization's overall enterprise infrastructure software strategy [9]. By introducing open source components, companies inherit the same challenges and attack surfaces as open source projects. They are now obligated to assess and mitigate the impact of vulnerabilities from open source components included in their products. As such, improving security and trust in the open source ecosystem leads to positive effects

down the whole dependency chain for both open source and commercial software.

These chain effects make the open source ecosystem an important field of research for the (security and privacy) community. With the introduction of more developer-centered research approaches arose the need for human-subject research considerations. Recent conflicts between the research and open source communities such as the "*hypocrite commits*" incident in early 2021 highlight the need for more respectful research approaches for investigating security and trust in open source projects [10]. In this work, we propose a more cooperative approach for researching open source, working together with committers towards a more secure and trustworthy ecosystem, instead of against them.

In addition to security, trust also plays an important role in software development and especially the open source community, as was probably best described in Ken Thompson's Turing Award Lecture "Reflections on Trusting Trust":

> "*To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.*" — K. Thompson [11].

As to err is only human, we consider contributors as trustworthy if they do not act with malicious intent, not necessarily that they contribute error-free code.

In this work, we aim to shed light on security and trust practices in open source projects — by exploring projects' behind-the-scene processes, provided guidance and security policies, as well as past security challenges and incident handling. We are especially interested in processes that are often not directly visible from the repository data, e. g., trust relationships, incident responses, and the handling of suspicious or malicious contributors. For this, we conducted 27 in-depth, semi-structured interviews with contributors, maintainers, and owners from a diverse set of open source projects.

Our research approach investigates security measures and trust processes in OSS based on the following research questions:

RQ1: *"How are open source projects structured behind-the-scenes?"* Due to their community-driven nature, open source projects include structures and processes that are not inherently visible on a repository level. We investigate the why and how of behind-the-scenes interactions and decisions, especially in the context of security and trust.

RQ2: *"If and what guidance and policies are provided by open source projects?"* Often changing contributors and loose team structures lead to challenges in distributing project-internal knowledge in open source projects. We examine guidance and (security) policies provided by open source projects of any size, as well as identify their established roles and responsibilities.

RQ3: *"How do open source projects approach security and trust challenges?"* Open source projects face unique challenges in terms of security and trust due to their open nature, including code submissions from mostly unknown entities. We investigate which organizational and technical measures open source projects employ to establish trust between contributors and how they react or plan to react to arising security and trust challenges.

This work is structured as follows: After this general introduction (Section I), we discuss related work in the areas of repository research, interviews in a security context, and open source security and trust (Section II). We then describe our interview approach (Section III) and highlight our findings (Section IV). Finally, we discuss our findings (Section V) and draw a conclusion (Section VI).

A. Replication Package

In line with the effort to support replication of our work and help other researchers build upon it, this publication has a companion website with a full replication package[1] and an artifact repository available.

II. Related Work

We present and discuss previous work in three areas: research involving data and artifacts from software repositories, interview studies in a security context, and investigations of security and trust in the open source community. We also put our work into context and illustrate the novel contributions of our research.

Research with Repositories: Open source repositories are an established data source in the (security and privacy) research community. This is corroborated by the large number of available datasets, e. g., of commits [12]–[14], contributors [15], and vulnerabilities [16], [17], as well as easy access via torrents [18], [19]. Early work describes case studies of then emerging open source projects such as Linux [20], Mozilla [21], and FreeBSD [22]. Due to freely accessible code and commits, open source repositories are a common source for vulnerability research, e. g., by matching Common Vulnerabilities and Exposures (CVEs) [23], [24], tracking vulnerability evolution over time or events [25]–[28], or for evaluating static analysis tools [29]–[32]. Both Deligiannis *et al.* and Bai *et al.* analyzed drivers in the Linux Kernel [33], [34]. Fixes and patches are essential for ensuring a secure code base, motivating previous work to investigate fix patterns and phases [35]–[37]. Piantadosi *et al.* linked 337 CVE entries to the corresponding patches, finding that developers who fix vulnerabilities are more experienced than average [38]. Related research focusing on social aspects investigated collaboration [39]–[41], gamification [42], donations [43], and pull requests [44]–[46]. Recently published work investigated repository artifacts such as programming languages [47], maintenance [48], [49], toxicity in comments [50], and related metadata [51], [52].

Unlike previous research focusing on repositories, we are more interested in aspects that are not directly visible on a repository level: trust processes, contributor hierarchy, and security considerations.

[1] https://publications.teamusec.de/2022-oakland-sec-oss/

Interview Studies in a Security Context: Interview studies are a well-established research approach for in-depth investigations in the (security and privacy) research community. In the past, research has utilized interviews to gain insights into the work and tools of experts such as security professionals [53], [54], administrators [55], [56], and security analysts [57]. Interviews were also conducted to establish the security needs of expert communities such as journalists [58], editors [59], and victim service providers [60]. As part of larger studies, interviews allow insights into specific mindsets and approaches, e. g., for encryption tasks [61] or Tor usage [62]. More recently, Gutfleisch *et al.* interviewed developers about security feature considerations in their software development process [63]. In the context of OSS and communities, Dabbish *et al.* examined the value of transparency for large-scale distributed collaborations and communities of practice in interviews [39]. As part of a larger study, Steinmacher *et al.* conducted semi-structured interviews with 36 developers from 14 different projects, identifying social barriers faced by first-time contributors [64]. Balali *et al.* interviewed mentors of 10 OSS projects, identifying both challenges and strategies related to recommending tasks for newcomers [65].

Similarly, we also decided on in-depth interviews for our research approach to gain detailed insights into participants' perceptions, behaviors, and reasoning.

Security and Trust in the Open Source Community: The open source community faces unique security and trust challenges compared to other ecosystems, making them a valuable subject for research [66]–[68]. Issues and commits are important structural features in the open source community, enabling evaluations of general statistics [69], security tactics [70], and emotions [71]. Antal *et al.* investigated commits of Python and JavaScript projects, finding that neither community reacts very fast to emerging security vulnerabilities in general [72]. Bosu *et al.* analyzed 267,046 code review requests from 10 open source projects, finding that less experienced contributors' changes were 1.8 to 24 times more likely to be vulnerable [73]. Published identification systems for open source projects include vulnerabilities [74]–[76] and toxic comments [77]. Trust is an important factor in public software collaboration. Research directions include trustworthiness measurements [78], [79] and factors influencing trust [45], [80], [81]. In line with our findings, prior research established (quality of) contributions, reputation, and employing organization as important trust factors. Code quality is an important factor for security in open source projects, with previous research investigating aspects such as code reviews [82], [83], quality assessment models [84], and discrepancies between vision and actual implementation [85]. Due to their important role in the open source ecosystem, committers are the focus of multiple works, e. g., for their pull requests [86], their motivations [87]–[90], or contribution barriers [64], [91]. Other works propose supporting aspects such as approaches for onboarding [65], [92]–[94] and mentoring [65], [95]. Blincoe *et al.* proposed a new method, *reference coupling*, for detecting technical dependencies between projects, finding that most ecosystems are centered around one project and are interconnected with other ecosystems [96]. Casalnuovo *et al.* explored the evidence for socialization as a precursor to joining GitHub projects, finding developers preferentially join projects where they have pre-existing relationships [97].

While our work utilizes certain repository artifacts to enrich our research, our focus is more on in-depth details from 27 interviews with contributors, maintainers, and project owners. Like previous research, we consider the open source ecosystem to be of major importance to the overall software world and hope to leverage 27 in-depth interviews as first steps towards supporting committers and maintainers in creating secure and trustworthy projects.

III. Methodology

In this section, we provide an overview of our study approach and the structure of the semi-structured interviews. We also detail the qualitative coding process, report on our data collection and ethical considerations, and discuss the limitations of our work.

A. Study Setup

To gain insights into the inner workings of open source projects, we conducted semi-structured interviews ($n = 27$) with contributors, maintainers, and owners of open source projects between July and November 2021. We decided on in-depth interviews as our research approach, as we were especially interested in processes that are often not directly visible from the repository data, e. g., trust relationships, incident responses, and the handling of suspicious or malicious contributors.

Interview Guide: We based the initial interview guide on our exploratory research questions. We also considered concepts investigated in previous related work and adapted them to our more in-depth interview approach. To establish additional areas of research and for feedback, we consulted and piloted the interview guide with open source contributors from our professional network. For the participants' convenience, we created both English and German versions of the interview guide, keeping both in sync during the study. During the study process, we continually iterated the interview guide based on the conducted interviews and the collected participant feedback. Changes were limited to the addition of a few follow-up questions and minor structural modifications, reaching saturation without any changes past the 15$^{\text{th}}$ interview. Our full interview guides in English and German are included in the replication package (cf. Section I-A).

Recruitment and Inclusion Criteria: We based our recruitment approach around reaching as many diverse OSS projects as possible. We decided on utilizing multiple recruitment channels to better reach a diverse set of projects from different historical and structural contexts: via our professional network, project- or technology-associated communication channels such as mailing lists, Discord instances, or IRC servers, as well as via contact details on public repository websites like

TABLE I: Detailed overview of interviewed contributors, their project background, as well as some project metadata. For reporting, participants were assigned an alias. We only report binned project metrics to preserve both our participants' and their projects' privacy.

Alias	Interview				Project[1]		
	Language	Duration	Codes[2]	Recruitment Channel	Commits	Contributors	Category
P01	German	0:40:49	68	Professional Network	100,000+	10+	Operating System
P02	German	1:03:51	76	Professional Network	1,000+	10+	Secure Messenger
P03	German	0:53:49	57	Contact Email	10,000+	100+	Virtualization/Containers
P04	English	0:33:59	62	Communication Channel	100+	10+	JavaScript Libraries
P05	English	0:36:35	42	Contact Email	1,000+	100+	Code Editor
P06	English	0:55:20	70	Communication Channel	100+	10+	JavaScript Libraries
P07	English	0:33:16	54	Contact Email	100+	10+	.NET Libraries
P08	English	1:06:18	67	Contact Email	100,000+	100+	Operating System
P09	English	0:30:37	95	Contact Email	10,000+	<10	Version Control System
P10	English	0:23:35	36	Contact Email	10,000+	100+	GUI Tool
P11	English	1:08:13	101	Contact Email	10,000+	1,000+	Orchestration
P12	German	0:35:12	61	Professional Network	10,000+	100+	Network Security Monitor
P13	English	0:29:23	39	Contact Email	10,000+	100+	Scientific Computing
P14	English	0:19:44	38	Communication Channel	1,000+	10+	Cryptocurrency Exchange
P15	German	0:26:32	44	Communication Channel	10,000+	100+	Operating System
P16	English	0:46:19	48	Contact Email	10,000+	100+	Code Analysis
P17	English	0:44:14	57	Contact Email	1,000+	1,000+	JavaScript Libraries
P18	English	0:32:46	45	Project Website	1,000+	10+	Scientific Computing
P19	German	0:40:59	40	Communication Channel	1,000+	10+	Scientific Computing
P20	German	0:38:14	63	Communication Channel	10,000+	100+	Network Protocol
P21	English	0:38:25	43	Contact Email	1,000+	100+	Virtualization/Containers
P22	English	0:37:09	73	Contact Email	1,000+	100+	Data Format
P23	English	0:23:19	62	Contact Email	10,000+	100+	Virtualization/Containers
P24	English	0:39:35	57	Contact Email	100+	100+	Orchestration
P25	English	0:52:23	83	Project Website	10,000+	1,000+	Operating System
P26	English	0:33:23	59	Contact Email	10,000+	100+	Scientific Computing
P27	English	0:37:52	78	Contact Email	1,000+	100+	Scientific Computing

[1] If multiple projects: largest project covered in the interview. [2] Total number of codes assigned to the interview after resolving conflicts.

GitHub. See also Table I for an overview of interviewed participants and corresponding recruitment channel.

Aside from our professional network and well-known open source projects, we utilized GitHub as a platform for selecting and contacting open source projects. We focused on GitHub, as it is widely used in the open source community and provides relevant metrics for gauging the activity as well as popularity of a project. We created our initial dataset based on data from July 2021, consisting of code repositories that received at least 40 commits from at least 20 distinct committers in the previous six months and gained new committers in July 2021. Our intent was to exclude inactive projects or small projects without contributors, for which our inquiry would either not reach active contributors or in which trust processes are irrelevant. The detailed selection process and criteria are described in the replication package (cf. Section I-A) and Appendix (cf. Appendix A). We then joined popularity and activity indicators to a combined ranking and divided the set of projects into quartiles, from which we then iteratively selected and contacted projects until we reached interview saturation:

1) Communication Channel. If the project provided a public communication channel such as a Slack workspace, Discord server, or Gitter chat, we asked the administrators for permission to post a call for participants.
2) Contact Email. Otherwise, we either contacted the project's contact email or the project's top contributor by number of commits in the past year via their public email address.

In addition to these channels, we asked our participants for their recommendations of interesting or unique open source projects, which we then contacted via the approaches described above.

Due to the previous filtering, we did not require any additional eligibility criteria from our participants beyond stating that we were looking for people involved in OSS. In total, we recruited 27 participants from equally as many distinct projects.

Interview Procedure: We conducted the 27 interviews in a lead/backup interviewer configuration between July and November 2021. To afford our participants a high level of comfort during the interview, we offered them the choice to conduct the interview either in English or German, as all interviewing researchers were proficient in both languages. We conducted the majority of interviews via our self-hosted Jitsi instance, though a few interviews were conducted via Zoom or the participant's service of choice. Interviews were advertised as lasting between 30–45 minutes in total, with the interview part lasting a median of 00:37:52 minutes.

The different interview sections were introduced with an open, non-leading question, allowing participants to elicit their own thoughts and reactions on their terms. Only if specific points were not addressed, did we follow up with

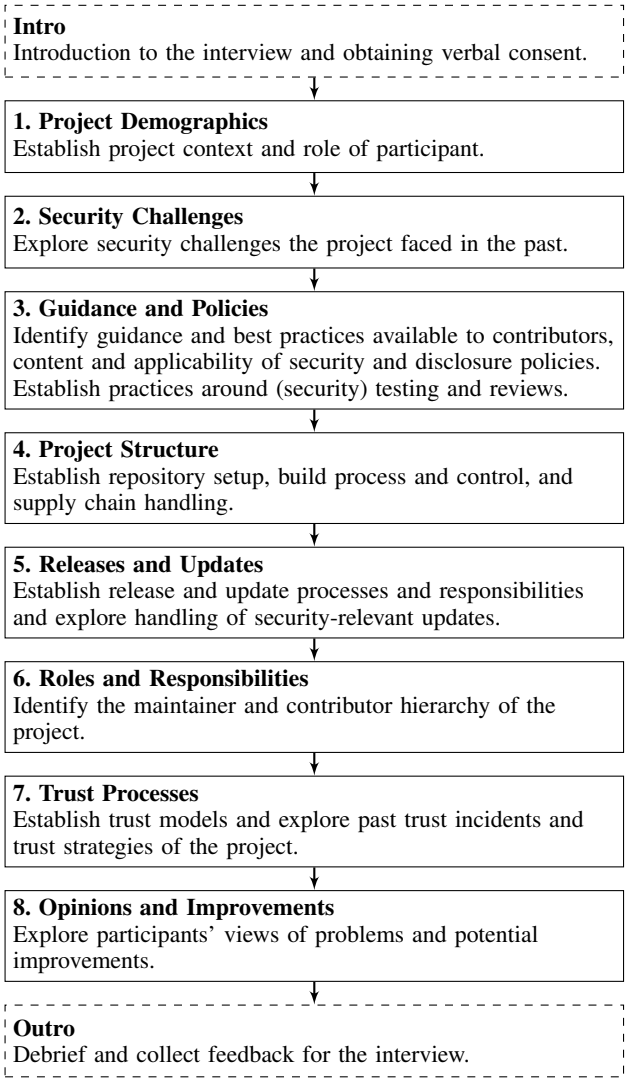

Fig. 1: Illustration of the flow of topics in the semi-structured interviews. In each section, participants were presented with general questions and corresponding follow-ups, but were generally free to diverge from this flow at will.

a more specific, non-leading sub-question. Interviewers were specifically instructed not to impart a sense that the project's security or insecurity was being judged and to not prime participants' answers in other ways.

B. Interview Structure

We outline our semi-structured interview structure below and in Figure 1. For reporting, we group the interview into eight sections, each consisting of 1–4 opening questions, corresponding follow-up questions, and in some cases additional nudges.

Before the actual interview part, we gave a short introduction of involved institutions and our motivations. We specifically highlighted to participants that our goal is not to judge the security of their projects, that it is okay not to be aware of all aspects of a project, and that we are explicitly interested in their personal thoughts and opinions. We went over how we intend to collect and handle the interview data and obtained the participant's consent for recording and data handling.

1. Project Demographics: The interview opens with a general question section about the project and our participant's relation to it. This section is intended both to ease nervous participants into the interview as well as establish some initial context to later combine with actual repository data. We report the demographics and combined data in Section IV-A and Table I.

2. Security Challenges: The "*Security Challenges*" section explores past security challenges encountered by our participants, as well as their opinion of a recent research conflict. To open this section with an example of a recent incident and to ease participants into this sensitive topic, we queried them about, and if necessary introduced them to, the "*hypocrite commits*" incident from early 2021 [10], [98], [99]. The incident is a recent, widely publicized example of well-intentioned actions resulting in potentially adverse outcomes. We selected this incident because we suspected that projects are more familiar with well-intentioned commits turning sour, compared to straight-up malicious attacks. We report these results in Section IV-B.

3. Guidance and Policies: The "*Guidance and Policies*" section establishes guidance provided for, and policies enforced by participants' projects. Follow-ups for guidance included specific guidance for infrastructure, programming style, and cryptography usage. Follow-ups for policies included the (coordinated) disclosure approach of the project, potential policies for handling security incidents, and policies for security aspects such as enforced (security) reviews. We report these results in Section IV-C.

4. Project Structure: The "*Project Structure*" section investigates behind-the-scenes structures and processes in the project. Specifically, we were interested in structures that are often not directly visible from repository artifacts, such as how build and deploy steps are set up, who controls them, and how the related secrets are managed. We also included follow-ups for supply chain handling such as selection criteria and vulnerability checks for dependencies. Lastly, we asked participants about additional infrastructure such as project websites and communication tools, as well as who controls these resources. We report these results in Section IV-D.

5. Releases and Updates: The "*Releases and Updates*" section explores release mechanisms within the project, as well as how end users or downstream dependencies receive updates to the latest version, with a special focus on security-relevant fixes. In particular, we were interested in release schedules, whether there were guidelines in place regarding the deprecation of older (insecure) versions, as well as if and how release binaries are secured. We report these results in Section IV-E.

6. Roles and Responsibilities: The "*Roles and Responsibilities*" section establishes the contributor hierarchy and security roles of the project. We were especially interested in how

decisions are formed and whether security-specific roles are assigned. We report these results in Section IV-F.

7. Trust Processes: The "*Trust Processes*" section considers established trust models in the project and how recently onboarded contributors can become trusted members. Follow-ups included questions about identity checks or the mandatory signing of Contributor License Agreements (CLAs). Additionally, we asked the participants about past trust incidents and, if applicable, what their mitigation strategy looked like. In cases without such an incident, we asked participants about their opinion on what would happen if an incident occurred. We report these results in Section IV-G.

8. Opinions and Improvements: The "*Opinions and Improvements*" section aims to elicit participants' personal opinions and beliefs about current open source practices regarding security and trust and how they would personally approach improving the status quo. We report these results in Section IV-H.

After the interview part, we debriefed the participants and collected additional feedback regarding covered topics and suggestions for interesting or unique open source projects to contact.

C. Coding and Analysis

For our study with interviews and repository artifacts, we evaluated both qualitative and quantitative data points. We recorded the interviews digitally, transcribed them via a GDPR-compliant service, and manually reviewed all transcripts for potential mistakes.

We analyzed all interview answers in an iterative open-coding approach [100]–[102]. All researchers together established an initial codebook based on the interview guide and interview impressions. Three researchers then iteratively coded the interviews in multiple rounds, resolving conflicts by consensus decision or by introducing new (sub)codes after each iteration. We continued with our iterative coding approach until no new codes or themes emerged [103], [104]. This approach does not necessitate the reporting of inter-coder agreement, as each conflict is resolved when it emerges (resulting in a hypothetical final agreement of 100%). In total, we assigned 1618 codes after resolving, resulting in a median of 59 codes per interview. The final codebook is included in our replication package described in Section I-A.

D. Ethical Considerations and Data Protection

This experiment was approved by the human subjects review board (IRB equiv.) of our institution. Research plan, study procedure, and all involved parties adhered to the strict German data and privacy protection laws, as well as the EU General Data Protection Regulation (GDPR). In addition, we modeled our study to follow the ethical principles of the Menlo report for research involving information and communications technologies [105]. All documents with personally identifiable data according to the GDPR were stored in a secure cloud collaboration software suite and were encrypted at rest and in transit. The transcription service we leveraged is based in the EU and fully complies with the GDPR. Our research approach agrees with the Researcher Guidelines for the Linux developer community introduced in response to the "*hypocrite commits*" incident in late March 2022, after the conclusion of our work [106].

We encouraged potential participants to familiarize themselves with consent and data handling information on a study website before agreeing to any interview participation. We obtained informed consent from all participants for participation in the study and having their interview's audio recorded. We contacted participants with a preprint and gave them an opportunity to suggest changes or veto this work in its current form. Before, during, and after the interview, (potential) participants were able to contact us at listed contact addresses for any questions or additional information. We consider the interview questions regarding certain security incidents to be of sensitive nature and explicitly highlighted to the participants that they could skip questions or terminate the interview at any time. Our participants did not receive any compensation, since we surmised that open source contributors likely would be more inclined to volunteer their time to research if they act out of intrinsic motivation.

E. Limitations

Our work includes a number of limitations typical for this type of interview study and should be interpreted in context. In general, self-report studies may suffer from several biases, including over- and under-reporting, sample bias, and social-desirability bias. We do note that our sample is a convenience sample and that our participants are not necessarily representative of contributors in the open source ecosystem. It is possible that contributors who agreed to speak with us are more (or less) security-conscious than those who declined.

During sampling, we focused on projects providing an English *Readme* document. We also offered and conducted seven interviews in German for participants' convenience. Thus, we can offer no direct insight regarding the generalizability of our results regarding non-English and non-German speaking open source contributors. During modelling of our study, we decided that this was an agreeable trade-off, with English serving as the "*working language*" of the international open source community, likely allowing us to communicate with a meaningful set of contributors.

Certain questions, e. g., about security and trust incidents, can be considered to be of sensitive nature. To reduce social-desirability bias in answers, we specifically highlighted to participants that we were only interested in information about their projects and not judging their security approaches and processes in any way. We also instructed participants that they were able to skip questions or to terminate the interview for any reason at any time.

IV. Results

In the following section, we report and discuss results for 27 semi-structured interviews with open source contributors,

maintainers, and owners. In our reporting, we mostly adhere to the structure of the interview guide described in Section III-A and summarize our key findings after each question block. We report participants' quotes as transcribed, with minor grammatical corrections and omissions marked by brackets ("*[...]*"). Quotes from German interviews were translated to English by native German speakers.

A. Project Demographics

In total, we interviewed 27 valid participants. In addition to this section, we report general interview and project demographics in Table I. As it is common in the open source community to be involved in multiple projects, we encouraged our participants to talk about the projects they considered most relevant during the interview. For the collected quantitative data, we considered the largest project mentioned during the interview, as a trade-off between concise reporting and applicability.

Due to our recruitment approach aiming for a high diversity in projects, our participants reported a wide range of projects and backgrounds, ranging from operating system components, over libraries, to scientific computing frameworks. For each individual participant, we report project categories and commits of the largest project they mentioned in Table I. Project contributors are often highly distributed, with five of 27 participants reporting to know other contributors only virtually. E. g., as P17 reported: "*Everybody that I've encountered has just been virtually: I can see the profile picture of some people, and that's the only image I have of them.*" (P17). Although this does not seem to impair collaboration: "*But to be honest, I don't really mind. As long as one has the same interests, it's still easy to collaborate if you have the same goal.*" (P17). At the other extreme, four participants mentioned very close connections such as working at the same company or university. We sorted our participants into their highest project role with a roughly ascending order of responsibility: contributors (4), maintainers (3), team leaders (7), and founders or owners (9).

Overall, we found our participants to be more experienced than we expected, often having been involved for multiple years and possessing high-level commit rights. We assume this high level of experience was due to our recruiting focusing on "expert channels" such as project-specific communication channels or dedicated contact addresses, as well as being referred further up in projects until reaching founders and owners.

> Summary: Project Demographics. The majority of our participants are highly experienced in the open source environment, often with multiple years of work and high-level commit rights.

B. Security Challenges

In this section, we explore past security challenges encountered by our participants, as well as their opinion of the widely reported "*hypocrite commits*" incident. More than half (16) of our participants reported never having encountered a direct security incident in the past. The most commonly reported security challenges (that did not necessarily lead to an incident) included: suspicious or low quality commits (15) and vulnerabilities introduced by dependencies (8). Overall, our participants seem to be mostly ambivalent about potentially malicious commits: "*I mean, there's definitely been people that have intentionally tried to put malicious code in projects, but it's always very easy to spot immediately. It's like those spam emails where they have bad grammar and stuff.*" (P06). Same holds true for vulnerabilities in dependencies, which apparently often turn out to be false positives or to be irrelevant for participants' projects:

> "*Most of the time, the vulnerabilities I deal with are transitive dependencies, have a CVE, and 99.99 percent of the time, they are false positives for every other use case: it's a real vulnerability in the dependency, but it's not in the way almost anyone uses it.*" — P06.

The majority of our participants were aware of the "*hypocrite commits*" incident in early 2021 (23 of 27). For the remaining four, we provided a short, factual summary of the incident during the interview. Of the 16 participants with a generally negative opinion of the incident, many considered the research approach as outright malicious: "*[t]he shocking and surprising part was, that an academic institution would essentially do evil and justify it by saying that the ends justify the means.*" (P06). This is likely a misconception, as the researchers stated that they did not intend to, and objectively did not, introduce any vulnerability in Linux [10]. Of the remaining participants with a mixed (7) or no opinion (4), some considered the research approach similar to that of a "*White Hat Hacker*", although with a flawed execution. E.g., "*I do understand both sides of this [...] It would be much better if this kind of research was done in cooperation with somebody at the Linux kernel, who knew that it's happening and without disclosing that to a lot of people.*" (P11). We could not identify a single participant with an outright positive opinion of the incident. We assume this skew was likely exaggerated by the generally negative, sometimes misinformed reporting by open source aligned news sources and communities.

> Summary: Security Challenges. Only few projects have experienced an outright security incident, although many of our participants were familiar with suspicious or low quality commits, as well as potential vulnerabilities introduced by dependencies. The majority of our participants were generally aware of the "*hypocrite commits*" incident and had an overall negative opinion of the research approach.

C. Guidance and Policies

In this section, we examine guidance and best practices provided by the projects, as well as the content and applicability of security and disclosure policies.

Guidance: Most commonly, our participants mentioned guidance for contributing to the project (14) and programming language-specific guidance such as style guides (13), followed

by general guidance for project setup and infrastructure (8). As reasons for not providing specific guidance documents, participants mention time and money constraints: "*Somebody would have to write the guide, and I am the only one who can write it. I mean, there is nobody paid to write it and I am also not paid to write it.*" (P26). More generally, our participants are somewhat divided in their opinions of the helpfulness of guidance for their projects, ranging from very positive: "*I personally think that documentation is one of the most important aspects of an open source project, both for users and also developers.*" (P27), to less helpful, as for P02's project: "*I'm also honestly not quite sure that's really that helpful [...] Of course, it's quite nice to have overviews and stuff like that somewhere, but there aren't too many people who then read something like that.*" (P02). Instead, P02 mentions that they prefer to coach new contributors: "*Most of the people who are interested show up in the communication channels. And then it depends on [project members] being communicative by helping the other person.*" (P02). Similarly, P11 mentions an approach outside of classical guidance documents: "*We answer very detailed answers to questions of users, which then become the kind of searchable result of answers for guides, including security fixes.*" (P11). This difference in approaching guidance appears to be between projects with a more technical developer audience preferring coaching or static testing, vs. projects with less technical contributors such as scientists preferring extensive guidance, although our interview coverage of these aspects was too low to statistically confirm this.

Security Policies: Next, we were interested in the content and applicability of our participants' security and disclosure policies. Of our 27 participants, eight mention that their projects do not have specific security policies. P06 offers one possible explanation for this:

> "*So in the same way as people don't make a security policy on their repo unless something pushes them to do it or unless they have a security incident, people aren't going to document security best practices unless they've had a problem. Part of that is because they may not know to do so. But part of that is also because is there a need?*" — P06.

The most commonly mentioned security policy aspect (10) was related to providing a security-specific contact for the project and/or to a dedicated security team. Less common security policies include air gapping: "*The policy of [the project] is that any released software has to be built on a machine controlled by the release manager.*" (P11) and programming language-specific policies: "*Everything that is related to crypto or network code or parsing and so on is all written in Rust. That's already a kind of policy.*" (P02).

Only four of our participants explicitly mentioned not having any form of disclosure policy or security contact. Disclosure approaches mentioned by the other participants included a policy or plan for coordinated disclosure (10), private channels for disclosure (5), and plans for full disclosure, e. g., as public issue (2). The often heated debate regarding coordinated disclosure in open source projects extends to our participants: "*[the projects] say: we're just putting our users at too much risk. We're not sitting on patches, the people out there have installations on the front line, and because somebody likes to coordinate something, we're not waiting three months longer.*" (P01).

Testing and Reviews: Being closely related to policies, we also queried our participants about their security testing and review setup, with many participants mentioning automated tests and mandatory reviews: "*There are standard practices like there is a test suite, we've unit tests, integration tests, and as soon as we find any bugs or you write regression tests and there are codes, there's peer reviews of our codes and larger reviews of bigger PRs as well.*" (P05).

> Summary: Guidance and Policies. Our participants appear to diverge in their opinions regarding the helpfulness of (written) guidance. For security policies, larger projects mentioned dedicated security teams, while smaller projects mentioned a security contact channel. Most projects included some type of disclosure policy or at least contact for security issues.

D. Project Structure

With this section, we wanted to explore structures that are often not directly visible from repository artifacts, such as how build and deploy steps are set up, selection criteria and vulnerability checks for dependencies, and any additional infrastructure such as project websites and communication tools. The specific project setups appear to be as diverse as our participants' projects. As probably expected of open source projects, most development approaches appear to be somewhat open:

> "*It's an open-source project, everything from [build] stages to CI is in the same repository, and everyone can contribute to it. However, no one has direct control over anything because everything executed is a series of scripts and tests in the main repository, meaning that anyone can send a pull request tomorrow and modify them.*" — P25.

Code submissions are at the heart of open source collaboration, making pull request handling and build pipeline setup part of the overall security and trust strategy.

Pull Requests: Specifically for incoming pull requests, projects provide a number of controls, e. g., by protecting the main branch: "*The main branch is protected. Of course, we do everything through forks. Meaning, each developer has their own fork, opens a pull request and there's a limited number of people who have the permissions to do the final merge.*" (P19).

Our participants opt for a number of different strategies for merging code contributions, such as only rebasing on main: "*We actually always require from the author to rebase their changes on top of the main, so that we don't have the whole complex structure of merges [...] which actually helps to pinpoint any kind of problems [...]*" (P11), a majority

vote before merging pull requests: "*So on each PR you can review it and then give a thumbs-up or thumbs-down. And that's done by at least three of the main contributors, [...] and that means that it's a majority of them think that it's a worthy contribution.*" (P17), or an optimistic merging approach with resolving problems in follow-up pull requests: "*[Y]ou optimistically merge code as long as it passes some basic sanity checks. If someone thinks that the code which is merged isn't actually perfect, there is some way to improve it, they need to send a follow-up pull request.*" (P16). Overall, project structure and code submission handling appear to be specialized to the project's needs and community.

Build Pipeline: In the interviews, 23 participants mentioned using CI/CD or or other automatic build systems in their projects, with the majority relying on GitHub Actions (10). Aside from GitHub Actions, many different systems were mentioned, sometimes even within the same project: "*But basically we use everything, like Travis, Azure Pipelines, GitHub Actions, CircleCI, custom build machines and so on. It's quite a hodgepodge.*" (P02). A few participants (3) mentioned that they prefer manual builds and publishing for a number of reasons, e. g., "*I don't like the one click deploy, I like to actually see, you know, things fly by in the console.*" (P04). Running tests as part of the build pipeline is a common practice, with some of our participants taking advantage of this, e. g., "*[...] we have a huge number of tests, actually. More than 10,000 tests and 70 static check analyses.*" (P11) and "*Every pull request automatically goes through our full test suite [...] There are at least 1,000 files, each testing one area.*" (P12). Thoroughly testing every commit might include some trade-offs in the context of attracting contributors, as pointed out by P16: "*If the tests run in five seconds, then people will contribute, if the tests run in five hours, then people will contribute less.*" (P16).

Only four participants mentioned that they PGP sign commits in their projects, although not always for security reasons: "*I PGP sign all my commits. The main reason I do that is because it gives me a pretty little verified badge on all my commits.*" (P06). Reasons for not signing commits included technical limitations: "*[Commit signing] is one of the things that is rather difficult to do if you are using the GitHub workflow*" (P11) and different workflows: "*I don't make everybody do it, because eventually, the commit will get squashed when I merge it, and then it's going to be signed by GitHub automatically.*" (P24). Some of these issues might be alleviated by a recent Git patch introducing SSH-based signatures and verification, although it remains to be seen if and how collaborative platforms will adapt.

Dependencies: Common criteria for selecting a dependency included activity: "*Our most important criteria, in general, is that we do not want to rely on inactive projects.*" (P25) and reputation metrics like GitHub stars: "*If somebody was pulling in a package and I go to their GitHub and its got two stars and it's only used in this project, I'm probably going to say: 'Let's avoid using that'.*" (P24). Other participants had more involved criteria for including a dependency:

> "*What I usually do before including any dependency is I send them a pull request fixing something. And if they don't react on this or don't merge that one, then they don't become my dependency because they are obviously not interested in improving the software.*" — P18.

Some of such elaborate selection criteria even benefited all involved parties: "*As it happened also with [dependency]: we reached out, we got a good response. We worked on a few issues together, even I personally fixed one of those issues [...]*" (P11). Few mentioned that they manually review third party dependencies: "*Whenever we include a library in a project, we examine the project beforehand and two or three core contributors actually need to confirm that it looks okay.*" (P03). One participant mentioned looking for usages of specific language features that may affect security: "*I always go to the source code. I searched for all uses of unsafe and I check if they are, if they are like, if they make sense or not.*" (P22).

> Summary: Project Structure. Our participants appear to fully utilize modern build systems, including during testing and deploy. Only few projects explicitly use signed commits, often due to incompatibilities with their workflow or threat model. Selection criteria for dependencies range from readily available metrics over security reviews, to elaborate collaborations or even rewrites.

E. Releases and Updates

In this question section, we were interested in the projects' release decisions and schedules, whether there were guidelines in place regarding release deprecation, how the releases are distributed, and whether releases are digitally signed. The release decisions of our participants broadly fit into two approaches: either as periodical releases (9) or when specific features or patches are ready (10). Different communities seem to favor different release approaches, as our participants describe both feature-driven and cycle-driven release schedules based on community input: "*Periodically, we'll reach consensus in the community, and say, 'Hey, we ought to do a release', and so we'll stop developing for a few days and just make sure there aren't any major bugs.*" (P09) and "*We try to aim for three times a year, mostly because the real reason for the three times a year rough cycle is that we polled the community and the kind of the averaging that three times a year seemed like what suited people the most.*" (P13). Some participants utilize both approaches, depending on, e. g., project maturity:

> "*Mainline development continues just normally under main branch, and we have this temporary release branch where we merge in only bug fixes that come in during this time. This is for the most mature projects [...] For projects that move faster and don't have for example, back-holding strategy for bugs*

and stuff like this, we basically once a month tag a version and push them out." — P18.

Aside from set release windows, participants often mentioned a more flexible approach to vulnerability fixes, e. g., "*If you have a vulnerability, Spectre, Meltdown or something like that, then it can also happen that updates are released completely unscheduled.*" (P01).

The majority of our participants does not seem to specifically advertise new releases, e. g., "*Most people who interact with this project don't actually even look at my GitHub. They don't look at the release assets or anything like that, they just use [package registry] and it just works from there. They pull it down and use it automatically.*" (P24). Of the ones that do advertise, preferred channels included social channels like Twitter, Slack, or IRC (3), mailing lists (3), and websites (2). Again, our participants seem to prefer a practical approach for deprecating insecure or out-of-date releases, e. g., by simply stopping support: "*We only guarantee that we will backport security fixes to the last two releases. So anything before that is not an LTS we will not fix, which could be seen as deprecated from this point of view.*" (P25) and "*I don't have any official policy of supporting old versions, so they're effectively deprecated as soon as I release a new version.*" (P27).

For distributing releases, 12 participants specifically mention that they utilize external infrastructure such as registries, app stores, or package managers. As a reason for not distributing binary releases, P15 points to their community composition: "*We have no [binary] releases. We always build the project ourselves, there are no pre-built binaries for end users, because there are practically no end users.*" (P15), as well as P25: "*All of our releases are done on GitHub tag, because we release via source code, not via binaries, so it's a software release in the form of a git tag.*" (P25).

Of our participants, 11 were aware of their projects' releases being signed. Their reasons for not or not correctly signing releases included technical limitations:

> "*The Mac build is signed by my developer key, but the builds for Raspberry Pi, Linux, Windows, they're unsigned. People just have to trust the integrity that I'm the only person who has access to those and I did it right. We'd love to have better solutions for that, but none are available right now.*" — P09,

Another reason was their general singing setup, which lead to key ownership problems:

> "*[. . .] because our release procedure checklist only states sign, meaning sign them in general. So people use their GPG signing keys, and there is no control where and how those keys are verified or belong to a particular key ring. So this is something we need to improve.*" — P25.

Generally, our participants seem to be aware of the security benefits of signing and releasing checksums of releases, but some are not utilizing it for (all) releases due to technical limitations and platform restrictions.

> Summary: Releases and Updates. Our participants mostly publish projects' releases and updates based on direct community input and feedback, often mentioning exceptions from their usual schedule for vulnerability fixes. Release distribution and deprecation appear to be oriented towards practicality, utilizing package registries and other distribution infrastructures, again depending on the need of their users.

F. Roles and Responsibilities

In this section, we sought to establish what hierarchies exist between contributors in the participants' projects and how they affect the decision making process. We were also interested in roles that directly deal with the projects' security and role-specific duties.

Somewhat unsurprisingly, participants involved in projects with corporate stakeholders frequently mentioned sophisticated management structures that oversee the project's development: "*At the top of the pyramid, there's the PMC, the project management committee and they're essentially the people who either funded the project or major industrial, or representatives of major industrial partners.*" (P13). Most of our participants described the contributor hierarchy in their projects as having two levels: The core team that is tasked with reviewing code submissions and that has permissions to merge new code into the source tree and everyone else whose code is subject to the code review process. The core team was often called a group of maintainers or simply committers:

> "*There's two classes of contributor. There are the maintainers and then there's pretty much everyone else. The maintainers are me and maybe seven other people who contribute regularly to the project [. . .] They can push directly to the main branch of the project.*" — P13.

Other projects make a distinction between the core team and the project's owners, or they even have a dedicated role for developers who have the ability to manipulate the repository itself, e. g., by pushing to branches corresponding to pull requests: "*[. . .] then there are about [a few] people who have maintainer status, so they can merge requests. And then there is about a hundred people who have developer access, so they can push to a branch inside the merge request.*" (P26). Some projects take centralization further and follow the so-called Benevolent Dictator for Life (BDFL) model, where the project's founder steers the overall direction of the project and has the final say in disputes: "*[The project] is what [one of our contributors] has dubbed a do-ocracy, and that is basically whoever's writing the code gets to decide how it's done, but our benevolent dictator has the final say so. We essentially have this benevolent dictator, and everybody else under that.*" (P09). Participants whose projects have not grown out of corporate contexts often mentioned a more relaxed contributor structure, with direct influences on the code review process:

> "*It is basically a much more peer-to-peer structure than a hierarchical structure. If you develop something, you don't need to submit it to somebody to*

get it into the tree. You do need to get a review from people who are competent in this area, but that's all." — P08.

Only five participants stated they were aware of roles within their projects that deal with security. P08 summarized the security team's obligations as follows: "*You can communicate privately with the security team. They would classify your issues and decide if it matches the criteria for the issued security notice, how to proceed with patches, and how to publish them.*" (P08). Three of the five participants mentioned roles that are not primarily or only indirectly involved with security, such as IT departments or sysadmins: "*We obviously have a IT department that would follow up on [security incidents].*" (P19). Relying on a security response team existing within the parent organization or foundation of the project was more uncommon, which two participants reported: "*There is a whole security team at [organization]. They are pre-vetting those issues, and filtering them, and contacting the PMC members of the projects involved, whenever they see there is a need to follow up on certain security issues.*" (P11).

> Summary: Roles and Responsibilities. Our participants' projects have a variety of contributor hierarchies which are mostly relatively flat with two levels. This practical approach seems to be prevalent in projects of any size, bar very small (single person) projects or ones that grew out of a corporate context. Most of the projects do not staff teams dedicated to project security, with some either relying on their organization's resources or leveraging members of other teams such as their IT admins.

G. Trust Processes

In this section, we explore the general trust model of projects, as well as their handling of, and strategies for trust challenges. We were also interested in how recently onboarded contributors can become trusted members and if identity checks or the the signing of a CLA is required.

Trust Models: Establishing trust for new committers is an important step in the OSS onboarding process. The majority of our participants described some form of meritocracy when asked how new contributors gain trust within the community, i. e., by making frequent, high-quality contributions to the project:

> "*So it's purely on contributions to the project, so it's meritocracy based. And this means that the person essentially starts usually either just helping out on filing issues like well documented issues, filing pull requests and again well documented, reviewing pull requests is also an important aspect of it.*" — P22.

A less common approach involves trusting unknown contributors by default and giving them access early in the hope of facilitating first-time contributions: "*I really want to empower people to contribute. [. . .] it's very easy to get access to [the project]. It's not like super easy, but you just submit patches and if you do some useful work, I default to just give you the commit access.*" (P16).

CLAs appear to be still somewhat rare, with only four participants mentioning that their projects require one, e. g.,

> "*For licensing purposes, we require a [CLA], because the project is licensed under the BSD license. We have to have people assign their copyright, so when people want to contribute, they fill out a form, just sign it. It says, 'Hey, I'm releasing my contributions under the Berkeley Style License'.*" — P09.

This low number agrees with the personal impressions of some of our participants, e. g., "*[. . .] I think that was the only time I ever had to [sign a CLA], and I've submitted lots of pull requests to many different projects. It doesn't seem to be very widely used.*" (P24). In our interviews, projects affiliated with corporate stakeholders or other organizations appear to be more likely to require a CLA.

Trust Incidents: The term "*trust incident*" can cover a wide field of potential incidents, including social conflicts due to open source project often being community-focused. Still, the majority of participants, 20, reported to never have experienced a trust incident (by their definition) in their projects. Described trust incidents included drive-by cryptocurrency miner commits, failed background checks, and a pro-active block after potential SSH key theft. Somewhat unsurprisingly, larger and likely older projects appear to have had more experience with trust incidents in the past.

The fact that most projects have never experienced a trust incident is also reflected in their incident handling strategy, with multiple participants reporting not having previously thought about such cases, e. g., "*[. . .] since it has never happened, it is not something I have thought about.*" (P26). Reported incident response strategies, especially by smaller projects, seem to be decided on a case-by-case basis, e. g., "*[Incident response] is decided dynamically from case to case. The infrastructures are so small that you can do this relatively quickly. So it's not like in the company that we have incident playbooks. There are too few people involved for that.*" (P01). Again, larger, and likely older projects appear to have a more codified incident handling strategy in place. Two participants pointed to their project's or organization's code of conduct, which codifies the steps to take in the case of a breach of trust:

> "*That is one place where then the code of conduct will start to kick in. We actually have an enforcement section for code of conduct with a step-by-step escalation, which basically ends up with everything from just asking someone not to do something through to banning them and removing access.*" — P27

> Summary: Trust Processes. Most of our participants use some form of meritocracy for establishing trust with new contributors, with some even assuming trustworthiness by default to facilitate first-time contributions. The majority of participants never experienced a trust incident in their projects and also did not establish specific trust incident strategies. Larger, likely older projects seem to have more

past experience with incidents, and often offer more specific strategies.

H. Opinions and Improvements

Lastly, we asked participants about both the internal and external reputation of their project in the context of security, as well as how they would personally like to improve security and trust in their projects.

With one exception, all participants reported a high internal reputation of their projects, e. g., "*Amongst the people on the project, everybody trusts it a lot.*" (P09) and "*We follow very, very high standards there, mainly because we have a few people who are very, very keen on that.*" (P11). The same generally holds true for the external reputation, although many participants are unsure about the actual awareness of the project outside of their community. Overall, our participants appear to take pride in their projects, but are quite humble about their importance and reach in the OSS ecosystem.

We also asked our participants how they would like to improve security and trust in their projects, assuming no limitations. For reporting, we roughly sorted the suggested improvements into mainly requiring more person-hours (15), requiring more money (9), or requiring a different infrastructure (9). Improvements requiring more person-hours focus on alleviating past software development decisions and technical debt, e. g., "*If I could, I would write the entire stack myself.*" (P14) and "*[. . .] I would rewrite a lot of the code. That's just a historical thing, because it has already become big and complex [. . .] It's just like building a house; you'd have to build it three times before it becomes good.*" (P20). Another focus was enhancing the review process, e. g., "*So the first thing I do is that a group of people would review every pull request exclusively from the view of security.*" (P25).

Some of the improvements mainly requiring more money also translated into necessitating more person-hours, just by buying the time, e. g., "*I could always use more participants in the review process and so if I could hire some people, if I had the disposable income to do that, I would probably hire people to get more eyes on pull requests than just myself [. . .]*" (P24) and "*I think getting more tools and more CI-type tools to watch for that, because I think humans are vulnerable [. . .] If I had unlimited budget and unlimited engineers, I'd really work on improving our testing systems.*" (P23). Other money-based improvements included the introduction of security bounties: "*[Projects] mentioned they tried all the different kinds of things, and the only thing that worked well was [a] bounty process, and having bounties, and being able to reward security researchers to bring up the security issues.*" (P11).

For improvements requiring infrastructure, participants mentioned improvements to build and test pipelines, e. g., "*with unlimited resources, I would like some more investment into automatic tools that are better in like finding vulnerabilities and problems with code.*" (P07) and "*I would like to build [the binaries] on my own machine and then ship the site final result. For anything binary related, that would be way better than what we have right now.*" (P18). Other participants mentioned transitioning their projects' codebases to other languages, e. g. Rust: "*What I'd like to do is oxidize [the project] over time, to integrate Rust and Rust code into the codebase – which is quite an undertaking [. . .] and an incredibly tedious task to do it well.*" (P03). Overall, even improvements initially requiring more money or a different infrastructure were traceable to the crux of all open source project: the need for more contributors.

> Summary: Problems and Improvements. Our participants take pride in their projects, but are quite humble about their importance and reach in the OSS ecosystem. Overall, even improvements initially requiring more money or a different infrastructure ended up targeting the project's need for more contributors.

V. Discussion

In this work, we investigated the security measures and trust processes of a diverse set of open source project. We conducted 27 in-depth, semi-structured interviews with open source contributors, maintainers, and owners to explore the following research questions:

RQ1: *"How are open source projects structured behind-the-scenes?"* Our participants described their contributor hierarchy as being mostly based on two levels: a core group of maintainers tasked with reviewing code submissions and with permissions to merge new code into the source tree and other contributors that are subjected to a code review process. Most of the projects do not staff dedicated security teams, with some relying on other teams for security, such as their IT admins or their organization's resources. Release processes appear to be oriented towards practicality, including decisions based on direct community input and feedback and utilizing package registries and other distribution infrastructures depending on the needs of their users. Our participants appear to fully utilize modern build systems, including during testing and deploy, with criteria for dependencies ranging from readily available metrics to elaborate reviews.

RQ2: *"If and what guidance and policies are provided by open source projects?"* Our participants appear to diverge in their opinion regarding the helpfulness of (written) guidance, with some preferring more hands-on approaches to knowledge transfer. For security policies, rather large projects described dedicated security teams, while smaller projects just offered a security contact point. Most projects mentioned some type of disclosure policy or contact for security issues.

RQ3: *"How do open source projects approach security and trust challenges?"* Most of our participants reported having experienced neither a security nor trust incident in the past, although many of our participants were familiar with suspicious or low quality commits, as well as potential vulnerabilities introduced by dependencies. Most of our participants use some form of meritocracy for establishing trust with new contributors, with some even assuming trustworthiness by default to facilitate first-time contributions. Participants with larger, older projects more frequently

reported incidents and approaches for incident handling.

Below we discuss some of our additional findings in greater detail. Open source projects are part of a larger connected ecosystem of components, libraries, and software registries. A single compromised dependency can introduce vulnerabilities into thousands of projects further down the chain, a fact that our participants were keenly aware of:

> "*What we don't have is the money to fix all the dependencies, like all the ones that depend on the project because every backward incompatible change that we will do in the project to address the security concern would have repercussions in the ecosystem that goes beyond our own project.*" — P22.

In general, project development as described by our participants appears to be highly community-driven and practical: important decisions such as release windows, announcements, and distribution infrastructure are all based on the input, feedback, and needs of contributors and users. Most projects appear to handle security and trust incidents "as they happen". This seems to be a pragmatic strategy, as it seems unlikely that a project could cover all possible incident types beforehand, especially with the limited personpower of smaller communities.

As mentioned by our participants, the combination of deep dependency chains and automatic testing can lead to many false positive security warnings. These false positives can lead to a habituation effect, as summarized by a participant:

> "*So one false positive is worse than missing a real vulnerability, in my opinion, because if you miss a real vulnerability, everyone's like, oh, we better care more about security. If there's a false positive, then everyone says, oh, security warnings are bullshit. It is much harder to unwind the security-warnings-are-bullshit attitude than it is to make people care about security.*" — P06

Fittingly, we can let one of our participant's words help with summarizing our general findings: "*Ultimately, I believe that people are the key. Automation is something that can help people. But in the end, the people are like the ultimate barrier between the harm and the intent.*" (P10).

VI. Conclusion

In 27 in-depth, semi-structured interviews with owners, maintainers, and contributors from a diverse set of open source projects, we investigate their security measures and trust processes. We explore projects' behind-the-scene processes, provided guidance & policies, as well as past challenges and incident handling. We find that our participants' projects are highly diverse both in deployed security measures and trust processes, as well as their underlying motivations.

As projects grow in scope and contributors, so grow their needs for security and trust processes. We argue for supporting projects in ways that their growth supports. A small three person project will never live up to security and trust measures provided by a 1,000+ maintainer project with corporate backing, yet it should not be left out of any support. Interesting aspects for future consideration include the type and applicability of support for small projects, as well as identifying measures with the best trade-off in working hours and security improvement.

Especially smaller projects handle security and trust incidents "as they happen". Elaborated incident playbooks and committee structures are likely of little use to these projects due to frequently changing committers and structures. We surmise that especially these smaller projects could be better supported with public, general example playbooks and resources for incidents, that they then can utilize when the need arises.

As researchers, we advocate against treating open source developers solely as data sources and review process black-boxes, and instead to consider them as valuable partners in bringing security and trust to OSS and software ecosystems as a whole. Overall, we argue for supporting open source projects in ways that better consider their individual strengths and limitations, especially in the case of smaller projects with low contributor numbers and limited access to resources.

Acknowledgements

With this, we want to acknowledge our interviewees for their participation: It was a great experience to interview you for this study. We appreciate your knowledge, project information, and most importantly your valuable time that you have generously given. We hope that with this work and your contribution, both the research and open source community are one step closer to more secure and trustworthy software. Last but not least, we thank the anonymous reviewers for their valuable feedback.

References

[1] GitHub, *The State of the Octoverse*, https://octoverse.github.com/, Accessed: 2021-10-15, 2020.

[2] Microsoft, *GitHub*, https://github.com, Accessed: 2021-09-29, 2008.

[3] GitLab Inc., *GitLab*, https://gitlab.com, Accessed: 2021-09-29, 2014.

[4] Cybersecurity and I. S. A. (CISA), *Malware discovered in popular npm package, ua-parser-js*, https://us-cert.cisa.gov/ncas/current-activity/2021/10/22/malware-discovered-popular-npm-package-ua-parser-js, Accessed: 2021-10-24, 2021.

[5] L. Abrams, *Popular npm library hijacked to install password-stealers, miners*, https://www.bleepingcomputer.com/news/security/popular-npm-library-hijacked-to-install-password-stealers-miners/, Accessed: 2021-10-24, 2021.

[6] M. Hanley, *Github's commitment to npm ecosystem security*, https://github.blog/2021-11-15-githubs-commitment-to-npm-ecosystem-security/#security-issues-related-to-the-npm-registry, Accessed: 2021-11-17, Nov. 2021.

[7] A. Sharma, *Npm fixes private package names leak, serious authorization bug*, https://www.bleepingcomputer.com/news/security/npm-fixes-private-package-names-leak-serious-authorization-bug/, Accessed: 2021-11-16, Nov. 2021.

[8] The Linux Foundation, "Open source software supply chain security," Tech. Rep., Feb. 2020, Accessed: 2021-11-16.

[9] RedHat, *The State of Enterprise Open Source 2020: Enterprise open source use rises, proprietary software declines*, https://www.redhat.com/en/blog/state-enterprise-open-source-2020-enterprise-open-source-use-rises-proprietary-software-declines, Accessed: 2021-06-23, Feb. 2020.

[10] Linux Foundation's Technical Advisory Board, *Report on University of Minnesota breach-of-trust incident*, https://lwn.net/ml/linux-kernel/202105051005.49BFABCE@keescook/, Accessed: 2021-11-27, May 2021.

[11] K. Thompson, "Reflections on trusting trust," *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984.

[12] A. Pietri, D. Spinellis, and S. Zacchiroli, "The software heritage graph dataset: Public software development under one roof," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19, Montreal, Quebec, Canada: IEEE Press, 2019, pp. 138–142.

[13] ——, "The software heritage graph dataset: Large-scale analysis of public software development history," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20, Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 1–5.

[14] A. Alali, H. Kagdi, and J. I. Maletic, "What's a typical commit? a characterization of open source software repositories," in *2008 16th IEEE International Conference on Program Comprehension*, 2008, pp. 182–191.

[15] G. Robles, L. Arjona Reina, A. Serebrenik, B. Vasilescu, and J. M. González-Barahona, "Floss 2013: A survey dataset about free software contributors: Challenges for curating, sharing, and combining," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, New York, NY, USA: Association for Computing Machinery, 2014, pp. 396–399.

[16] A. Gkortzis, D. Mitropoulos, and D. Spinellis, "Vulinoss: A dataset of security vulnerabilities in open-source systems," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18, Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 18–21.

[17] M. Shahzad, M. Z. Shafiq, and A. X. Liu, "A large scale exploratory analysis of software vulnerability life cycles," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12, Zurich, Switzerland: IEEE Press, 2012, pp. 771–781.

[18] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, ser. MSR '12, Zurich, Switzerland: IEEE Press, 2012, pp. 12–21.

[19] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean ghtorrent: Github data on demand," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, Hyderabad, India: Association for Computing Machinery, 2014, pp. 384–387.

[20] Q. Tu *et al.*, "Evolution in open source software: A case study," in *Proceedings 2000 International Conference on Software Maintenance*, IEEE, 2000, pp. 131–142.

[21] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.

[22] T. T. Dinh-Trong and J. M. Bieman, "The freebsd project: A replication case study of open source development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 481–494, 2005.

[23] N. Edwards and L. Chen, "An historical examination of open source releases and their vulnerabilities," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12, Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, pp. 183–194.

[24] A. D. Householder, J. Chrabaszcz, T. Novelly, D. Warren, and J. M. Spring, "Historical analysis of exploit availability timelines," in *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, 2020.

[25] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "When are oss developers more likely to introduce vulnerable code changes? a case study," in *Open Source Software: Mobile Open Source Technologies*, L. Corral, A. Sillitti, G. Succi, J. Vlasenko, and A. I. Wasserman, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 234–236.

[26] P. Anbalagan and M. Vouk, "Towards a unifying approach in understanding security problems," in *20th International Symposium on Software Reliability Engineering*, 2009, pp. 136–145.

[27] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, "Bug characteristics in open source software," *Empirical software engineering*, vol. 19, no. 6, pp. 1665–1705, 2014.

[28] J. Walden, "The impact of a major security event on an open source project: The case of openssl," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 409–419.

[29] K. Altinkemer, J. Rees, and S. Sridhar, "Vulnerabilities and patches of open source software: An empirical study," *Journal of Information System Security*, vol. 4, no. 2, pp. 3–25, 2008.

[30] M. Alenezi and Y. Javed, "Open source web application security: A static analysis approach," in *2016 International Conference on Engineering & MIS (ICEMIS)*, 2016, pp. 1–5.

[31] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, "How open source projects use static code analysis tools in continuous integration pipelines," in *Proceedings of the 14th International Conference on Mining Software Repositories*, ser. MSR '17, Buenos Aires, Argentina: IEEE Press, 2017, pp. 334–344.

[32] M. Zahedi, M. Ali Babar, and C. Treude, "An empirical study of security issues posted in open source projects," in *Proceedings of the 51st Hawaii International Conference on System Sciences (HICSS18)*, 2018, pp. 5504–5513.

[33] P. Deligiannis, A. F. Donaldson, and Z. Rakamaric, "Fast and precise symbolic analysis of concurrency bugs in device drivers," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2015, pp. 166–177.

[34] J.-J. Bai, J. Lawall, Q.-L. Chen, and S.-M. Hu, "Effective static analysis of concurrency use-after-free bugs in linux device drivers," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 255–268.

[35] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, May 2005.

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

[37] R. Ramsauer, L. Bulwahn, D. Lohmann, and W. Mauerer, "The sound of silence: Mining security vulnerabilities from secret integration channels in open-source projects," in *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, ser. CCSW'20, Virtual Event, USA: Association for Computing Machinery, 2020.

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

[39] L. Dabbish, C. Stuart, J. Tsay, and J. 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*, ser. CSCW '12, Seattle, Washington, USA: Association for Computing Machinery, 2012, pp. 1277–1286.

[40] B. Vasilescu, K. Blincoe, Q. Xuan, *et al.*, "The sky is not the limit: Multitasking across github projects," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16, Austin, Texas: Association for Computing Machinery, 2016, pp. 994–1005.

[41] K. Constantino, M. Souza, S. Zhou, E. Figueiredo, and C. Kästner, "Perceptions of open-source software developers on collaborations: An interview and survey study," *Journal of Software: Evolution and Process*, e2393, 2021.

[42] L. Moldon, M. Strohmaier, and J. 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)*, 2021, pp. 549–561.

[43] C. Overney, J. Meinicke, C. Kästner, and B. 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*, 2020, pp. 1209–1221.

[44] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, Hyderabad, India: Association for Computing Machinery, 2014, pp. 345–355.

[45] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, Hyderabad, India: Association for Computing Machinery, 2014, pp. 356–366.

[46] D. Ford, M. Behroozi, A. Serebrenik, and C. Parnin, "Beyond the code itself: How programmers really look at pull requests," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society*, ser. ICSE-SEIS '19, Montreal, Quebec, Canada: IEEE Press, 2019, pp. 51–60.

[47] W. Li, N. Meng, L. Li, and H. Cai, "Understanding language selection in multi-language software projects on github," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 256–257.

[48] H. Hata, R. G. Kula, T. Ishio, and C. Treude, "Research artifact: The potential of meta-maintenance on github," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 192–193.

[49] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, Paderborn, Germany: Association for Computing Machinery, 2017.

[50] C. Miller, S. Cohen, B. Vasilescu, and C. Kästner, ""Did You Miss My Comment or What?" understanding toxicity in open source discussions," in *In 44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022*, Pittsburgh, PA, USA: ACM, May 2022.

[51] D. Sondhi, A. Gupta, S. Purandare, A. Rana, D. Kaushal, and R. Purandare, "Dataset to study indirectly dependent documentation in github repositories," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 215–216.

[52] R. Li, P. Pandurangan, H. Frluckaj, and L. Dabbish, "Code of conduct conversations in open source software projects on github," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, no. CSCW1, Apr. 2021.

[53] D. Botta, R. Werlinger, A. Gagné, *et al.*, "Towards understanding it security professionals and their tools," in *Proceedings of the 3rd Symposium on Usable Privacy and Security*, ser. SOUPS '07, Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2007, pp. 100–111.

[54] M. Silic and A. Back, "Information security and open source dual use security software: Trust paradox," in *Open Source Software: Quality Verification*, E. Petrinja, G. Succi, N. El Ioini, and A. Sillitti, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 194–206.

[55] L. Bauer, L. F. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea, "Real life challenges in access-control management," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 899–908.

[56] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker, "Field studies of computer system administrators: Analysis of system management tools and practices," in *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, 2004, pp. 388–395.

[57] R. A. Bridges, M. D. Iannacone, J. R. Goodall, and J. M. Beaver, "How do information security workers use host data? a summary of interviews with security analysts," *arXiv preprint arXiv:1812.02867*, 2018.

[58] S. E. McGregor, P. Charters, T. Holliday, and F. Roesner, "Investigating the computer security practices and needs of journalists," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 399–414.

[59] S. E. McGregor, E. A. Watkins, M. N. Al-Ameen, K. Caine, and F. Roesner, "When the weakest link is strong: Secure collaboration in the case of the panama papers," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 505–522.

[60] C. Chen, N. Dell, and F. Roesner, "Computer security and privacy in the interactions between victim service providers and human trafficking survivors," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 89–104.

[61] W. Bai, M. Namara, Y. Qian, P. G. Kelley, M. L. Mazurek, and D. Kim, "An inconvenient trust: User attitudes toward security and usability tradeoffs for key-directory encryption systems," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016, pp. 113–130.

[62] K. Gallagher, S. Patil, and N. Memon, "New me: Understanding expert and non-expert perceptions and usage of the tor anonymity network," in *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, 2017, pp. 385–398.

[63] M. Gutfleisch, J. H. Klemmer, N. Busch, Y. Acar, M. A. Sasse, and S. Fahl, "How does usable security (not) end up in software products? results from a qualitative interview study," in *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*, IEEE Computer Society, May 2022.

[64] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15, Vancouver, BC, Canada: Association for Computing Machinery, 2015, pp. 1379–1392.

[65] S. Balali, U. Annamalai, H. S. Padala, *et al.*, "Recommending tasks to newcomers in oss projects: How do mentors handle it?" In *Proceedings of the 16th International Symposium on Open Collaboration*, ser. OpenSym 2020, Virtual conference, Spain: Association for Computing Machinery, 2020.

[66] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, *Understanding free/open source software development processes*, 2006.

[67] K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/libre open-source software development: What we know and what we do not know," *ACM Comput. Surv.*, vol. 44, no. 2, Mar. 2008.

[68] S.-F. Wen, "Software security in open source development: A systematic literature review," in *2017 21st Conference of Open Innovations Association (FRUCT)*, 2017, pp. 364–373.

[69] L. P. Hattori and M. Lanza, "On the nature of commits," in *2008 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*, 2008, pp. 63–71.

[70] J. C. S. Santos, A. Peruma, M. Mirakhorli, M. Galstery, J. V. Vidal, and A. Sejfia, "Understanding software vulnerabilities related to architectural security tactics: An empirical investigation of chromium, php and thunderbird," in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 69–78.

[71] D. Pletea, B. Vasilescu, and A. Serebrenik, "Security and emotion: Sentiment analysis of security discussions on github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, New York, NY, USA: Association for Computing Machinery, 2014, pp. 348–351.

[72] G. Antal, M. Keleti, and P. Hegedűns, "Exploring the security awareness of the python and javascript open source communities," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20, New York, NY, USA: Association for Computing Machinery, 2020, pp. 16–20.

[73] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "Identifying the characteristics of vulnerable code changes: An empirical study," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, 2014, pp. 257–268.

[74] H. Perl, S. Dechand, M. Smith, *et al.*, "Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, ACM, 2015, pp. 426–437.

[75] I. Abunadi and M. Alenezi, "Towards cross project vulnerability prediction in open source web applications," in *Proceedings of the The International Conference on Engineering & MIS 2015*, ser. ICEMIS '15, Istanbul, Turkey: Association for Computing Machinery, 2015.

[76] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, New York, NY, USA: Association for Computing Machinery, 2017, pp. 914–919.

[77] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. 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*, ser. ICSE-NIER '20, Seoul,

South Korea: Association for Computing Machinery, 2020, pp. 57–60.
[78] S. Bugiel, L. V. Davi, and S. Schulz, "Scalable trust establishment with software reputation," in *Proceedings of the sixth ACM workshop on Scalable trusted computing*, 2011, pp. 15–24.
[79] M. Syeed, J. Lindman, and I. Hammouda, "Measuring perceived trust in open source software communities," in *Open Source Systems: Towards Robust Practices*, F. Balaguer, R. Di Cosmo, A. Garrido, F. Kon, G. Robles, and S. Zacchiroli, Eds., Cham: Springer International Publishing, 2017.
[80] M. Antikainen, T. Aaltonen, and J. Väisänen, "The role of trust in oss communities — case linux kernel community," in *Open Source Development, Adoption and Innovation*, J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti, Eds., Boston, MA: Springer US, 2007, pp. 223–228.
[81] V. S. Sinha, S. Mani, and S. Sinha, "Entering the circle of trust: Developer initiation as committers in open-source projects," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11, Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 133–142.
[82] A. Bosu and J. C. Carver, "Impact of developer reputation on code review outcomes in oss projects: An empirical investigation," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14, Torino, Italy: Association for Computing Machinery, 2014.
[83] C. Thompson and D. Wagner, "A large-scale study of modern code review and security in open source projects," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, Toronto, Canada: Association for Computing Machinery, 2017.
[84] A.-K. Groven, K. Haaland, R. Glott, and A. Tannenberg, "Security measurements within the framework of quality assessment models for free/libre open source software," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ser. ECSA '10, Copenhagen, Denmark: Association for Computing Machinery, 2010, pp. 229–235.
[85] J. Ryoo, B. Malone, P. A. Laplante, and P. Anand, "The use of security tactics in open source software projects," *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1195–1204, 2016.
[86] V. N. Subramanian, I. Rehman, M. Nagappan, and R. G. Kula, "Analyzing first contributions on github: What do newcomers do," *IEEE Software*, 2020.
[87] A. Hars and S. Ou, "Working for free? motivations for participating in open-source projects," *Int. J. Electron. Commerce*, vol. 6, no. 3, pp. 25–39, Apr. 2002.
[88] C. Hannebauer and V. Gruhn, "Motivation of newcomers to floss projects," in *Proceedings of the 12th International Symposium on Open Collaboration*, ser. OpenSym '16, Berlin, Germany: Association for Computing Machinery, 2016.
[89] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 112–123.
[90] C. Miller, D. G. Widder, C. Kästner, and B. Vasilescu, "Why do people give up flossing? a study of contributor disengagement in open source," in *Open Source Systems*, Springer International Publishing, 2019, pp. 116–129.
[91] S.-F. Wen, "Learning secure programming in open source software communities: A socio-technical view," in *Proceedings of the 6th International Conference on Information and Education Technology*, ser. ICIET '18, Osaka, Japan: Association for Computing Machinery, 2018.
[92] I. Steinmacher, T. U. Conte, C. Treude, and M. A. Gerosa, "Overcoming open source project entry barriers with a portal for newcomers," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16, Austin, Texas: Association for Computing Machinery, 2016, pp. 273–284.
[93] I. Steinmacher, C. Treude, and M. A. Gerosa, "Let me in: Guidelines for the successful onboarding of newcomers to open source projects," *IEEE Software*, vol. 36, no. 4, pp. 41–49, 2019.
[94] J. Dominic, J. Houser, I. Steinmacher, C. Ritter, and P. Rodeghero, "Conversational bot for newcomers onboarding to open source projects," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW'20, Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 46–50.
[95] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "Who is going to mentor newcomers in open source projects?" In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12, Cary, North Carolina: Association for Computing Machinery, 2012.
[96] K. Blincoe, F. Harrison, and D. Damian, "Ecosystems in github and a method for ecosystem identification using reference coupling," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15, Florence, Italy: IEEE Press, 2015, pp. 202–207.
[97] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov, "Developer onboarding in github: The role of prior social links and language experience," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015, Bergamo, Italy: Association for Computing Machinery, 2015, pp. 817–828.
[98] J. Salter, *Linux kernel team rejects university of minnesota researchers' apology*, https://arstechnica.com/gadgets/2021/04/linux-kernel-team-rejects-university-of-minnesota-researchers-apology/, Accessed: 2021-11-27, Apr. 2021.
[99] T. Holz and A. Oprea, *IEEE S&P'21 program committee statement regarding the "hypocrite commits" paper*, https://www.ieee-security.org/TC/SP2021/downloads/2021_PC_Statement.pdf, Accessed: 2021-11-27, May 2021.
[100] K. Charmaz, *Constructing Grounded Theory*. Sage, 2014.
[101] A. Strauss and J. M. Corbin, *Grounded theory in practice*. Sage, 1997, p. 288.
[102] J. Corbin and A. Strauss, "Grounded theory research: Procedures, canons and evaluative criteria," *Qualitative Sociology*, vol. 19, no. 6, pp. 418–427, 1990.
[103] C. Urquhart, *Grounded theory for qualitative research: A practical guide*. Sage, 2012.
[104] M. Birks and J. Mills, *Grounded theory: A practical guide*. Sage, 2015.
[105] E. Kenneally and D. Dittrich, "The Menlo report: Ethical principles guiding information and communication technology research," *SSRN Electronic Journal*, Aug. 2012.
[106] *Guidelines for research on the kernel community*, https://lwn.net/Articles/888891/, Accessed: 2022-03-31, Mar. 2022.

Appendix A
Recruitment Criteria

Our general recruitment approach in the repository channel was a stratified sampling in quartiles of GitHub repositories ranked by both a "*popularity*" and an "*activity*" score. We based this score on repository-level metrics provided by the GitHub API such as the number of commits and committers as well as the number of stars and forks.

Our initial repository dataset was downloaded in July 2021 from *GH Archive* (https://www.gharchive.org/), a service providing historical GitHub repository data, publicly available for further analysis. We limited our dataset to code repositories that received at least 40 commits from at least 20 distinct committers in the previous six months, which sets a minimum threshold for any given selected project's activity. This was done with the intent of excluding inactive and personal projects, in which our inquiry would either not reach active contributors or where interpersonal trust processes are irrelevant.

The resulting 15,256 repositories were enriched with up-to-date data from GitHub's API, such as programming language usage, topic tags, as well as star and fork counts. Users usually give out stars as a means of bookmarking a project or to explicitly value a project's merit. A project is "forked" into a user's namespace for them to be able to make changes

to its code base and consequently create a pull request for their changes to be accepted into the main source tree. The combination of the number of a project's stars and forks can thus serve as a proxy for its popularity. To ensure that the selected projects recently went through the onboarding of new contributors, we only proceeded with those that gained new committers in July 2021, and which had not contributed to the project before. After excluding duplicate repositories as well as repositories exclusively containing markup languages, we arrived at a set of 4,456 projects for final consideration.

We joined the popularity and activity indicators to a combined ranking and divided the set of projects into quartiles. This ensured high diversity across the indicators, while minimizing the amount of strata. We then iteratively selected and contacted projects from each stratum (e. g., first project from 1st quartile, first project from 2nd quartile, and so on) until we reached interview saturation.