Security Considerations in the Open Source Software Ecosystem

Von der Fakultät für Elektrotechnik und Informatik

der Gottfried Wilhelm Leibniz Universität Hannover

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften

(abgekürzt: Dr. rer. nat.)

genehmigte Dissertation

von Herrn

Dominik Wermke, M.Sc.

2023

1. Referent:	Prof. Dr. Sascha Fahl
2. Referentin:	Prof. Dr. Yasemin Acar
3. Referent:	Prof. Dr. Esfandiar Mohammadi
Vorsitz:	Prof. Dr. Ziawasch Abedjan

Tag der Promotion: 15.05.2023

Summary

Open source software plays an important role in the software supply chain, allowing stakeholders to utilize open source components as building blocks in their software, tooling, and infrastructure. But relying on the open source ecosystem introduces unique challenges, both in terms of security and trust, as well as in terms of supply chain reliability.

In this dissertation, I investigate approaches, considerations, and encountered challenges of stakeholders in the context of security, privacy, and trustworthiness of the open source software supply chain. Overall, my research aims to empower and support software experts with the knowledge and resources necessary to achieve a more secure and trustworthy open source software ecosystem. In the first part of this dissertation, I describe a research study investigating the security and trust practices in open source projects by interviewing 27 owners, maintainers, and contributors from a diverse set of projects to explore their behind-the-scenes processes, guidance and policies, incident handling, and encountered challenges, finding that participants' projects are highly diverse in terms of their deployed security measures and trust processes, as well as their underlying motivations. More on the consumer side of the open source software supply chain, I investigated the use of open source components in industry projects by interviewing 25 software developers, architects, and engineers to understand their projects' processes, decisions, and considerations in the context of external open source code, finding that open source components play an important role in many of the industry projects, and that most projects have some form of company policy or best practice for including external code. On the side of end-user focused software, I present a study investigating the use of software obfuscation in Android applications, which is a recommended practice to protect against plagiarism and repackaging. The study leveraged a multi-pronged approach including a large-scale measurement, a developer survey, and a programming experiment, finding that only 24.92% of apps are obfuscated by their developer, that developers do not fear theft of their own apps, and have difficulties obfuscating their own apps. Lastly, to involve end users themselves, I describe a survey with 200 users of cloud office suites to investigate their security and privacy perceptions and expectations, with findings suggesting that users are generally aware of basic security implications, but lack technical knowledge for envisioning some threat models.

The key findings of this dissertation include that open source projects have highly diverse security measures, trust processes, and underlying motivations. That the projects' security and trust needs are likely best met in ways that consider their individual strengths, limitations, and project stage, especially for smaller projects with limited access to resources. That open source components play an important role in industry projects, and that those projects often have some form of company policy or best practice for including external code, but developers wish for more resources to better audit included components.

This dissertation emphasizes the importance of collaboration and shared responsibility in building and maintaining the open source software ecosystem, with developers, maintainers, end users, researchers, and other stakeholders alike ensuring that the ecosystem remains a secure, trustworthy, and healthy resource for everyone to rely on.

Keywords

Open Source, Software Supply Chain, Usable Security, Software Developers

Zusammenfassung

Open-Source-Software spielt eine wichtige Rolle in der Software-Versorgungskette, da sie Beteiligten ermöglicht, Open-Source-Komponenten als Bausteine in ihrer Software, Werkzeugen und Infrastruktur zu verwenden. Die Verwendung von Open-Source-Komponenten bringt jedoch auch spezielle Herausforderungen mit sich, sowohl in Bezug auf die Sicherheit und Vertrauen, als auch im Hinblick auf die Zuverlässigkeit.

Diese Dissertation untersucht Ansätze, Überlegungen und Herausforderungen von Beteiligten im Zusammenhang mit der Sicherheit, dem Datenschutz und der Vertrauenswürdigkeit der Open-Source-Versorgungskette. Meine Forschung zielt darauf ab, Software-Experten mit dem Wissen und den Ressourcen zu unterstützen, die für ein sicheres und vertrauenswürdiges Open-Source-Ökosystem erforderlich sind. Im ersten Teil dieser Dissertation untersuche ich die Sicherheits- und Vertrauenspraktiken in Open-Source-Projekten in 27 Interviews mit Eigentümern, Maintainern und Beitragenden aus Open-Source-Projekten über Prozesse hinter den Kulissen, Richtlinien, den Umgang mit Zwischenfällen und Sicherheits-Herausforderungen. Die untersuchten Open-Source-Projekte besaßen hierbei eine weite Spanne an Sicherheitsmaßnahmen, Vertrauensprozesse und zugrunde liegende Motivationen. Um die Verwenderseite der Versorgungskette mit einzubeziehen, untersuche ich den Einsatz von Open-Source-Komponenten in Industrieprojekten in 25 Interviews mit Softwareentwicklern, Architekten und Ingenieuren, um die Prozesse, Entscheidungen und Überlegungen ihrer Projekte im Zusammenhang mit externem Open-Source-Code zu verstehen. Open-Source-Komponenten spielen hierbei in vielen Projekten eine wichtige Rolle, die meisten Projekte haben Richtlinien für externen Code. Auf der Endnutzer-Seite stelle ich eine Studie zur Verwendung von Obfuskation in Android-Anwendungen vor. Die Studie nutzt einen mehrgliedrigen Ansatz bestehend aus Messungen, einer Umfrage mit Entwicklern und einem Programmier-Experiment. Nur 24,92% der Apps wurden von ihren Entwicklern obfuskiert, die Entwickler befürchten dabei generell nicht den Diebstahl ihrer eigenen Apps, und haben Schwierigkeiten ihre Apps zu obfuskieren. Um auch Endnutzer selbst einzubeziehen, stell ich eine Umfrage unter 200 Nutzern von Cloud-Office-Suiten vor, welche ihre Wahrnehmungen und Erwartungen im Bezug zu Sicherheit und Datenschutz untersucht. Die Nutzer sind sich dabei den grundlegenden Sicherheits-Implikationen bewusst, es mangelt ihnen aber an technischem Wissen zum Verständnis von einige Bedrohungsmodelle.

Zu meinen wichtigsten Erkenntnissen gehört, dass Open-Source-Projekte sehr unterschiedliche Sicherheitsmaßnahmen, Vertrauensprozesse und zugrunde liegende Motivationen haben, dass ihre Sicherheits- und Vertrauensbedürfnisse wahrscheinlich am besten auf eine Art und Weise erfüllt werden, die ihre individuellen Stärken, Grenzen und das Projektstadium berücksichtigt, insbesondere bei kleineren Projekten mit begrenztem Zugang zu Ressourcen. Open-Source-Komponenten spielen ausserdem in Industrieprojekten eine wichtige Rolle, und die meisten Projekte verfügen über Richtlinien für die Verwendung von externem Code. Aber die Entwickler wünschen sich auch Ressourcen, um die einbezogenen Komponenten besser überprüfen zu können. Diese Dissertation unterstreicht die Bedeutung von Zusammenarbeit und gemeinsamer Verantwortung von Entwicklern, Maintainern, Endnutzern, Forschenden und anderen Beteiligten, welche alle dazu beitragen, dass das Open-Source-Ökosystem eine sichere und verlässliche Ressource bleibt.

Schlüsselwörter

Open Source, Software Supply Chain, Nutzbare IT-Sicherheit, Software-Entwickler

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Open Source Software plays an important role in many software ecosystems. Whether in operating systems, network stacks, or low-level system drivers, open source software is found as the foundation, glue, or tooling in many systems and processes, constituting important links of the software supply chain. This wide-spread usage is to be expected, as utilizing open source code allows stakeholders to concentrate on delivering features and achieving faster development cycles, rather than investing time into building foundational solutions in-house. The general openness and community-based development approach of the open source ecosystem also introduce unique security challenges: code submissions might come from unknown entities, open source projects often only have limited developer-hours to review pull requests or update dependencies, and including third-party code introduces obligations for stakeholders to continuously vet the included components. In this dissertation, I investigate approaches, considerations, and encountered challenges in the context of security, privacy, and trustworthiness of the open source software supply chain. For this, I conducted research involving software stakeholders such as maintainers, contributors, developers, software architects, and end users. The research presented in this dissertation empowers and supports software experts involved in the software supply, towards a more secure and healthy open source software ecosystem.

This chapter provides a general introduction to the motivation and challenges for the research described in this dissertation (Section 1.1), as well as to provide an overview of the structure and components of this dissertation and the research (Section 1.2). For a more in-depth description of, and introduction to, the individual topics of software supply chain security, opportunities and challenges in the open source ecosystem, and the research area of usable security for experts, see Chapter 2: Background.

1.1 Motivation

With the continued advancement of digital innovation, our daily lives are becoming increasingly intertwined with technology and software. While these technological advancements have undoubtedly brought about unprecedented convenience, efficiency, and connectivity, they have also resulted in a reliance on underlying software infrastructures and ecosystems. An important part of these underlying infrastructures is the software supply chain, which involves the creation, distribution, and integration of software components. A complex, connected system, the software supply chain enables stakeholders to utilize reusable abstractions and processes, supporting and speeding up the development and deployment of software products. As part of the software supply chain, reusable abstractions like libraries, frameworks, and infrastructure templates enable stakeholders to focus on the specific functionalities of their application, rather than having to write their whole software stack from scratch. Libraries and frameworks, for instance, provide pre-built code that developers can incorporate into their applications, saving time and effort. Infrastructure templates allow developers to access and utilize pre-built services and processes in their software, such as data storage, build pipelines, and authentication. For

stakeholders however, relying on third-party code also introduces security challenges, requires a certain level of trust and reliance, and introduces obligations in vetting included components. This trust requirement was probably best summarized by Ken Thompson:

> "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 [1])

Software supply chain vulnerabilities pose significant risks to stakeholders, historically being exploited by threat actors who target systems with unpatched components containing known vulnerabilities. Because of these components' reusability and connectedness, a vulnerability in, or successful compromise of, a single important component can have cascading security effects on dependent components and systems, creating ripple effects impacting individuals, communities, enterprises, and even entire industries. High-impact exploits targeting vulnerabilities like Heartbleed [2], Shellshock [3], and more recently Log4Shell [4] have highlighted weaknesses in both commercial and open source software, affecting individuals, enterprises, and governments.

Over time, attacks on the software supply chain became more targeted, evolving from simply exploiting vulnerabilities in unpatched systems to directly leveraging the connectedness of the supply chain for targeting more victims such as in the Solarwinds Orion [5] and Kaseya VSA [6] attacks, where attackers successfully compromised maintenance tools to target thousands of users' systems. Following these impactful attacks, governments worldwide recognized the high impact of attacks targeting the software supply chain responding in a number of ways: the European Commission published a draft of the Cyber Resilience Act [7], the German IT Security Act 2.0 was approved [8], and two Presidential Executive Orders were introduced in the United States to better protect critical federal infrastructure from cyberattacks [9], [10]. These acts, laws, and orders aim to improve the security of products with digital elements throughout their entire life cycle, facilitate compliance for hardware and software producers, enhance transparency of the security properties of digital products, and enable businesses and consumers to securely use digital products.

These targeted attacks have evolved, with threat actors now exploiting the software supply chain through targeting upstream dependencies, build systems, and even developers directly through their accounts and computers. The complexity of today's build systems, continuous integration and continuous delivery (CI/CD) pipelines, and the involved number of stakeholders increases the risk of misuse, misconfiguration, or leakage of secrets. This has resulted in third-party services and developers involved in building and deploying software becoming high-value targets for attackers. Recent security incidents at Codecov [11], Slack [12], Okta [13], LastPass [14], and CircleCI [15] highlight the vulnerability of these links in the software supply chain.

As one of these links, the open source ecosystem plays an important role in the software supply chain, allowing stakeholders to utilize open source components as building blocks in their software, tooling, and infrastructure. It also allows users to customize the software to meet their specific needs, which includes benefits in efficiency and cost savings, making open source components a common occurrence, even in the commercial software industry. The interactions in the open source ecosystem differ from a commercial software supply chain, working more like a community of many smaller communities, instead of the more linear, often contract-based, supply chain. But relying on the open source ecosystem brings unique challenges, both in terms of security and trust, as well as in terms of supply chain reliability and non-existing warranties or contracts.

In the past, the security research field has produced advanced technologies and approaches for improving security such as public key cryptography and end-to-end encryption. However, adoption

of such mechanisms has often been slow, and despite the development of advanced cryptographic algorithms, access control, and memory-safe applications that can offer provable strong security, vulnerabilities and attacks continue to happen. One reason for this gap between theoretical security and low actual security in practice is the lack of consideration of human factors during the development of these solutions. Security mechanisms can be difficult to use, interfere with users' priorities, or make unrealistic assumptions about users' security knowledge. Traditional approaches to security have often prioritized technical solutions that focus on functionality over usability, resulting in systems that are difficult to navigate and understand for both non-experts and even experts. While the field of usable security encompasses a wide range of stakeholders, including end users, software experts play a crucial role in developing and implementing secure software solutions. As software experts are responsible for designing, implementing, and maintaining software systems, enabling and supporting a single expert with usable secure approaches can result in security benefits for tens of their projects, hundreds of deployments, and thousands of end users using the software.

In this dissertation, I investigate approaches, considerations, and encountered challenges of stakeholders in the context of security, privacy, and trustworthiness of the open source software supply chain. Specifically, in the first research part of this dissertation, I present research concerning the security and trust practices in open source projects by interviewing 27 owners, maintainers, and contributors from a diverse set of open source projects to explore their behind-the-scenes processes, guidance and policies, incident handling, and challenges they encountered in the past. In the second part, I present research concerning the "consumer" side of the open source software supply chain, specifically the use of open source components in industry projects. For this, I interviewed 25 software developers, architects, and engineers about their projects' processes, decisions, and considerations around the usage of external open source components. In the third part, more focused on end-user facing software, I investigate the use of software obfuscation in Android applications, which is a recommended practice to protect against plagiarism and repackaging. Therefore, my study leveraged a multi-pronged approach including a large-scale measurement, a developer survey, and a programming experiment. In part four, to involve end users and their perceptions, I surveyed 200 users of cloud office suites to investigate their security and privacy perceptions and expectations. Overall, the research presented in this dissertation aims to empower and support the involved software experts, towards a more secure and reliable open source software ecosystem.

1.2 This Dissertation

At the time of writing this dissertation, I have published several peer-reviewed, high quality conference research papers, both as first author and co-author with the invaluable help of many colleagues. I am deeply grateful to my co-authors for helping me to work on these ideas. Without them, a large part of my research would not have been possible.

The research-based chapters in this thesis are either based on, or inspired by, research that was also published as peer-reviewed research publications. I thus provide a preamble before each research-based chapter, outlining my personal, as well as my co-authors' contribution to each of these research projects (Sections 4.1, 5.1, 6.1, and 7.1). In addition, for the chapters based on previous research, some sections include disclaimer boxes highlighting certain facts about the following section, e.g.,

Disclaimer: This related work section reflects the state of prior research in early 2022 and is provided to highlight the state of research during the time of the research project. For related and concurrent work at the time of this dissertation, see Chapter 3: Related and Concurrent Work.

Other sections, specifically the sections describing results, include summary boxes to highlight findings in the related area:

> Summary: Projects and Participants.The majority of our participants had worked on multiple projects in a diverse set of software areas, and in different team configurations and sizes. Only about half mentioned security-specific roles in the development loop.

The rest of this section is structured as follows: I provide the research statement of this dissertation (Section 1.2.1). I then introduce the research that chapters in this dissertation are based on, provide a short summary, and highlight my contributions, as well as list some of the research projects with my contribution that served as inspiration for some sections (Section 1.2.2). Lastly, I summarize the individual chapters of this dissertation and outline their content (Section 1.2.3).

1.2.1 Statement

The overarching goal of my research is to empower and enable software experts through usable security research, providing them with knowledge, processes, and tooling that support them in building secure and trustworthy software, resulting in potential benefits for thousands of end users. The central thesis of this dissertation is:

> "Open source software plays an important role in the software supply chain, allowing stakeholders to utilize open source components as building blocks in their software, tooling, and infrastructure. Relying on the open source ecosystem introduces unique challenges, both in terms of security and trust, as well as in terms of supply chain reliability. By identifying challenges, approaches, and considerations in the context of security, we can empower and support the involved software experts and stakeholders, towards a more secure, trustworthy, and reliable open source software ecosystem."

Based on this statement, I conducted usable security research with software experts and other stakeholders of the software supply chain. The research presented in this dissertation investigates security, trustworthiness, and perceptions in the context of the (open source) software supply chain, involving open source maintainers and contributors, industry software stakeholders, Android developers, and end users of cloud office software.

1.2.2 Contributions

Four research projects with me as team lead and lead author contributed to the research and research chapters presented in this dissertation (Chapters 4, 5, 6, and 7). As usual for collaborative research, this work would have been impossible without the significant contributions of the co-authors. For all publications, the authors are listed in order of contribution, as is customary in the security research field in general, and the usable security and privacy field in particular. The publications are listed below in the same order as their related chapters appear in this dissertation.

1. Chapter 4: Security & Trust in Open Source Software Projects of this dissertation is based on research that also lead to the publication "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects" [16].

   > **Dominik Wermke**, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects," in *43rd IEEE Symposium on Security and Privacy (IEEE S&P'22)*, San Francisco, CA, USA: IEEE, May 2022

   Short Summary: In 27 in-depth, semi-structured interviews with owners, maintainers, and contributors from a diverse set of open source projects, we explored their security measures and trust processes. 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.

   Contributions to the Project: I came up with the initial idea for this study based on my desire to conduct open source research in a more developer-inclusive and cooperative manner and further refined the idea with input from Sascha Fahl. I set up the initial concept and research approach for this research project. I lead the design of the study and interview guide and iterated it with the rest of the team. I implemented the landing page and contact templates for this study, and iterated them with the group. Noah Wöhler, Jan Klemmer, and I invited participants via GitHub and other communication channels. Together with Noah Wöhler, Jan Klemmer, Marcel Fourné, and Sascha Fahl, I conducted or supported the majority of interviews. In joint work with Noah Wöhler and Jan Klemmer, we qualitatively coded the interview transcripts. I analyzed the coded text passages and code counts. I compiled the paper for publication with minor contributions from the remaining team and we jointly discussed the work's implications. I presented the publication at IEEE S&P 2022 and included it in some of my talks.

   Recognition: The publication was awarded a Distinguished Paper Award at IEEE S&P 2022. I presented aspects of this research as part of an invited talk at USENIX ENIGMA 2023 and included it in my other invited talks and a guest lecture.

2. Chapter 5: Security Challenges of the Open Source Supply Chain of this dissertation is based on research that also lead to the publication ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain" [17].

   > **Dominik Wermke**, J. H. Klemmer, N. Wöhler, J. Schmüser, H. S. Ramulu, Y. Acar, and S. Fahl, ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain," in *44th IEEE Symposium on Security and Privacy (IEEE S&P'23)*, San Francisco, CA, USA: IEEE, May 2023

   Short Summary: In a study consisting of 25 interviews with software developers, architects, and engineers from industry software projects, we found that open source components play an important role in many projects, and that most projects have policies or best practices for including external code, but many developers desire more resources for auditing included components.

   Contributions to the Project: I came up with the initial idea for this study based on a logical follow-up to the previous paper "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects" [16]. I lead the design of the study and interview guide with

the rest of the team. I implemented the landing page and contact templates for this study, and iterated them with the group. Jan Klemmer and I invited participants via the team's professional network and from job postings on Upwork. I conducted the majority of interviews either alone or with support from the rest of the team. In joint work with Jan Klemmer, Noah Wöhler, Juliane Schmüser, and Harshini Sri Ramulu, we qualitatively coded the interview transcripts. I analyzed the coded text passages and code counts. I compiled the paper for publication with contributions from the remaining team and we jointly discussed the work's implications.

Recognition: I presented aspects of this research as part of an invited talk at USENIX ENIGMA 2023.

3. Chapter 6: Large Scale Investigation of Obfuscation Use in Android of this dissertation is based on research that also lead to the publication "A Large Scale Investigation of Obfuscation Use in Google Play" [18].

   Dominik Wermke, N. Huaman, Y. Acar, B. Reaves, P. Traynor, and S. Fahl, "A Large Scale Investigation of Obfuscation Use in Google Play," in *34th Annual Computer Security Applications Conference (ACSAC'18)*, San Juan, PR, USA: ACM, Dec. 2018, pp. 222–235

   Short Summary: For this large-scale analysis of obfuscation in Android, we analyzed 1.7 million free Android apps and found that only 24.92% of them were obfuscated. To investigate reasons, we surveyed 308 Google Play developers about their experiences and attitudes towards obfuscation and found that while developers feel that apps are at risk of plagiarism, they do not fear theft of their own apps, and many report difficulties obfuscating their own apps. In a follow-up programming experiment with 70 developers, we found that the vast majority failed to obfuscate a realistic sample app.

   Contributions to the Project: Sascha Fahl, Yasemin Acar, and Brad Reaves came up with the initial idea and iterated it with me. The full team came up with the initial concept and research approach for this research project. I implemented the analysis tooling and storage for the large-scale analysis. Nicolas Huaman and I created the tasks and example apps for the programming experiment. Yasemin Acar, Nicolas Human, and I created the survey guide for the developer survey, and iterated it with the rest of the team. I analyzed the large-scale analysis results. Together with Nicolas Huaman, I qualitatively coded the programming task solutions. I compiled the paper for publication with contributions from the remaining team and we jointly discussed the work's implications. I presented the publication at ACSAC'18 and included it in some of my talks.

4. Chapter 7: Security & Privacy Perceptions of Cloud Office Suites of this dissertation is based on research that also lead to the publication "Cloudy with a Chance of Misconceptions: Exploring Users' Perceptions and Expectations of Security and Privacy in Cloud Office Suites" [19].

   Dominik Wermke, C. Stransky, N. Huaman, N. Busch, Y. Acar, and S. Fahl, "Cloudy with a Chance of Misconceptions: Exploring Users' Perceptions and Expectations of Security and Privacy in Cloud Office Suites," in *Sixteenth Symposium on Usable Privacy and Security (SOUPS'20)*, Aug. 2020

Short Summary: We surveyed 200 office users from the U.S. and German-speaking countries about their experiences with and perceptions of cloud office tool such as Google Docs or Microsoft Office 365. We find that our participants are aware of basic general security implications, storage models, and access by others, although some of their threat models seem somewhat underdeveloped, often due to lacking technical knowledge. Our participants have strong opinions on how comfortable they are with the access of certain parties, but are somewhat unsure about who actually has access to their documents.

Contributions to the Project: I came up with the initial idea for this study based on the then-prevalent privacy issues with using U.S.-based cloud applications in German education and industry. I setup the initial concept and research approach involving U.S. and German participants for this research project. I lead the design of the study and survey guide with the rest of the team. Christian Stransky and I invited participants via Amazon's Mechanical Turk. I analyzed and visualized the survey counts together with Nicolas Huaman. In joint work with Christian Stransky, Nicolas Huaman, and Niklas Busch, we qualitatively coded the free text answers. I compiled the paper for publication with contributions from the remaining team and we jointly discussed the work's implications. I presented this publication at SOUPS'20.

In addition to these four main research projects with me as team lead and lead author, I list a number of supporting projects with my involvement below. This research is not directly part of, or included in this dissertation, but provided some general themes and ideas, especially for the background and conclusion sections. The supporting publication are ordered by time of publication, with the latest publications being first.

(a) N. Huaman, A. Krause, **Dominik Wermke**, C. Stransky, J. H. Klemmer, Y. Acar, and S. Fahl, "If You Can't Get Them to the Lab: Evaluating a Virtual Study Environment with Security Information Workers," in *Eighteenth Symposium on Usable Privacy and Security, (SOUPS'22)*, Boston, MA, USA, Aug. 2022

The research conducted for "If You Can't Get Them to the Lab: Evaluating a Virtual Study Environment with Security Information Workers" [20] inspired some of the research approaches described in the Background Section 2.3: Usable Security. This work will likely contribute to Nicolas Huaman's dissertation. My contributions to this work included creating and testing an initial prototype of *OLab* (prototype name of "Project Leine"), programming some of *OLab*'s features, as well as to contribute some texts for the publication.

Short Summary: This work tackles the challenges of conducting lab studies in usable security and privacy research, such as the difficulty of recruiting skilled participants and limited resources. We created a virtual study environment prototype called *OLab*, which enables researchers to conduct lab-like studies remotely using a commodity browser. The prototype was evaluated and found to be effective in supporting a variety of lab-like study setups and received positive feedback from participants.

(b) N. Huaman, B. von Skarczinski, **Dominik Wermke**, C. Stransky, Y. Acar, A. Dreißigacker, and S. Fahl, "A Large-Scale Interview Study on Information Security in and Attacks against Small and Medium-sized Enterprises," in *30th USENIX Security Symposium (USENIX Sec'21)*, Vancouver, B.C., Canada: USENIX Association, Aug. 2021

The research conducted for "A Large-Scale Interview Study on Information Security in and Attacks against Small and Medium-sized Enterprises" [21] emphasized some of the software supply challenges in the industry context described in the Background Section 2.1: Software Supply Chain. This work will likely contribute to Nicolas Huaman's dissertation. My contributions to this work included supporting Nicolas Huaman in the data analysis, as well as writing and revising some of the publication's texts.

Short Summary: We conducted a study of 5,000 small and medium enterprisess (SMEs) in Germany to investigate their experiences with cybercrime and information security. Our findings show that while many technical security measures and basic awareness have been implemented by most companies, there are differences in reporting cybercrime incidents based on industry sector, company size, and security awareness.

(c) C. Stransky, **Dominik Wermke**, J. Schrader, N. Huaman, Y. Acar, A. L. Fehlhaber, M. Wei, B. Ur, and S. Fahl, "On the Limited Impact of Visualizing Encryption: Perceptions of E2E Messaging Security," in *Seventeenth Symposium on Usable Privacy and Security (SOUPS'21)*, Aug. 2021

The research conducted for "On the Limited Impact of Visualizing Encryption: Perceptions of E2E Messaging Security" [22] informed some of the distinctions between experts and end users discussed in the Background Section 2.3: Usable Security. This work contributed to Christian Stransky's dissertation. My contributions to this work among others included supporting Christian Stransky in qualitative coding and data analysis, as well as writing parts of the publication.

Short Summary: Through a series of five online studies, we investigated whether making an app's end-to-end encryption more visible improves perceptions of trust, security, and privacy. We found that simple text disclosures that messages are "encrypted" are sufficient, while icons negatively impacted perceptions. User perceptions depend more on preconceived expectations and an app's reputation than visualizations of security mechanisms.

(d) Y. Acar, C. Stransky, **Dominik Wermke**, C. Weir, M. L. Mazurek, and S. Fahl, "Developers Need Support, Too: A Survey of Security Advice for Software Developers," in *IEEE Cybersecurity Development (SecDev'17)*, IEEE, Boston, MA, USA, Sep. 2017, pp. 22–26

The research conducted for "Developers Need Support, Too: A Survey of Security Advice for Software Developers" [23] informed some of the challenges developers face described in the Background Section 2.1: Software Supply Chain. This work contributed to Yasemin Acar's dissertation. My contributions to this work included supporting the team in qualitative coding and data analysis, as well as writing parts of the publication.

Short Summary: This paper has taken a first step in understanding and improving the security guidance ecosystem for developers by analyzing 19 general advice resources, identifying gaps in the current ecosystem and providing a basis for future work to evaluate existing resources and develop new ones to fill these gaps.

(e) Y. Acar, C. Stransky, **Dominik Wermke**, M. L. Mazurek, and S. Fahl, "Security Developer Studies with GitHub Users: Exploring a Convenience Sample," in *Thirteenth Symposium on Usable Privacy and Security (SOUPS'17)*, Santa Clara, CA, USA, Jul. 2017, pp. 81–95

The research conducted for "Security Developer Studies with GitHub Users: Exploring a Convenience Sample" [24] informed some of the decisions around conducting research with open source contributors, both for research described in the chapters of this dissertation, as well as approaches described in the Background Section 2.3: Usable Security. This work contributed to Yasemin Acar's dissertation. My contributions to this work included supporting Yasemin Acar with the data analysis and visualization.

Short Summary: We conducted an experiment to examine the performance of 307 active GitHub users on security-related programming tasks. While we found differences in performance based on self-reported years of experience, we did not find statistically significant differences based on the participants' status as a student, professional developer, or security background.

1.2.3 Structure

The remainder of this dissertation is structured as follows:

Chapter 1 — Introduction: This chapter is intended to give a general introduction to motivation and challenges, as well as to provide an overview of the structure and components of this dissertation and the research it is based on.

Chapter 2 — Background: This chapter aims to equip readers with the fundamental knowledge and concepts needed to comprehend this dissertation, namely, software supply chain security, the open source ecosystem, and usable security research, especially involving software experts.

Chapter 3 — Related and Concurrent Work: This chapter identifies and summarizes the most relevant research publications and studies at the time of this dissertation. It is intended to establish the context for the presented research in this dissertation, to demonstrate the significance of the underlying research questions, and to provide a foundation for understanding of the research methodology and relevance.

Chapter 4 — Security & Trust in Open Source Software Projects: This chapter investigates security and trust challenges associated with decentralized development and collaboration in open source projects. For this, I conducted 27 interviews with owners, maintainers, and contributors of various open source projects to investigate their security and trust practices, finding that projects had diverse measures and motivations. Based on these findings, I argue for supporting individual open source projects based on their strengths and limitations, particularly smaller projects with limited resources. The findings have implications for improving trust and security in the open source software ecosystem. This chapter is based on research that also resulted in the previously published work "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects" [16].

Chapter 5 — Security Challenges of the Open Source Supply Chain: This chapter investigates the associated security challenges of using open source components in software development based on 25 in-depth interviews with industry professionals. I found that open source components play an important role in many projects, but developers wish for more resources to better audit included components. The findings have implications for usage and security of open source components in industry software projects. This chapter is based on research that also resulted in the previously published work ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain" [17].

Chapter 6 — Large Scale Investigation of Obfuscation Use in Android: This chapter presents an analysis of the use of software obfuscation in Android applications using a multi-pronged study approach including a large-scale measurement, a developer survey, and a programming experiment. I found that only 24.92% of apps are obfuscated by their developer and that they do not fear theft of their own apps but have difficulties obfuscating their apps. The findings have implications for improving the security of Android apps and tools to help developers write more secure software. This chapter is based on research that also resulted in the previously published work "A Large Scale Investigation of Obfuscation Use in Google Play" [18].

Chapter 7 — Security & Privacy Perceptions of Cloud Office Suites: This chapter describes a survey with 200 users of cloud office suites to investigate their security and privacy perceptions and expectations. Findings suggest that users are generally aware of basic security implications but lack technical knowledge to envision some of the more advanced threat models. They had strong opinions on certain parties accessing their data, but were unsure who actually has access to their documents. I provide recommendations for different groups associated with cloud office suites to improve future standards, regulations, implementations, and configuration options. This chapter is based on research that also resulted the previously published work "Cloudy with a Chance of Misconceptions: Exploring Users' Perceptions and Expectations of Security and Privacy in Cloud Office Suites" [19].

Chapter 8 — Conclusion and Future Work: This chapter provides a conclusion for the research conducted for this dissertation and discusses some of the possible directions for future work that could build upon my findings.

In addition, a number of appendices for the individual research studies (Appendices A, B, C, and D), references and glossaries are provided in the later chapters of this dissertation.

1.2.4 Typesetting and Typography

This dissertation was compiled and typeset utilizing John Collins' latexmk (version 4.79) [25], LuaTex (version 1.16.0) [26], and Markus Kohm's KOMA-Script scrbook class (version 3.38) [27]. References are included via biblatex, using the default ieee style, slightly modified to include titles as they appear in the bib source. In terms of font choices, this thesis utilizes a serif font, EB Garamonds with old style numerals, for the body text. A serif font, Inter, for headings and subheadings. And a mono font, Source Code Pro at 0.76 scale, for code inserts and certain tool names.

Chapter 2

Background

Human factors play an important role in securing the software supply chain, especially for the community-focused open source ecosystem. This chapter provides a background on the fundamental concepts mentioned in the subsequent chapters of this dissertation. The chapter is divided into three main sections providing background on the software supply chain, the open source ecosystem, and the area of usable security research.

Section 2.1, Software Supply Chain, discusses the various aspects of securing the software supply chain. This section covers vulnerabilities (Section 2.1.1), metrics and frameworks (Section 2.1.2), and attacks directly targeting the supply chain (Section 2.1.3). Section 2.2, Open Source Software, focuses on the open source ecosystem, including the unique challenges and opportunities related to using and securing open source software. This section discusses how the open source model works and the potential security risks associated with it, the role of dependencies (Section 2.2.1), and both challenges and opportunities unique to the open source software supply chain (Section 2.2.2). Section 2.3, Usable Security, introduces the concept of usable security and discusses its importance in securing software. This section introduces common research approaches and populations (Section 2.3.1), as well as the subfield of usable security for software experts (Section 2.3.2), relevant for the research presented in this dissertation. Finally, Section 2.4, Summary, provides a summary of the key points covered in this chapter.

By understanding the concepts and background information presented in this chapter, readers will be better equipped to understand the subsequent chapters of this dissertation, which delve deeper into research involving topics such as securing the software supply chain, security challenges in open source software, and usable security research with experts.

2.1 Software Supply Chain

The emerging software supply chain concept is inspired by the supply chain system from logistics: in logistics, supply chain refer to the network of organizations, people, activities, information, and resources involved in the creation and delivery of a product or service to customers or end users. The supply chains encompass processes from the sourcing of raw materials to the delivery of finished goods to end users, and includes all the logistical and operational processes in-between.

As an supply chain example adjacent to computer science, consider the supply chain involved in the production and delivery of a laptop computer: the chain begins with the sourcing of raw materials, such as metals, plastics, and electronic components, from suppliers around the world. These materials are then transported to manufacturing facilities, where they are either first refined, or directly processed into individual components like wires, transistors, or plastic brackets. The individual components are combined into assemblies, e.g., key caps, chip boards, or cables. These key caps, chip boards, and cables are then assembled into functional components like a keyboard, hard drives, or screens. The individual functional components are combined according to a specification and branded to form

a fully functional laptop. The assembled laptops themselves are then packaged with accessories and shipped to warehouses, where they are stored until they are ordered by retailers or individual customers. Finally, the laptops are shipped to their destination, where they are unpacked, sold, and used by their new owners. Throughout this process, a complex web of logistics, communication, and coordination is required to ensure that the laptop is produced and delivered efficiently and effectively. Each step in the supply chain depends on the successful completion and availability of the previous step, and any disruptions or delays can have a significant impact on the overall process.

Analogous to a supply chain in logistics, the software supply chain concept covers the network of components, tools, and processes by which software is composed, developed, tested, and finally deployed to customers or end users. The software supply chain's processes can be divided into several stages, including planning, design, development, testing, deployment, and maintenance. Similarities to the supply chain system in logistics involve the coordination of multiple suppliers and stakeholders, establishing contracts and quotas, as well as the management of (virtual) assets like code and data. Some key differences include that the software supply chain is typically less reliant on physical materials and more focused on digital assets, such as code and data. This results in transportation and storage of goods being less of a concern, with a focus more on ensuring the integrity and security of the digital assets.

Unlike physical goods, software products are often developed iteratively, with multiple teams working on different parts of the product at the same time. This often results in stakeholders having to collaborate and coordinate between different teams over long time-spans, as well as the need to conduct continuous testing and quality assurance processes to ensure that the resulting software is stable and secure in every deployment cycle and release. With many different teams and stages being involved, the interactions between stakeholders and their individual types and specialities can be quite complex. The cross-sector working group of the Enduring Security Framework (ESF) under the Critical Infrastructure Partnership Advisory Council (CIPAC) released a number of "Recommended Practices" for securing the software supply chain, targeting software suppliers [28], developers [29], and customers [30]. Analogues to the 3 types, but less oriented towards a federal government agency and with a more general focus, this dissertation considers the major stakeholders of the software supply chain to fit into the following categories (the open source ecosystem being an exception, see Section 2.2 and Chapter 8):

Customers are responsible for defining the requirements for the software product they need, evaluating various software products to determine the best fit for their needs, acquiring and deploying the software product, and maintaining it throughout its life cycle. The customer's responsibilities also include monitoring the software's performance, identifying issues, and applying updates and patches as necessary to ensure the product remains functional and secure.

Developers are involved in designing, building, and releasing software products that meet the needs of customers while maintaining the security and integrity of the codebase. Developers are also responsible for identifying and remediating any security vulnerabilities that are identified during the development process.

Suppliers act as an intermediary between developers and customers. Their responsibilities include maintaining the integrity of delivered software products, as well as ensuring that customers receive high-quality software that meets their needs. For this, they provide software packages and updates to ensure that they are safe, stable, and effective. Suppliers should accept reports of issues or newly discovered vulnerabilities from customers and notify developers for remediation.

Figure 2.1: Logos for impactful software vulnerabilities. From left to right: Heartbleed [32], Shellshock (also known as Bashdoor) [33], ProxyLogon (part of the Microsoft Exchange Attack) [34], and the (ironically bad) logo for the Log4Shell vulnerability [35].

To proactively identify and address potential vulnerabilities that may impact the security of the software supply chain, stakeholders need to maintain clear communication channels, implement strong security measures, and conduct regular risk assessments to identify potential vulnerabilities. By taking a shared responsibility approach towards security, the different types of stakeholders can work together to ensure the integrity and reliability of the software supply chain for everyone involved.

2.1.1 Vulnerabilities

Compared to a physical product supply chain, the software supply chain faces unique cybersecurity challenges: due to the resulting software products being typically distributed and deployed digitally, they often present a large attack surface to potential cyberattacks, a fact that attackers are keenly aware of. According to Sonatype's 2021 "State of the Software Supply Chain" report, the number of recorded supply chain attacks in the past year had increased by over 650% to 12,000 [31].

One of the benefits of participating in the software supply chain are reusable abstractions. Reusable abstractions refer to software components such as libraries, frameworks, or infrastructure that can be used to build more complex applications quickly and efficiently. These reusable abstractions enable developers to focus on the specific functionalities of their application, rather than having to write their whole software stack from scratch. Libraries and frameworks, for instance, provide pre-built code that developers can incorporate into their applications, saving time and effort. And existing external infrastructure like data storage, data serving, or authentication frameworks allows developers to access and include pre-built services with additional features and benefits in their software.

Trusting external code from reusable abstractions can come with security risks. Vulnerabilities in third-party code can be exploited by attackers to gain access to sensitive information or compromise systems and vulnerabilities in one component can have cascading effects on other components and systems, creating a ripple effect that can impact individuals, organizations, and entire industries.

According to the ESF working group, common methods of compromise used against software supply chains include exploitation of software design flaws, incorporation of vulnerable third-party components into a software product, infiltration of the supplier's network with malicious code prior to the final software product being delivered, and injection of malicious software that is then deployed by the customer [29]. In addition to these attack methods, the software supply chain with its wide-spread reuse of software components is especially vulnerable for zero day vulnerabilities ("zero days"). The term refers to the fact that there is zero time between the discovery of the vulnerability and the first attack, leaving the affected software or system at risk until a patch or fix is developed and implemented. This type of vulnerability can be exploited by attackers to gain unauthorized access to all systems with the component, steal sensitive information, or carry out other malicious activities. An overview of impactful vulnerabilities and past attacks leveraging zero days as examples:

Heartbleed (CVE-2014-0160 [2]) was a security vulnerability in the OpenSSL cryptography library publicly disclosed in April 2014 [32]. The vulnerability allowed attackers to exploit a missing bound check in the input validation of a keep-alive feature in OpenSSL, affecting both instances running as Transport Layer Security (TLS) server or client, potentially leaking passwords and encryption keys stored in the memory. A patch was released on the same day the vulnerability was announced. The Heartbleed vulnerability was considered one of the most significant security incidents in recent history due to its widespread impact and the sensitive data that could have been compromised.

Shellshock (CVE-2014-6271 [3] and other related vulnerabilities), also known as Bashdoor, was a vulnerability in the Bash shell, a widely used command-line interface in Unix-based systems, and was discovered in September 2014 [36]. It allowed attackers to remotely execute arbitrary code on vulnerable systems, giving them unauthorized access to sensitive data and control over affected systems. The Shellshock vulnerability was particularly severe due to the widespread use of Bash in web servers, routers, and other network devices, making it a potential target for cybercriminals seeking to gain access to critical infrastructure. The vulnerability also affected many Internet of Things (IoT) devices, including cameras, routers, and other connected devices, making it difficult to patch all the affected systems. The discovery of Shellshock led to a worldwide effort to patch the vulnerability, with many organizations and security experts working to identify and mitigate the risks posed by the exploit. It also raised concerns about the security of open-source software and the need for more rigorous code reviews and security audits to prevent similar vulnerabilities from being introduced in the future.

The **Microsoft Exchange Server attack** involving the ProxyLogon vulnerability [34] (CVE-2021-26855 [37]) and others, was a global cyberattack wave targeting Microsoft Exchange Server, a widely used email and calendaring software. The attack began in January 2021 and was detected by Microsoft in early March. The multiple attackers, believed to include a Chinese state-sponsored hacking group known as Hafnium, exploited 4 zero-day vulnerabilities in Microsoft Exchange Server to gain access to email accounts, steal sensitive data, and install malware [38]. The attack affected tens of thousands of organizations globally, including government agencies, businesses, and non-governmental organizations (NGOs).

Log4Shell (CVE-2021-44228 [4] and other related) was a vulnerability found in the Log4j Java logging library, which is widely used in many applications and services [39]. The vulnerability allows remote attackers to execute arbitrary code on a server running a vulnerable version of the library. The vulnerability was discovered in early December 2021, and the details were publicly disclosed on December 9th. Within hours of the disclosure, attackers began exploiting the vulnerability to gain unauthorized access to servers around the world. The Log4Shell incident was considered a critical cyberattack, and the impact was widespread, affecting thousands of organizations and businesses globally. The vulnerability was rated 10 out of 10 on the Common Vulnerability Scoring System (CVSS), the highest possible score. The incident led to an urgent global response to patch the vulnerability, with many organizations implementing emergency measures to protect their systems. It also sparked discussions around the importance of secure coding practices, vulnerability management, and the need for better cybersecurity strategies to prevent similar incidents in the future.

Exploits targeting vulnerabilities like Heartbleed, Shellshock, and Log4Shell have highlighted weaknesses in relying on external code, both in commercial and open source software, affecting individuals, enterprises, and governments worldwide.

2.1.2 Metrics and Frameworks

In the software supply chain, stakeholders address security vulnerabilities that are specific to their area of responsibility. However, some security challenges or vulnerabilities may require a collaborative approach, involving multiple stakeholders, dependencies, and software deployments. In the context of a secure software supply chain, metrics, and frameworks that classify security vulnerabilities (CVE [40], CVSS [41], EPSS [42], VEX [43]), attacks and weaknesses (CWE [44]), or coding practices (Open Source Security Foundation (OpenSSF) Scorecard [45]) play an important role in communicating between stakeholders. These metrics and frameworks provide a standard way to measure and evaluate the security of software systems, helping stakeholders to identify and mitigate security risks.

CVE, CVSS, and EPSS help assess the severity or exploitability of security vulnerabilities: Common Vulnerabilities and Exposures (CVE) is a standardized convention for identifying and tracking publicly disclosed vulnerabilities [40]. It provides a unique identifier for each vulnerability and allows security researchers, product vendors, and users to share information about vulnerabilities across different platforms and organizations. Common Vulnerability Scoring System (CVSS) is a framework for rating the severity of security vulnerabilities based on their impact on confidentiality, integrity, and availability of a system [41]. The CVSS score is a numerical value ranging from 0 to 10, with higher values indicating greater severity. The score is calculated based on several factors, including the attack vector, the attack complexity, the impact on the system, and the availability of mitigations. CVSS allows organizations to prioritize which vulnerabilities to patch first, as vulnerabilities with higher CVSS scores are considered more severe and should be addressed with greater urgency. Exploit Prediction Scoring System (EPSS) is used to predict the likelihood of a vulnerability being exploited by attackers [42]. The scoring consists of a numerical score between 0 and 1 for each vulnerability, estimating the probability that a vulnerability will be exploited in the next 30 days. The scoring is based on a number of factors such as the severity of the vulnerability, the complexity of the attack needed to exploit it, and the potential impact of a successful exploit. The Vulnerability-Exploitability Exchange (VEX) is used by to exchange information about vulnerabilities and their exploitability [43]. A VEX document includes a Product Tree, which lists all the products referenced in the Common Vulnerability Reporting Framework (CSRF) document. The VEX framework also includes an Affected Status field, which describes whether a product is affected by the vulnerability, regardless of whether the vulnerability is present or not. The status can be one of fixed, known affected, known not affected, and under investigation In addition to the product information, the VEX framework requires at least one vulnerability identifier, such as a CVE or an ID, to be associated with each vulnerability.

Analogous to vulnerabilities, there are a number of frameworks to track and communicate attacks and weaknesses. Common Weakness Enumeration (CWE) tracks weaknesses in software and hardware, providing a description of the weakness, examples of how it can be exploited, and mitigations that can be taken to prevent or reduce the risk of exploitation [44]. Common Attack Pattern Enumeration and Classification (CAPEC) catalogues common attack patterns that can be used to identify, describe, and classify various types of attacks [46]. The CAPEC taxonomy includes a hierarchical structure of attack patterns, with each pattern having a unique identifier and a detailed description of its characteristics and potential impact. MITRE ATT&CK is a framework for understanding and categorizing adversary behaviors during cyberattacks [47]. The framework consists of a matrix of tactics and techniques that adversaries utilize to achieve their objectives, as well as a knowledgebase of detailed descriptions and examples of real-world attacks.

In supply chain logistics, a bill of materials (BOM) provides a list of materials, assemblies, and components required to manufacture a final product. Analogous for the software supply chain, a Software Bill of Materials (SBOM) is a comprehensive inventory of all the components, dependencies, and other

third-party software used in building a particular software application or system [48]. It provides a detailed list of all the software and hardware components that make up a product or system, including their version numbers, licenses, and any known vulnerabilities. A SBOM is useful for inventory, vulnerability, and license management by suppliers, developers, and consumers in the software supply chain (SSC) [49]. In May 2021, the United States "Executive Order on Improving the Nation's Cybersecurity" mandated that the National Institute of Standards and Technology (NIST) issue guidance within 90 days to enhance the security of the software supply chain, including providing a SBOM for each product [10]. However, the diversity of software development and use across different organizations makes it challenging to develop a one-size-fits-all approach to providing transparency for software assurance. To address these challenges, the National Telecommunications and Information Administration (NTIA) published the minimum elements for an SBOM, consisting of three broad categories: data fields, automation support, and practices and processes [49]. The data fields category includes baseline information about each software component, such as component name, version number, and licensing information. The automation support category focuses on the ability to generate SBOMs in both machine and human-readable formats. The several data format specifications for generating and consuming SBOMs include The Linux Foundation Projects' Software Package Data eXchange (SPDX) [50], OWASP's CycloneDX [51], and NIST's Software Identification (SWID) tags [52]. The practices and processes category includes a set of guidelines and procedures for generating and maintaining the SBOM throughout the software development life cycle, covering aspects such as frequency, depth, known unknowns, and distribution and delivery, access control, and accommodation of mistakes. The NTIA notes that these minimum elements are only the initial steps, and that SBOMs are an emerging technology and practice [49]. They also highlight that modern software applications provided as a service present unique challenges for the SBOM format, with risk management responsibilities being on the side of service providers due to lack of control by the user.

2.1.3 Targeted Attacks

In recent times, attacks on the software supply chain became more targeted, evolving from simply exploiting vulnerabilities in unpatched systems to directly leveraging the connectedness of the supply chain for targeting more victims. The European Union's European Union Agency for Cybersecurity (ENISA) published a report on "Threat Landscape for Supply Chain Attacks" in July 2021, finding that of 24 supply chain attacks between January 2020 to July 2021, around 62% of the attacks on customers took advantage of trust in their supplier and 66% of the incidents attackers focused on the suppliers' code to target further victims [53]. Both the SolarWinds Orion attack and the Kaseya VSA ransomware attack highlight the approach of these targeted attacks:

The **SolarWinds** Orion attack was a major supply chain attack in late 2020, affecting a wide range of organizations, including government agencies and major corporations [54]. SolarWinds is a software company providing system management tools for network and infrastructure monitoring. The attackers targeted the performance monitoring system SolarWinds Orion, which had privileged access to IT systems to obtain log and system performance data. The attackers exploited a backdoor in a component of the Orion software framework, named SUNBURST by FireEye (CVE-2020-10148 [5]), allowing the attackers to inserted malicious code into software updates for the component. After a dormant period of up to two weeks, SUNBURST retrieved and executed commands, including the ability to transfer files, execute files, profile the system, reboot the machine, and disable system services [55]. In addition to the SUNBURST malware, a

second attack, later dubbed SUPERNOVA, was discovered during investigations into the SolarWinds incident [56]. On December 19, 2020, Microsoft announced that it had found evidence of this attempted supply chain attack, which was distinct from the SUNBURST attack that inserted malware into Orion binaries [57].

The **Kaseya VSA** ransomware attack occurred in July 2021, targeting the customers of Kaseya, a provider of IT management software used to manage clients' systems remotely [58]. The attackers exploited a vulnerability in Kaseya's VSA (Virtual System Administrator) software to distribute ransomware to its customers' systems (CVE-2021-30116 [6]). The vulnerability was originally identified in April 2021 by the Dutch Institute for Vulnerability Disclosure (DIVD) [59]. They worked with Kaseya experts to fix four of the seven vulnerabilities, but the remaining issues could not be resolved in time before the attack. The attack affected between 800 and 1,500 businesses globally, mostly small to medium-sized businesses and managed service providers. The attackers demanded a ransom payment of $70 million in Bitcoin to release keys for the encrypted data. On July 23, 2021, Kaseya announced that it had received a universal decryptor tool from an unnamed "trusted third party" and was working to restore the victims' files [60].

These high-impact attacks, particularly the SolarWinds Orion incident, did not go unnoticed by governments worldwide, and a number of acts, laws, and orders were introduced in response: The European Commission published a draft of the Cyber Resilience Act (CRA) on September 15, 2022, setting out the specific objectives of improving the security of products with digital elements throughout their entire life cycle, facilitating compliance for hardware and software producers by creating a coherent cybersecurity framework, enhancing transparency of the security properties of digital products, and enabling businesses and consumers to securely use digital products [7]. The German "Second act on increasing the security of IT systems (German IT Security Act 2.0)" was approved in May 2021, allowing the Federal Office for Information Security (German: Bundesamt für Sicherheit in der Informationstechnik (BSI)), to set standards and requirements for federal authorities, mobile network operators, and companies of special public interest, as well as allowing the BSI to act as a advisory body for consumer protection related to IT security [8]. In the United States, two Presidential Executive Orders were introduced to better protect critical U.S. federal infrastructure from cyberattacks: "Executive Order on America's Supply Chains" (EO14017 [9], with a focus on the general supply chain, but specific considerations for software products) and "Executive Order on Improving the Nation's Cybersecurity" (EO14028 [10]), establishing new requirements to secure the federal government's software supply chain, such as systematic reviews, process improvements, and security standards for software suppliers, developers, and customers that acquire the software for the Federal Government.

More recently, attackers began to directly exploit the software supply chain by targeting upstream dependencies and build systems to inject malicious code into downstream software. Today's build systems and CI/CD pipelines can interact and chain with other systems and third-party services, allowing for the creation of complex, multi-step build and distribution processes for software. This complexity also increases the risk of misuse, misconfiguration, or leakage of secrets, and as software is increasingly being built and deployed using third-party services, these services are becoming high-value targets for attackers seeking to infect all customers and compromise the software supply chain. This vulnerability of the software supply chain was further highlighted by a large number of recent security incidents, targeting Codecov, Slack, Okta, LastPass, and CircleCI:

The **Codecov** security incident was publicly disclosed on April 15, 2021 by Codecov, a company that provides code coverage and software testing tools [11]. An attacker leveraged an error in Codecov's public Docker image setup, allowing them to extract a Google Cloud Storage account

key from an intermediate image layer. With account access, the attacker modified Codecov's Bash Uploader in the Google Cloud Storage with malicious changes [61]. Each time a developer downloaded the Codecov testing script, the malicious software would begin running on the customer organization's test machines, allowing the attackers to exfiltrate credentials and other sensitive data stored in the victim's continuous integration environments [62].

The **Slack** security incident was published on the messenger service's blog on December 31, 2022 [12]. Slack was notified of suspicious activity on their GitHub account and discovered that a limited number of Slack employees' tokens were stolen and misused to gain access and download externally hosted GitHub repositories on December 27. Slack claimed that none of the downloaded repositories contained customer data, means to access customer data, or Slack's primary codebase.

The **Okta** security incident. On January 20, 2022, the identity and access management company Okta was notified that a new password factor was added to a Sitel customer support engineer's Okta account [13]. At the time, Okta assumed that the individual's attempt to access the account was unsuccessful, they reset the account, and informed Sitel, who engaged a forensic firm to investigate. Based on this investigation, Okta actually concluded that a small percentage of customers, approximately 2.5%, may have had their data viewed or acted upon [63].

The **LastPass** password manager experienced two connected security incidents in 2022 [14]. The first incident, which occurred in August, was due to a keylogger malware infection on software engineer's corporate laptop, which provided the hacker with access to the company's cloud-based development environment. According to LastPass, no customer data or vault data was taken during this incident, but information stolen in the first incident was used to identify targets and initiate a second attack. The second incident was disclosed on December 22, 2022, and it was caused by a vulnerability in third-party software that was exploited by an attacker. The attacker targeted a senior DevOps engineer and used the vulnerability to deliver malware, bypass existing controls, and gain unauthorized access to cloud backups. The stolen data included system configuration data, API secrets, third-party integration secrets, and encrypted and unencrypted LastPass customer data.

The **CircleCI** security incident was disclosed by the CI/CD platform on January 4, 2023 [15]. An attacker used malware on a CircleCI engineer's laptop to gain unauthorized access to a subset of CircleCI's production systems. The malware allowed the attacker to steal a valid two-factor, authentication-backed single sign-on (SSO) session and execute a session cookie theft, enabling the attacker to impersonate the targeted employee in a remote location and escalate access to a subset of CircleCI's production systems. The targeted employee had privileges to generate production access tokens, which allowed the unauthorized third party to access and exfiltrate data from a subset of databases and stores, including customer environment variables, tokens, and keys.

In addition to highlighting the overall vulnerability of the software supply chain, these incidents especially revealed the large potential attack surface presented by third-party services and employees involved in building and deploying software.

To summarize, the software supply chain encompasses the processes of software composition, development, testing, deployment, and maintenance. Collaboration and coordination are necessary due to the iterative development process with multiple teams. High-impact attacks, such as the SolarWinds

Orion incident, have targeted the software supply chain, leading to the introduction of laws and orders. Attackers now target upstream dependencies and build systems to inject malicious code into downstream software. Third-party services are becoming high-value targets for attackers seeking to compromise the software supply chain, as seen in recent security incidents at Codecov, Slack, Okta, LastPass, and CircleCI.

In conclusion, the software supply chain is a complex and interconnected network that is essential for software development and deployment. However, recent attacks targeting the supply chain have highlighted its vulnerability to malicious actors. As software continues to play an increasingly critical role in our lives, it is essential to prioritize research and innovation in this area to address the evolving security challenges and ensure the integrity and trustworthiness of the software supply chain. Only by understanding the risks and implementing effective measures can we build a resilient and secure software ecosystem for the future.

2.2 Open Source Software

Open source software is a type of computer software that allows users to use, modify, and distribute the source code of the software freely. Open source code plays a major role in the software supply chain, with a 2023 report finding that of 1,700 codebases in 17 industries, 96% of codebases contained open source, and 76% of total code was open source code [64].

The general concept of open source software is known under many different, sometimes diverging, terms: free software, libre software, Free and Open Source Software (FOSS or F/OSS), or Free/Libre and Open Source Software (FLOSS). The term "free software" predates "open source software" but can sometimes be misunderstood to mean software that is free of charge, whereas the intended meaning is freedom of use. The free software community is divided into two political camps: the free software movement and the open source movement [65]. The free software movement advocates for computer users' freedom and sees non-free programs as an injustice to users, while the open source movement focuses on practical benefits rather than the issue of justice for users. To remain neutral between the two political camps, some use the combined term "FLOSS," which stands for "Free/Libre and Open Source Software" and includes both free and libre [65]. This dissertation uses the more general term "open source software (OSS)" and related concepts throughout all chapters, both to highlight the more general focus on the entire ecosystem in this dissertation, as well as the community-spanning, practical utilization of open source components.

While the legal and philosophical interpretation of the open source concept is highly relevant for the open source ecosystem itself and for companies or other supply chain stakeholders that want to utilize open source components, this dissertation, and the research described in it, focuses more on the practical aspect of open source software, namely that its code is available for public use, modification, and distribution. Thus, the underlying definition for open software used in this dissertation follows the one provided in the recent United States S.4913 bill introduced to the senate:

> "Open source software means software for which the human-readable source code is made available to the public for use, study, re-use, modification, enhancement, and re-distribution." (S.4913: Securing Open Source Software Act [66])

The open source ecosystem plays an important role in the software supply chain, allowing stakeholders to utilize open source components as building blocks in their software, tooling, and infrastructure. Whether in operating systems, network stacks, or low-level system drivers, open source software is found as foundation, glue, or tooling in many systems and processes, constituting important links of

the software supply chain. It also allows users to customize the software to meet their specific needs, which includes benefits in efficiency and cost savings during software development. This made open source software ubiquitous across the software industry, with no area considered too critical to exclude it. In RedHat's 2022 "The State of Enterprise Open Source" they interviewed 1296 IT leaders world wide, finding that 82% are more likely to select a vendor who contributes to the open source community and that 89% of IT leaders believe enterprise open source is as secure or more secure than proprietary software [67].

Governments have taken notice of this special position of open source software: the "Securing Open Source Software Act" (S.4913 [66]) was introduced to the senate in September 2022, and is placed on the Senate Legislative Calender as of December 2022. The bill sets forth the duties of the Cybersecurity and Infrastructure Security Agency (CISA) regarding open source software security, including performing outreach and engagement, supporting federal efforts, coordinating with non-federal entities, serving as a point of contact, and encouraging efforts to bolster open source software security. CISA is also required to publish a framework for assessing the risk of open source software components and update it annually [66].

Unlike contract relations in a (software) supply chain, the open source ecosystem consists of less linear, more connected interactions, with stakeholders commonly acting in multiple roles. For example, a developer can contribute some patches to an open source project, while being one of the users of the project, or even using some of the components as maintainer in one of their own projects.

Maintainers in open source projects play a critical role in ensuring that the project is healthy and sustainable over time. The exact responsibilities can differ from project to project, but generally they are responsible for managing the community of contributors, reviewing code contributions, resolving issues and bugs, and making decisions about the direction of the project.

Contributors are individuals who contribute to open source projects by submitting code, documentation, bug reports, and other contributions. They play an important role in the success of open source projects by helping to improve the quality of the software, fixing bugs, adding new features, and enhancing the user experience.

Users in the open source ecosystem utilize and provide feedback on the software. They may also contribute to the project by providing feedback, suggesting new features, or reporting bugs, potentially moving into the role of a contributor depending on their involvement with the project.

Even aside from the differing stakeholder roles, the open source ecosystem functions quite differently compared to a traditional software supply chain, working more like a community of many smaller communities, instead of the more linear, often contract-based, supply chain. Unlike commercial software development, the open source community is made up of individuals who often have never met or spoken to one another, but share a common goal of working towards creating or improving a software project. Contributors often have different ideas of what the project should do and what their individual motivation behind their commitment to the project is. Many open source projects start informally and without any legal entity, but some later grow into formal organizations or join umbrella organizations. For potential contributors, Joining or contributing to an open source project is often quite straightforward and often requires no formal process.

2.2.1 Dependencies

The ability to rely on external dependencies as building blocks for software is likely one of the most impactful advantages of participating in the software supply chain. Acting as part of the software supply chain, the open source ecosystem include many ways for developers to leverage external dependencies as building blocks for their software, e.g., on code platforms like GitHub and GitLab or from package repositories like npm or PyPI.

But as recent incidents described in the previous sections have highlighted, relying on external dependencies also introduces new attack surfaces for each new component. A number of recent attacks on open source package infrastructures highlight the vulnerability and the impact of an attacker hijacking popular dependencies. For this, the attackers try to leverage several of potential attack surfaces:

Developer Accounts, as in an incident that occurred in June 2020, involving the user-agent parsing library ua-parser [68]. The incident was caused by a malicious actor who gained access to the npm publishing account of one of the library's maintainers and published a version of the library that contained a backdoor. This backdoor was designed to steal the npm tokens of developers who installed the library, which would allow the attacker to gain access to their other projects and resources. The backdoor was discovered and reported by a security researcher, and the affected version of the library was quickly removed from npm. The maintainer of the library also revoked all the compromised npm tokens and urged all users of the library to update to a safe version as soon as possible.

Malicious Packages waiting to be downloaded by potential victims, as in the noblesse malware family that involved several malicious PyPI packages discovered by the JFrog security team in July 2021m, which were estimated to be downloaded about 30,000 times [69]. The malicious packages used basic obfuscation techniques to avoid detection and included a number of different payloads: some payloads targeted Discord authentication tokens, likely to impersonate the user on the communication platform. Others attempted to steal browser autocomplete data like passwords and credit cards, or collected various system informations from a victim's computer.

Typo-squatting or dependency confusion attacks, which flood package registries with malicious packages named very similar to already existing, popular packages in an attempt to trick victims. As happened in August 2022 with more than 200 cryptominer packages flooding npm and PyPI [70] and again in February 2023 with thousands of PyPI packages containing a Windows trojan [71].

These incidents highlighted the potential security risks of relying on third-party libraries, especially those that are widely used and have multiple contributors. They also served as a reminder of the importance of implementing good security practices, such as using two-factor authentication and regularly monitoring for unusual activity on package registry accounts. In response to these incidents, Python's PyPI and GitHub have began to require two-factor authentication (2FA) for developer accounts with critical projects. While such approaches increase the overall security of package repositories and dependencies, they might also negatively affect usability. E.g., if 2FA is required, but the authentication process is too complicated or time-consuming, developers may try to find ways to bypass it, which would undermine the intended security benefits. By examining the common challenges and usage patterns involved in using and providing external dependencies, researchers can identify ways to improve the adoption of security processes, ultimately enhancing the security of the software supply chain.

2.2.2 Unique Challenges and Opportunities

The benefits of open source software include collaboration and knowledge sharing among developers, which can lead to faster development cycles and more innovative solutions. It also allows users to customize the software to meet their specific needs, which can result in increased efficiency and cost savings for users. In addition, open source software is often free (as in money) or significantly cheaper to run than proprietary software, making it accessible to a wider range of teams and users. Open source software also presents unique challenges connected to the social and community focus of the ecosystems, as well as in terms of supply chain reliability and non-existing warranty or contracts, as a number of recent incidents highlighted:

colors and **faker**, two popular npm libraries with 20 million and 2.8 million weekly downloads respectively, started to print corrupted text and run in a loop in January 2022 [72], [73]. Major open source projects like Amazon's Cloud Development Kit, Facebook's Jest, and the Node.js Open CLI Framework were impacted by this and expressed concerns regarding a potential hijack [74]. The changes appear to be intentionally introduced by the developer, who had previously expressed frustration with the lack of compensation by major companies for maintaining these widely-used libraries [74], [75].

The **node-ipc** package turned malicious in March 2022 (CVE-2022-23812 [76]), when its npm maintainer intentionally added malware targeting Russian and Belarusian IP addresses [77]. Depending on the version, the malicious dependency would overwrite user files with heart emojis or just create files with an anti-war message.

Aside from these obvious incidents, some open source software is so ubiquitous in the development and operation of IT systems that its existence is hardly noticed. Its absence, or even just unfixed bugs, could wreak large costs and other damages for big organizations. This is specifically true for many open source projects, which are often done by default as hobbies, not contributing significantly to the income of their main developers.

Being heavily community-based, there are a number of efforts of improving security in the open source ecosystem. The Open Source Security Foundation, also known as OpenSSF, is a collaborative effort between leading companies in the technology industry to improve the security of open-source software [78]. The foundation was founded with the goal of bringing together industry leaders, developers, and open source foundations to promote secure coding practices and identify and address vulnerabilities in open source software.

2.3 Usable Security

Usable security is an interdisciplinary research field that combines human-factor concepts with the technology and development of secure systems, with the goal of creating software, tools, interfaces, and workflows that are both secure and accessible to users of all levels of technical expertise. Security research has resulted in advanced technologies and approaches such as public key cryptography and end-to-end encryption. However, adoption of such mechanisms has often been slow, and despite the development of advanced cryptographic algorithms, access control, and memory-safe applications that can offer provably strong security, cyberattacks continue to happen. One reason for this gap between theoretical security and low actual security in practice is the lack of consideration of human factors during the development of these solutions. Security mechanisms can be difficult to use, interfere with users' priorities, or make unrealistic assumptions about users' security knowledge. Traditional approaches to

security have often prioritized technical solutions that prioritize functionality over usability, resulting in systems that are difficult to navigate and understand for non-experts. A theme also reflect in the following quote by Bruce Schneier:

> "In the past, computer security research has focussed on technical defences to safeguard systems. But it has become clear that technical measures are not enough: People are the weakest link in the security chain." (Bruce Schneier in "Secrets and Lies: Digital Security in a Networked World" [79])

The field of usable security grew out of concerns about the usability of security systems. In the early days of computing, security was typically implemented through complex passwords and other technical measures that were difficult for users to understand and use.

Zurko and Simon's 1996 paper "User-Centered Security" proposed an agenda for creating user-friendly security systems [80]. They argued that users would not use or buy security products they did not understand, and suggested conducting usability testing, developing security models, and considering user needs during system development. They also proposed formal usability testing of security mechanisms to ensure both usability and security, departing from the traditional assumption that security mechanisms were understandable without testing.

Following this, two influential papers for the usable security research were published in 1999: Adams and Sasse's widely cited "Users are not the enemy" addresses the issue of blaming users for compromising system security [81]. The authors found that users often pick weak passwords and do not change them unless forced to do so. Technical measures to enforce stronger security practices were found to be ineffective, as users found ways to bypass them. Users were also challenged to comply with conflicting password requirements and had confused beliefs about password strength and security threats. The paper argues that by explaining the rationale behind security measures to users, they can become the first line of defense against security threats. And Whitten and Tygar's "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0." conducted a usability study usability of an email encryption tool for PGP [82]. The study subjects were given a scenario that required using and understanding the encryption software, but none were able to complete the task. This validated the widespread perception that even easy-to-use encryption programs were too complex for the general public. Referencing this seminal paper's title, a number of papers from unrelated authors conducting usability studies with encrypted messages use titles with "Johnny," e.g., "Johnny 2: a User Test of Key Continuity Management with S/MIME and Outlook Express" [83], "Why Johnny still can't encrypt: evaluating the usability of email encryption software" [84], or "Helping Johnny 2.0 to encrypt his Facebook conversations" [85].

In 2005, Cranor and Garfinkel co-authored the book *Security and Usability: Designing Secure Systems That People Can Use*, which was one of the first comprehensive works to address the topic of how to make security mechanisms more usable [86]. And in 2014, Garfinkel and Lipford released "Usable Security: History, Themes, and Challenges," one of the first structured overviews covering the field of usable security [87].

In the following years, the field of usable security continued to grow and mature, with researchers investigating topics such as phishing, password management, and privacy. Today, usable security is an established area of research with its own conferences, journals, and professional organizations. As technology continues to evolve, the need for usable security will only continue to grow, making usuable security an important area of research and practice.

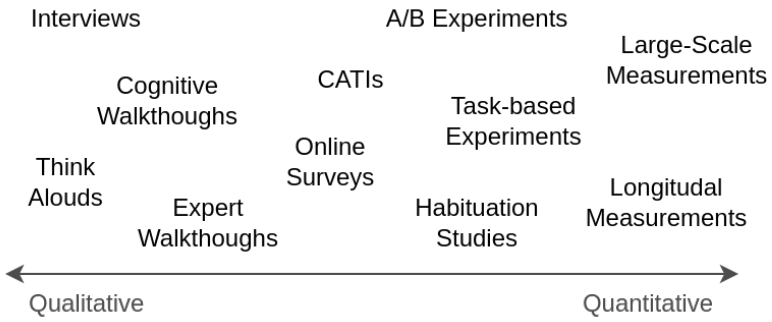


Figure 2.2: Range of research approaches I have used during my usable security research. Ranging from quantitative approaches to more qualitative methods.

2.3.1 Approaches and Populations

The field of usable security is a multidisciplinary domain that covers a broad range of research approaches and diverse populations to gather data and insights on how to create secure and user-friendly computer systems and applications.

Empirical approaches for gathering data can range from quantitative measurements to more qualitative methods like interviews. For an exemplary range of methods in usable security research, see the range of methods I have utilized in my research in Figure 2.2. One common approach to data collection in usable security research is through large-scale measurement studies. These studies often use quantitative methods to analyze data from large sample sizes, providing a broad understanding of user behaviors and attitudes towards security. Another approach is task-based experiments, where participants are given programming tasks to solve in controlled environments. These experiments allow researchers to gather detailed data on user behavior, performance, and decision-making processes. A/B experiments and online surveys are also popular empirical methods in usable security research. A/B experiments involve randomly assigning participants to different experimental conditions and comparing the results to identify which design or feature is most effective. Online surveys are used to gather self-reported data from participants, providing insights into their attitudes, perceptions, and behaviors related to security. In addition to quantitative methods, qualitative approaches are also valuable in usable security research. Semi-structured interviews are often used to collect detailed information from experts or users with specialized knowledge in a particular area of security. Process walkthroughs involve observing and interviewing participants as they perform security-related tasks, providing valuable insights into user behaviors and needs. Overall, usable security research involves a wide range of empirical methods and techniques that can be tailored to the specific needs of the research question and population being studied.

Because the software ecosystem involves complex and interconnected network of stakeholders, usable security research encompasses a wide range of populations. For an exemplary range of populations in usable security research, see the range of populations I have studied in my research in Figure 2.3. One of the primary user-focused populations studied in this field are the end users of specific apps or software. These users play a critical role in determining the success of software products, and their behaviors and preferences can greatly impact the security and usability of these products. In addition to end users, usable security research also encompasses more specialized roles like application developers and Data Protection Officers (DPOs). Application developers are responsible for creating and maintaining software products, and their decisions can have a significant impact on the security and usability of

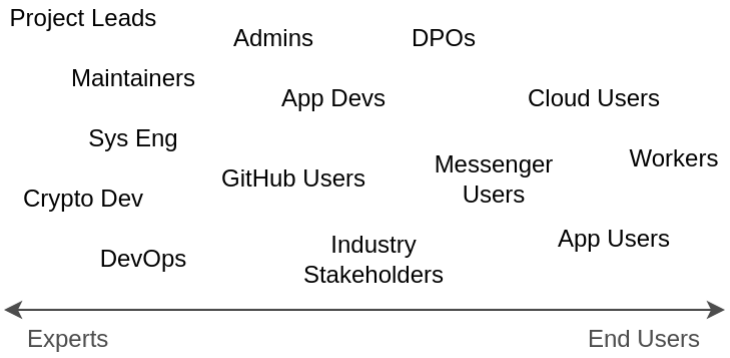


Figure 2.3: Range of populations in the software ecosystem I have worked with during my usable security research, ranging from experts to end users.

those products. DPOs are responsible for ensuring that organizations comply with data protection regulations, and their expertise in this area is crucial for developing secure and privacy-preserving software. Furthermore, usable security research also extends to extremely specialized and niche populations like developers of cryptographic libraries. These developers are responsible for creating the fundamental building blocks of secure software, and their work plays a critical role in protecting sensitive information and ensuring the privacy of users. Beyond these specific populations, usable security research also involves understanding the broader social and cultural contexts in which software is used. For example, researchers may examine the impact of cultural norms on user behavior, or study the role of social influence on the adoption of security measures. Overall, the multidisciplinary nature of usable security research requires considering many different populations and their specific needs and behaviors.

To summarize, by applying a wide range of approaches for studying the software ecosystem and its various stakeholders, researchers can gain valuable insights into how to create secure and user-friendly software products that meet the needs of a diverse range of users.

2.3.2 Usable Security for Software Experts

The IT security field involves various stakeholders, including end users and software experts. Among these stakeholders, software experts have a crucial role to play in developing and implementing secure software solutions. Since they are responsible for designing, implementing, and maintaining the software systems that support various aspects of modern life, their expertise is essential to ensure that these systems are both secure and usable.

The field of usable security recognizes that traditional security approaches have often prioritized technical solutions over usability, resulting in systems that are difficult to implement and comprehend. However, software experts are uniquely positioned to bridge this gap by integrating human-centered design principles into the development of secure software systems. Similar to end users, developers may struggle with security and privacy. Rather than blaming them, the goal of usable security research for software experts is to support and empower them to build secure, trustworthy, and privacy-respecting software, benefiting not only them, but also all users of their software. As the software ecosystem is vast, this idea extends to all types of software experts, such as developers, administrators, and maintainers. The importance of considering the experts was also highlighted in this quote by Kevin Mitnick:

> "Companies spend millions of dollars on firewalls and secure access devices, and it's money

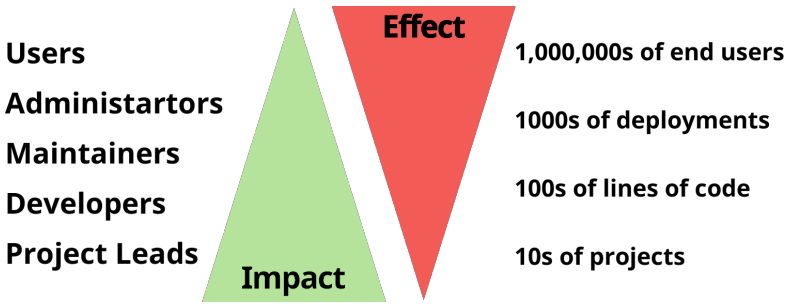


Figure 2.4: Supporting software experts with usable security approaches can have a significant impact on the security of software applications. By enabeling software experts to create secure, usable software, any users benefit as well, resulting in improved security for a larger number of software projects and deployments.

> wasted because none of these measures address the weakest link in the security chain: The people who use, administer and operate computer systems." (Kevin Mitnick [88])

Being an expert does not necessarily make one knowledgeable about security, as recent software vulnerabilities have demonstrated. For example, the Log4Shell vulnerability has affected many services and the recent LastPass and CircleCI breaches were traced back to leaked data from employees. In addition, recent incidents of fake packages in the Python ecosystem have preyed on careless developers who inadvertently install them. It is essential to recognize that experts are not necessarily security experts and may require support and education to minimize the risk of becoming the weakest link in the software supply chain. Experts are not only not security experts, they might end up as the weakest link in the software supply chain.

As with many aspects of security, the human factors involved in the design, implementation, and use of supply chain security measures might not have received the attention they deserve. Recent high-profile attacks on the software supply chain have highlighted the role of individual developers as the weakest link in the chain, and simply securing dependencies and build systems with technological approaches is not sufficient to prevent such attacks. The main challenge is that developers are all different, with various projects, setups, and attack surfaces that need to be protected. Each individual developer involved in the software supply chain serves as a potential attack surface, which can have a significant impact on the overall security of the ecosystem.

Experts, including project leads, developers, and maintainers, have a significant impact on the software ecosystem. By supporting these experts, we can bring security and privacy improvements to many individual users of their software. For example, empowering an expert to include a specific security feature in their code could potentially impact tens of their projects, hundreds of lines of code they write, which could then be potentially deployed thousands of times, resulting in software that might ultimately be used by millions of end users.

Although tech-only approaches, such as firewalls, password managers, or dev and analysis tools, can provide some level of security, they might not be able to solve security problems alone. Ultimately, these tools depend on security-conscious experts for their effectiveness. Therefore, it is critical to recognize that security is not only a technical problem but also a human problem that requires support and education for security-conscious experts. By empowering and educating these experts, usable security research can enable more secure software systems that are resilient to attacks, more trustworthy, and better protect their users' data privacy.

2.4 Summary

The software supply chain is a process that encompasses the development, testing, deployment, and maintenance of software using various components, tools, and processes. Reusable abstractions, such as libraries, frameworks, and cloud infrastructure, allow stakeholders to quickly build and develop complex applications without reinventing the wheel. However, relying on third-party code and services can lead to security risks, which can have cascading effects on systems and organizations.

Open source software is a critical aspect of the software supply chain but introduces unique challenges in terms of security, trust, and supply chain reliability. Open source software provides benefits such as collaboration, knowledge sharing, and customization that can result in faster development and increased efficiency for software development. Some open source software is even so embedded in IT systems that unfixed bugs or the software being no longer supported could lead to significant damages for large organizations.

Supply chain security is a critical research concern, including the human factors involved in design, implementation, and use of security measures. While technical measures are essential, developers and other software experts might actually be the weakest security link in the software supply chain. Supporting these experts can bring security and privacy improvements to many individual users of their software, making it essential to recognize that security is not only a technical problem but also a human problem that requires support for security-conscious experts. Empowering and educating these experts can lead to the building of a secure and reliable open source software ecosystem, resilient to attacks, trustworthy, and protecting of end users' data and privacy.

Chapter 3

Related and Concurrent Work

This chapter identifies and summarizes the most relevant research publications and studies at the time of this dissertation. It is intended to establish the context for the presented research in this dissertation, to demonstrate the significance of the underlying research questions, and to provide a foundation for understanding of the research methodology and relevance.

To allow for a better reading experience, my research and research with my contribution is listed inline with, and in the same style of, general related work, but specifically highlighted in text by "In work with my contribution [...]", "As part of my research [...]", or similar.

3.1 Software Supply Chain Security

Supply-chain risk, attacks, and vulnerabilities have been systematized [89]–[91] and analyzed to inform the development of protective measures [92], to improve the accuracy of vulnerability alerts and security bug reports [93], [94], and to better understand the factors that influence dependency vulnerability remediation in software projects [95].

Research investigating security aspects with developers, architects, and engineers working on industry projects provide important insights into the security of the overall software supply chain. Past research investigated the security impact of different aspects such as decision-making [96], [97], organizational changes [98], [99], and information sources [100], [101]. Stevens *et al.* conducted a multi-stage study with 25 industry employees investigating aspects of threat modeling [102]. Assal and Chiasson surveyed 123 software developers about software security processes, finding that the real issues frequently stem from a lack of organizational or process support [103].

CI/CD and build systems play an important role of integrating and combining software supply chain components in software projects. As such, they are a high-value target for an attacker looking to stealthily injecting malicious code. Past research into the security of CI/CD and build systems include hardening [104], as well as infrastructure as code [105], [106]. Koishybayev *et al.* analyzed 447,238 workflows spanning 213,854 GitHub repositories finding that 99.8% of workflows are overprivileged and have read-write access (instead of readonly) to the repository [107]. In 2023, Gu *et al.* systematically studied potential security threats in continuous integration (CI) workflows with multiple stakeholder, conducting a large-scale measurement with over 1.69 million repositories and revealing four novel attack vectors [108].

Dependencies are a popular (security) software research topic, as they can hide critical attack vectors and attack surfaces. Kula *et al.* conducted an empirical study on library migration with over 4,600 GitHub software projects and 2,700 library dependencies, finding that although many of these systems rely heavily on dependencies, 81.5% still keep their outdated dependencies and 69% of surveyed developers claimed to be unaware of their vulnerable dependencies [109]. Xu *et al.* surveyed 49 developers from GitHub and F-Droid to analyze reasons why developers replace own code with a library, i. e. re-use, or

re-implement a library's functionality [110]. Larios Vargas *et al.* identified 26 technical, human, and economic factors that developers consider in their dependency selection processes based on 16 interviews and a survey with 115 developers [111]. Dependency ecosystems are a common data source for measurement studies in this field, e. g., for package repositories like JavaScript's npm [112]–[118], Python's PyPI [119], Ruby's gem [120], R's CRAN [121], and the wider software ecosystems like for Apache [122], Gentoo [123], Java [124], [125], or Android [126]. In 2023, Gu *et al.* conducted a measurement study spanning one year over six registries and seventeen popular mirrors, covering over 4 million packages, finding that multiple threats exist in every ecosystem, and some have been exploited by attackers [127]. The propagation of vulnerabilities within the npm ecosystem has been studied with the help of dependency trees [128] and dependency graphs [117]. Ferreira *et al.* proposed a lightweight permission system that protects Node.js applications by enforcing package permissions at runtime [129]. As part of the npm ecosystem, many webpages deliver third-party Javascript, which brings challenges as a compromised script will be delivered to all website visitors. Jueckstock and Kapravelos presented VisibleV8, a dynamic analysis framework hosted inside V8, the JS engine of the Chrome browser, which allows for isolating and identifying namespace artifacts used by JS code in the wild to detect automated browsing platforms [130] An approach to separate these third-party scripts from the privileged browser execution model is sandboxing [131], [132]. The inclusion of third-party dependencies and the associated technical challenges have been studied and compared across a variety of software ecosystems [133]–[136]. Reducing the number of included dependencies, and with that the potential attack surface, is a common approach. E.g., by Koishybayev and Kapravelos, who presented Mininode, a static analysis tool (SAT) for Node.js applications that measures and removes unused code and dependencies [137].

A common form of third-party code stems from libraries and other APIs, allowing developers to speed up their development process by (re)utilizing established implementations of software behavior. In 2010, Mileva *et al.* mined and evaluated API popularity and trends for 200 Java projects [138]. The authors demonstrate that it is possible to give adoption recommendations based on past usage trends. Similarly, libraries that are already included in a project can also be used for further library recommendations, as Nguyen *et al.* demonstrated with the library recommendation system *CrossRec* [139]. Zapata *et al.* inspected 60 npm projects for three cases of high severity vulnerabilities, finding evidence that up to 73.3% of the projects depending on outdated dependencies were actually safe from the threat because they did not call vulnerable code [140]. As part of the library ecosystem, cryptographic APIs are especially relevant for ensuring a secure software supply chain. Nadi *et al.* analyzed 100 StackOverflow posts, 100 GitHub repositories, and survey input from 48 developers to find challenges and obstacles faced by developers using Java cryptography APIs, finding that while developers find it difficult to use certain cryptographic algorithms correctly, they feel confident in selecting the right cryptography concepts [141]. Acar *et al.* investigated the usability of Python cryptographic libraries in a controlled experiment with 256 developers, finding participants struggling with both basic functional correctness and security. López de la Mora and Nadi proposed a metric-based approach for informed adoption decision when selecting and comparing libraries [143]. With my contribution, Gorski *et al.* conducted a controlled online experiment with 53 participants, in which we study the effectiveness of cryptographic API-integrated security advice, finding that integrated advice improves code security [144]. Based on this research, Gorski *et al.* went on to conduct a participatory design study with 25 software developers in focus groups [145]. In 2022, Jancar *et al.* conducted a survey with 44 developers of 27 open source cryptographic libraries investigating if and how the developers ensure that their code executes in constant time. They found that developers are aware of timing attacks and of their potentially dramatic consequences and yet often prioritize other issues [146].

Software obfuscation has been studied as defense against reverse engineering [147], to prevent intellectual property attacks [148], as disguise for malware [149], and to avoid user profiling [150]. Code ob-

fuscation techniques allowed to successfully avoid detection tools such as anti-malware software [151]–[153], repackaging detection algorithms [154], and app analysis tools [155]. On the detection side, obfuscation-resilient detection of libraries in Android apps has been advanced for in-depth analyses of apps with a focus on malicious third-party libraries, malware detection, and repackaging [156]–[159]. Both Fahl *et al.* and Oltrogge *et al.* conducted developer surveys and interviews, revealing deficits in the handling of TLS/SSL and suggesting several improvements [160]–[162]. More on the user side, privacy policies of Android apps have been evaluated in a number of works [163]–[166]. Balebako *et al.* performed interviews and online surveys to investigate how app developers make decisions about privacy and security, identifying several hurdles and suggesting improvements that would help user-privacy [167], [168]. Jain and Lindqvist suggested design changes to the Android Location API based on the results of a developer lab study [169].

In conclusion, the software supply chain presents a complex ecosystem of risks, attacks, and vulnerabilities that require continuous attention to ensure the security and integrity of software systems. Researchers have systematically analyzed these risks and vulnerabilities to inform the development of protective measures. Dependency ecosystems are a common data source for measurement studies in this field and security of the overall software supply chain is greatly impacted by the practices and decisions made by developers, architects, and engineers working on projects. The research approaches described in this dissertation provide additional and enhanced insights into the real-world experiences of software teams, complementing and extending the results of previous, more measurement-focused studies.

3.2 Security and Trust in the Open Source Ecosystem

The open source community faces unique security and trust challenges compared to other ecosystems, making them a valuable subject for research, including reviews and systematizations: Crowston *et al.* reviewed the empirical research on Free/Libre and open source software (OSS) development and assess the state of the literature [170]. Wen provided a literature review of software security in open source development [171]. Ohm *et al.* analysed 174 malicious software packages that were used in real-world attacks on open source software supply chains, finding that 56% of packages trigger their malicious behavior on installation and 61% leveraged typo squatting [172]. In 2023, Ladisa *et al.* presented a taxonomy of attacks on open source supply chains validated by user surveys with 17 domain experts and 134 developers [173].

In past case studies, researches closely studied development in early open source projects such as Linux [174], Mozilla [175], and FreeBSD [176]. Antikainen *et al.* surveyed 95 respondents of the Linux kernel community about factors influencing their trust, finding significance for (developer) skills, reputation, and established practices [177]. Deligiannis *et al.* analyzed 16 drivers from the Linux 4.0 kernel [178]. Bai *et al.* proposed and evaluated a static analysis approach, finding 640 use-after-free bugs in Linux driver code [179].

Trust is an important factor in public software collaboration. Bugiel *et al.* presented an approach to asses the trustworthiness of software based on their security history [180]. Syeed *et al.* investigate the different aspects of measuring trust in OSS communities, providing further avenues to develop trust-based measurement tools [181]. Murdoch and Leaver discussed the UK government's Cyber-Security Information Sharing Partnership (CiSP), an online collaboration environment for security information [182]. Sinha *et al.* empirically studied in an automated approach the induction of external developers as code committers in Eclipse projects. They find that developers establish trust and credibility in a project by contributing to the project in a non-committer role or, as lesser factor, by their

employing organization [183].

Another important factor is maintaining code quality with many involved contributors and software life cycles. Groven *et al.* applied first and second generation software quality assessment models to the case of Asterisk, a FLOSS implementation of a telephone private branch exchange [184]. Bosu and Carver developed an approach to divide OSS developers into core and periphery groups based on centrality measures. They then compared the outcome of the code review process for members of the two groups, finding that the core developers receive quicker first feedback on their review request, complete the review process in shorter time, and are more likely to have their code changes accepted into the project codebase [185]. Ryoo *et al.* studied the extent of discrepancy between an architect's vision of what security tactics need to be adopted in the software and the actual implementation [186]. Thompson and Wagner investigated 3,126 projects to study the relationship between code review coverage and participation and software quality and security, finding a significant effect [187]. Moldon *et al.* examined how the behavior of software developers changes in response to removing gamification elements from GitHub [188].

The openness of the ecosystem has lead to open source repositories being established data source in the (security and privacy) research community. This is mirrored by the large number of available datasets, e.g., of commits [189], [190], contributors [191], vulnerabilities, and accessible via torrents: both Gousios and Spinellis and Gousios *et al.* provided torrents of datasets as an alternate access method that have been used for research in a number of papers [192], [193]. Alali *et al.* characterized what a typical commit looks like, finding that 75% of commits are quite small with respect to number of files, number of lines, and number of hunks committed [194]. But the openness of open source repositories also introduces challenges, e.g., when secrets like API keys or credentials are involved. Meli *et al.* conducted a six-month scan of real-time public GitHub commits and a public snapshot covering 13% of open source repositories, finding that secret leakage is pervasive with over 100,000 repositories, and that thousands of new, unique secrets are leaked every day [195]. With my involvement, Krause *et al.* surveyed 109 developers and conducted 14 in-depth interviews with developers, finding that 30.3% have encountered secret leakage in the past [196]. Feng *et al.* presented PassFinder, an automated approach based on deep neural networks for detecting password leakage from public repositories [197]. Basak *et al.* investigated Internet artifacts, such as blog articles and question and answer posts, identifying 24 practices of secret management for developers [198].

Due to the amount of freely available code, open source repositories are a common source for vulnerability research. In 2004, Bosu *et al.* analyzed peer code review data of the Android Open Source Project (AOSP) to understand whether code changes that introduce security vulnerabilities occur at certain intervals [199]. In 2008, Hattori and Lanza studied commits in 9 large open source projects [200]. Altinkemer *et al.* collected software vulnerability data for open source and proprietary operating system software and analyzed if significant differences exist for multiple metrics [201]. Anbalagan and Vouk classified 43,710 vulnerabilities from the Open Source National Vulnerability Database, Bugzilla, and Fedora [202]. Edwards and Chen examined historical releases of Sendmail, Postfix, Apache httpd, and OpenSSL with static source code analysis and the entry-rate in the CVE dictionary for a release [203]. Shahzad *et al.* explored a large software vulnerability dataset disclosed since 1988 till 2011 [204]. Tan *et al.* investigated software bug characteristics by sampling 2,060 real world bugs in three large, representative open source projects, finding that while software evolves, semantic bugs increase, while memory-related bugs decrease [205]. Bosu *et al.* analyzed 267,046 code review requests from 10 open source projects and identified 413 vulnerable code changes [199]. Pletea *et al.* performed sentiment analysis on commits and pull requests from 90 GitHub projects, finding more negative emotions for security-related discussions compared to others [207]. Abunadi and Alenezi present an empirical study clarifying how useful cross project prediction techniques are in predicting software vulnerabil-

ities [208]. Alenezi and Javed tested several open source web applications against common security vulnerabilities [209]. Zampetti *et al.* studied the usage of SATs for Java code in 20 GitHub repositories [210]. Santos *et al.* investigated vulnerabilities associated with security tactics in Chromium, PHP, and Thunderbird [211]. Gkortzis *et al.* presented a dataset of reported vulnerabilities of 8694 open source project versions [212]. Zahedi *et al.* empirically identified security issues posted on 200 GitHub repositories [213]. In 2020, Walden investigated how the Heartbleed vulnerability changed the software evolution of OpenSSL [214]. Householder *et al.* analyzed vulnerabilities with CVE IDs, finding that 4.1% of IDs have public exploit code associated with them within 365 days [215].

Identifying and patching or fixing vulnerable dependencies is an important task of open source maintainers and users. Plate *et al.* proposed an approach to facilitate the impact assessment for vulnerabilities in OSS libraries [216]. Antal *et al.* investigated commits of Python and JavaScript projects, finding that neither community reacts fast to appearing security vulnerabilities in general [217].

Automated approaches and generated metrics are a common approach to better tackle the complexity of open source dependency chains and repository maintenance. Perl *et al.* trained a SVM classifier to flag suspicious commits based on a large-scale mapping of CVEs to GitHub commits [218]. Zhou and Sharma described an automatic vulnerability identification system based on commit messages and bug reports in open source projects [219]. Imtiaz *et al.* analyzed five open source projects that have been using the SAT Coverity, finding that severity and fix complexity may correlate with an alert's lifespan in some of the projects [220]. Hogan *et al.* presented an automated method for labeling vulnerability-contributing commits (VCCs) [221]. In 2022, Zahan *et al.* worked with OpenSSF Scorecards in two preprints, both for investigating security features in npm and PyPI, as well as their impact on security outcomes, highlighting some impactful features [222], [223].

On the side of vulnerability fixes, Śliwerski *et al.* analyzed CVS archives for fix-inducing changes, finding distinct patterns with respect to their size and the day of week they were applied [224]. Li and Paxson investigated fixes for vulnerabilities in 682 open source projects, finding that a third of all security issues were introduced more than 3 years prior to remediation [225]. Piantadosi *et al.* linked 337 CVE entries to the corresponding patches, finding that developers who fix software vulnerabilities are more experienced than the average [226]. Ramsauer *et al.* present a data-mining based approach to detect fixes for vulnerabilities that bypass the standard public process, finding 29 commits that address 12 vulnerabilities [227].

Because of the open source ecosystem's reliance on communities and communication, related aspects like interactions with newcomers, first submissions, or donations are a common subject of research. Among the social aspects found in open source projects, researchers investigated toxic comments [228] and metadata [229], [230] as well as programming languages [231] and their general maintenance [232], [233]. Wen studied knowledge sharing and learning about secure programming knowledge in projects by a socio-technical approach [234]. Hannebauer and Gruhn surveyed newcomers to Mozilla and GNOME, finding that most newcomers modify a component because they need the modification for themselves [235]. Steinmacher *et al.* conducted semi-structured interviews with 36 developers from 14 different projects, identifying social barriers faced by first-time contributors [236]. Pinto *et al.* conducted two surveys aimed at understanding what motivates casual contributors, finding that although casual contributors are rather common they are responsible for only 1.73% of the total number of commits [237]. Steinmacher *et al.* proposed and evaluated a portal created to support newcomers to OSS projects [238]. Canfora *et al.* proposed an approach aimed at identifying and recommending mentors in software projects by mining data from mailing lists and versioning systems [239]. Steinmacher *et al.* provide guidelines for newcomers to open source projects based on previous studies [240]. Dominic *et al.* proposed a conversational bot that would recommend projects to newcomers and assist in the onboarding to the open source community [241]. Subramanian *et al.* investigated first pull requests

by contributors, finding a mixture of trivial and non-trivial changes [242]. Balali *et al.* interviewed mentors of 10 well-established OSS projects and qualitatively analyzed their answers to identify both challenges and strategies related to recommending tasks for newcomers [243]. Overney *et al.* conducted a mixed-method study to investigate donations in open source, finding d 25,885 projects asking for donations on GitHub, often to support engineering activities. However, they find no clear evidence that donations influence the activity level of a project [244].

In conclusion, the open source ecosystem has become an essential resource for the security and privacy research community due to the openness and accessibility of its repositories. The large number of available datasets has enabled researchers to conduct studies that were not possible before. While open source repositories have become an established data source, the focus of this dissertation is more on exploring the less visible aspects of the ecosystem, such as trust processes, contributor hierarchy, and security considerations. By examining these aspects, we can gain a better understanding of how open source projects operate and how they can be made more secure and trustworthy.

3.3 Interview Studies in a Security Context

Interview studies are a well-established qualitative research approach for in-depth evaluations in the (security and privacy) research community.

Interviews allow researchers access to data that is not readily available from technical systems: thoughts and procedures. Johnson *et al.* conducted interviews with 20 developers to investigate why developers are not widely using SATs and how current tools could potentially be improved, finding that although all participants felt that use is beneficial, false positives and warning presentation are barriers to use [245]. Combined with other approaches in larger studies, interviews can provide additional in-depth insights: Bai *et al.* asked 52 participants to complete encryption tasks using both a traditional key-exchange model and a key-directory-based registration model [246]. Gallagher *et al.* conducted 17 semi-structured interviews, finding that experts and non-experts view, understand, and use Tor in notably different ways. [247].

Past research also utilized interviews to gain insights into the work and tools of experts such as security professionals, app developers [248], administrators, and security analysts. Specifically, as part of a larger study in 2004, Barrett *et al.* conducted 12 interviews with sysadmins, managers, team leads, and others in various roles about their issues and concerns, challenges in their work [249]. Botta *et al.* interviewed 12 security management professionals, finding that the job of IT security management is distributed across multiple employees, often affiliated with different organizational units or groups within a unit and responsible for different aspects of it [250]. Bauer *et al.* conducted a series of interviews with thirteen administrators that manage access-control policy. Based on these interviews, they identified three sets of real-world requirements that are either ignored or inadequately addressed by technology [251]. Silic and Back conducted 14 semi-structured interviews with information security professionals and programmers, finding that the professionals generally trust OSS [252]. Bridges *et al.* interviewed 13 security analysts about host data, how tools are used, and how tools are evaluated [253]. Haney *et al.* conducted 21 interviews in organizations including cryptography in products, finding an uniquely strong security mindset in those companies [254]. With my contribution,Huaman *et al.* conducted 5,000 computer-assisted telephone interviews with small and medium enterprises in Germany, finding that security awareness has arrived in all companies [21]. These experts also include expert communities that rely on security, such as journalists, editors, and victim service providers: McGregor *et al.* investigated computer security practices of 15 journalists in the U.S. and France in semi-structured interviews, finding that existing security tools fail not only due to usability issues but when they ac-

tively interfere with other aspects of the journalistic process [255]. As part of a larger study, McGregor *et al.* interviewed five of the personnel with significant editorial or technical input on the systems used during the Panama Papers project [256]. Chen *et al.* conducted 17 semi-structured interviews with staff members at victim service providers and survivors of trafficking, investigating the role technology plays in their interactions as well as related computer security and privacy concerns and mitigations [257].

Past research of open source ecosystem leveraged interviews to gain additional insights to mindsets and opinions, often in combination with conducted measurements. 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 [236]. Also as part of a larger study, Balali *et al.* interviewed mentors of 10 OSS projects, qualitatively analyzing their answers to identify both challenges and strategies related to recommending tasks for newcomers [243]. Zhou *et al.* investigated and classified 15,306 hard forks on GitHub and conducted interviews with 18 owners of hard forks, finding that hard forks often evolve out of social forks rather than being planned deliberately [258]. Bogart *et al.* conducted interviews and a survey combined with measurements to investigate how breaking change decisions are made in 18 open source ecosystems, finding shared values like stability and compatibility, as well as differences in other values between the projects [259]. In recent work from 2021 and 2022 respectively, both Jansen *et al.* and Ghofrani *et al.* conducted smaller-scale interview studies with industry developers investigating the trust aspect of external software [260], [261]. Butler *et al.* interviewed company experts to identify the value of reproducible builds for businesses [262]. Also in 2022, Gutfleisch *et al.* interviewed developers about usability considerations in their secure software development processes, identifying a high impact of contextual factors [263]. As part of my research and foundation for chapters in this thesis, Wermke *et al.* investigated both trust and security practices of open source projects in 27 interviews [16] and considerations and experiences around open source components (OSCs) in 25 interviews with industry project [17]. In 2023 with my contribution, Fourné *et al.* interviewed 24 participants from the reproducible-builds.org project, finding that self-effective work by highly motivated developers and collaborative communication with upstream projects are key contributors [264].

In conclusion, the use of in-depth interviews as a research method has been well-established in the security and privacy research community, and has been effectively used in prior studies to gather detailed information from experts. Some of the research approaches described in this dissertation employed in-depth interviews as a means to gain deep insights into the perceptions, behaviors, and reasoning of study participants. By allowing software experts to share their perceptions and experiences in their own words, interviews can facilitate a more personal and engaging conversation that can help build trust and rapport between researchers and participants. This can be particularly important in the context of security and privacy research, where developers may be hesitant to disclose sensitive information or express their true thoughts and feelings about certain issues. Overall, the use of in-depth interviews as a research method can provide valuable insights that are difficult to obtain through other means, and can play a critical role in advancing our nuanced understanding of the issues at hand, which can inform the development of effective solutions and strategies to enhance security and privacy in a variety of contexts.

Chapter 4

Security & Trust in Open Source Software Projects

Open Source Software is an important link in the software supply chain. But the openness and community-based development approach of the open source ecosystem introduce unique security challenges: code submissions might come from unknown entities and projects often only have limited developer-hours to review pull requests or update dependencies. This chapter presents research investigating security and trust practices in open source projects, including the unique challenges of decentralized development and open collaboration in these projects.

As this research project was conducted as a team consisting of me, Noah Wöhler, Jan Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl, this chapter utilizes the academic "we" to mirror this fact. We conducted 27 in-depth interviews with owners, maintainers, and contributors of diverse open source projects to investigate their security and trust practices, including guidance and policies, incident handling, and challenges encountered, finding that the projects have highly diverse security measures and trust processes, as well as underlying motivations. Based on our findings, we argue for supporting open source projects in ways that consider their individual strengths and limitations, especially for smaller projects with limited access to resources. This research has implications for the open source software ecosystem and how the research community can better support open source projects in trust and security considerations.

4.1 Preamble

This chapter is based on research that was also published as "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects" [16], which appeared and was presented by me at the top-tier security conference 43rd IEEE Symposium on Security and Privacy (IEEE S&P 2022, "Oakland") in May 2022. The publication was awarded one of four Distinguished Paper Awards (out of 147 papers) at IEEE S&P 2022.

Dominik Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects," in *43rd IEEE Symposium on Security and Privacy (IEEE S&P'22)*, San Francisco, CA, USA: IEEE, May 2022

The original abstract for the publication is as follows:

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.

The publication included the following acknowledgements:

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.

In addition, I would like to express my sincere gratitude to everyone who has contributed to the completion of this project. I would also like to thank the participants again, who generously gave their time and shared their impressions and experiences around their open source projects with us. Their willingness to participate made this research possible, and I am deeply grateful for their contributions.

4.1.1 Contribution

The research presented in this chapter was conducted as a team consisting of me as a team lead, Noah Wöhler, Jan Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. I am grateful for the contributions of each member, which have been integral to the success of this research project. Without their expertise, hard work, and dedication, this research project would not have been possible.

I came up with the initial idea for this study based on my desire to conduct open source research in a more developer-inclusive and cooperative manner and further refined the idea with input from Sascha Fahl. I set up the initial concept and research approach for this research project. I lead the design of the study and interview guide and iterated it with the rest of the team. I implemented the landing page and contact templates for this study, and iterated them with the group. Noah Wöhler, Jan Klemmer, and I invited participants via GitHub and other communication channels. Together with Noah Wöhler, Jan Klemmer, Marcel Fourné, and Sascha Fahl, I conducted or supported the majority of interviews. In joint work with Noah Wöhler and Jan Klemmer, we qualitatively coded the interview transcripts. I analyzed the coded text passages and code counts. I compiled the paper for publication with minor contributions from the remaining team and we jointly discussed the work's implications. I presented the publication at IEEE S&P 2022 and included it in some of my talks.

4.1.2 Structure

The remainder of this chapter is structured as follows: After a general introduction (Section 4.2), I provide the related work at the time of this project in 2022, covering the areas of repository research, interviews in a security context, and open source security and trust (Section 4.3). I then describe our interview approach (Section 4.4) and highlight our findings (Section 4.5). Finally, I discuss our findings (Section 4.6) and provide a summary for this chapter (Section 4.7).

4.2 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 [265]. In recent years, collaborative version control platforms such as GitHub [266] and GitLab [267] introduced a wide field of developers to open source projects (OSPs). As the complexity of modern software development increased, so did the number of dependencies and involved contributors. Decentralized development and open collaboration of OSPs 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 [265]. 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 [268]. An attacker gained access to the maintainer's account and released three manipulated versions executing a Monero cryptocurrency miner and password-stealing trojans [269]. 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 [270]. While GitHub stated with "high confidence" that the vulnerability had not been exploited maliciously, telemetry data was only available from September 2020 onwards [271]. Analogous to a 2020 report from The Linux Foundation [272], we consider the software supply chain in this research 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 [273]. By introducing open source components, companies inherit the same challenges and attack surfaces as OSPs. 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 [274]. In this chapter, 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 [1])

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 chapter, we aim to shed light on security and trust practices in OSPs — 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 OSPs structured behind-the-scenes?"* Due to their community-driven nature, OSPs 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 OSPs?"* Often changing contributors and loose team structures lead to challenges in distributing project-internal knowledge in OSPs. 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 OSPs approach security and trust challenges?"* OSPs 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 OSPs employ to establish trust between contributors and how they react or plan to react to arising security and trust challenges.

4.3 Related Work

> **Disclaimer:** This related work section reflects the state of prior research in early 2022 and is provided to highlight the state of research at the time of this research project. For related and concurrent work at the time of this dissertation, see Chapter 3: Related and Concurrent 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.

4.3.1 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 data sets, e. g., of commits [189], [190], [194], contributors [191], and vulnerabilities [204], [212], as well as easy access via torrents [192], [193]. Early work describes case studies of then emerging open source projects such as Linux [174], Mozilla [175],

and FreeBSD [176]. Due to freely accessible code and commits, open source repositories are a common source for vulnerability research, e. g., by matching Common Vulnerabilities and Exposuress (CVEs) [203], [215], tracking vulnerability evolution over time or events [199], [202], [205], [214], or for evaluating static analysis tools [201], [209], [210], [213]. Both Deligiannis *et al.* and Bai *et al.* analyzed drivers in the Linux Kernel [178], [179]. Fixes and patches are essential for ensuring a secure code base, motivating previous work to investigate fix patterns and phases [224], [225], [227]. Piantadosi *et al.* linked 337 CVE entries to the corresponding patches, finding that developers who fix vulnerabilities are more experienced than average [226]. Related research focusing on social aspects investigated collaboration [275]–[277], gamification [188], donations [244], and pull requests [278]–[280]. Recently published work investigated repository artifacts such as programming languages [231], maintenance [232], [233], toxicity in comments [228], and related metadata [229], [230].

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.

4.3.2 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 [250], [252], administrators [249], [251], and security analysts [253]. Interviews were also conducted to establish the security needs of expert communities such as journalists [255], editors [256], and victim service providers [257]. As part of larger studies, interviews allow insights into specific mindsets and approaches, e. g., for encryption tasks [246] or Tor usage [247]. More recently, Gutfleisch *et al.* interviewed developers about security feature considerations in their software development process [263]. 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 [275]. 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 [236]. Balali *et al.* interviewed mentors of 10 OSS projects, identifying both challenges and strategies related to recommending tasks for newcomers [243].

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

4.3.3 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 [170], [171], [281]. Issues and commits are important structural features in the open source community, enabling evaluations of general statistics [200], security tactics [211], and emotions [207]. Antal *et al.* investigated commits of Python and JavaScript projects, finding that neither community reacts very fast to emerging security vulnerabilities in general [217]. 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 [206]. Published identification systems for open source projects include vulnerabilities [208], [218], [219] and toxic comments [282]. Trust is an important factor in public software collaboration. Research directions include trustworthiness measurements [180], [181] and factors influencing trust [177], [183], [279]. 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 [185], [187], quality assessment models [184], and discrepancies between vision and actual implementation [186]. Due to their important role in the open source ecosystem, committers are the focus of multiple works, e. g., for their pull requests [242], their motivations [235], [237], [283], [284], or contribution barriers [234], [236]. Other works propose supporting aspects such as approaches for onboarding [238], [240], [241], [243] and mentoring [239], [243]. 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 [285]. 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 [286].

While our research 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.

4.4 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.

4.4.1 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 OSPs 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 15th interview. Our full interview guides in English and German are included in the appendix (cf. Sections A.1, A.2).

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 GitHub. See also Table 4.1 for an overview of interviewed participants and corresponding recruitment channel.

Table 4.1: 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.

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 data set 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.

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 data set 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 data set 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.

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 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.

4.4.2 Interview Structure

We outline our semi-structured interview structure below and in Figure 4.1. 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

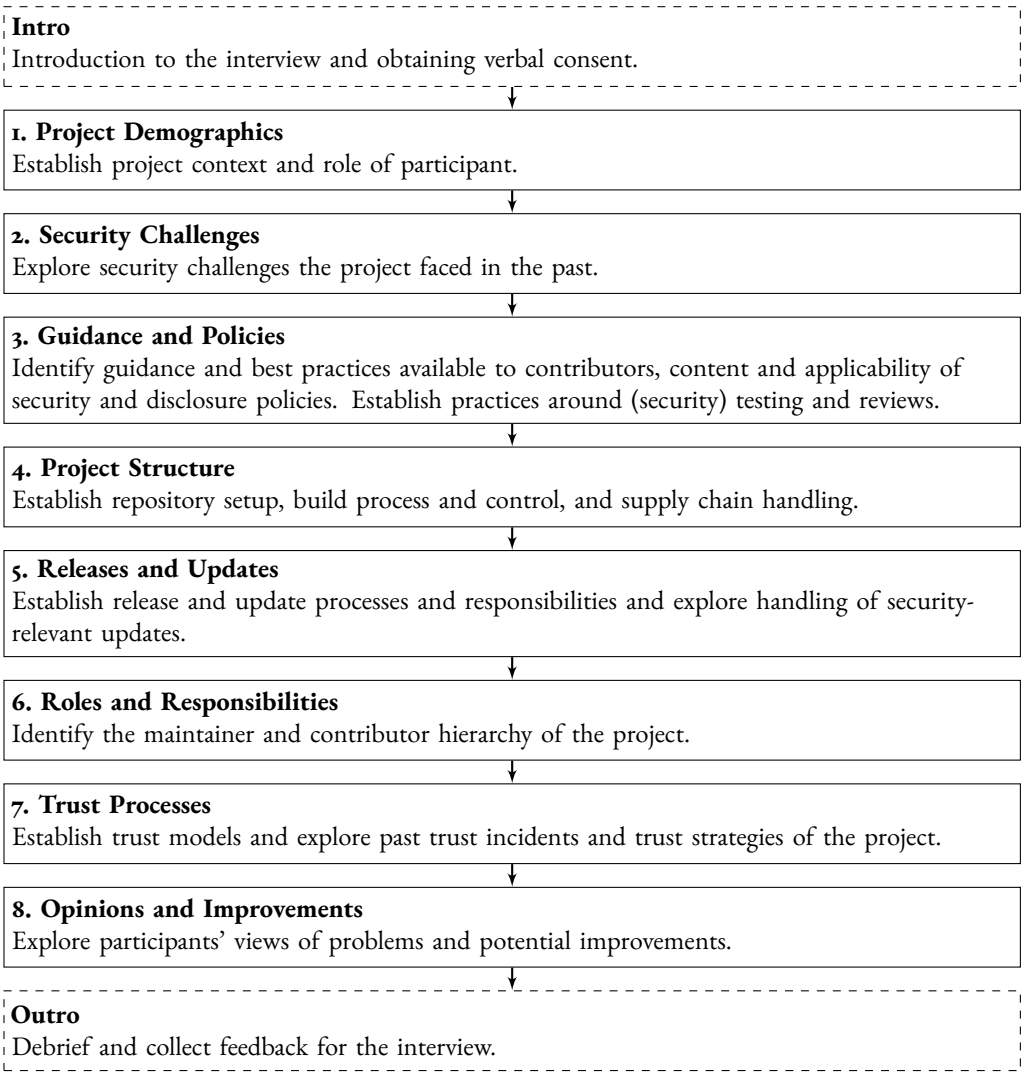


Figure 4.1: Illustration of the flow of topics in the semi-structured interviews. In each section, participants were presented with general questions and corresponding follow-ups, but were generally free to diverge from this flow at will.

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 4.5.1 and Table 4.1.

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 [274], [287], [288]. 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 4.5.2.

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 4.5.3.

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 4.5.4.

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 4.5.5.

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 4.5.6.

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 4.5.7.

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 4.5.8.

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

4.4.3 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 [289]–[291]. 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 [292], [293]. 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 the Appendix at Section A.3.

4.4.4 Ethical Considerations and Data Protection

This experiment was approved by the human subjects review board (Institutional 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 [294]. 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 [295].

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. 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.

4.4.5 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.

4.5 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 4.4.1 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.

4.5.1 Project Demographics

In total, we interviewed 27 valid participants. In addition to this section, we report general interview and project demographics in Table 4.1. 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 4.1. 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.

4.5.2 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 [274]. 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.

4.5.3 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 OSPs 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.

4.5.4 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.

4.5.5 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.

4.5.6 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.

4.5.7 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 OSP 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 proactive 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.

4.5.8 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 OSP: 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.

4.6 Discussion

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

RQ1. *"How are OSPs 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 OSPs?"* 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 OSPs 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. OSPs 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)

4.7 Summary

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 and 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.

The research presented in this chapter focused on the "producer side" of the open source software supply chain, interviewing owners, maintainers, and contributors of open source projects, investigating security and trust processes. Based on this idea, we extended this research with a project investigating the "other side" of the software supply chain, covering security considerations in industry projects around included open source components. I present this follow-up research is in the following chapter Security Challenges of the Open Source Supply Chain (Chapter 5).

Chapter 5

Security Challenges of the Open Source Supply Chain

Open Source Components hold important roles in companies' and software teams' products, setups, and processes. While external software components allow companies to focus on features and faster delivery, they also introduces unique security challenges and attack surfaces, such as code from potentially unvetted contributors and the obligation to assess and mitigate the impact of vulnerabilities in external components. This chapter investigates security challenges and considerations in the context of utilizing open source components in companies.

As this project was conducted as a team consisting of me, Jan Klemmer, Noah Wöhler, Juliane Schmüser, Harshini Sri Ramulu, Yasemin Acar, and Sascha Fahl, this chapter utilizes the academic "we" to mirror this fact. We conducted 25 in-depth interviews with software developers, architects, and engineers from industry projects to investigate their processes, decisions, and considerations when using open source code. Our fidings include that open source components play an important role in many projects, and most projects have some form of company policy or best practice for including external code. However, developers wish for more resources to better audit included components.

5.1 Preamble

This chapter is based on research that was also published as ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain" [17], which will appear at the top-tier security conference 44th IEEE Symposium on Security and Privacy (IEEE S&P 2023, "Oakland") in May 2023.

> **Dominik Wermke**, J. H. Klemmer, N. Wöhler, J. Schmüser, H. S. Ramulu, Y. Acar, and S. Fahl, ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain," in *44th IEEE Symposium on Security and Privacy (IEEE S&P'23)*, San Francisco, CA, USA: IEEE, May 2023

The original abstract for the publication is as follows:

> **Abstract:** Open source components are ubiquitous in companies' setups, processes, and software. Utilizing these external components as building blocks enables companies to leverage the benefits of open source software, allowing them to focus their efforts on features and faster delivery instead of writing their own components. But by introducing these components into their software stack, companies inherit unique security challenges and attack surfaces: including code from potentially unvetted contributors, as well as the obligation to assess and mitigate the impact of vulnerabilities in external components.

In 25 in-depth, semi-structured interviews with software developers, architects, and engineers from industry projects, we investigate their projects' processes, decisions, and considerations in the context of external open source code. We find that open source components play an important role in many of our participants' projects, that most projects have some form of company policy or at least best practice for including external code, and that many developers wish for more developer-hours, dedicated teams, or tools to better audit included components. Based on our findings, we discuss implications for company stakeholders and the open source software ecosystem. Overall, we appeal to companies to not treat the open source ecosystem as a free (software) supply chain and instead to contribute towards the health and security of the overall software ecosystem they benefit from and are part of.

The paper includes the following acknowledgements:

Acknowledgements: This work is supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972, NSF grant CNS-2206865, and the Google Research Scholar program. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies. We want to thank all interviewees for their participation and appreciate the industry-insider knowledge and valuable time that they have generously given. We also thank the anonymous reviewers for their valuable feedback.

In addition, I would like to express my sincere gratitude to everyone who has contributed to the completion of this project. I would also like to thank the participants again, who generously gave their time and shared their experiences regarding open source components in their industry projects with us.

5.1.1 Contribution

The research presented in this chapter was conducted as a team consisting of me as team lead, Jan Klemmer, Noah Wöhler, Juliane Schmüser, Harshini Sri Ramulu, Yasemin Acar, and Sascha Fahl. I am grateful for the contributions of each member, which have been integral to the success of this research project. Without their expertise, hard work, and dedication, this research project would not have been possible.

I came up with the initial idea for this study based on a logical follow-up to the previous paper "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects" [16]. I lead the design of the study and interview guide with the rest of the team. I implemented the landing page and contact templates for this study, and iterated them with the group. Jan Klemmer and I invited participants via the team's professional network and from job postings on Upwork. I conducted the majority of interviews either alone or with support from the rest of the team. In joint work with Jan Klemmer, Noah Wöhler, Juliane Schmüser, and Harshini Sri Ramulu, we qualitatively coded the interview transcripts. I analyzed the coded text passages and code counts. I compiled the paper for publication with contributions from the remaining team and we jointly discussed the work's implications.

5.1.2 Structure

The remainder of this chapter is structured as follows: After a general introduction (Section 5.2), I provide the related work at the time of this project in early 2023, covering the areas of dependency analysis and selection, security research with software developers, and security interview studies (Section 5.3). I then describe our interview approach (Section 5.4) and highlight our findings (Section 5.5). Finally, I discuss our findings (Section 5.6) and provide a summary for this chapter (Section 5.7).

5.2 Introduction

Open source components (OSCs) play an important role in many companies' and software teams' setup and processes. Whether as libraries and packages included in their software, as foundation or glue for their development and deployment processes, or as part of an even longer software supply chain itself: Utilizing external software components as building blocks in their processes and products enables companies to leverage the benefits of open source software (OSS), allowing them to focus their efforts on features and faster delivery. According to a 2020 RedHat report, 95% of IT departments and companies consider OSS as strategically important to their organization's overall enterprise infrastructure software strategy [273].

By introducing these external components into processes and their stack, industry projects inherit the unique challenges and attack surfaces from open source projects: companies are including code from potentially unvetted contributors and sources and are now obligated to assess and mitigate the impact of vulnerabilities from external code included in their software. While not strictly open source, the impactful SolarWinds Orion attack wave highlighted the industry's vulnerability to compromised external code components [296]. In December 2020, cybersecurity company FireEye discovered that advanced persistent threat actors had created a backdoor hidden in a software update of SolarWinds' Orion system, affecting almost 18,000 customers worldwide [297]. Malicious actors are aware of the widespread use of OSCs in the industry and have tried to leverage this attack vector in the past: In August 2022, 10 packages on the popular Python package index PyPI were found to be malicious by Checkpoint [298]. Installing one of these packages triggered a malicious script, crawling the local browser storage for passwords, cookies, and crypto-currency wallets, extracting them via a Discord server hook. The integrity of OSCs is not only threatened by malicious external actors: In a so-called protestware incident, the JavaScript *node-ipc* library (dependency of, e. g., *@vue/cli* and the Unity game engine) was updated by the maintainer as protest to Russia's invasion of Ukraine in early March 2022. Depending on version and if the machine's IP matched a list of Russian or Belarusian addresses, the library would replace all of the user's system files with heart emojis [77]. This and other recent protestware incidents highlighted that even initially well-meaning changes can be conceived as threats to the software supply chain and harm the trust in OSS.

In this chapter, we aim to shed light on the security challenges and considerations of companies and software teams around including OSCs in their projects and processes — by exploring industry projects' behind-the-scene processes, provided guidance and security policies, as well as past security challenges and incident handling. Our research approach utilizes 25 in-depth interviews with software developers, architects, and engineers from a diverse sample of industry projects and companies, to investigate the importance of OSCs in companies' software stacks, as well as related security challenges and considerations, guided by the following research questions:

RQ1. *"How are Open Source Components included in companies' tech stacks in terms of position, importance, and security effects?"* OSCs hold an important role in many companies' software stacks. We are interested in the specific roles of these components in the software stack, as well as if and how these components are considered in the update and security processes of the projects.

RQ2. *"What are companies' awareness, experiences, and attitudes regarding the security of including external open source code?"* Including external OSCs in industry projects introduces unique security challenges and attack vectors such as code contributions from unvetted sources. We are interested in companies' awareness surrounding the security of including external open source code, as well as their experiences with, and past challenges of, including external code in the context of security and updates. We are also interested in the companies' attitudes about including, managing, and contributing back

to open source projects (OSPs).

RQ3. *"If and how do stakeholders make decisions and considerations around security and trust challenges of including Open Source Components?"* The major impact of security challenges in OSCs justifies specific considerations. We are interested in measures that companies use to decide on including OSCs, what decisions and considerations they have in place for the external code, and which improvements and changes stakeholders consider.

5.3 Related Work

> **Disclaimer:** This related work section reflects the state of prior research in early 2023 and is provided to highlight the state of research at the time of this research project. For related and concurrent work at the time of this dissertation, see Chapter 3: Related and Concurrent Work.

In this section, we present and discuss related work in three areas: research investigating dependencies and the selection thereof, security research involving software developers and similar stakeholders, as well as interview studies with a focus on security. We also put our work into context and illustrate the novel contributions of our research.

5.3.1 Dependency Analysis & Selection

Dependencies are a popular (security) software research topic, as they can hide critical attack vectors and attack surfaces. Dependency ecosystems are a common data source for measurement studies in this field, e. g., for package repositories like JavaScript's npm [112]–[117], Python's PyPI [119], [299], Ruby's gem [120], R's CRAN [121], and the wider software ecosystems like for Apache [122], Gentoo [123], Java [124], [125], or Android [126]. The inclusion of third-party dependencies and the associated technical challenges have been studied and compared across a variety of software ecosystems [133]–[136]. In 2020, Ponta *et al.* presented a novel method for detecting vulnerabilities in OSS dependencies [300]. The propagation of vulnerabilities within the npm ecosystem has been studied with the help of dependency trees [128] and dependency graphs [117]. The obfuscation-resilient detection of libraries in Android apps has been advanced for in-depth analyses of apps with a focus on malicious third-party libraries and malware detection [156]–[158]. The selection of dependencies is crucial for supply chain security. However, it is also a major challenge because approaches, criteria, and metrics for good and secure choices are hard to generalize and various exist. In 2010, for example, Mileva *et al.* mined and evaluated API popularity and trends for 200 Java projects [138]. The authors demonstrate that it is possible to give adoption recommendations based on past usage trends. Similarly, libraries that are already included in a project can also be used for further library recommendations, as Nguyen *et al.* demonstrated with the library recommendation system *CrossRec* [139]. In 2020, Xu *et al.* surveyed 49 developers from GitHub and F-Droid to analyze reasons why developers replace own code with a library, i. e. re-use, or re-implement a library's functionality [110]. In 2018, López de la Mora and Nadi proposed a metric-based approach for informed adoption decision when selecting and comparing libraries [143], [301]. Kula *et al.* conducted an empirical study on library migration covering 4,600 GitHub software projects and 2,700 library dependencies, finding that 81.5% of the studied systems still keep their outdated dependencies [109]. Supply-chain attacks and vulnerabilities that propagate through the dependency chain have been systematized [91] and analyzed to inform the development of protective measures [92], [172], to improve the accuracy of vulnerability alerts [93], and to better

understand the factors that influence dependency vulnerability remediation in software projects [95]. Larios Vargas *et al.* identified 26 technical, human, and economic factors that developers consider in their dependency selection processes based on 16 interviews and a survey with 115 developers [111]. More recently, in two preprints, Zahan *et al.* utilized Open Source Security Foundation (OpenSSF) Scorecards, both for investigating security features in npm and PyPI, as well as their impact on security outcomes, highlighting some impactful features [222], [223].

Compared to prior work investigating dependencies utilizing measurements and systematization, we leveraged interviews to investigate in-depth the real-world selection and inclusion practices of, and experiences around, open source components in companies and software teams, providing additional and enhancing insights to previous measurement results.

5.3.2 Security Research with Software Developers

Research investigating security aspects with developers, architects, and engineers working on industry projects provide important insights into the security and health of the overall software ecosystem.

Past research investigated the security impact of different aspects such as decision-making [96], [97], organizational changes [98], [99], and information sources [100], [101]. Stevens *et al.* conducted a multi-stage study with 25 industry employees investigating aspects of threat modeling [102]. Assal and Chiasson surveyed 123 software developers about software security processes, finding that the real issues frequently stem from a lack of organizational or process support [103]. More recently, Ladisa *et al.* introduced a taxonomy for attacks on open source supply chains, validating their taxonomy by surveying 17 domain experts and 134 software developers [173].

Similar to prior work with software developers, we consider industry developers and software teams to play an important role in the overall security and health of the software supply chain.

5.3.3 Security Interview Studies

A common approach for in-depth, qualitative research in the security community are interview studies. Prior interview studies were conducted to establish the security needs of expert communities such as journalists [255], editors [256], and victim service providers [257]. As part of larger studies, interviews allow insights into specific mindsets and approaches, e. g., for encryption tasks [246] or Tor usage [247]. Huaman *et al.* conducted 5,000 computer-assisted telephone interviews with small and medium enterprises in Germany, finding that security awareness has arrived in all companies [21]. More related to this research, past research has utilized interviews to gain insights into the work and tools of experts such as security professionals [252], app developers [248], administrators [249], [251], and security analysts [253]. Specifically, Botta *et al.* interviewed 12 security management professionals, finding that the job of IT security management is distributed across multiple employees [250]. Haney *et al.* conducted 21 interviews in organizations including cryptography in products, finding an uniquely strong security mindset in those companies [302]. More recently, both Jansen *et al.* and Ghofrani *et al.* conducted smaller-scale interview studies with industry developers investigating the trust aspect of external software [260], [261]. Compared to these smaller-scale, preliminary works, our work focuses less on specific trust aspects, with our approach covering the broader topic of OSC in companies, covering real-world usage, company policies, and security considerations. Gutfleisch *et al.* interviewed developers about usability considerations in their secure software development processes, identifying a high impact of contextual factors [263]. Wermke *et al.* interviewed 27 open source maintainers about security and trust considerations in their projects, finding that the projects were highly diverse both in deployed

security measures and trust processes [16]. Similar to Wermke *et al.*, we also decided on leveraging 25 in-depth interviews to gain detailed insights into participants' perceptions, behaviors, and reasoning, focusing on the "other" end of the open source software supply chain, interviewing stakeholders of industry projects in the context of OSCs they use.

Overall, we leveraged 25 interviewes with participants from industry projects to investigate the broader picture of OSCs in companies and software teams, covering topics including, but not limited to, real-world usage, company policies, and security considerations.

5.4 Interview Study

In this section, we outline the interview approach including the structure of our interview guide, the subsequent coding and analysis steps, ethical considerations, and potential limitations of our research approach. The full interview guide and codebook are included in the appendix (cf. Sections B.1 and B.2).

5.4.1 Study Setup

To investigate security considerations and experiences around OSCs in companies and software teams, we conducted semi-structured interviews ($n = 25$) with software developers, architects, and engineers experienced in industry software projects between May and October 2022. We opted for interviews as a qualitative approach, because we wanted to focus our investigation on processes not necessarily visible on a software level and rationales, e. g., how a decision for or against including a component is made, how incidents are handled internally, or what the (potentially unwritten) policies for including external code look like. Conducting this research study as interviews also allowed us to explore participants' decisions and considerations in-depth by asking follow-up questions.

Interview Guide. We conducted the interviews with an established interview guide based on our research questions. In addition to our research questions, we also considered concepts and findings from previous and ongoing related work and adapted them for in-depth interviews. We gathered feedback from, and tested the initial interview guide with, pilot interviews in the team and with industry stakeholders from our professional network. After the initial pilot interviews, we only performed relatively minor changes: Adding a few minor follow-up questions to improve coverage of interesting areas as well as updating some question wording and moving questions between interview guide sections for better interview flow. No further changes beyond minor grammatical modifications were added after the 8th interview. The full interview guide is included in the appendix (cf. Section B.1).

Recruitment and Inclusion Criteria. We based our recruitment approach around covering a diverse set of industry projects utilizing OSCs. For recruitment, we utilized multiple recruitment channels to better reach a diverse set of companies from different historical, structural, and industry contexts. This included recruiting expert talent via Upwork and our professional network:

1. Industry Experts. For recruiting expert talent, we turned to Upwork, a platform for professional developers and freelancers. We posted a job posting for our interviews and specifically selected participants based on their experiences working in company projects utilizing some form of open source, aiming for a diverse sample with a broad coverage of the industry.

2. Professional Network. In addition to Upwork, we enhanced our sample with professionals from our own network, specifically targeting software solutions that are less commonly encountered in industry but still play important roles, such as embedded hardware or research software projects.

Table 5.1: Detailed overview of interviewed software developers, their projects, as well as some project metadata. According to our interview guidelines, participants were assigned an alias and their projects' metadata was binned to preserve their privacy.

Alias	Interview			Projects		
	Duration	Codes[1]	Recruitment Channel	Position[2]	Area	Software Stack[2]
P01	46:21	31	Professional Network	Developer	Machine Learning	Python, Flask, AWS
P02	50:59	34	Professional Network	Sec Engineer	Finance, VR	JavaScript
P03	33:54	29	Professional Network	Lead Dev	Embedded	C, STM32
P04	29:02	29	Professional Network	Team Lead	Mobile	Android
P05	39:19	31	Professional Network	Lead Engineer	Framework	Python
P06	30:21	26	Professional Network	Developer	Industry	Java, Spring
P07	38:20	25	Industry Expert	Senior Engineer	Finance	Node.js, SQL
P08	54:15	24	Industry Expert	Lead Dev	Web Apps	PHP, Laravel, MySQL
P09	35:17	36	Industry Expert	Lead Dev	Web Apps	Angular JS, ASP.NET, Python, C#
P10	40:33	27	Industry Expert	Architect	Various	Java, Maven, Terraform
P11	41:01	26	Industry Expert	Senior Engineer	Enterprise Apps	Java, Node.js, Angular JS
P12	52:04	27	Industry Expert	Founder	Enterprise Apps	Java
P13	25:20	30	Industry Expert	Developer	Web Apps	PHP, WordPress
P14	36:50	28	Industry Expert	Developer	Backend	React
P15	49:51	19	Industry Expert	Consultant	Various	Java
P16	49:24	35	Industry Expert	Developer	Finance	Angular JS, Vue.js
P17	26:25	30	Industry Expert	Architect	Various	ASP.NET, Angular JS, React Native
P18	39:25	32	Industry Expert	Developer	Mobile	Android, Spring, Angular JS
P19	27:39	30	Industry Expert	Expert, Architect	Embedded	Terraform
P20	29:04	33	Industry Expert	Developer	Enterprise Apps	JavaScript, Ruby on Rails
P21	33:29	22	Industry Expert	Developer	Health & Wellness	Java, PostgreSQL
P22	51:04	28	Industry Expert	Team Lead	Web Apps	JavaScript, React, Node.js
P23	28:09	25	Industry Expert	Developer	Web Apps	.NET, C#, React
P24	43:04	28	Industry Expert	Developer, Auditor	Mobile	C++, C-Basic, C#, Flutter
P25	33:56	30	Industry Expert	Developer	Blockchain	React, Python

[1] Total number of codes assigned to the interview after resolving conflicts.
[2] Based on self-reporting of participants and binned to preserve their privacy.

See also Table 5.1 for an overview of interviewed participants and corresponding recruitment channels. Due to the previous filtering, we did not require any additional eligibility criteria from our participants beyond stating that we were looking for professionals working on industry projects utilizing OSCs. In total, we recruited 25 participants from equally as many distinct companies and projects. As compensation for their valuable time as domain experts, we offered each participant $60 or the equivalent value in local Amazon vouchers.

Interview Procedure. We conducted the 25 interviews either in a solo interviewer or lead and backup interviewer configuration. We chose the lead and backup interviewer setup so that the lead interviewer can fully concentrate on asking questions and listening to the interviewee, and the backup interviewer could ensure that no questions are forgotten, ask additional follow-up questions that emerge, or take over in case of any connection issues. We conducted all interviews virtually; mostly via our self-hosted Jitsi instance, or any other tool of the participant's choice (e. g., *Zoom*, *Google Meet*, etc.). We advertised the interviews with a duration between 35–45 minutes depending on answer duration and scheduled one hour interview appointments for some time to spare. Overall, the median duration of the actual interview part, excluding short introduction, consent gathering, and debriefing, was 38:20 minutes.

In general, the interviews were based around non-leading, open questions, allowing interviewees to elaborate their thoughts and answers. Each interview section started with a general question, allowing participants to freely state what they had on their mind. Only if specific points were not already addressed by that time, we asked more specific sub-questions as follow-ups. All interviewers were

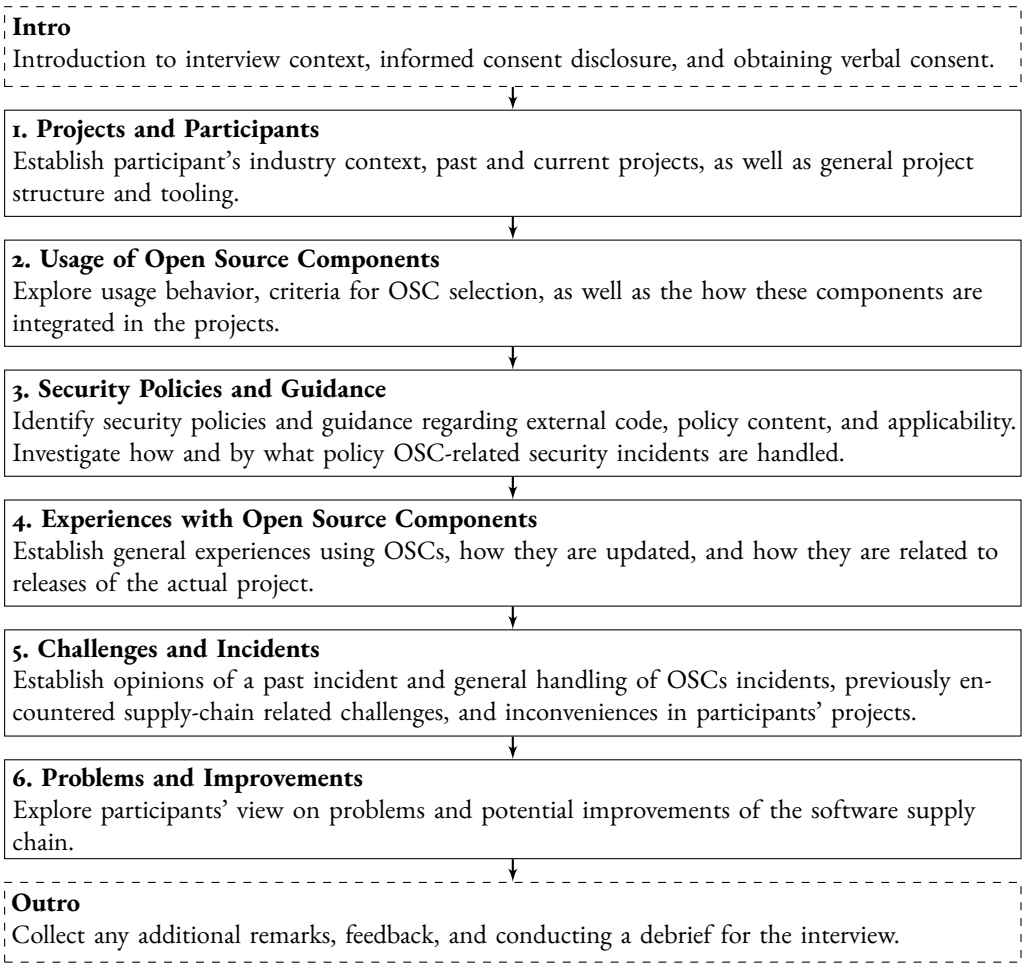


Figure 5.1: Overview of the semi-structured interviews' flow and topics. After introducing each section with a general question, we followed-up with specific questions (if not already covered). Due to our semi-structured interview approach, participants were allowed to diverge from this flow at any time.

instructed not to prime participants through the questions and not to impart any sense of judging, e. g., regarding specific OSC choices or security practices.

5.4.2 Interview Structure

We report on the structure of the semi-structured interviews below and in Figure 5.1. The interviews were structured in six main sections consisting of one to four opening questions, corresponding follow-up questions, and sometimes additional nudges or explanations. We also report on the results from the interview in sections corresponding to the interview guide's question section (cf. Section 5.5).

Before starting the interview, we provided participants with a general introduction of ourselves and our research project, followed by an explanation of our goals, the interview process, and the interview's role in that process. We specifically affirmed participants that participation in the interview is voluntary, that they could skip any question for any reason, that we were not judging their projects in terms of security or privacy, and that we were also very interested in their personal thoughts and opinions

about processes. We guaranteed full de-identification of any quotes we might use and offered to send participants a preprint of the potential scientific publication based on their interviews.

After answering any remaining questions and obtaining consent for data handling and recording from the participant, we started a recording and began the actual interview with the following structure:

1. Projects and Participants. In the first interview section we asked our participants to describe their projects, their relation to it, as well as project structures and tooling. This section intends to both ease nervous participants into the interview as well as to establish some initial context about the participant and their projects. Specifically, we prompted for project context and structure, team size and tools, as well as for the development process and stages. We report these results in Section 5.5.1.

2. Usage of Open Source Components. Both the *"Usage of Open Source Components"* and *"Thoughts about Open Source Components"* sections investigate the usage of open source components in our participants' projects. Specifically, we were interested in the technical implementation and processes, selection and exclusion criteria for open source components, as well as whether they contributed back to open source projects in some way. We report these results in Sections 5.5.2 and 5.5.3.

3. Security Policies and Guidance. Our third block of questions covers security policies and guidance for including external code like OSC in projects. We asked about company policies and project-specific guidance or documentation for the inclusion of external code, and the participants' personal opinions and wishes regarding these. Additionally, we investigated the general processes and policies for security incidents in external components. We report these results in Section 5.5.4.

4. Experiences with Open Source Components. The fourth section focuses on the participants' personal experiences with OSC in their projects. Our questions covered aspects such as the developer experience of using OSCs and if components had to be customized for the projects. We also investigated the update, release, and deprecation procedures of the projects in the context of external components. Lastly, we asked whether our participants would use the same components again and why. We report these results in Section 5.5.5.

5. Challenges and Incidents. In the fifth interview section *"Challenges and Incidents"*, we are interested in specific OSC-related security incidents and inconveniences experienced by our participants in the past. To ease our participants into this sensitive topic, we asked them about their opinion regarding open source software supply chain security of the "node-ipc protestware" incident from March 2022 [77]. We inquired about participants' opinion of the incident, as well as for their strategies to deal with similar incidents in their project. We then specifically asked them about any OSC-related security incidents or inconveniences their projects might have encountered in the past. We report these results in Section 5.5.6.

6. Problems and Improvements. In our final interview section, we investigate our participants' opinion of their projects' security, as well as problems they see with the current software supply chain and their suggested solutions. We report these results in Section 5.5.7.

Following the interview sections, we asked our participants for any additional insights and aspects that we might have missed or they wanted to talk about. After completing the interview, we thanked them for their valuable time and offered them an opportunity for questions and comments, concluding the interview with a debriefing.

5.4.3 Coding and Analysis

For our evaluation of the interviews, we recorded the audio of interviews digitally, removed identifying information from recordings, transcribed them via a GDPR-compliant service, and manually reviewed

all transcripts for potential transcription mistakes. We analyzed all interview answers in an iterative semi-open coding approach [289]–[291]. All researchers together established an initial codebook based on the interview guide and interview impressions. Five researchers then iteratively coded the interviews according to the codebook 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 [292], [293]. Our approach does not necessitate the reporting of intercoder agreement, as each conflict is resolved when it emerges (resulting in a hypothetical final agreement of 100%) [303]. In total, we assigned 715 codes after resolving, resulting in a median of 29 codes per interview. The final codebook is included in the appendix (cf. Section B.2). As part of discussing our results, we report on some counts. We want to highlight that counts from a qualitative interview study with a sample selected for diverse background are not intended to be representative counts for the wider developer population, but are included to give some general idea about the distribution of codes and to highlight especially prevalent or underrepresented themes in the interviews.

5.4.4 Ethical Considerations & Data Protection

This interview study was approved by our institutions' Institutional Review Board (IRB) as well as Human Subjects Review Board (IRB equivalent). Our study was modeled after the ethical principles for research involving information and communication technologies outlined in the Menlo report [294]. The research plan, study procedure, and all involved research parties adhered to the strict German data and privacy protection laws as well as the General Data Protection Regulation (GDPR). Before signing up for interviews, we provided participants extensive information about our study procedure and data handling, encouraged them to get informed before making a decision, and offered to answer any questions they may have had. We emphasized to participants that they could skip any question for any reason such as not knowing an answer, not wanting to answer, or not being allowed to answer, as well as that they could drop out of the interview at any time. We offered participants to provide them with a preprint of the paper before publication, allowing them to request changes or to correct misunderstandings. To compensate our domain expert participants, we offered them $60 or the equivalent value in local Amazon vouchers.

All data was collected, handled, and stored in compliance with the EU's GDPR. In accordance, any personally identifiable data was stored in a secure cloud collaboration software, encrypted at rest as well as in transit. For transcribing the interviews, we commissioned an EU-based, fully GDPR-compliant transcription service.

5.4.5 Limitations

A number of limitations typical for this kind of interview study apply to our work, including potential over- and under-reporting, self-reporting, recall, and social-desirability biases, as well as sampling bias. Our sample is a convenience sample which may not be representative of the larger population of developers working on industry projects utilizing OSC. Experts who agreed to participate in our study might be more or less open source or security-oriented than those did not sign-up for an interview. We conducted our interviews in English, so we cannot provide insights into non-English-speaking industry projects. As English is the de-facto "working language" of international software projects, we consider this a negligible drawback that still allows us to reach a meaningful set of developers. Since questions about security practices and incidents can be considered sensitive, we attempted to mitigate social desirability bias by emphasizing that we were not going to judge the participants or their answers

in any way but were genuinely interested in their processes and opinions. We also reminded them that they could skip questions as desired and for any reason.

5.5 Results

We report and discuss results for 25 semi-structured interviews with software developers, architects, and engineers. In our reporting, we mostly adhere to the structure of the interview guide described in Section 5.4.2 and summarize our key findings after each question block. We report participants' quotes as transcribed, with de-identified information, minor grammatical corrections, and omissions highlighted by brackets ("[...]").

5.5.1 Projects and Participants

In total, we interviewed 25 valid participants, reporting on their projects and background in this section and Table 5.1. We only report binned project metrics to preserve both our participants' and their projects' privacy. Due to our recruitment approach aiming for a high diversity in projects, our participants reported a wide range of projects and backgrounds, ranging from web applications, over embedded devices, to scientific computing frameworks.

As the vast majority of our participants (23) had worked on multiple projects in the past, we encouraged them to highlight aspects of their projects as they saw fit. The majority of our participants (22) worked or had worked on projects in teams, specifically with two to five (9) or more than five (13) developers. About half (13) mentioned having worked on projects with multiple teams, e. g., "We are a very flat team, so basically everyone is on the same level. We discuss things together and we work together. We have a development team for back-end code, for front-end code, and the design team." (P13) We were also interested whether this included specific teams or members with a security background: 11 participants mentioned security-specific roles on their projects, e. g., "Yes, we have online software security engineer or cybersecurity engineer [...]. Then you have more dedicated roles for the information assurance processes and some of the other cloud based [services]." (P19) These 11 also include security-specific roles provided by clients, e. g. for P12: "Actually, most of the time, our clients are enterprise clients and we hand over the codebase to them and then their security team." (P12) About half of our participants (13) specifically mentioned not having someone with security-specific background or experience in the loop, e. g., "No, I haven't worked with any company [that] had something like [a security-specific role]." (P18)

Overall, we found our participants to be quite knowledgeable about their projects, with many years of experience in different areas of software development. This allowed us in-depth insights into the considerations around OSCs in their projects, a goal we hoped to archive with our recruitment strategy. Based on our findings, an interesting area for future research could be how security processes differ between the different industry areas.

> Summary: Projects and Participants.The majority of our participants had worked on multiple projects in a diverse set of software areas, and in different team configurations and sizes. Only about half mentioned security-specific roles in the development loop.

5.5.2 Usage of Open Source Components

In this interview section, we were interested in the general usage and selection criteria of OSCs (for specific experiences with OSCs, see also Section 5.5.5). All 25 of our participants mentioned using OSCs in their projects (unsurprisingly, as we specifically selected for this), e. g., "Every solution that we have built, we heavily use open-source components." (P10) Some participants even voiced a philosophical attachment to the idea, e. g., "We are using a lot of open-source things because, by philosophy, we like to embrace open-source community" (P07), including perceived benefits such as reduced maintenance burden:

> "The software are well maintained, and we don't have to focus on maintaining the work. We can just use them, and if we see any problems, we can contribute to this and we don't have to do any in-house maintenance." (P07)

For pulling in OSCs into their build processes, our participants have a number of different approaches, often relying on the (semi-)official package repositories such as PyPI or npm, e. g.,

> "So for builds we use npm and also some other package managers. Usually it's a mixed project with different code bases, and we pull the packages directly from the package managers. I don't think we currently have any that are not managed by package managers, which is great." (P02)

Aside from package repositories, some participants mentioned directly pulling from repositories if there are no other options available: "[For] some of the dependencies, if either the current version is not maintained [on PyPI] or for some machine learning tools [...] we pull them directly from the Git repo for them." (P01) Others mentioned directly pulling from GitHub as a potential security concern:

> "I think some components are fairly secure because we draw from sources posted and maintained by hardware manufacturers. [...] These are open source components, but they have major corporate backing behind them, and we're not pulling it from GitHub." (P04)

Some participants specifically mentioned configuring and modifying OSCs to fit their needs and the requirements of their projects: "We pull them in, and then all of these open source components, we take them and then we do a bit of work on it ourselves. So, for example, we run packet managers and we add an external code." (P17)

Updates & Releases. As for keeping their OSCs up-to-date, our participants seem to follow the same pattern for pulling in components, i.e., relying on the package management tools of their software stack, including tools like *npm audit*: "We even have a mechanism that lets the build fail if there is a component that could not be updated if an update is available or if there is a vulnerability reported by npm, for example." (P02) Others routinely revisited included components:

> "It's some of the external dependencies that, yes, can go out of date or can disappear, that are more of a cause of concern. That's why we periodically revisit that whenever we upgrade or whenever we make a new release." (P04)

Outside of the build and update process, some participants mentioned to keep track of included components via their package managers, but only one specifically mentioned maintaining a Software Bill of Materials (SBOM):

> "We do regular scans, we build Software Bill of Materials [...] and we put those Bill of Material files into Dependency-Track which is free software, open source provided by OWASP that we also use for vulnerability management and overview." (P02)

Both tooling and research around the recently established SBOM format might present a promising research opportunity.

Of our 25 participants, 14 mentioned that they (at least partially) use internal mirrors for pulling software into their build processes, e. g.,

> "We do use internal mirrors mainly for speed and convenience, especially large code bases. [...] Usually, when we clone from those internal repositories, we're going to use fixed commits from it, so it makes development a lot easier.," (P04)

and three participants mentioned other solutions like local build caches: "We have a local cache. So we try to have everything pulled only once and put it in our cache, but sometimes we get upstream changes, so we pull it from there." (P03).

Selection Metrics. We were interested in what metrics and criteria our participants use for deciding on and selecting OSCs. Most commonly mentioned metrics included: Some form of popularity measure like downloads or GitHub stars (16), a large and active community (11), specific features (10), and activity measures like commit frequency and recent releases (10). We were also interested in what criteria would exclude an component from being used by our participants. Most common exclusion criteria included: the project being visibly inactive (5), a low number of contributors (4), and specific company policies (3). On the security side, participants mentioned looking for a positive security history (8) and exclusion based on vulnerable or malicious code (3), e. g.,

> "if the vulnerability score is ridiculously high, obviously, then we would not allow that [component]. Also, if certain vulnerabilities exist for functionalities that we want to actually use, if those are compromised, then obviously it's not a good choice." (P02)

Lastly, we were interested in whether our participants had used or heard of recently emerging automatic metric tools before, such as the OpenSSF Scorecards for repositories. The majority (17) had not, and only four had heard of (but not used) the OpenSSF Scorecards project specifically: "Yes, I have heard of those, but we have not used them yet." (P02) For other metric tools, e. g., P12 mentioned: "We are using some integrated code quality tools, and there are some predefined, maybe hundreds of rules to check the quality of the code. These tools also provide a feedback on code security." (P12)

Overall, our participants' selection metrics and criteria seem to focus on quickly accessible numbers and facts, such as downloads, GitHub stars, and time since the last release. Understandably so, as there a often many different open source packages to be considered for each use case.

> Summary: Usage of Open Source Components.All of our participants included OSCs in their projects. About half maintained internal mirrors or caches for their builds. Common selection and exclusion criteria included easily visible metrics like activity, number of contributors, or GitHub stars.

5.5.3 Thoughts about Open Source Components

Aside from our participants' usage of OSC in their projects, we were also interested in their thoughts about supporting the open source ecosystem, company policies prohibiting packages, and which metrics they would like to use for selecting components.

We asked participants if their company contributed back to open source projects in some form including pull requests or issues, which 14 do and five would like to, e. g., "We rarely submit PRs, but we submit issues, and if we find some bugs or enhancements regularly, because, often, there are some bugs, and also, we found a few security-related, actually." (P01) Some participants even mentioned contributing back as a company policy:

> "We also do heavy contribution on anything that is being leaky, or if something is not looking right, we always create issues. This is something that we also have in our company policy: Always contribute back." (P07)

Sometimes, this contribution was out of necessity:

> "[A feature] never made to production because the [open source] project went dead completely. When we were thinking about the operation of the new release, we basically had to take over, fork it, and then implement that feature on our own." (P10)

Some participants suggested that their management or legal departments do not fully understand the open source ecosystem.

For the three cases where company policies prohibit specific packages, reasons often involved the package's license: "Many open source projects are using different licenses, some of the licenses are not okay for big projects [...] we check if this package is using restricted license [...] " (P24) and "There are no rules, except that the license has to be compliant with what we do." (P01) Some mentioned that they include their clients in these decisions: "There are certain exclusion criteria based on our customers' concerns. For example, if they don't want us using products from certain companies, then we don't do that." (P04) We were also interested in what selection criteria our participants *would like* to use if they could. The wishes included specific (free) software solutions:

> "I think there are many interesting software solutions as far as I could see. So most of them, obviously, combined with costs. I saw a lot of software that is supposed to help with managing third party vulnerabilities and so on and used for scanning. It's always a question of the price and most of the time I would say it's not really worth it." (P02)

as well as security-focused metrics: "It would be nice to have some security metrics. How much security leaks [the project] has, or in what time frame it will be fixed, or what time frame fixed for last issues." (P11) Based on our findings, we suggest developers might require metrics and tools that are still simple and free, yet cover more aspects of an OSC than just popularity.

Overall, our participants seemed to have a very positive attitude about open source projects and are aware of the importance of contributing back.

> Summary: Thoughts about Open Source Components.Most projects contributed back to open source projects in some form or would at least like to, with some participants suggesting that their management or legal departments do not fully understand the open source ecosystem. Some participants mentioned that their company's policy prohibited them from using certain open source packages, mostly due to non-permissive licenses.

5.5.4 Security Policies and Guidance

In this interview section we were interested in the policies around OSC usage in projects, as well as provided guidance and documentation for external components. More than half (16) of our participants

mentioned some form of company or team policy for including external code in their projects. These policies range from quite strict: "Every single third party library that is used, installed, involved in our projects is vetted and must be approved." (P02), over somewhat more lenient: "We can use anything that is signed off by our CTO and our project lead. Like any piece of code that has been vetted by them, we are free to use it on our projects." (P07), to fully placing trust on the individual:

> "[The policy is to] make sure that the plugins or open source components that we use are still updated and they're still supported. We wouldn't want to include something very old, but it very much depends on the developer to make sure of that." (P13)

This number also includes more or less informal policies that are nonetheless applied by the developers of the project, such as: "Not on a systematic basis, but most of the time my team if we are using a new library, that we are going to use it for the first time, we are actually checking it in vulnerability databases." (P12) Participants also mentioned policies influenced by external laws and standards such as HIPAA: "If we are including something which is not self-hosted, then we have to take [HIPAA] into consideration with what they claim happens with the data on their end." (P17) as well as ISO: "In some [client] companies, [policy] follows the standard security programs like ISO and things like that." (P10) Other policies are less concerned with actual security and more with copyright, e. g., "at [company], we were allowed to use things like npm and Angular, but we had to basically extract all the legal stuff, so the licenses basically, compile a list and give it to them." (P16)

We were also interested in what our participants wished to see in a (security) policy for external code, which included some more general advice "You need to have clear guidance on how to select packages, which quality, how you would define the quality of the package." (P16) as well as specific security considerations such as: "[...] doing searches to see if the software that we're considering using has had any prior issues, whether it's security issues, whether they've been disclosed through penetration testing or through some other means [...]." (P09) Both P17 and P18 mentioned why such a policy might not be relevant or even a good idea for every project size: "For small projects, I don't think those policies would help because they would create more governance and more red tapes." (P18) and

> "With the size of our team I don't believe it's really needed, because if a developer wants to include a component, the code is generally reviewed before it goes in by people who would be in the know whether or not this component can be included. And in general, during the initial discussion of when we are explaining what needs to be done and they are giving their thoughts on it, is when we decide what needs to be included and what doesn't." (P17)

As part of policies, we were interested in how a potential security incident in an external component would be handled, by what policy, and by whom (for actually encountered incidents, see Section 5.5.6). Only six of our participants mentioned that they would involve or hand a hypothetical incident to a security team, e. g., "We'll tell [the security team], hey, this needs to be patched pretty soon, and if it doesn't happen then we escalate it to the management above it, and we exert pressure on both ends." (P04) P18 provided us some insights on why a security team does not necessarily make sense for every company structure:

> "There's no security team that specifically would do that because if there is a team that exists that only does that, they would probably just sit on their hands with nothing to do a lot of time, and this team needs to work across different projects across the company, then that they can handle that because the different projects in a company probably have a lot of languages, different languages, different frameworks. There's no way either they

> will know enough to poke in the project and respond fast enough, or there's no way the company is willing to pay money for that for a team that doesn't produce anything." (P18)

We were also interested in whether companies had included a disclosure policy for security issues. Only seven participants mentioned something akin to a disclosure policy for the public or their clients, e. g., "Yes, we do. Well, a policy for coordinated disclosure [...]. Internally, we try to fix it as quickly as possible and only then disclose it." (P05) Other companies appear to be in the process of implementing a disclosure policy, e. g., as mentioned by one participant:

> "I guess the organization that kind of writes the standards that we follow is trying to adopt widespread disclosures of historical and current cybersecurity threats, but at the moment I have not seen any of those come out yet." (P19)

Overall, most participants mentioned some form of policy or common best practice in their teams, although some of the company policies seem more focused on fulfilling external standard or law requirements instead of ensuring the security of included external components.

Guidance & Documentation. A small majority of our participants (14) did mention not having specific documentation or guidance for OSCs in their projects. As reasons for not providing specific documentation, participants mentioned extensive approval processes and sufficiently experienced developers:

> "It sounds like it would be just something to add to the pile because we already have approval processes and usually our developers are at least somewhat experienced and they have seen many things before [...] For other firms, for other companies, it definitely might make sense." (P02)

Reasons for providing documentation included easier onboarding and supporting new developers by providing them with wrapped, documented versions of OSCs: "We document [wrapped OSCs], and we present these in-house made components to our new developers with a good documentation and they know how they operate." (P12) Our participants seemed to agree on the usefulness of having documentation for these components and processes, e. g., P13 said: "But [having some documentation for included OSCs is] a very good point. We should definitely have something like that because it's not always going to be me." (P13)

Overall, our participants appear to provide documentation for included components based on their given requirements and team context, with accessibility for mostly new developers in larger teams on the one side of the spectrum, to experienced developers with other processes in smaller teams on the other side.

> Summary: Policies and Guidance.Most projects had some form of company policy or at least best practices for including external code. Relatively few had dedicated security teams or a disclosure policy. Perspectives on providing documentation appear to depend on the team context, with larger documentation support being seemingly correlated with larger team size and the number of less experienced developers.

5.5.5 Experiences with Open Source Components

Aside from general usage and policies for external code, we were also interested in participants' past experiences with OSCs. Overall, our participants voiced a positive to very positive opinion about their experiences with the open source ecosystem, e. g., "[i]t's great. It's a very vibrant ecosystem and you

have a plethora of options to use," (P16) "I would say pretty good. I like [the open source ecosystem] a lot. It's really easy to have issues and get them resolved," (P23) and "open source technology and components are very attractive. They have to be because if not, you're not going to use them." (P15) Multiple participants mentioned a very important or key role of OSS in the overall software industry, e. g.,

> "I think open-source components or open-source software in general has [a] very important role in overall software development industry. Whether companies are developing commercial solutions or building open-source or free solutions, they play a very important role." (P10)

as well as: "So [the ability to talk with maintainers] is why I feel that they really help us to accelerate our process of development and are pretty much a cornerstone of the software industry today." (P17) A common theme for the usage experience of OSCs was the friendliness and openness of the open source communities, e. g., for questions "['open-source project'] says if we find any question, there is always a community back there to answer whatever we need," (P07) and for better understanding the structure:

> "[...] when we talk to them and try to understand why something is built a certain way or not, most of them will be open to sitting down and having a discussion, and even in cases allowing us to help them change something for the better or for a completely new feature." (P17)

This theme also included the ability to easily file issues (and get them resolved), e. g.,

> "[t]here are always issue trackers where you can flag any problems you have. I think I went into one case where we didn't have to put the issue up ourselves because that was already flagged by other users of the component" (P06)

and "If there is a breaking change, we don't have to think about it, we just have to create an issue on the parent or the maintenance repository and they take care of making this in the next version." (P07) Further positive attributes of OSCs mentioned by our participants included speeding up development of their projects, e. g., "[open source software] has allowed us to develop much quicker or develop applications more quickly using a lot of open source tools as part of the overall application, so I'm a big proponent of open source software" (P19) and overall good code quality: "So I have always particularly really liked the open source industry and what they provide, because if you go to see the code quality in most of these projects it is really good and they do cover a lot of use cases." (P17)

Importance of Documentation. We specifically inquired about the setup experience of open source components, and multiple participants mentioned that they see a good documentation as quite crucial for a good experience when using OSCs, e. g., "I remember when I was junior or new in this open-source side [...] It was quite hard to set up or test or check or find the documentation, etc., but when you get used to it, it's mostly intuitive" (P11) and

> "It depends on how good the documentation is [...] Some people just write terrible documentation or they just don't write it at all. I think depending on how good they are, that can make your experience either very good or very bad." (P23)

Our participants seem to be divided on the actual state of documentation for OSCs, with some negative experiences on the one side, e. g.,

> "[m]y general experience with this technology is that there's a lack of documentation sometimes [...] there's more documentation and examples from the private software because of course, you're paying for the documentation as a client" (P15)

and some positive expressions on the other side, e. g.,

> "[i]f those are popular components, they're usually very comfortable to use because they have examples on their website. They have a little demo version [...] and [you can] even edit the code on it and then just copy and paste it into your project." (P25)

As mentioned by P25, the quality and available documentation for OSCs appears to be often directly correlated with the popularity of the project, with more popular projects likely having more maintainer-hours available for creating documentation.

Customization and Using Components Again. More than half of our participants (14) mentioned that they had to customize an OSC for their projects beyond basic configuration changes, e. g., "Yes. There's been times where contributing or even bringing in our own package, there's been a few times where I've forked and customized the open source repository, too." (P20) Participants also mentioned contributing back some of their customization and improvements to the open source projects, e. g., "We've also contributed back to the code base or the open source project to try to get changes implemented as well." (P19) When asked whether they would select the same components again for a project, 17 responded positively, e. g., "we are using our popular frameworks, our popular open-source components again and again and again. We have already set up a documentation for that." (P12) Of the remaining, six responded somewhat negatively, e. g.,

> "I think we could do with a few dependencies less because they are not really critical and they just add a nice-to-have feature [...] Some dependencies were pulled in that if starting over, I would probably try to avoid." (P05)

Overall, our participants had quite positive experiences with OSCs. Their highlights include, among others, the ability for fast iteration in their projects, the lessened maintenance burden, and the general openness of the communities and code, allowing them to understand, modify, and contribute to open source project our participants utilize in their software. We see promising research venues in what constitutes a high-quality open source documentation and how to best support the customization of components without sacrificing security.

> Summary: Experiences.Our participants mentioned almost exclusively positive experiences, although some highlight the varying quality of documentations. Mentioned positive attributes included the ability to open issues (that get resolved) and the ability to directly talk to maintainers. Most of our participants would select the same components in some form again, given the choice.

5.5.6 Challenges and Incidents

Almost all of our participants (24) reported to have encountered some form of security challenge or annoyance related to OSCs in the past. Our participants mentioned challenges related to updates: "One day, all of a sudden, the system stopped responding because the PHP updates didn't follow in that particular package," (P08) as well as out-dated and potentially vulnerable components. Another common theme was OSCs being no longer maintained or deprecated, e. g., as described by P17: "Yes, dependency is no longer maintained is a big challenge" (P17) but they also highlight their way forward of forking or looking for alternatives: "At that time we will start maintaining it ourselves privately, or else

see if somebody else has started a version two." (P17) Other participants mentioned that they updated their development process when they became aware of prevalent incidents (but were not affected) in the open source ecosystem, e. g.,

> "I believe it is also was a Node.js developer who deleted all their repositories [...] and that is when we implemented cache for everything that we have a local copy for every open source component we are integrating in our build chain to be locally available." (P03)

The participant is likely referring to the "left-pad" incident from early 2016, which involved a maintainer deleting their popular npm packages, including the widely-used "left-pad" package included in, and thus breaking, many other npm packages [304].

Incident Opinion and Strategy. To investigate our participants' strategies for handling trust incidents, we introduced and asked them for their opinion of the March 2022 "node-ipc protestware" incident [77]. In this protestware incident, the JavaScript node-ipc library was updated by the maintainer as protest to Russia's invasion of Ukraine to, depending on library version and IP address, replace all of the user's system files with heart emojis. The majority of our participants had a mostly negative opinion of the incident (16), followed by a neutral opinion (6), and no opinion (3). No participant had a mostly positive opinion of the incident. Negative opinions mostly focused on the potential damage done to trust in the open source ecosystem and the potential to harm bystanders, e. g., "I don't think that's appropriate when we're talking about security and trust [...] I don't really consider that inclusive of all people trying to use open source. Sorry, I don't really agree with that," (P23) the overall malicious look of the changes: "That is just straight up malicious that is a very black hat thing to do, and that should not even have reached a package manager," (P02) as well as the overall damage to OSS's reputation:

> "It's bad for reputation of open-source software, but these things happen in commercial software also. [...] Some people want to use them for, as you said, for protesting purposes and some people want to use them for malicious activities." (P12)

With this recent incident as background, we asked our participants, what they would do if one of their projects depended on this package and how their general strategy for incidents would look like if a component or maintainer lost their trust. Most participants mentioned that they had not encountered something like that before, e. g., "No. We had never had this incident or something like that, so we never thought about what we should do if this ever occurs." (P03) Common strategies for handling such incidents included finding an (open source) alternative, e. g.,

> "If we lose trust in a component, we'd also try to find an alternative. I guess it's a trade off, [if] it's the only alternative and we really need it, then we would have to think about how to make it more trustworthy or maybe contribute upstream," (P05)

or assessing the damage first before taking any further steps, as mention by P12:

> "Try to minimize the bad effects and try to contain the bad effects. Then I can maybe complain about the reputation of open-source software, but my first priority is to go ahead and fix the issue if it affects us." (P12)

Overall, most participants mentioned not having considered or encountered such an incident before. In general, their first strategy would consist of either finding an alternative, stepping up and forking the project, maintaining the project internally, or assessing the damage first before any further steps. Providing tooling and strategies that support developers in handling such incidents present a promising opportunity for both researchers and industry.

Summary: Challenges and Incidents. Almost all of our participants had encountered (security) challenges or inconveniences related to OSCs, often mentioning broken updates and vulnerabilities in (out-dated) components. Our participants had mostly a negative opinion of the "node-ipc protestware" incident, mostly due to harming the trust in the open source ecosystem. Most did not mention a specific strategy for reacting to such incidents and would generally look for alternatives.

5.5.7 Problems and Improvements

In the final question section, we asked our participants what they think the perceived security of their projects is (both by internal and external actors), as well as how they would like to improve the software supply chain security of their projects. Regarding the perception within their team or company, seven mentioned a mostly positive perception regarding their security, four a mostly negative, with the rest reporting either a neutral (5) or no perception. Regarding the perception of external actors (e.g., their clients, their users, or the public), nine mentioned a mostly positive perception regarding their security, zero a mostly negative, and again the rest reporting either a neutral (3) or no perception.

Improving Security. As for suggestions for improving the software supply chain security of their projects assuming no limitations, we roughly sorted our participants' ideas by theme, with the most mentioned including the auditing of their dependency graph and the code of external components (8), e. g.,

> "It would be nice to have independent audits of everything that we use, that way, we can have some level of assurance that at least the software that we're using or components we're using meets some particular standard" (P09)

and in general more developer hours (3) for testing and securing their projects or adopting OSCs, e. g., "[...] I would like to have enough developers that we do not have to go through some of those dependencies which are not highly rated on GitHub or which are nearing the end of their maintenance lifecycle, and be able to develop those in-house." (P17) Other ideas for improvement included hiring a dedicated security team (2): "If I have unlimited money, then a security team would be fine, but that's not a reality in most enterprise" (P18) or establishing a set of best practices and documentations for open source communities (2):

> "I think having some set of best practices out there that is more widely accepted among the open source development community and I guess rigid guidelines in such a way would improve what we use it for and how we use it." (P19)

Another suggestion was the creation of a foundation or entity that could verify the security of OSCs:

> "There should be an entity. Just like there's an Apache foundation, there can be a security foundation that can offer this analysis and certification for the open source if you pay. I can be more comfortable or more confident about the technology that I'm going to propose to my boss or to the client. I can say, hey, this software is open source, but it has already been tested by this other open source foundation, but focus on security." (P15)

Such a recently formed organization is the OpenSSF, which aims to improve open source software security through a collaborative effort, potentially highlighting a need to raise even more awareness for such efforts.

Overall, our participants' ideas for improving the software supply chain security of their projects mostly centered around having more developer-hours or tools to audit included components, as well

as general security checks and pen tests of their projects and OSCs. Providing and enabling the tooling for both auditing and testing OSC provide an opportunity for both researchers and industry going forward.

> Summary: Problems and Improvements.If they had an opinion about it, most participants thought that their projects' security is perceived as positive, both by internal and external actors. For improving the software supply chain security of their projects, participants often suggested manual and automatic audits of code and dependencies.

5.6 Discussion

In this chapter, we qualitatively investigated the role and importance of OSCs in companies and software teams, as well as the related security challenges and consideration, by conducting 25 in-depth interviews with software developers, architects, and engineers to answer the following research questions:

RQ1. *"How are Open Source Components included in companies' tech stacks in terms of position, importance, and security effects?"* Our participants mentioned OSCs in many positions in their projects, including as project components, as foundation and frameworks for their software, and as tools in their development infrastructure. OSCs appeared to play quite important roles in participants' projects, with some reporting using OSC for key features or foundation in their software or development processes. Some even specifically mentioned OSCs and the open source community as an important or key part of their overall software ecosystem. As for security effects, some participants reported updating their development processes in response to news about vulnerabilities in, or the abandoning of, popular open source projects, e. g., "[...] and [the left-pad incident] is when we implemented cache for everything [so] that we have a local copy for every OSC [...]." (P03)

RQ2. *"What are companies' awareness, experiences, and attitudes regarding the security of including external open source code?"* Overall, our participants consisting of software developers, architects, software developers, architects, and engineerss appeared to be quite aware of the security implications of including OSC in their software, although some reported management not allowing or understanding the concept of open source, e. g.,

> "I think the responsible people just didn't understand the whole scope of OSC options that a developer has, because they're mostly managers and legal people, and they don't have so much insight in technical stuff." (P16)

Almost all participants reported positive to very positive experiences with open source code, although all except one mentioned experiencing some form of challenge or inconvenience by OSCs in the past, mostly originating from an unmaintained project, a botched patch, or an upstream vulnerability. Our participants seemed to have somewhat ambivalent attitudes about the security of OSCs, with many mentioning that they would or could only handle incidents from OSC if/when they happen, while their most common security wishes included large-scale audits of their dependencies and OSC projects.

RQ3. *"If and how do stakeholders make decisions and considerations around security and trust challenges of including Open Source Components?"* The decision and selection processes around OSCs reported by our participants appear to span the whole spectrum from purpose-build, in-house components modified by specific teams wrapping and documenting open source projects, to whatever component an individual developer thought right for the job. As for considerations around security, our participants

appeared in general to be optimistic, while still acknowledging the large potential attack surfaces of using external code.

Aside from answering our research questions, we discuss some of the broader themes and our interview-spanning findings in greater detail:

Securing a Bowl of Spaghetti. The "chain" part of the software supply chain analogy lends itself to convey an overall image of linear relations, with clear start (producer) and end (consumer) points, with some additional chain links in-between. But in reality, a better fitting picture for the software supply chain in general, and OSS in particular is that of a giant bowl of spaghetti, with many intertwined strands, impossible to discern beginning and ends, even when closer investigating some string. Some companies in our study tackled this problem by focusing only on the security aspects on their plate, namely by maintaining in-house versions or caches of included OSCs, which separates them from many attack vectors in the whole bowl, and allows them to better check and audit the local components. Promising research venues include both the underlying concepts for maintaining such a software stack separation, as well as the necessary tooling like for static analysis, reproducible builds, and package signing. Based on our findings, our recommendations for industry projects include considering established available approaches like version pinning and including static analysis tools (SATs) in their build pipeline, as well as to evaluate some of the more recently emerging technologies, like SBOM and OpenSSF Scorecards. Other participants mentioned that they see this security and complexity problem more as a journey: "I think that security isn't something that a lot of people, I think even in IT, view security as something that is a destination and not a journey, so to speak. They don't think of it as an ongoing process." (P09)

Community of Communities. Our participants seemed to have quite positive attitudes about OSCs, with many mentioning their software or team benefiting from using them, e. g., through reduced maintenance burden, fast iterations, and open communities and code. This exchange can quickly become one-sided, especially as it is not always feasible for both companies to provide, and open source communities to receive, the most common exchange equalizer in the industry: Money. Promising future research opportunities involve identifying ways to best support both individual open source projects in different growth stages and communities, as well as the open source ecosystem as a whole. Based on our current findings, it might be beneficial for companies to approach the open source ecosystem with the mindset of being just another community among the many different open source communities, instead of treating it as another software supplier. In practice, this could involve the open sourcing of their internal components if feasible, providing guidance and help with issues just as most open source projects, and contributing back if the chance arises. The software industry can also benefit from supporting open source communities in terms of cultivating developer talent: By supporting and enabling open source projects included in their software stack, they allow a world-wide developer community to learn from, and participate in their software stack, allowing a wide group of people the access to industry technologies, allowing them to grow into expert developers. e. g., as it happened to P19:

> "I started programming when I didn't have a lot of money to buy software, so finding free tools on the Internet has always been cost effective for me. Then later on in, I guess, my professional career, it has allowed us to develop much quicker or develop applications more quickly using a lot of open source tools as part of the overall application, so I'm a big proponent of open source software." (P19)

Not Your Typical Supply Chain. Companies treating the open source ecosystem as any other of their (software) supply chains will likely lead to bad surprises for both sides down the line: Companies might need to scramble if open source components they had relied on for years are suddenly abandoned by

the maintainer or don't implement direly needed features, while open source communities might be punished for their openness by being (mis)treated as a cheaper support desk and alternative for in-house development teams. Unlike a company's other (software) supply chains, the open source ecosystem rarely operates based on contracts, and if a company is not able to provide a value exchange equivalent in money for utilized OSCs, they might want to consider offering some of their developer time or code back to the open source ecosystem. Future researcher venues could involve the legal challenges of the open source ecosystem, best approaches for different company types to support or get involved in open source, and how companies could improve their development processes around involved OSC. With industry's great power of utilizing freely available OSCs in their software comes also the great responsibility of keeping the open source ecosystem healthy and secure, or as one of our participants formulated it fittingly: "This is something that we also have in our company policy: Always contribute back." (P07)

5.7 Conclusion

We investigated the use of OSCs in software companies and teams during 25 in-depth, semi-structured interviews with software developers, architects, and engineers. We explored challenges and considerations of software companies and teams around including OSCs in their projects by exploring their behind-the-scene processes, provided guidance and security policies, as well as security challenges encountered in the past and their incident handling. We found that most of our participants' projects had some form of company policy or at least best practices for including external code, with selection and exclusion criteria for OSCs being commonly based on easily visible metrics like activity, number of contributors, or GitHub stars. We also found that most projects contribute in some form back to open source projects, or our participants would at least like to, with some suggesting their management or legal departments do not fully understand the open source ecosystem.

This chapter presented research investigating considerations around OSCs in industry projects by interviewing 25 software developers, architects, and engineers. But the requirement for a secure supply chain and software is not limited to software experts in companies: especially Android app developers, with apps being widely utilized by end users, can benefit from improving the security and protection of their apps. I present research on the use of obfuscation in the Android ecosystem, consisting of a multi-pronged study approach with measurements, a survey, and a programming experiment in the following chapter Large Scale Investigation of Obfuscation Use in Android (Chapter 6).

Chapter 6

Large Scale Investigation of Obfuscation Use in Android

Smartphones have changed society in countless ways, especially by enabling millions of end users the access to applications. But Android applications are frequently plagiarized or repackaged, resulting in security and privacy risks for users. Software obfuscation is a recommended protection against these malicious practices. At the time of this research, there was little prior data and insights on how and why Android apps are obfuscated in practice apart from limited or small-scale studies. This chapter presents a comprehensive analysis of the use of software obfuscation in Android applications with a multi-pronged approach consisting of a large-scale app measurement, a developer survey, and a programming experiment.

As this research project was conducted as a team consisting of me, Nicolas Huaman, Yasemin Acar, Brad Reaves, Patrick Traynor, and Sascha Fahl, this chapter utilizes the academic "we" to mirror this fact. We analyzed 1.7 million free Android apps from Google Play and found that only 24.92% of apps are obfuscated by their developer. To better understand this rate, we surveyed 308 Google Play developers and found that while developers think that apps in general are at risk of plagiarism, they do not fear theft of their own apps. We then conducted a follow-up study where the majority of 70 participants failed to obfuscate a realistic sample app, even though many mistakenly believed they had been successful. The presented findings have broad implications for improving the security of Android apps and for tools that aim to help developers write more secure software.

6.1 Preamble

This chapter presents research that also resulted in the previously published paper "A Large Scale Investigation of Obfuscation Use in Google Play" [18], which appeared in the proceedings of, and was presented by me at, the 34th Annual Computer Security Applications Conference (ACSAC'18) in December 2018.

> **Dominik Wermke**, N. Huaman, Y. Acar, B. Reaves, P. Traynor, and S. Fahl, "A Large Scale Investigation of Obfuscation Use in Google Play," in *34th Annual Computer Security Applications Conference (ACSAC'18)*, San Juan, PR, USA: ACM, Dec. 2018, pp. 222–235

The original abstract for this publication is as follows:

> **Abstract:** Android applications are frequently plagiarized or repackaged, and software obfuscation is a recommended protection against these practices. However, there is very little data on the overall rates of app obfuscation, the techniques used, or factors that lead to developers to choose to obfuscate their apps. In this paper, we present the first comprehensive analysis of the

use of and challenges to software obfuscation in Android applications. We analyzed 1.7 million free Android apps from Google Play to detect various obfuscation techniques, finding that only 24.92% of apps are obfuscated by the developer. To better understand this rate of obfuscation, we surveyed 308 Google Play developers about their experiences and attitudes about obfuscation. We found that while developers feel that apps in general are at risk of plagiarism, they do not fear theft of their own apps. Developers also report difficulties obfuscating their own apps. To better understand, we conducted a follow-up study where the vast majority of 70 participants failed to obfuscate a realistic sample app even while many mistakenly believed they had been successful. These findings have broad implications both for improving the security of Android apps and for all tools that aim to help developers write more secure software.

The paper was published with the following acknowledgements:

Acknowledgements: This work was supported in part by the National Science Foundation under grant numbers CNS-1526718 and CNS-1562485. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

In addition, I would like to express my sincere gratitude to everyone who has contributed to the completion of this project. I would also like to thank the participants again, who generously gave their time and shared their experiences about obfuscation in the Android ecosystem with us. Their willingness to participate made this research possible, and I am deeply grateful for their contributions.

6.1.1 Contribution

The research prresented in this chapter was conducted as a team consisting of me as team lead, Nicolas Huaman, Yasemin Acar, Brad Reaves, Patrick Traynor, and Sascha Fahl. I am grateful for the contributions of each member, which have been integral to the success of this research project. Without their expertise, hard work, and dedication, this research project would not have been possible.

Sascha Fahl, Yasemin Acar, and Brad Reaves came up with the initial idea and iterated it with me. The full team came up with the initial concept and research approach for this research project. I implemented the analysis tooling and storage for the large-scale analysis. Nicolas Huaman and I created the tasks and example apps for the programming experiment. Yasemin Acar, Nicolas Human, and I created the survey guide for the developer survey, and iterated it with the rest of the team. I analyzed the large-scale analysis results. Together with Nicolas Huaman, I qualitatively coded the programming task solutions. I compiled the paper for publication with contributions from the remaining team and we jointly discussed the work's implications. I presented the publication at ACSAC'18 and included it in some of my talks.

6.1.2 Structure

The remainder of this chapter is structured as follows: After a general introduction (Section 6.2), I provide the related work at the time of this research project in 2018 (Section 6.3). I then provide background for the areas of Android obfuscation techniques (Section 6.4) and the detection of obfuscated apps (Section 6.5). I then describe our multi-pronged study approach including a large-scale analysis

(Section 6.6), a developer survey (Section 6.7), and a programming task experiment involving obfuscating an app (Section 6.8). Finally, I discuss our findings (Section 6.9) and provide a summary for this chapter (Section 6.10).

6.2 Introduction

While smartphones have changed society in countless ways, application markets are perhaps an underappreciated development. These markets enable the simple distribution of new software, but they have also enabled numerous studies of application security [163], [164], [305] and provided mechanisms to identify malware before or after infection [306], [307]. Much of this research depends on software analysis techniques, and these techniques face challenges in the presence of *software obfuscation* [148], [159], [308]–[310], software transformations designed to frustrate automatic or manual analysis.

Despite the impacts of obfuscation, to-date there is very little *data* on how Android apps are obfuscated in practice apart from limited or small-scale studies [305], [311]. In this chapter, we present the first holistic, comprehensive analysis of the state of the use of software obfuscation in Android applications. We begin with a study of obfuscation usage (and techniques) on over 1.7 million apps collected from Google Play. We follow this with a survey of 308 application developers about their experiences and perceptions of software obfuscation. We conclude with a development study with 70 professional Android developers to investigate usability issues with ProGuard, which is by a large margin the most popular obfuscation tool for Android. We address three research questions:

RQ1: *How many apps are obfuscated, and what techniques are used?* For researchers who develop automated analysis tools, it is critical to understand how often and what types of obfuscation are commonly applied so they can ensure correct analysis of apps. It is also an important measurement for the Android ecosystem. Software obfuscation is a defense against app repackaging, an abusive practice where applications are cloned and redistributed to build trojan apps or steal ad revenue. App repackaging is an epidemic threat to the entire ecosystem: in recent studies, 86% of malware samples collected were repackaged versions of benign applications [312], and apps are repackaged by the thousands [313], [314]. Up to 13% of entire third party markets consist of repackaged apps [315], [316]. Thus, software obfuscation protects not just individual apps and developers, but also users and the ecosystem at large.

We find that roughly 25% of apps are obfuscated, but that number rises to 50% for the most popular apps with more than 10 million downloads. This is high enough that it would have a significant impact on research – especially for projects that ignore obfuscated apps [161], [317]. However, it is also still low enough to indicate that the vast majority of apps are unprotected.

RQ2: *What are developers' awareness, threat models, experiences, and attitudes about obfuscation?* These factors provide insight into root causes of the low rates of obfuscation in Android. We examine whether developers are aware of obfuscation, whether they have attempted or successfully used obfuscation, which tools they have used, and whether they found the tools were sufficiently easy to use. We find that while developers are aware of the benefits of obfuscating their apps on a theoretical level, a perceived negligible personal impact and the time-consuming use of obfuscation tools for real applications is a large deterrent to obfuscation.

RQ3: *How usable is the leading obfuscation tool ProGuard?* Our developer survey also found that 35% of our participants reported difficulty obfuscating their apps, while over 61% — more than double the Play market average — claim to obfuscate their apps. To better understand this paradox, we asked 70 developers to obfuscate two sample apps. We found that while most developers successfully managed to complete a simple obfuscation task, 78% failed to correctly use ProGuard in a more complex and realistic scenario. Moreover, 38% mistakenly believed they had successfully obfuscated their app. This highlights that even when developers attempt to use obfuscation, tool usability likely has a negative impact on its effectiveness.

We conclude our paper with a discussion of lessons learned and recommendations in Section 6.9. While software obfuscation is by no means a perfect defense against reverse engineering, previous work shows that even simple forms of obfuscation (like identifier renaming) significantly increase the effort

required to successfully reverse engineer software [318], [319]. Additionally, the significant challenges obfuscation presents researchers (as shown in prior work [159], [308]–[310]) make this topic worthy of study. Our focus is on obfuscation used by legitimate applications; we leave the topic of obfuscation of malware for future work.

We note that the implications of this study go beyond the Android ecosystem. In contrast to other secure practices with a variety of costs and trade offs, software obfuscation is in an ideal position for adoption: ProGuard is one of the very few secure development tools in existence that is free, already available in the IDE of most developers, and can automatically enhance security while simultaneously improving performance. *Understanding why developers do or do not use such an ideal tool has broad implications both for the development of better developer support and as a measure of barriers to a more security-conscientious software development community.*

6.3 Related Work

Disclaimer: This related work section reflects the state of prior research in late 2018 and is provided to highlight the state of research at the time of this research project. For related and concurrent work at the time of this dissertation, see Chapter 3: Related and Concurrent Work.

Software obfuscation has been studied as defense against reverse engineering [147], to prevent intellectual property attacks [148], as disguise for malware [149], and to avoid user profiling [150]. Researchers successfully employed code obfuscation techniques to avoid detection tools, including anti-malware software [151]–[153], repackaging detection algorithms [154], and app analysis tools [155], although performance of anti-malware software improved in a more recent study [320]. A number of works detail different obfuscation techniques in general [147], [149], [321], [322], for the Java programming language [323]–[325], and for Android apps in particular [151], [326], [327]. Research on Android app obfuscation has focused on reversing obfuscation [328], [329], analyzing an app in spite of obfuscation [159], [308]–[310], the detection of repacked malware [330]–[333], or identification of third-party libraries [158], [334].

Previous Android developer studies were performed in the context of privacy, Trusted Layer Security/Secure Sockets Layer (Transport Layer Security (TLS)/SSL) security, and cryptographic Application Programming Interfaces (APIs). Balebako et al. performed interviews and online surveys to investigate how app developers make decisions about privacy and security, identifying several hurdles and suggesting improvements that would help user-privacy [167], [168]. Jain et al. suggested design changes to the Android Location API based on the results of a developer lab study [169]. Fahl et al. and Oltrogge et al. conducted developer surveys and interviews, revealing deficits in the handling of TLS/SSL and suggesting several improvements [160]–[162]. Nadi et al. found in a study that Java developers struggle with perceived low-level cryptography APIs [141]. Concerning obfuscation on the Android platform, Ceccato et al. assessed in experiments the impact of Java code obfuscation on the code comprehension of students, finding that obfuscation delays, but not prevents tampering [318], [319]. Pang et al. surveyed 121 developers about their knowledge concerning app energy consumption [335]. Compared to these works, our root cause analysis focuses on obfuscation knowledge and ability to use the obfuscation tool ProGuard among Google Play developers. Similar to recent work on how information sources influence code security [100], we find that developers are generally aware of benefits and basic use, but fail to correctly obfuscate in complex scenarios.

Finally, in a pre-print concurrent with our research, Dong et al. also investigate the use of obfuscation in the Android ecosystem [336]. While that work is solely focused on technical measurements

Table 6.1: Selected features of popular obfuscation software for the Android environment.

		Obfuscation							Other				
Name	License	Package name	Class name	Method name	Field name	Overloading	Debug Data	Annotations	String Enc.	Class Enc.	Optimization	Minimization	Watermarking
Allatori[1,†]	$290	●	●	●	●	●	●	○	●	○	●	●	●
DashO[†]	On request	●	●	●	●	●	●	●	●	○	●	○	●
DexGuard[2,†]	On request	●	●	●	●	●	●	●	●	●	●	●	○
DexProtector	$800	●	●	●	●	○	○	○	●	●	○	○	○
GuardIT	On request	●	●	●	●	●	●	○	●	●	○	○	○
Jack[2,†]	Free	●	●	●	●	○	●	●	○	○	●	●	○
ProGuard[†]	Free	●	●	●	●	●	●	●	○	○	●	●	○
ReDex[2,†]	Free	●	●	●	●	●	●	●	○	○	●	●	○
yGuard[†]	Free	●	●	●	●	●	●	○	○	○	○	●	○

[1] Multiple obfuscation patterns, default can be detected
[2] Mirrors ProGuard's obfuscation with same configuration format
[†] Obfuscation features (partially) detected by Obfuscan

of obfuscation (similar in focus to our Sections 3 and 4), our research works with the developers responsible for obfuscation to determine the root causes of why apps are or are not obfuscated. Our app measurements are more comprehensive (1,762,868 apps from Google Play market vs. 114,560 apps) and use measurement techniques grounded in specifications of the most common obfuscation tools (instead of machine learning approaches).

6.4 Android Obfuscation Techniques

Available obfuscation tools for the Android ecosystem range from free, open-source obfuscation solutions providing only basic obfuscation features such as ProGuard, up to premium tools with high licensing fees such as DexGuard (cf. Table 6.1). Basic obfuscation features include the following:

Name obfuscation. *Package*, *class*, *method*, and *field* names are commonly obfuscated by replacing their original values with meaningless labels. For example, ProGuard implements name obfuscation by generating name replacements using characters from the `[a-zA-Z]` alphabet. Obfuscated names are generated by iterating through the alphabet resulting in the following renaming patterns: `[a, b, ..., z]`, `[A, B, ..., Z]`, `[aa, ab, ..., zz]`, and so on. Allatori and DexGuard build on ProGuard's name obfuscation alphabet and add reserved Windows keywords ("AUX", "NUL"). Some of the tools allow users to add their own word lists to the renaming alphabet.

Name overloading. Obfuscation tools commonly use Java method overloading to assign the same name to methods with different signatures (i.e., different arguments or return types). In addition to using the same name for different methods, method parameters are also renamed to common names.

Source

```
public class Matrix {
  private int M;
  public Matrix(int M);
}
```

Obfuscated

```
public class a {
  private int a;
  public a(int b);
}
```

Listing 6.1: Example code before and after obfuscation with ProGuard.

Debug data obfuscation. Removing debug information like line numbers or method names complicates the reverse engineering of code structures. Obfuscation tools often include means to reverse this information removal to allow for debugging by developers.

Annotation obfuscation. Another information removal feature strips annotations from classes and methods. Annotations provide additional functional context in class bytecode, including annotations for inner classes or methods that contain "throws" statements. Similar to debug information, the removal of class file annotation and the removal of class source file information complicates the reverse engineering of code structures by tracing class attributes.

String encryption. Strings can be encrypted to hide information. A trade-off has to be made between encryption strength and performance impact by decryption. The decrypter has to be provided in the program, making encryption unsuitable to hide sensitive information. Strings are encrypted to deter simple string searches over the code base and hide information about the program flow.

DEX file encryption. The *classes.dex* file can be encrypted to increase the difficulty of decompilation. Decryption of encrypted classes at run time can cause large performance impacts.

6.4.1 Complications for Obfuscation

While the previous section has discussed a number of techniques for transforming software, configuring obfuscation tools for Android is more complicated than merely choosing from the available features. In fact, there are a number of complicating situations that make it difficult or impossible to obfuscate certain pieces of code, and if that code happens to be obfuscated the app can no longer function. These situations for partial obfuscation include classes that need to be accessible from an outside context: the names and class names of native methods and similarly classes that extend native Android classes such as activities, services or content providers should remain unobfuscated in most cases so that the library/system can invoke callbacks.

6.4.2 ProGuard

The free ProGuard enjoys preferential treatment in the Android ecosystem. It is included with the Android SDK and the official Android Studio IDE. In addition, other obfuscation tools inherit most of their functionality from ProGuard; the now deprecated alternative tool chain Jack is configured by ProGuard configuration files and provides ProGuard's obfuscation with reduced options. Similarly, ReDex accepts ProGuard's configuration files and mirrors the renaming functionality closely. DexGuard is a commercial ProGuard extension and utilizes name obfuscation with the same basic functionality as ProGuard, but with some advanced features.

```
-optimizationpasses 5

-dontusemixedcaseclassnames
-overloadaggressively
-printmapping mapping.txt

-keep public class * extends project.Interface
-dontwarn project.example.**
```

Listing 6.2: Example ProGuard configuration. Configuration path is set in the build system, e.g. in a *gradle.build* file.

ProGuard was integrated with the Android Software Development Kit (SDK) in August 2009 and can be activated in the build setup of a project. The "`minifyEnabled`" option activates ProGuard obfuscation for the release build of an app. Additional configuration files can be specified with the "`proguardFiles`" option. In the ProGuard configuration file, different program options are activated/deactivated by setting a number of flags that are relevant to later presented results (cf. Listing 6.2). Some processing steps of ProGuard can be completely disabled with flags such as "`-keep`".

6.5 Detecting ProGuard Obfuscation

To answer "**RQ1:** *How many apps are obfuscated, and what techniques are used?*" we built a tool we call Obfuscan to conduct a large scale measurement study of obfuscation practices. Obfuscan is able to detect a number of obfuscation features in compiled Android binaries. In particular, Obfuscan is able to detect all of ProGuard's obfuscation features and many features of other obfuscation tools (as shown in Table 6.1).

6.5.1 How Obfuscan Works

Obfuscan takes an Android binary as input and analyzes certain parts of the binary to detect specific obfuscation features and outputs the list of all detected features. Obfuscan analyzes package, class, method and field names to detect name obfuscation. To detect method name overloading, Obfuscan analyzes the distribution of obfuscated method names for duplicates and relies on the content of debug entries to detect debug information removal. Annotation removal is detected by analyzing an app binaries for the removal of corresponding class attribute fields. To detect further obfuscation features, Obfuscan relies on the *classes.dex* file format and specific function calls (see below).

6.5.2 Feature Detection

Obfuscan implements many heuristics to detect obfuscation features. For accuracy, many of these are developed deterministically and directly from the ProGuard source code.

For name obfuscation, Obfuscan detects both lower- and upper-case obfuscated names by simulating the obfuscation process of ProGuard and comparing the generated names to the actual names encountered on the app, package, or class level. Obfuscan also considers possible flags such as the usage of mixed-case characters if corresponding strings are detected in the scope. Finally, Obfuscan also looks for instances where tools replace class names with restricted keywords in the Windows operating system utilized by DexGuard and some Allatori configurations. To detect method name overloading,

Table 6.2: Performance of Obfuscan for sample set of 200 APKs. Shown are true positive (TP), true negative (TN), false positive (FP), false negative (FN) predictions, and Matthews correlation coefficient (MCC).

Feature	TP	TN	FP	FN	MCC
Class name obfuscation	98	100	0	2	0.980
Method name obfuscation	99	100	0	1	0.990
Field name obfuscation	100	92	8	0	**0.923**
Method name overloading	99	100	0	1	0.990
Debug information removed	100	100	0	0	1.000
Annotations removed	100	88	12	0	**0.886**
Source files removed	100	100	0	0	1.000

Obfuscan investigates names that follow the obfuscation pattern and occur multiple times on the same class level. Obfuscan detects missing debug information by parsing and storing the entries of the Java *LineNumberTable* which maps bytecode instructions to source code line numbers. Similarly, the removal of the source file data from classes removes information about the source file where the class (or at least its majority) is defined. Obfuscan detects this feature by directly accessing the source file attribute of classes and storing the string content of the attribute. Removal of annotations is detected by Obfuscan by directly accessing and storing the attribute field of classes.

6.5.3 Other Tools

Although we built Obfuscan with a focus on detecting the use of ProGuard, it is able to detect apps that were obfuscated with other tools (cf. Table 6.5). Obfuscan is able to detect apps that were obfuscated using ReDex, Jack and DexGuard name obfuscation using Obfuscan's name obfuscation detection feature since all three tools use name obfuscation patterns that are identical with ProGuard's name obfuscation. Additionally, Obfuscan is able to detect DexGuard's more advanced removal of debug line numbers and annotations obfuscation features. We extended Obfuscan's name obfuscation detection feature to also cover the name obfuscation patterns implemented by yGuard and DashQ. To be able to detect Allatori's non-alphanumeric name obfuscation scheme, we extended Obfuscan and added detection support for restricted Windows keywords such as "AUX" or "NUL".

6.5.4 Evaluation

We implemented Obfuscan in Python and evaluated its efficacy by conducting a lab experiment using 100 real Android applications randomly selected from the F-Droid open source app market. We compiled two different versions of each sample app: One version did not use any means of obfuscation and one version that had ProGuard's name obfuscation for all application scopes, method name overloading, debug information removal, annotation removal, and source file removal enabled. Additionally, we acquired and tested 26 apps obfuscated with DexGuard, an expensive commercial tool, correctly identifying obfuscation in all 26.

Obfuscan correctly identifies nearly all obfuscation features of the 200 APKs dataset with a low false-positive rate and a high correlation coefficient (cf. Table 6.2). We manually investigated false positives and false negatives. Obfuscan falsely detected few class and method names as not obfuscated. In these cases, structures of the app were exempt from obfuscation, e.g., due to classes being marked

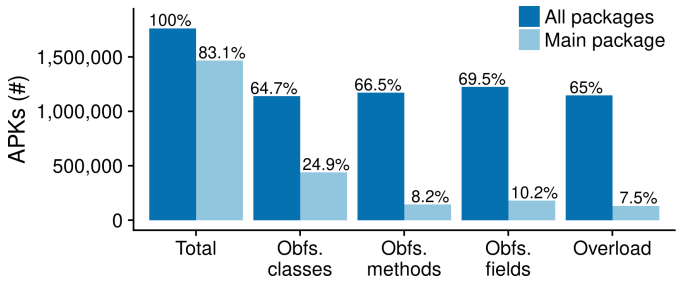


Figure 6.1: Comparison of obfuscation for different app structures including all packages and main package only. Overall obfuscation of apps considering all packages is increased due to library obfuscation.

as an interface. The false positive rate for field names is slightly higher than for other features. This is because ProGuard uses short strings for names (e.g., `a` and `b`) that are sometimes used as variables in unobfuscated apps. Obfuscan had no false positives for the debug information and source files removal feature. However, it falsely detected 12 apps as using the annotations removal feature. These false positives affect apps that do not use the code characteristics that are compiled to annotations (like inner classes).

6.5.5 Limitations

There are several obfuscation features that Obfuscan does not measure. Since Obfuscan focuses on the detection of the benign application of obfuscation, we do not look for packers or other techniques specifically used by malware. We excluded the heuristics for resource name and content obfuscation from our large scale measurement study for performance reasons. We evaluated a test set of 1,000 random apps from Google Play and could not find a single app using these features. Additionally, we did not implement class and string encryption detection. Both are advanced features and DexGuard, DexProtector, or GuardIT provide them as extensions to the more basic name obfuscation features. Finally, Obfuscan focuses on the detection of name obfuscation as implement by common tools. These heuristics conservatively estimate the prevalence of obfuscation at the cost of missing the use of name obfuscation algorithms by less popular tools. However, because Obfuscan reliably detects the removal of debugging information, we believe that this estimates a strong upper bound of the potential uses of other tools that are not ProGuard-related.

Obfuscan's annotation removal detection looks for app packages that do not include annotations. However, this heuristic might mislabel unobfuscated apps that naturally do not use annotations. Since it is hard to estimate the false positive rate for this heuristic, we excluded it from our measurement study in Section 6.6.

To test Obfuscan, we used apps from F-Droid rather than Google Play because we needed source code. While there is a chance that F-Droid apps differ from Google Play apps, this methodology was better than alternatives like writing test apps.

Table 6.3: Top 10 obfuscated libraries by total number of packages and number of APKs containing the libraries. Our analysis considers both main application code and libraries separately to determine the actual rates of use by end developers.

Scope	Packages	**Unique APKs**
com.google.ads.*	1,919,976	**681,102**
com.google.android.gms.*	24,095,920	**651,952**
android.support.v4.*	1,811,806	**192,497**
com.unity3d.*	432,856	**152,668**
org.fmod.*	135,524	**135,524**
android.support.v7.*	992,843	**117,680**
com.facebook.*	1,309,276	**106,178**
com.startapp.*	2,234,609	**88,242**
com.chartboost.*	491,612	**87,781**
com.pollfish.*	537,046	**44,851**

6.6 Large Scale Obfuscation Analysis

With Obfuscan we can answer our "**RQ1:** *How many apps are obfuscated, and what techniques are used?*" Therfore, we analyzed 1,762,868 [1] current free Android apps from Google Play to investigate the real-world use of the ProGuard family of Android obfuscation tools. To the best of our knowledge, this is the largest obfuscation detection analysis to-date for Android applications. Of those applications, Obfuscan detected the renaming obfuscation pattern implemented by the ProGuard family of obfuscation tools (cf. Section 6.4) in 1,137,228 (64.51%) apps.

Main Application Code Obfuscation. The high percentage of apps with obfuscated code would seem to indicate that many developers are obfuscating their apps. However, this statistic is misleading because a large percentage of apps are not intentionally obfuscated by the original developer. Instead, many apps simply include third-party libraries that use obfuscation, and the presence of an obfuscated library does not indicate that core application code is obfuscated. This fact means we need additional analysis to determine how often developers are actually obfuscating their apps.

To distinguish between apps that are obfuscated by their developer and apps that simply include obfuscated libraries, we analyze the obfuscation used by the declared main package of the application[2]. The main package is used as the universal identifier of the application (e.g. *com.google.maps*) and is necessarily implemented by the developer, so a choice to obfuscate the main package strongly indicates a choice to obfuscate at least some (if not all) of the original application code. We note that determining whether code is from a library or written by the developer is non-trivial, and this approach has the advantage of being scalable to millions of apps while not relying on potentially incomplete lists of libraries [334].

Our main package analysis found that only 24.92% of apps (439,232 apps) are intentionally obfuscated by the developer. In other words, *the vast majority of apps — representing millions of man-hours of development — are not protected using ProGuard as recommended for use in the official Android developer documentation [337].*

Obfuscation in Libraries. To get a better understanding of the included libraries in the Android ecosystem, we investigated the names of Android packages in all apps. Android packages follow Java naming conventions, allowing for the identification of larger scopes (e.g. the *com.google.ads.interactive-*

[1] All free Android applications we were able to download from our geographic location.
[2] This distinction of main package vs. other packages was also performed by Linares-Vásquez et al. [310]

media.v3.api package can be traced to the *com.google.ads.** scope). Examining the included packages, we find that most of the external library obfuscation stems from a few, popular library frameworks (cf. Table 6.3). Examples include the Google Ad framework and the Google Mobile Service (GMS) framework used for Google authentication and search. Other commonly included obfuscated frameworks include the Facebook integration library and the FMOD audio playback library. The presence of these very popular libraries explains why many applications have obfuscated code, yet so few main packages are obfuscated.

Obfuscation Feature Popularity. Obfuscan provides the ability to examine use of individual ProGuard obfuscation features, and the use of name features for both entire applications and main packages only is shown in Figure 6.1. The "all package" category is measured as the number of apps containing any package with the obfuscation feature. This includes all libraries and the declared main package. The "main package" category is the number of apps with the obfuscation feature considering only the app's main package. We note that percentages of features used in the main package results are only among those apps with code in the main package.

We see first that class name obfuscation is the most popular feature, with 64.7% of all packages and 24.9% of main packages using it. Looking at other features shows a marked difference in feature use between libraries and main packages. While features that obfuscate method names, field names, and exploit function name overloading are used about as often as class name obfuscation in the all package analysis, they are infrequently used in main packages. One explanation is that library developers have a greater incentive to protect proprietary or sensitive internal APIs.

Overall, our findings indicate that the vast majority of app developers do not obfuscate their core code, and even when they do they do not use all of the available features. These results might indicate that developers either only obfuscate critical parts of their application or do not fully understand the concept of obfuscation.

Non-Proguard Obfuscation. While Obfuscan comprehensively covers features used by ProGuard, it also provides information about other forms of obfuscation. First, apps that do not contain debug info or source files are likely obfuscated, and so looking for those characteristics provides an upper bound on the number of apps in our dataset that are obfuscated by any non-ProGuard tool. As shown in Figure 6.1, we find that between 7.4 and 7.5% of apps in our data have these features for the main package, while between 11.7 and 13.2% of apps have these features for any class in the application. Additionally, we found 2,799 (0.16%) apps that use the advanced obfuscation feature of replacing class names with restricted keywords of the Windows operating system (e.g. "AUX", utilized by DexGuard and some Allatori configurations). By analyzing *classes.dex* files, we found 794 (0.05%) apps that were obfuscated with DexProtector and 207 (0.01%) apps obfuscated with Bangcle. Ultimately, these results together allow us to conclude that ProGuard is far more popular than any other obfuscation tool. This is because the classes using ProGuard-style name obfuscation greatly outnumber the scrubbed debugging or source files, which provide an upper bound on all other obfuscation tools.

6.6.1 Obfuscation Trends

By comparing our obfuscation findings with Google Play metadata for all analyzed apps, we can develop further insights into the use of obfuscation in Android. In this subsection, we consider an app "obfuscated" if classname obfuscation is used, as this is the most common obfuscation feature supported by most obfuscation tools. As before, we distinguish between "all packages" and "main packages" for our analysis. We investigate the following trends in app obfuscation: main package obfuscation rate in

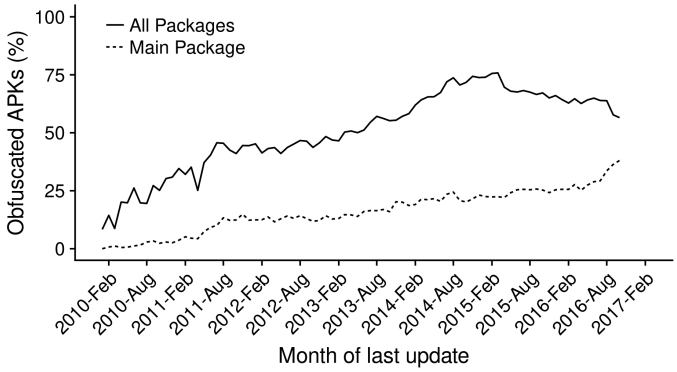


Figure 6.2: On average, more recently updated apps are more likely to be obfuscated.

Table 6.4: Distribution of main package obfuscation for different download counts. More popular apps have a higher rate of main package obfuscation.

Download Counts	Total Apps	**Obfs. Main Package**
0+	115,683	**27.30%**
10+	343,652	**26.34%**
100+	499,018	**24.74%**
1,000+	383,046	**24.13%**
10,000+	234,213	**23.95%**
100,000+	80,302	**25.50%**
1,000,000+	16,335	**29.15%**
10,000,000+	1940	**36.80%**
100,000,000+	160	**50.00%**

relation to download numbers; average main package obfuscation by number of apps per developer; and obfuscation by app update date.

App Popularity. Google Play apps range from rarely downloaded side projects to popular and complex apps with dozens of developers and millions of installs. Hence, different apps will have different incentives to obfuscate their code. We hypothesized that popular apps would be more likely to obfuscate their code as these apps are often more sophisticated and complex and face the greatest risks of plagiarism. To test this hypothesis, we examine the obfuscation rates for each download count category reported by Google Play.

Table 6.4 shows these results. We find that most apps — the 98.9% (1,655,914 apps) of apps with less than 1 million downloads — are obfuscated at roughly the same rate, ranging from 23.9% – 27.3%. As download counts increase further, we see an increase in obfuscation in the most downloaded apps from 29.15% of apps with more than one million downloads to 50.0% of apps with more than 100 million downloads. While this does confirm our initial expectation, we were surprised that even the most popular apps are only obfuscated on average half of the time.

Obfuscation by Google Play account. We also investigated if the number of published apps per Google Play account plays a role in the decision to obfuscate apps. Our hypothesis was that accounts with more submitted apps either belong to experienced developers or even companies specialized in app development and that apps from these accounts would show a higher obfuscation rate either due to a higher awareness or even previous experience of intellectual property theft or due to a higher perceived investment.

Table 6.5: Average main package obfuscation for number of apps by Google Play account. Accounts with more apps have a higher average rate of main package obfuscation.

Apps per Account	Unique Accounts	**Avg. Obfs. of MP**
1	311,908	**21.83%**
2+	155,220	**21.24%**
10+	27,397	**26.50%**
100+	642	**34.37%**
250+	112	**35.29%**
500+	36	**68.41%**

Table 6.5 shows the results. We find that apps from accounts with less than 100 apps have roughly the same average obfuscation rate between 21.8% – 26.5%. For accounts with 100 or more submitted apps this increases to about 35% and even to 68.4% for accounts with 500 and more apps. This increase in average app obfuscation seems to confirm our hypothesis that experienced developers or specialized companies with a large number of submitted apps use obfuscation more often. A likely explanation for this could be that more experienced developers and companies want to protect their intellectual property further. This could be the results from previous experiences of intellectual property theft, or the result of placing a higher value on their apps, as they are likely an important source of income for professional developers and specialized app companies.

Update Date. Figure 6.2 shows how all package and main package obfuscation rates vary when compared to the month of their most recent update; recent updates on average imply frequent maintenance of apps [338].[3] ProGuard is distributed with the Android SDK starting August 2009. The base ProGuard name obfuscation algorithm remained functionally unchanged, allowing Obfuscan to detect obfuscation for all included apps over the study period.

The figure shows a clear upward trend for both all packages and main packages, though as seen previously the overall obfuscation rate for all packages is much greater than main package obfuscation rate. More recently updated apps are more likely to be obfuscated as well. This could be indicative of greater developer sophistication or greater investment in terms of development time and intellectual property. In any case, it is clear that more recently updated apps are more likely to be obfuscated, though still at a low rate overall.

RQ1 Conclusions: This section addressed our **RQ1:** *How many apps are obfuscated, and what techniques are used?*. We found that a significant minority of apps are obfuscated by developers (24.92%), though obfuscated libraries are present in far more apps (64.51%). We also found that ProGuard was overwhelmingly the most popular obfuscation tool. Although these numbers a low compared to an ideal high rate of adoption, they are high enough that software tools and research should be compatible with obfuscated apps.

6.7 Developer Survey

To answer our second research question, "**RQ2:** *What are developers' awareness, threat models, experiences, and attitudes about obfuscation,*" we conducted an online survey of Android developers covering their obfuscation experience, the tools they use and their general knowledge and risk assessment concerning obfuscation and reverse engineering. We asked them if they had heard of obfuscation, if they

[3] Unfortunately, our data collection only allowed us to collect the most recent data on an application, preventing us from getting ground truth on the changes in obfuscation of individual apps over time.

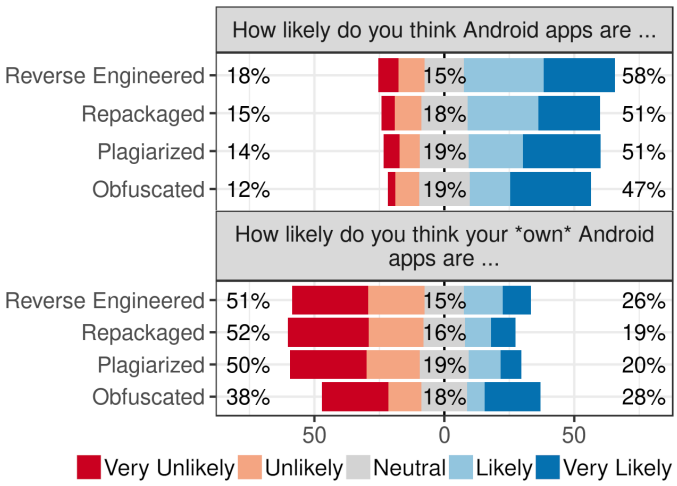


Figure 6.3: Likert plots of questions on risks of apps in general and risks to the participant's apps show that developers see themselves at much lower risk than the "average" app.

knew what it was, if they had ever used it or decided against using it, and why. Additionally, we measured their awareness of "repackaging," "reverse engineering," "software plagiarism," and "obfuscation". We asked how strongly they feel that apps in general and their own apps in particular are threatened by the first three concepts. We followed this with a set of general questions about their Android development practices.[4] In this section, we briefly discuss the design of this survey as well as the results. The online study was approved by the Institutional Review Boards of all involved universities (See Section 6.9.1 for more details).

Depending on participants' prior answers, we asked up to three free text questions, the results of which we analyzed by using open coding with two researchers, developing an initial codebook and refining it iteratively, using it independently on the answers and resolving all conflicts with the help of a third researcher [289].

6.7.1 Recruiting

We collected a random sample of 62,462 email addresses of Android application developers listed in Google Play. We emailed these developers, introducing ourselves and asking them to take our online survey. A total of 561 people clicked on the link to our survey, visited our website and agreed to the study's consent form. Of these 561, 186 dropped out before answering the first question; another 67 participants were removed for dropping out later during the survey or providing answers that were nonsensical, profane, or not in English. Results for our survey are presented for the remaining 308 valid participants.

As common for developer studies [100], we compared participants to the larger population from which we sampled: we compared metadata of 3,159 Android apps associated with our survey participants to the metadata of 1.1M free and paid applications associated with the 62,462 email addresses to which we sent survey invitations (shown in Figure 6.4).

We found a close resemblance in download counts per app (mean invited: 5.75, mean participated: 5.89, category 5 corresponds to 100–500 downloads, category 6 to 500–1,000 downloads), the average user rating (mean invited: 3.07, mean participated: 3.29) and the date of the last update as a measure

[4] Full questionnaire included in the appendix

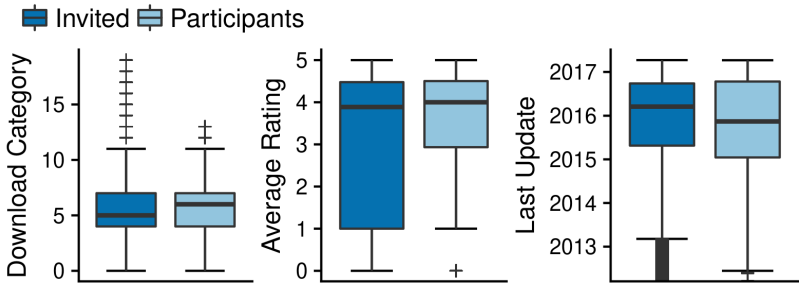


Figure 6.4: App metadata associated with invited email addresses compared to metadata from our participants: We find a close resemblance in download category (as classified by Google Play), ratings and currentness of last update. We compared the distributions using Mann-Whitney-U tests, but our results were inconclusive due to a number of outliers. Nonetheless, we observe similar interquartile ranges: while the invited population leans to being more spread out than our participant population, the populations are similar in median and mean with the invited population having heavier tails.

of app age and long-term developer support (mean invited: 2015-11-18, mean participated: 2015-09-01). These similarities suggest that the developers who opted into our survey study resemble the random sample of Google Play Android developers.

6.7.2 Results and Takeaways

Obfuscation Experience

We found that the majority (241, 78%) of our participants had heard of software obfuscation in general, while 210 (68%) knew about obfuscation techniques for Android in particular. 187 (61%) had considered obfuscating one or more of their applications, of whom 148 (48%) actually did obfuscate one or more applications. While the majority of developers had heard of reverse engineering (253, 82%), software plagiarism (201, 65%) and software repacking (189, 61%) and felt that Android applications in general were severely threatened by plagiarism and malicious repacking, they had the impression that their own apps were less likely to face those threats than apps "in general". Figure 6.3 shows a Likert plot of these responses.

Reasons to obfuscate

The following results are reported for 101 developers who voluntarily specified reasons for using obfuscation in a free text answer. 63 developers (62.3%) used obfuscation to protect their intellectual property against malicious reverse engineering and theft. Interestingly, 14 (13.9%) participants used ProGuard only because it came pre-installed with Android Studio and was easy to use. 18 (17.8%) participants needed ProGuard's optimization features and stated that adding obfuscation was trivial. 4 (4%) participants apparently misunderstood the concept of obfuscation and enabled ProGuard to provide their users additional security, similar to encrypting files or using secure network connections. 7 (6.93%) configured obfuscation because there was a policy (either given by the company they worked for or a customer) that required it.

Verifying that obfuscation works

The following results are reported for the 69 participants who gave a free text answer on their method of verifying the success of obfuscating their app. 48 (69.6%) developers verified the correct use of obfuscation by decompiling the application and manually looking for obfuscation features (e.g., obfuscated package, class or method names). Six (8.7%) participants relied on the Android Studio toolchain and interpreted no warning or error messages as successful obfuscation. Four (5.8%) participants checked their apps' logfiles to verify their obfuscation. Finally, six (8.7%) other participants verified obfuscation by comparing the size of the non-obfuscated with the obfuscated version of an application.

Reasons to not obfuscate

Out of the 185 developers who gave reasons to not obfuscate in a free text answers, 81 (54.8%) thought about obfuscation and then decided against using it because they saw no reason to protect their application(s) against malicious reverse engineering, either because they open sourced their applications (17) or included no valuable intellectual property (64). 52 (35%) participants tried to use obfuscation and gave up because they felt overwhelmed by ProGuard's complexity. They could not get third party libraries working or had other issues such as non-working JavaScript interfaces. Five (3.2%) tried to understand the concept of obfuscation but failed. Eight (5.8%) participants mentioned company policies that did not allow them to obfuscate code. However, no one elaborated on those policies in more detail.

Use of Obfuscation Tools

Furthermore, 148 participants gave details on the obfuscation tools they had used. Most of them (127, 85.8%) had used ProGuard. 12 participants (8.1%) used the Jack toolchain[5]. 11 participants (7.4%) used DexGuard and 6 participants (4%) used ReDex. 4 participants mentioned other less popular obfuscation tools with only one appearance, like an obfuscation tool built into the Unity engine. Overall, 144 (97.3%) of the participants had used ProGuard or similar tools.

RQ2 Conclusions: This section addressed **RQ2:** *What are developers' awareness, threat models, experiences, and attitudes about obfuscation?* We found that survey participants are aware of obfuscation, but estimated the risk to their own apps as low. Many participants noted that obfuscation was simply not worth the extensive effort.

We also learned that many Android developers suffer from misconceptions (e.g., using obfuscation to secure network connections) and seem to be overwhelmed by using obfuscation correctly (e.g., the inability to obfuscate an app, but exclude certain components from obfuscation). Generally, we also observed the lack of a threat model: one participant explicitly stated "I wasn't sure my apps would be even popular enough so that someone would bother to copy them. If they would get popular, I'd release an update with obfuscation on." Many developers did not see a reason to obfuscate their own app(s) despite being aware of an abstract risk. One participant explicitly spoke of their experiences with piracy, stating "I see it as highly unlikely, that someone is actually interested in reverse engineering my code. However, I have encountered several fraud cases as an Android developer. All consisted of minimum reverse engineering efforts, i.e. people decompiled my app, changed the advertising ID code, repacked it, and published it under a different name." We find that the lack of concrete threat models explains a low motivation to obfuscate; to obtain a better understanding of the barriers to obfuscation,

[5]The Jack toolchain was deprecated in March 2017 (cf. https://android-developers.googleblog.com/2017/03/future-of-java-8-language-feature.html)

we decided to investigate the usability issues mentioned by a substantial number of participants in depth.

6.8 Obfuscation Experiment

The large scale measurement study and developer survey described above raised an interesting paradox: Roughly 50% of our survey participants claimed to have tried obfuscation in the past, but only 25% of the apps in our measurement study were obfuscated. We hypothesized that this discrepancy may be explained by the fact that developers may *attempt* obfuscation, but be unsuccessful due to difficulties in using their obfuscation tool. To test this hypothesis that the leading obfuscation tool might suffer from *usability problems*, we conducted an online experiment to investigate how developers interact with the ProGuard obfuscation framework. This study addresses our **RQ3:** *How usable is the leading obfuscation tool?*

6.8.1 Study Design

We designed an online, within-subjects study to compare how effectively developers could quickly write correct, secure ProGuard configurations. Again, we recruited developers with demonstrated Android experience from Google Play. Participants were assigned to complete a short set of Android obfuscation tasks, using ProGuard. All participants completed the same set of two ProGuard tasks. After finishing the tasks, participants completed a brief exit survey about the experience. We examined participants' submitted ProGuard configuration for functional correctness and security. The study was approved by our institutions' ethics review boards (see Section 6.9.1 for more details).

We chose to use ProGuard as the obfuscation tool for our experiment because it is pre-installed with Android Studio, the standard IDE for Android app development, and also because our online survey participants overwhelmingly used ProGuard.

Recruitment and Framing

Similar to our survey, we recruited Android developers from Google Play to participate in our developer study. We emailed 91,177 developers in batches, asking them to volunteer for a study on how Android developers use ProGuard to obfuscate apps. We did not mention security or privacy in the recruitment email. We assigned each invitee a unique pseudonymous ID to link their study participation to Google Play metadata without directly identifying them. Recipients who clicked the link to participate were directed to a page containing a consent form. After affirming they were over 18, consenting to the study, and indicating comfort completing a study in English, they were introduced to the study, given access to an Android Study project containing our skeleton app and instructions (including screenshots) on how to import it and set it up. We also provided brief instructions about the study infrastructure, which we describe next.

Experimental Setup

After reading the study introduction, participants were instructed to work on the tasks themselves. Our aim was to have developers write and test ProGuard configurations. We wanted to capture the ProGuard configuration and the Android application code that they typed. To achieve this, we prepared a Gradle-based Android application development project for Android Studio as a skeleton, compressed

the project to a zip file, and provided a download link. We asked participants to download the zip file, import the project into their Android Studio development environment, work on the tasks, put their solutions in a new zip file, and upload this file to our study server. After uploading the solution zip file, we provided a link to the exit survey that allowed us to connect the ProGuard solutions to the survey responses.

6.8.2 The Tasks

To investigate possible usability issues with ProGuard, we asked participants to use ProGuard to complete two obfuscation tasks on the skeleton app we provided in the zip file.

We designed tasks that were short enough so that uncompensated participants would be likely to complete them before losing interest, but sufficiently complex to offer insights into the usability of ProGuard. Most importantly, we designed tasks to model real world problems that Android developers using ProGuard could reasonably be expected to encounter. We chose both tasks after investigating ProGuard centered StackOverflow discussions and GitHub repositories. Both tasks are amongst the most popular ProGuard related discussions on StackOverflow and represent the most popular modifications in ProGuard configuration files on GitHub.

For each task, participants were provided with stub code and some commented instructions. These stubs were designed to make the task clear without providing too much scaffolding and to facilitate our evaluation. We also provided Android application and ProGuard code pre-filled so participants could test their solutions.

Task 1 - Configure: The first task required participants to activate ProGuard within the default Gradle configuration file. The goal was to fully obfuscate the Android application. Participants were asked to solve this task so we could investigate their ability to complete a basic ProGuard configuration. Possible errors include the inability to activate obfuscation or a misconfiguration of ProGuard that disables obfuscation.

Task 2 - Obfuscate and Keep: The second task required developers to configure ProGuard to obfuscate one specific class (*SecretClass*) of our app, while keeping a second class (*OpenClass*) and its function (*doStuff()*) unobfuscated. To solve this task, developers were expected to use ProGuard's "`-keep`" flag for the *OpenClass* class.

The challenge for this task was to correctly use the "`-keep`" flag. Depending on the specified arguments, developers could potentially leave the *SecretClass* unobfuscated or obfuscate *OpenClass* instead.

Exit Survey

Once both tasks had been completed and the zip file was uploaded, participants were directed to a short exit survey.[6] We asked for opinions about the completed tasks, their assessment of their configurations for both tasks, general questions related to obfuscation and reverse engineering, and their previous experience with ProGuard and other Android obfuscation tools.

Evaluating Solutions

Once participants submitted solutions, we evaluated their correctness. Every solution was independently reviewed by two coders, using a codebook prepared ahead of time based on the official ProGuard configuration documentation. Differences between the two coders were reconciled by a third coder.

[6] We used LimeSurvey for this; the full questionnaire is available in the Appendix.

We assigned correctness scores to valid solutions only. To determine a correctness score, we considered several different ProGuard parameters. A participant's solution was marked correct (1) only if their solution was acceptable for every parameter; an error in any parameter or a parameter that weakened the ProGuard configuration security resulted in a correctness score of 0. To assess the correctness of Task 1, we evaluated the Gradle and ProGuard flags in participants' solutions. Whenever participants enabled ProGuard using both the "`minifyEnabled true`" and "`proguardFiles proguard-rules.pro`" options in the configuration file, we rated the solution correct. Solutions that did not specify one of these options or included the "`-dontobfuscate`" flag were rated incorrect.

For Task 2 correctness, we evaluated whether participants enabled obfuscation for the *SecretClass* class and its *doSecretStuff()* method but left the *OpenClass* class and its method *doStuff()* unobfuscated. Similar to Task 1, we required participants to enable obfuscation by using the "`minifyEnaled true`" and the "`proguardFiles proguard-rules.pro`" options. Additionally, correct solutions had to specify one of the following options "`-keep`", "`-keepclassmemebers`", "`-keepclasseswithmembers`", "`-keepnames`", "`-keepclassmembernames`", or "`-keepclasseswithmembernames`" for both the *OpenClass* class and the *doStuff()* method without including the *SecretClass* and its *doSecretStuff()* method. Solutions that did not meet these criteria were considered incorrect.

6.8.3 Results and Takeaways

In total, we sent 91,177 email invitations. Of these, 999 (1.9%) requested to be removed from our list, a request we honored.

766 people clicked on the link in the email. Of these, a total of 280 people agreed to our consent form; 202 (72.1%) dropped out without taking any action. We received zip files from the remaining 78 participants. We excluded eight submissions from further evaluation: one participant submitted a broken zip file, five submitted zip files without a ProGuard configuration file included, two submitted unmodified ProGuard configuration files.

The remaining 70 participants proceeded through at least one ProGuard task; of these, 66 started the exit survey, and 63 completed it with valid responses. Unless otherwise noted, we report results for the remaining 63 participants, who proceeded through all tasks and completed the exit survey with valid responses. Almost all (60, 95%) of our participants had heard of the concept of software obfuscation before, and 54 (85%) had been using ProGuard at least for one Android application in the past.

Most participants (49, 77%) mentioned an abstract threat of reverse engineering or malicious repackaging for Android applications in general. However, similar to the online survey we conducted in Section 6.7, only a small number of participants estimated a high risk for malicious repackaging for their own app(s).

Surprisingly, all of the 70 participants who changed the configuration for Task 1 submitted a correct solution by adding both the "`minifyEnabled true`" and "`proguardFiles proguard-rules.pro`" options.

Task 2 was correctly solved by only 17 (22%) participants, all of whom correctly solved Task 1 as well. Of the 53 incorrect solutions for Task 2, 30 solutions did not include the *-keep* option for the *OpenClass* class. These mistakes resulted in obfuscated classes that should be kept unobfuscated. 17 of the 53 incorrect solutions did include the *-keep* option but misspelled the package name for the *OpenClass* class. Six of the 53 incorrect solutions included the wildcard option for class names which disabled obfuscation for the *SecretClass* class.

41 of our participants rated their solutions as correct. However, only 11 of them actually submitted correct solutions for both tasks. Overall, 52 participants self-reported previous experience with

ProGuard of which 13 correctly solved both tasks. Only one of the 11 participants with no previous ProGuard experience was successful.

RQ3 Conclusions: This section addressed our **RQ3:** *How usable is the leading obfuscation tool?* We found that all participants, regardless of their experience with ProGuard, were able to solve the trivial task to obfuscate the complete app with ProGuard. However, we found a low success rate for the task that required more complex configuration, which substantiated the usability problems mentioned in our developer survey. Being unfamiliar with ProGuard use essentially disqualified participants from configuring partial obfuscation. Critically, participants were unable to verify whether ProGuard had been configured correctly and whether it obfuscated successfully. These results underline a critical usability problem with ProGuard that likely contributes to low obfuscation rates in the wild.

6.9 Discussion

Security through insignificance? Our large-scale analysis showed that the majority of developers do not take basic steps to protect their apps. Even for the most popular apps with upwards of 10,000,000 downloads, who are high risk candidates for obfuscation-related threats, the intentional obfuscation percentage remains below 50%. In our studies, participants assigned a low risk of obfuscation-related attacks to their apps while assuming a greater risk for the whole app ecosystem. Through provided write-ins we learned that many developers perceive their apps as too insignificant to ever fall prey to intellectual property theft or plagiarism. This "security through insignificance"-approach could prove fatal to the increasing number of small developers in the Android ecosystem.

Optional obfuscation: In addition to low initial motivation, the complexity of correctly using obfuscation further contributes to developer unwillingness to obfuscate. Cryptic error messages and confusing documentation do not increase motivation. Perhaps as a result, a certain mind-set seems to have contributed further to the rejection of obfuscation: some participants voiced concerns that obfuscation would destroy their "completed" applications. This view of obfuscation usage as an optional — not essential — development practice could play a larger role in hampering the acceptance of software obfuscation among developers.

Recommendations: Our findings indicate that there are two critical problems preventing widespread adoption of obfuscation in the Android ecosystem. The first is technical, and may have a technical solution: ProGuard is difficult to use correctly. We believe that it may be possible to automatically detect complicating factors (like WebView use) and automatically generate valid ProGuard configurations for developers. If successful, this would allow obfuscation to be enabled by default within Android Studio and other development environments. The second problem is that developers are not motivated to deploy obfuscation given a low perceived risk and high perceived effort. Developers also view obfuscation as an optional, possibly "app destroying" step instead of an integral part of the build process. While improved interfaces and automation for obfuscation may improve the perceptions of effort, more research and education regarding the risks of plagiarism is needed. A technical solution may take the form of new obfuscation techniques or obfuscations applied by the market instead of relying on developers to protect themselves, their users, and the ecosystem at large.

6.9.1 Ethical Considerations

We conducted two user studies in the context of this research. Both the survey presented in Section 6.7 and the developer study in Section 6.8 were approved by the Institutional Review Board (US) and ethical review board (Germany) of all involved universities. Additionally, the strict data and privacy

protection laws in Germany were taken into account for collecting, processing and storing participants' data. Our user studies were targeted towards Android developers who had made their app public by offering it on Google Play. For ecological validity reasons we decided against recruiting local computer science students. To reach this rather specific group of Android developers, we gathered email addresses from developers who had published apps on Google Play from their public Google Play profiles. We selected a random sample and emailed them an invitation to one of our studies (This participant recruitment procedure is in line with work by Acar *et al.* [100]). Our invitation email included a link to our website, where they could access information about the purpose of our research, a consent form that explained how participant data would be used and a contact form. The email further included a link to be blacklisted; hashes of the blacklisted email addresses are shared across several research groups participating in similar developer studies.

6.9.2 Threats to Validity

In this section, we detail issues that may have affected the validity of our results and the steps we have taken to ensure that our results are as accurate as possible.

App Analysis. Our dataset of 1.7 million apps was downloaded from public accessible Google Play Android apps. This is a common methodology, and like all similar studies we run the risk that paid apps or apps in other markets have different properties. These populations (paid apps in particular) may have additional incentives to obfuscate. However, we believe that the high overlap of apps that are available as both free and paid apps, and identical apps available in multiple markets, minimizes this risk.

Our choice of measuring main package obfuscation is not perfect; it is possible that a developer does not obfuscate the main package but obfuscates the remainder of the app. To estimate the frequency of this practice, we examine how many apps without main package obfuscation have obfuscated packages that do not have multiple occurrences in the overall dataset. We found that only 22,868 apps (1.30% of all apps in the dataset) meet this criteria. This establishes an upper bound on the error of this heuristic. We note that an alternative approach to main package analysis would have been to remove third-party library packages after identification with obfuscation-resistant library detection tools such as LibRadar [158], LibScout [334], or LibD [157]. This whitelist approach to package filtering would by design miss new or rarely used libraries, so we opted for the conservative approach of main package analysis.

Online Survey and Developer Study. As with any user study, our results should be interpreted in context. We chose an online study because it is difficult to recruit "real" Android application developers (rather than students) for an in-person lab study at a reasonable cost. Conducting an online study resulted in less control over the study environment, but it allowed us to recruit a geographically diverse sample.

Because we targeted developers, we could not easily take advantage of services like Amazon's Mechanical Turk or survey sampling firms. Managing online study payments outside such infrastructures is very challenging; as a result, we did not offer compensation and instead asked participants to generously donate their time. As might be expected, the combination of unsolicited recruitment emails and no compensation may have led to a strong self-selection effect, and we expect that our results represent developers who are interested and motivated enough to participate. However, as the recruitment in Figure 6.4 demonstrates, while our participants have higher average app ratings, the sample represents Google Play developers both in app popularity and frequency of updates.

In any online study, some participants may not provide full effort or may answer haphazardly. In this case, the lack of compensation reduces the motivation to answer non-constructively; unmotivated participants typically do not opt in to the study. We attempted to remove obviously low-quality data (e.g., responses that are entirely invective) before analysis, but cannot discriminate perfectly.

6.10 Summary

This paper presents the first comprehensive evaluation of the state of software obfuscation for benign Android applications. We built Obfuscan to analyze the use of obfuscation in 1,762,868 free Android applications available in Google Play. Our investigation reveals that 439,232 were obfuscated by their developers, leaving more than 75% unprotected against malicious repacking. In an online survey with 308 Google Play developers, 78% of the participants had heard of obfuscation while only 48% actually used software obfuscation – more than 85% of the participants used ProGuard – in the past. Interestingly, the majority of the participants recognized that software obfuscation in general is a laudable approach to protect against malicious repackaging. However, only few of them saw a reason to protect their own apps. Finally, in a within-subjects study with 70 real Android developers, we learned that 78% of the participants could not correctly complete a realistic ProGuard obfuscation task. Participants who self-reported no previous experience with ProGuard had a negligible chance to correctly obfuscate the study application beyond the trivial option to obfuscate it entirely.

Overall, our studies show that the current use of software obfuscation for benign Android applications leaves manifold challenges for future research. We find that both misconceptions about software obfuscation many of our participants suffered from and the challenges in using ProGuard correctly seem to be the root cause for the low adoption rate of software obfuscation in the Android ecosystem. Hence, future research needs to find more effective ways to make the concept and relevance of software obfuscation concepts accessible to Android developers and should work on more usable software obfuscation tools.

This chapter presented research on the use of obfuscation in the Android ecosystem, consisting of a multi-pronged study approach with measurements, a survey, and a programming experiment. Aside from software experts themselves, end users can provide important insights into their perceptions and reasoning, allowing developers and administrators to match their approaches accordingly. I present research involving a survey with 200 cloud office users from Germany and the U.S., investigating their experiences and perceptions of cloud office suites in the following chapter Security & Privacy Perceptions of Cloud Office Suites (Chapter 7).

Chapter 7

Security & Privacy Perceptions of Cloud Office Suites

Cloud Office Suites such as Google Docs or Microsoft Office 365 are widely used tools, but introduce unique security and privacy risks and challenges for documents and sensitive user information. This chapter describes an investigation into the security and privacy perceptions and expectations of 200 users of cloud office suites such as Google Docs and Microsoft Office 365 from Germany and the U.S.

As this project was conducted as a team consisting of me, Nicolas Huaman, Christian Stransky, Niklas Busch, Yasemin Acar, and Sascha Fahl, this chapter utilizes the academic "we" to mirror this fact. Our survey found that users are generally aware of basic security implications, storage models, and access by others, but some threat models are underdeveloped due to a lack of technical knowledge. Users have strong opinions on certain parties accessing their data but are unsure who actually has access to their documents. The chapter provides recommendations for different groups associated with cloud office suites to inform future standards, regulations, implementations, and configuration options.

7.1 Preamble

This chapter is based on research that was also published as "Cloudy with a Chance of Misconceptions: Exploring Users' Perceptions and Expectations of Security and Privacy in Cloud Office Suites" [19], which appeared and was presented by me at the Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020) in August 2020.

> **Dominik Wermke**, C. Stransky, N. Huaman, N. Busch, Y. Acar, and S. Fahl, "Cloudy with a Chance of Misconceptions: Exploring Users' Perceptions and Expectations of Security and Privacy in Cloud Office Suites," in *Sixteenth Symposium on Usable Privacy and Security (SOUPS'20)*, Aug. 2020

The original abstract for the publication is as follows:

> **Abstract:** Cloud Office suites such as Google Docs or Microsoft Office 365 are widely used, and introduce security and privacy risks to documents and sensitive user information. Users may not know how, where and by whom their documents are accessible and stored, and it is currently unclear how they understand and mitigate risks. We conduct surveys with 200 cloud office users from Germany and the U.S. to investigate their experiences and behaviours with cloud office suites. We explore their security and privacy perceptions and expectations, as well as their intuitions for how cloud office suites should ideally handle security and privacy. We find that our participants seem to be aware of basic general security implications, storage models,

and access by others, although some of their threat models seem underdeveloped, often due to lacking technical knowledge. Our participants have strong opinions on how comfortable they are with the access of certain parties, but are somewhat unsure about who actually has access to their documents. Based on our findings, we distill recommendations for different groups associated with cloud office suites, which can help inform future standards, regulations, implementations, and configuration options.

I would like to express my sincere gratitude to everyone who has contributed to the completion of this project. I would also like to thank our participants, who generously gave their time and shared their perceptions regarding cloud office software with us. Their willingness to participate made this research possible, and I am deeply grateful for their contributions.

7.1.1 Contribution

The research presented in this chapter was conducted as a team consisting of me as team lead, Nicolas Huaman, Christian Stransky, Niklas Busch, Yasemin Acar, and Sascha Fahl. I am grateful for the contributions of each member, which have been integral to the success of this research project. Without their expertise, hard work, and dedication, this research project would not have been possible.

I came up with the initial idea for this study based on the then-prevalent privacy issues with using U.S.-based cloud applications in German education and industry. I setup the initial concept and research approach involving U.S. and German participants for this research project. I lead the design of the study and survey guide with the rest of the team. Christian Stransky and I invited participants via Amazon's Mechanical Turk. I analyzed and visualized the survey counts together with Nicolas Huaman. In joint work with Christian Stransky, Nicolas Huaman, and Niklas Busch, we qualitatively coded the free text answers. I compiled the paper for publication with contributions from the remaining team and we jointly discussed the work's implications. I presented this publication at SOUPS'20.

7.1.2 Structure

The rest of this chapter is structured as follows: after a general introduction (Section 7.2), I provide a background to cloud office suites in Section 7.3. I then describe the setup and structure of our two surveys in Section 7.4 and report our results in Section 7.5. I discuss related work at the time of this research in 2020 in Section 7.6. Finally, I discuss findings and give recommendations in Section 7.7 and summarize this chapter in Section 7.8.

7.2 Introduction

During the 1970s, office software began to emerge in the world of personal computing. Early word processors such as Electric Pencil for the MITS Altair in 1976 , WordStar for the CP/M in 1978 , and later dedicated spreadsheet applications such as VisCalc were considered "killer applications" for their respective systems. These dedicated office tools helped the adoption of personal computers over more dedicated or mechanical systems for word processing. In recent years, another major shift is happening in the world of office applications. With Microsoft Office 365, Google Drive, and projects like LibreOffice Online, most major Office suites have moved on to provide some sort of cloud platform that allows for collaboration between multiple editors, automatic real-time storage on cloud or internal network servers, and easy access through the browser without requiring the installation of software.

The major selling points for these cloud office platforms might as well be their biggest (security & privacy) weaknesses: easy sharing of documents, cloud storage of data, and the high similarity in design and UI to previously prevalent offline office software hide a large array of potential privacy and security trapdoors from the average office user.

With the shift from offline to cloud, many cloud office providers also moved from a pay-once model to a subscription-based model with a trial period or even a completely free payment model. This shift accompanied a questionable change in business model drive for these companies: the processing and storing of documents in the cloud provides the possibility of large-scale privacy intrusion by the providers for both end users and businesses that utilize the cloud. Due to the similarity in design to offline office software, this major impact on their privacy is likely not fully realized by the end user. This impact on privacy gets further amplified by governments and administrations updating their infrastructure to cloud-based solutions, potentially processing and uploading the data of citizens in the cloud without their explicit consent. In a recent example, the Department of Defense awarded a $7.6 billion contract to General Dynamics to provide the Pentagon with the cloud-based Microsoft Office 365 [339].

Another major selling point of cloud office applications is the ease of access, often from almost anywhere on earth with an internet connection, without requiring any additional installation of software. While the actual location of the underlying servers is rarely mentioned in cloud advertisements, it has large implications on privacy and security. In July 2019, the central German State of Hesse declared that schools may not legally to use Microsoft Office 365 and similar cloud office platforms due to collected telemetry and the potential access to stored data on U.S. servers by U.S. officials [340], [341]:

> "What is true for Microsoft is also true for the Google and Apple cloud solutions. The cloud solutions of these providers have so far not been transparent and comprehensibly set out. Therefore, it is also true that for schools the privacy-compliant use is currently not possible." (Hessian commissioner of Data Protection and Freedom of Information [341].)

In August 2019, Microsoft announced that it will be able to provide cloud services from data centers in Germany in late 2019 "to meet evolving customer needs" and to being "committed to making sure that the Microsoft Cloud complies with [the European General Data Protection Regulation] GDPR" [342]. As of February 2020, Microsoft offers Office 365 and Dynamics 365 from new German data center regions [343].

In this chapter, I investigate privacy and security misconceptions by end users of cloud office applications in a user study including participants from both Germany and the U.S. For this, we conducted two online surveys with 200 crowd workers from Amazon's Mechanical Turk and ClickWorker. With a combination of qualitative and quantitative methods, we modeled the two surveys to explore the following research questions:

Table 7.1: Overview of the most common cloud office suites and their related features.

	Storage	Offline Mode	Versions available				Mobile Version		Sharing				Document Recovery
			Self Hosted	Free	Paid	Trial	Android	iOS	E-Mail	Link	Read Only	Read & Comment	
Office 365	●	○	○	◐[1]	●	●	●	●	●	●	●	○	●
Google Drive	●	●	○	●	●	●	●	●	●	●	●	●	●
iWork for iCloud	●	○	○	●	●	-	○	●	●	●	●	○	●
LibreOffice Online	○	○	●	●	◐[2]	-	◐[3]	○	◐[4]	◐[4]	◐[4]	○	◐[4]
OnlyOffice	●	○	●	●	●	●	●	●	●	●	●	●	●

● Feature available ○ Feature not available ◐ Feature partially available

[1] Students and teachers receive a free online only version. [2] Support is only provided by third party companies and not directly by The Document Foundation. [3] Only a viewer is available. [4] Depends on underlying software.

RQ1: "How and why do our participants interact with cloud office applications?" Today's office suites are compelling to use with features such as collaboration between multiple editors, automatic real-time storage, and easy online access without installation. We are interested why and how our participants interact with office applications both in a home and organizational setting.

RQ2: "What are end users' awareness, perceptions, and attitudes about privacy in cloud office applications?" The switch from an offline to cloud environment in both home and organizational settings introduced immense changes for privacy and security assumptions for office suites. We examine our participants' security and privacy perceptions and expectations, as well as their intuitions for how cloud office suites should ideally handle security and privacy.

RQ3: "What is the participants' understanding and related mental models regarding protection and security of their cloud documents?" The actual server location, access by providers or governments, and handling of deletions has an enormous impact on the privacy of cloud office applications. We survey the extend of our participants' understanding and their basic mental models regarding cloud office documents.

7.3 Cloud Office Suites

For this research, we define cloud office suites as cloud-based office applications that allow view, edit and comment on documents, spreadsheets and presentations in the browser.

Table 7.1 provides an overview of the most popular cloud office suites and their features relevant for this work. Prominent providers of cloud office suites are Google (Google Drive) [344], Microsoft (Office 365) [345], Apple (iWork for iCloud) [346], The Document Foundation (Libre Office Online) [347], and Ascensio System SIA (OnlyOffice) [348]. In contrast to traditional office suites such as Microsoft Office, cloud office suites provide browser based user interfaces. Users are no longer limited to work on desktop computers using native office applications, but can access their files using any device that provides a modern browser. Hence, modern cloud office suites support mobile devices such as smartphones and tablets and allow easy access to their cloud applications wherever users have access to the internet.

In contrast to traditional office suites, cloud office suites allow users to easily share documents with multiple collaborators and edit the same document simultaneously. Cloud office documents can be shared using e-mail addresses or direct links to a document. For better user experience, all cloud office

suite providers allow their users to recover deleted documents. In addition to online access to their documents, Google Drive provides an offline mode that stores documents in the local browser storage and makes them available for offline editing. Offline documents are pushed to the cloud as soon as users have Internet access.

The three major providers Microsoft, Google, and Apple only provide cloud-hosted solutions while Ascensio System also provides a self-hosted community edition which allows keeping the data under their users' control. Every hosted cloud office solution provides storage capabilities in the cloud. The amount of storage included depends on the license purchased and can be upgraded at any time. LibreOffice Online by The Document Foundation supports no storage by itself and is dependent on the underlying software like OwnCloud or NextCloud to provide the storage and authentication.

While all cloud office suites provide rudimentary access control for sharing, only Google Drive and OnlyOffice provide an option to share documents with read-only access that still allows to comment on documents.

7.4 Methodology

In this section we provide details on the procedure and structure of the two surveys we conducted with crowd workers from Amazon's Mechanical Turk ($n = 105$) and ClickWorker ($n = 95$). We also detail the coding process for our qualitative questions as well as the statistical analysis approach for our quantitative data. We also report on our data collections and ethics, and discuss the limitations of our work.

Note that while our two surveys may include participants living in the U.S. or in Germany, Austria, or Switzerland respectively, we refer to them as "U.S." and "German(y)" for a more succinct reporting.

7.4.1 Study Procedure

Both the German-speaking participants from ClickWorker and the English-speaking participants from Mechanical Turk were administered an almost identical survey, with the German survey being a direct translation from the English one by multiple native German speakers.

Questionnaire Development. The questionnaire development was guided by our established research questions. We included pre-tested and evaluated survey questions from previous work where appropriate to allow for a greater comparability between studies. In addition, we performed 5 in-depth, free-form interviews with both experts and non-experts to establish additional areas of interest for our survey.

Pre-Testing. Before we conducted the surveys, we pre-tested our questionnaires following the principle of cognitive interviews [349]. This allowed us to glean insights into how survey respondents might interpret and answer questions. We asked participants to share their thoughts as they answered each survey question and used our findings to iteratively revise and rewrite the survey questions to minimize bias and maximize validity. This first pre-test was conducted internally in both German and English with members of the groups, students of our university, and friends. In addition, we refined the surveys in multiple pilots with participants on Mechanical Turk ($n = 9$) and ClickWorker ($n = 20$) until a satisfactory convergence was reached.

Recruitment and Inclusion Criteria. We recruited participants for our study from Amazon's Mechanical Turk and ClickWorker during September 2019. We did not mention security or privacy in the initial recruitment ad to avoid certain recruitment biases. We generally required participants to be age 18 or older and to have used cloud office software before. For Amazon's Mechanical Turk, we

additionally required participants to live within the United States. To ensure sufficient data quality, we also required them to have completed a minimum of 1,000 hits and to have a task approval rate of at least 95% [350]. For ClickWorker, we additionally required participants to speak German and to live within Germany, Austria, or Switzerland.

A total of 229 people responded to our surveys. Of those, 22 did not finish and 7 were excluded due to low-quality answers or due to failing at least one of our quality checks, resulting in 200 final participants whose responses we consider.

7.4.2 Survey Structure

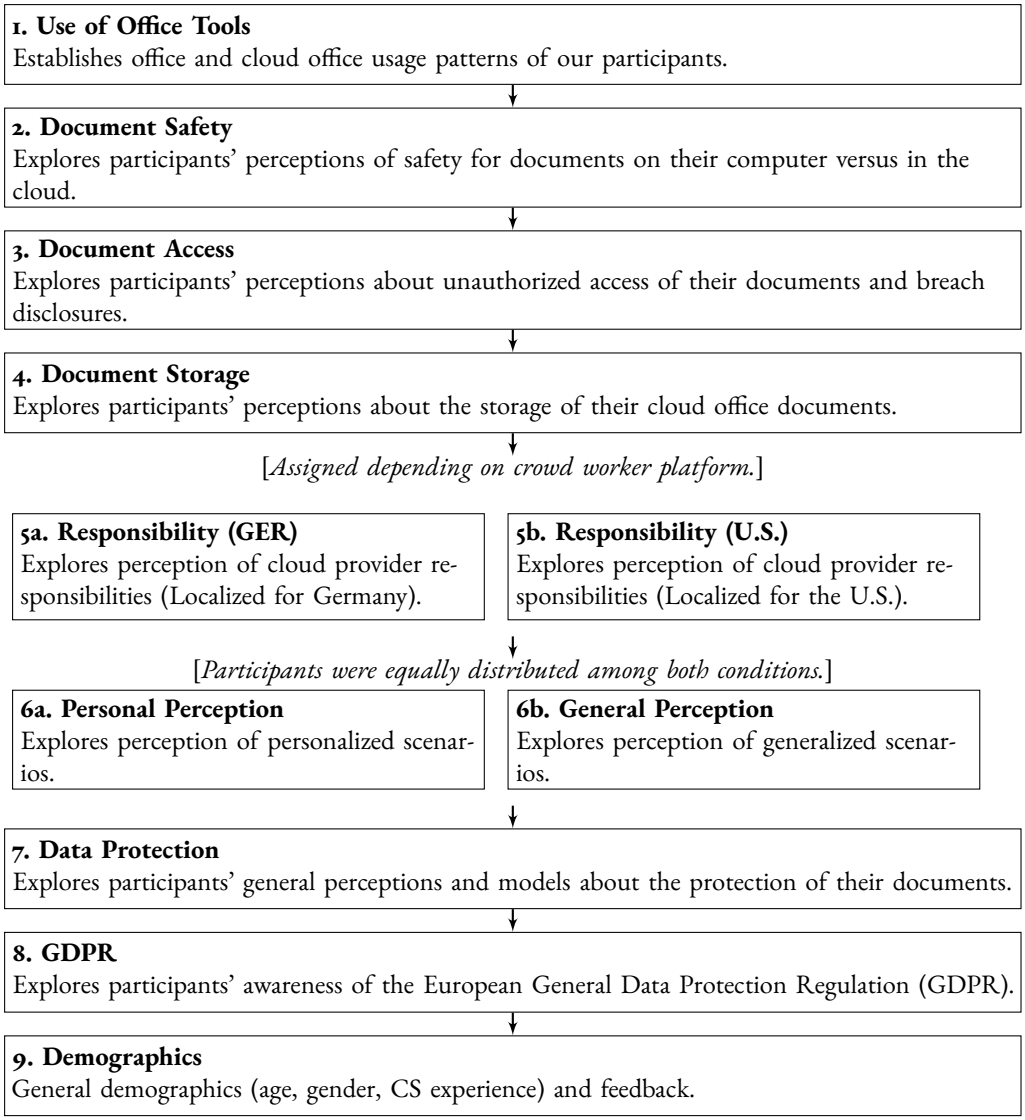


Figure 7.1: Illustration of the survey flow for both German and English surveys. Splits in the flow include a localized version of the "Responsibility" block for Germany and the U.S. and a split for generalized scenarios vs. personalized that were randomly assigned to participants.

We outline the survey structure in Figure 7.1 and below. Both our surveys consisted of a total of 9 sections, ranging from general cloud office questions to personal beliefs about the responsibilities of cloud office providers. The two survey versions differed slightly due to localized answer options (e.g., localized names for government agencies) and changes to concepts that do not exist or have a different privacy implication in German-speaking countries (e.g., social security number).

1. Use of Office Tools: Our surveys open with questions in which we explore the general usage patterns of offline and cloud office applications by our participants in both private and professional context. We report general demographics and office-specific demographics of our participants in Section 7.5.1 and Table 7.3.

2. Document Safety: The "Document Safety" section explores how participants perceive the security of their documents in the cloud vs. locally on their computer and why. We report these results in Section 7.5.2.

3. Document Access: The "Document Access" section investigates participants' mental concepts and perceived risks related to the access of their documents. Questions related to which parties they think have access to their documents, who already might have accessed their documents without their authorization, and if the risk of unauthorized access by different parties is higher in the cloud or on their computer. Further, the section asks participants about who they think *would* inform them in case of an unauthorized access to their data and who they think *should* inform them and how. We report the results related to the access of cloud office documents in Section 7.5.3.

4. Document Storage: This section explores our participants' perception about the storage of their cloud office documents. We asked our participants about the number of copies they think exist of their documents and with whom they think copies remain after deleting their own versions. In addition, we asked who they think can delete their documents. We report the results for these questions in Section 7.5.4.

5a/b. Responsibility: In this section, we investigate our participants' perceptions about responsibilities of cloud office providers regarding access and protection of documents. The "Responsibility" section differs slightly between the German and English survey to allow for the localization of certain answer options such as law-enforcement agencies and government names. We report the results in Section 7.5.5.

6a/b. Perception: The "Perception" section contains questions to three different scenarios related to the processing of sensitive data in cloud document applications, either in a more personal or more generalized condition.

1. Data of children. The first scenario described the use of a cloud office application in an educational setting. We asked our participants to assess how much they felt at ease with using cloud office applications for handling data of children in schools, e.g., for storing grades or writing tasks.
2. Health data. The second scenario had a focus on health information. A general practitioner used a cloud office application to handle sensitive patient information including a patient's name, age, weight, diagnosis, and treatment plan. Again, we asked our participants to rate their level of comfort with the scenario.
3. Financial data. In the third scenario we illustrated a use case involving financial data. A financial advisor used a cloud office application to process client data. The processed documents include private information such as the client's name, social security number, and detailed financial information.

Participants of the study were equally distributed between both conditions and the order of scenarios was randomized for each participant. Results for the different scenarios are reported in Section 7.5.6.

7. Data Protection: The "Data Protection" section explores participants' mental models about the protection of their documents in the cloud. We asked our participants' which data they think is collected when they process documents in cloud office applications and how they think their data is protected. We report these results in Section 7.5.7.

8. GDPR:. In the "GDPR" section we explored our participants' general knowledge about the European General Data Protection Regulation (GDPR) and what they know about the protection offered by it. These questions link back to "responsibility block", which asked participants about cloud office provider responsibilities directly implied by the GDPR. We report the general results for this block together with other demographics in Section 7.5.1 and combined it for our analysis of the responsibility section in Section 7.5.5.

9. Demographics: We administered demographic questions at the end of the questionnaire to prevent stereotype bias [351], [352]. Our demographic questions included age, gender, age, and previous experiences in CS education and CS jobs. Additionally we asked respondents for general feedback for the survey questionnaire. We report general demographics and office-specific demographics of our participants in Section 7.5.1 and Table 7.3.

7.4.3 Coding and Analysis

Our collected data includes both qualitative and quantitative data points.

Qualitative Coding. We analyzed all open-ended questions in an iterative open-coding process [290], [291]. Two researchers established an initial codebook [289], coded all open-ended questions together, and resolved emerging coding conflicts immediately in a consensus discussion or by introducing new codes. If new codes were introduced, all previous answers were revisited and re-coded if necessary. Due to the immediate resolving, reporting an intercoder agreement and reliability is uncommon for this approach [353]. The codebook remained stable once both researchers were satisfied that all important themes and concepts in the responses could be captured with the codes. Both surveys were coded with the same codebook and codes for the German survey were assigned by two native speakers.

Quantitative Analysis. We use the non-parametric Kruskal-Wallis H test (KW; non-parametric equivalent to the one-way ANOVA) to compare multiple independent groups. For multiple tests on paired groups, we use the Mann-Whitney U test (MWU) and control the results for multiple testing. We assume an alpha level of $\alpha = .05$ for significance in hypothesis tests. Where appropriate, we controlled our hypothesis tests for the multiple comparison problem with the conservative Bonferroni correction and report the "adjusted"/"adj." values. For certain tests, we map five-point Likert scale answers to numbers (-2, -1, 0, 1, 2).

We present the outcomes of our regressions in tables where each row contains a factor and the corresponding change of the analyzed outcome in relation to the baseline of the given factor. Linear regression models measure change from baseline factors with a coefficient (Coef.) of zero for the value of the outcome. For each factor of a model, we also list a 95% confidence interval (C.I.) and a p-value indicating statistical significance. We highlight p-values below a cut-off of .05 with a star ($*$).

As our regression analyses are intended to be exploratory, we consider a set of candidate models and select the final model based on the lowest Akaike Information Criterion (AIC) [354]. We consider candidate models consisting of the required factors "Country", "Condition", and "Scenario", as well as every possible combination of the optional variables. Required factors, optional factors, and corresponding baseline values are described in Table 7.2. In cases when we consider results on a per-scenario

Table 7.2: Factors used in candidate regression models. Model candidates always included the required factors and covered all possible combinations of optional factors. Final models were selected based on lowest AIC. Categorical factors are individually compared to the baseline.

Factor	Description	Baseline
Required		
Country	Germany or U.S., participants assigned based on crowd working platform.	U.S.
Condition	General or Personal. Scenario condition, participants evenly distributed between both conditions.	General
Scenario	Child, Health, or Financial. Type of scenario, all 3 shown to each participant.	Child
Participant	Random effect accounting for repeated measures (due to the 3 scenarios per participant).	n/a
Optional		
Office at work	True or False, uses office software at work, self-reported.	False
CS Education	True or False, has a CS education, self-reported.	False
CS Job	True or False, has a CS job, self-reported.	False
Age	Age in years, self-reported.	n/a

rather than a per-participant basis, we use a mixed linear model that adds a random intercept to account for multiple scenarios from the same participant.

7.4.4 Data Collection and Ethics

Our institutions did not require a formal IRB process for the studies conducted in this research project. Nonetheless, we modeled our research plan and study procedures after an IRB-approved study, adhered to the strict German and U.S. data and privacy protection laws and the General Data Protection Regulation in the E.U., and structured our study following the ethical principals of the Menlo report for research involving information and communications technologies [355]. All participants approved to a consent form that informed them about the study purpose, the data we collected and stored and included an e-mail address and phone number to contact the principal investigators in case of questions or concerns.

Recently, researchers faced issues with low data quality on Amazon MTurk [356]. Therefore, we included a number of filters to identify low-quality answers. During data cleaning and analysis, we identified 7 participants who did not pass our quality measures and excluded these invalid participants from further analysis.

We calibrated participants' compensations based on an average piloting time of 10 minutes and payed participants on Amazon's Mechanical Turk $1.70 and on ClickWorker €1.70 for an hourly wage of $10.20 and €10.20, respectively.

7.4.5 Limitations

As any study with online surveys, our work includes a number of limitations typical for this type of 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. However, while we utilize self-report data, our central claims are not about the accuracy of respondents' answers to a given question, but rather about the concepts and misconceptions conveyed by their answers.

Conducting user studies on crowd working platforms like Amazon's Mechanical Turk and ClickWorker is a commonly used and generally accepted procedure for human-computer interaction and usable security and privacy research [357]. While the quality of answers can suffer in a crowd worker context, we tried to ensure a high data quality by following best practices by limiting access to our surveys to high-reputation cloud workers [350] and by manually filtering low quality answers.

This study focuses on the responses of German and U.S. Internet users, and thus, we can offer no insight into the generalizability of results for international participants. We aimed to improve the internal validity of our study by providing localized answer options.

We explicitly ignored the implications of meta data collection and third party data of cloud office providers to allow participants to focus on their mental model of cloud document processing and access.

7.5 Results

In the following section we report and discuss results for all 200 valid participants of both the U.S. and German survey. Generally, participants were aware of certain security and privacy implications of writing their documents in cloud office applications, but were unaware or had severe misconceptions about others. Our reporting of results mostly follows the actual order of survey sections described in Section 7.4.2. After each question section, we summarize our key findings.

7.5.1 Use of Office Tools

We report the general demographics of both surveys in Table 7.3. Overall, 127 participants responded to our survey on Amazon's Mechanical Turk (U.S.) and 102 on ClickWorker (German). Of those, 105 and 95 respectively completed the survey and were considered valid for a combined total of 200 participants of whom we report results.

Our participants identified predominantly as male (64.5%) with a median age of 33.0 years (mean = 35.7, σ = 10.7). Across both surveys, 28.0% of our participants classified themselves as having a CS education and 22.5% as having worked in a CS-related job. CS experiences are similar for both the U.S. and the German survey, with the exception of CS education (38.1% vs. 16.8%). We assume this discrepancy might be related to general differences in education systems, as the German school curriculum focuses less on IT education compared to the U.S. The majority of both our U.S. and German participants have a job that involves using office applications regularly with 80.0% and 77.8%, respectively.

The majority (97.1%) of our U.S. participants have used Google Drive (with its related cloud office tools such as Google Docs or Google Sheets) before, followed by Microsoft Office (Offline) (86.7%) and Microsoft Office 365 (Cloud) (70.5%). The majority of our German participants (87.4%) is more familiar with Microsoft Office (Offline), followed by Google Drive (80.0%) and Microsoft Office 365 (Cloud) (64.2%). We assume this difference is likely due to the extensive, almost exclusive usage of Microsoft Office products in German businesses and government[1]. These differences even out for office tools used in the last months where Google Drive prevails among both the U.S. and German participants (82.7%, 70.5%), followed by Microsoft Office (Offline) (50.5%, 65.3%) and Microsoft Office 365 (Cloud) (50.5%, 55.8%). The majority of our U.S. participants use office tools to process Spreadsheets

[1] E.g. the city of Munich decided to migrate to Windows 10 after it's 2003 decision to adopt Linux, partially due to incompatibility and communication problems with other organizations [358].

Table 7.3: Demographics for all valid participants from the U.S. survey (Amazon's Mechanical Turk), German survey (ClickWorker), and combined.

	U.S.	German	Combined
Participants			
Started	127	102	229
Finished	110	97	207
Valid (n =)	105	95	200
Gender			
Male	66.7%	62.1%	64.5%
Female	33.3%	37.9%	35.5%
Other (Free text)	0.0%	0.0%	0.0%
Age in years			
Mean	35.3	36.1	35.7
Std. dev. (σ)	9.9	11.5	10.7
Median	33.0	33.0	33.0
Computer Science			
CS Education	38.1%	16.8%	28.0%
CS Job	24.8%	20.0%	22.5%
Professional Usage			
Office software at work	80.0%	77.9%	79.0%
Office Usage*			
Google Drive	97.1%	80.0%	89.0%
Microsoft Office (Offline)	86.7%	87.4%	87.0%
Microsoft Office 365 (Cloud)	70.5%	64.2%	67.5%
LibreOffice Offline	18.1%	25.3%	21.5%
Apple's iWork Web (Offline)	9.5%	20.0%	14.5%
Apple's iWork Web (Cloud)	6.7%	17.9%	12.0%
LibreOffice Online	4.8%	9.5%	7.0%
Other	3.8%	5.3%	3.0%
OnlyOffice	1.0%	1.1%	2.5%
Document Usage*			
Spreadsheets	89.5%	82.1%	86.0%
Text	76.2%	90.5%	83.0%
Emails	68.6%	55.8%	62.5%
Presentations	49.5%	65.3%	57.0%
Calendar and Appointments	57.1%	50.5%	54.0%
Other	1.0%	2.1%	1.5%
Document Storage*			
Locally on my computer	73.3%	82.1%	77.5%
Google Drive	88.6%	52.6%	71.5%
Dropbox	33.3%	35.8%	34.5%
OneDrive	30.5%	29.5%	30.0%
iCloud	18.1%	24.2%	21.0%
Network Share	21.0%	16.8%	19.0%

* Multiple answers allowed, may not sum to 100%

(89.5%), Text (76.2%), and Emails (68.6%). As document types, the German participants process Text (90.5%), followed by Spreadsheets (82.1%) and Presentations (65.3%).

Most of our U.S. participants prefer to store their documents in Google Drive (88.6%), followed by locally (73.3%), and Dropbox (33.3%). While the majority of German participants prefers local storage (82.1%), followed by Google Drive (52.6%), and Dropbox (35.8%). This mirrors the distribution of most used office tools for U.S. participants (Google Drive Office → Google Drive Storage) and German participants (Offline Microsoft Office → Local Storage).

Participants of both the U.S. and German survey agree on the top reasons why they (would) use cloud office applications over local office applications: easy remote access of documents (76.2%, 70.5%), ease of collaboration (58.1%, 59.0%), and free or cheap access (52.4%, 43.2%).

> Summary: Demographics.Somewhat unsurprisingly, participants prefer to store their documents on the platform they edit them with (e.g., locally for offline office). All of our participants agree on the benefits of cloud office applications: free access and easy collaboration for remote documents.

7.5.2 Document Security

In this question section we asked our participants to think about where their documents are more secure from any unauthorized access, on their personal computer or in the cloud. Most participants reported that they feel their personal computer is more secure than for their documents than the cloud (54.5% vs. 19.5%).

In addition to the quantitative questions, we asked our participants to explain their assessment. Most (94) of the participants who said they felt their documents would be more secure against unauthorized access on their personal computers mentioned that an attacker would require physical access to their machines to acquire access to documents, e.g., P30 said "You would have to physically breach my computer to get to the documents, the drive is encrypted, no one can access it.". Similarly P47 explained "I think that documents are more secure on my computer because I'm the only one that can access them; and if there were any threats on my PC, I would use programs to get rid of them." (P47). Some participants (21) who said files were more secure on their computer thought that it was easier to attack a cloud system than their personal computer, e.g., P27 mentioned that "Because I know what security I have on my pc, but don't know about Google. Of course, I assume they've got top of the line security, but I don't actually know." (P27).

Participants who thought documents in the cloud were more secure (39; 19.5%) mostly mentioned two reasons. First, they believe that cloud office suite providers have more security expertise than they personally do. For example, P79 said "The cloud is managed by big corporations. They probably take security more serious than individuals. They always have to worry about hackers so there security is likely very powerful." (P79). Second, some participants assessed cloud office suites to be more "secure" than their personal computers because they have backups and losing data is less likely, e.g.,

> "Local computers can be hacked and can crash. It happens. Too often, backups are not made regularly, so data can be lost in either case. With automatic backup to the cloud, documents are more secure in case of local computer issues.." (P74)

Other reasons for believing in a secure cloud seem often to be based on insufficient technical knowledge, e.g., "because I think it is not possible to hack the cloud." (P219). Few participants (3; 1.5%) mentioned the use of two-factor authentication (2FA) and the application of encryption by cloud office suite providers as important security factors.

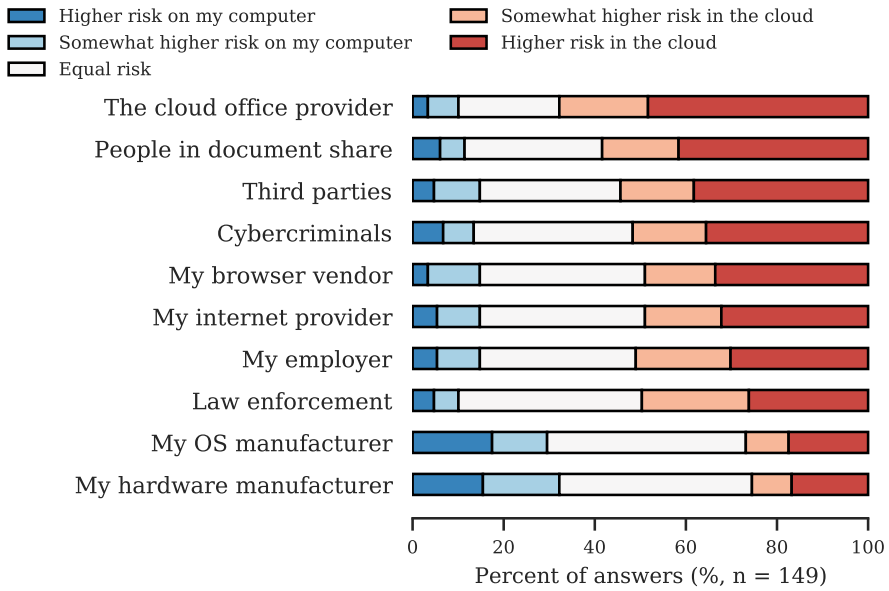


Figure 7.2: Likert-Scale for participants' associated risk of unauthorized access between their local computer and their cloud office documents for different parties.

> Summary: Document Security. Our participants seem to be aware of some general security implications of processing their documents in the cloud. They seem to prefer their local system in terms of security against unauthorized access, although some of their threat models seem less developed.

7.5.3 Document Access

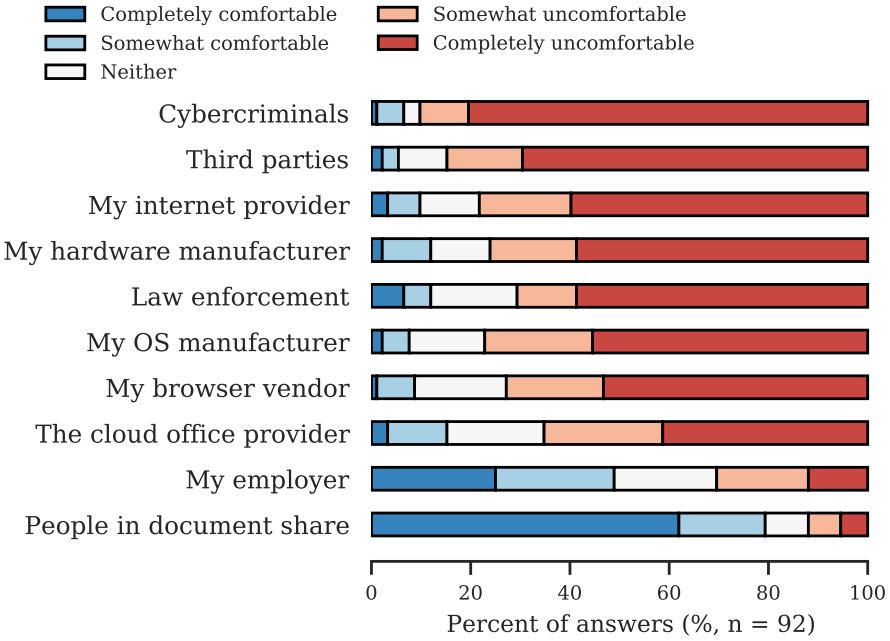
In this question section we explore perception, misconceptions, and mental models of our participants regarding the (unauthorized) access of specific parties to their potentially sensitive documents processed in cloud office applications.

We found a significant difference in the risk of different parties accessing the participants documents (KWH; $H = 102.33$; $p < 0.01$). This might indicate that participants seem to be aware of the changed attack surface for cloud office documents and associate a higher risk of unauthorized access by cybercriminals and third parties such as advertisers and plugin developers in the cloud (cf. Figure 7.2).

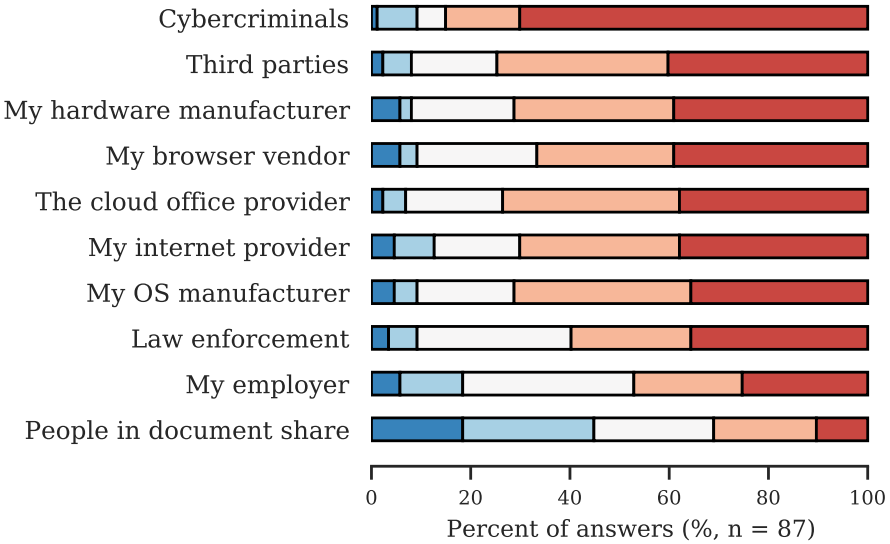
These answers coincide with parties of which participants thought that they already accessed their documents, although some participants have the misconception that their browser vendor and operating system provider also have accessed their cloud documents. Figure 7.3 shows the comfort level of our participants related to the access of different parties to their cloud office documents.

We also asked participants who would inform them if their cloud office documents are accessed by an unauthorized party and who should inform them. Participants' answers point at a responsible party here: While the German and U.S. participants are split on the cloud office provider (73; 69.5%) and nobody (39; 41.1%) as most common answer on who *would* inform them respectively, both groups agree that it is the cloud office provider that *should* inform them (153; 76.5%).

A large number of participants explicitly told us that they like to be informed about unauthorized access of their cloud office documents by email (119; 59.5%). In addition, some participants provided us with their wishes about the information they want to receive in case of such a data breach, e.g., P69 insisted that



(a) Survey participants from MTurk (U.S.).



(b) Survey participants from CrowdWorker (German).

Figure 7.3: How comfortable our participants are with different parties accessing their cloud documents. "I don't know" answers were omitted.

"[I] [n]eed to know basically everything that the person saw. When they saw it, what they saw, where they're from. I don't care who gives the analysis, just that its an accurate analysis and that they let me know.." (P69)

> Summary: Document Access.Overall, our participants seem to have a clear idea on by whom and how they should be informed about unauthorized access of their cloud documents: the cloud office provider via (secure) email. Our participants seem to have strong opinions on how comfortable they are with the access of certain parties, but are somewhat unsure about who actually has access to their documents.

7.5.4 Document Storage

The majority of German participants believe that multiple copies of their cloud office documents exist (49; 51.6%), while most U.S. participants admit that they do not know (51; 48.6%). Of those that assume multiple copies exist (83; 41.5%), the majority thinks that only their copies are deleted if they delete a document (30 of 83; 36.1%), or they are unsure (21 of 83; 25.3%). Unsurprisingly the majority of our participants assume that their cloud provider can delete their documents (138; 69.0%), followed by people they shared the documents with for U.S. participants (43 of 105; 41.0%) and cybercriminals for German participants (46 of 95; 48.4%).

Some of our participants assume a rather basic mental model of why copies of their cloud documents might exist, e.g., P123 believes "[...] that these copies exist just in case that [sic] the original documents get lost." (P123). Other participants had a less utilitarian view on the existence of potential copies, e.g., P79 had some rather dystopian thoughts about why copies of their documents are created: "[T]o use against me when the time is right." (P79). For why not all of the copies are deleted, some participants had some very convincing arguments: "[They are] used to train artificial intelligence or to make a profile of me for the future." (P79), "possibly to sell to 3rd-party vendors for advertising" (P96), and "so they can be used for law enforcement." (P97).

> Summary: Document Storage.Overall, our participants seem to be rather unsure about the actual number of copies, access rights, and deletion procedures of their cloud documents. They seem pessimistic on why additional copies are kept.

7.5.5 Document Responsibility

In this section we asked participants about which party they think is responsible for the protection of their documents. The majority of U.S. participants sees the cloud provider as responsible (83; 79.0%), while the majority of the German participants sees themselves as responsible (69; 72.6%),

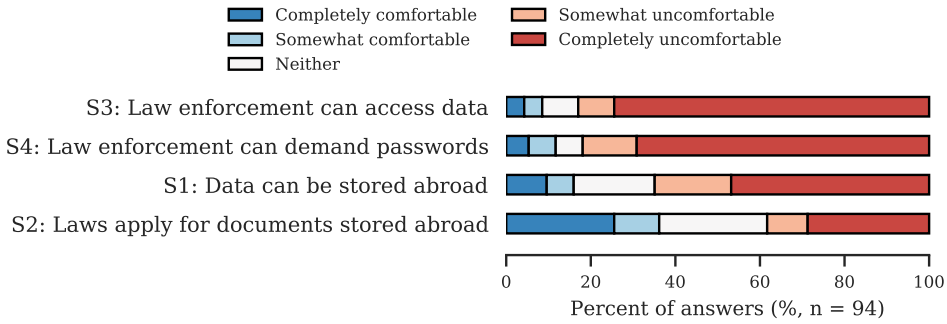
We also compared U.S. and German participants in their agreement regarding four scenarios exploring the responsibilities of cloud office providers:

S1: "Cloud office providers should offer adequate protection for cloud office documents." (MWU; $U = 4445$; adj. p = 1)

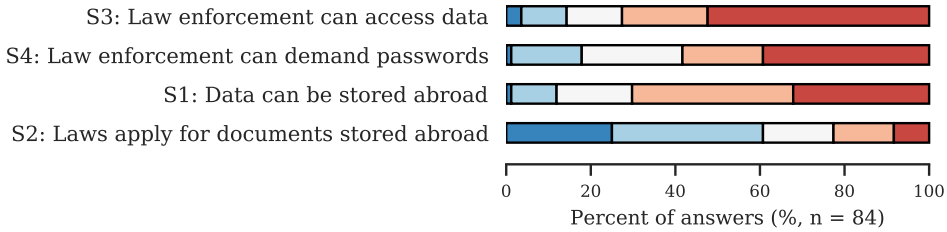
S2: "I should have the right to demand a full overview of my data collected by cloud office providers." (MWU; $U = 4419$; adj. p = 1)

S3: "Upon my request, cloud office providers should have to show what they do with my documents and who has or had access." (MWU; $U = 4181$; adj. p = 1)

S4: "Cloud office providers must be able to modify or delete any data they have on private individuals." (MWU; $U = 4566$; adj. p = 1)



(a) Survey participants from MTurk (U.S.).



(b) Survey participants from CrowdWorker (German).

Figure 7.4: Participants' comfort with potential privacy violations by their government.

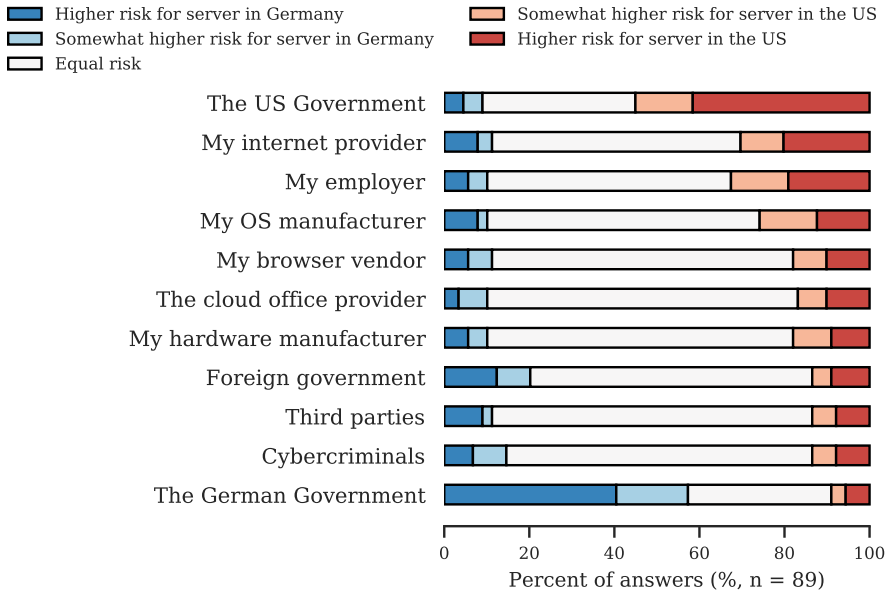
and find no significant differences between our U.S. and German participants. Similarly, we compared U.S. and German participants regarding their (dis)comfort with the following statements (Note that the statements were localized, e.g., an U.S. participant would be presented with "US regulation"):

S1: "Cloud providers can store my documents on servers outside of the US/Germany without legal repercussions." (MWU; $U = 4151$; adj. $p = 1$)
S2: "US/German regulations and laws still apply if the documents are stored on servers outside of the US." (MWU; $U = 4817$; adj. $p = 0.04$)
S3: "US/German law enforcement can access my cloud documents without a court order." (MWU; $U = 4768$; adj. $p = 0.02$)
S4: "US/German law enforcement can force me to give up my cloud office password." (MWU; $U = 5104$; adj. $p < 0.001$)

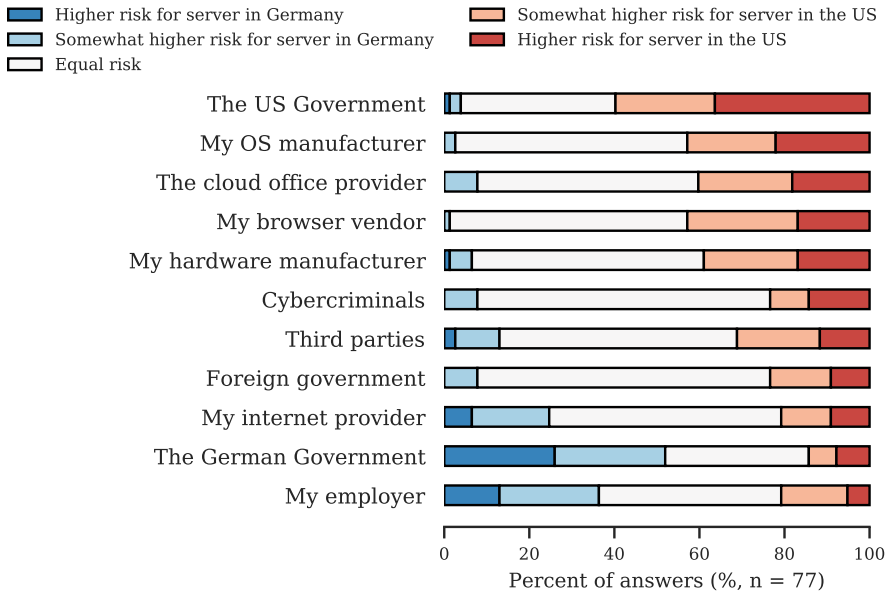
and find significant differences for S2, S3, and S4. These differences can be mostly attributed to U.S. participants being more uncomfortable with privacy violations by their government compared to the Germans (cf. Figure 7.4).

We further investigated differences between U.S. and German participants by asking them where they do think the risk is higher of different parties obtaining unauthorized access to their documents if they are either stored on a server in the U.S. or Germany (cf. Figure 7.5).

> Summary: Document Responsibility. While participants from the U.S. and Germany agree on the responsibilities of cloud providers, U.S. participants are comparably more uncomfortable regarding potential privacy violations by the government.



(a) Survey participants from MTurk (U.S.).



(b) Survey participants from CrowdWorker (German).

Figure 7.5: Risk of unauthorized parties accessing participants' documents on servers in the U.S. vs. Germany.

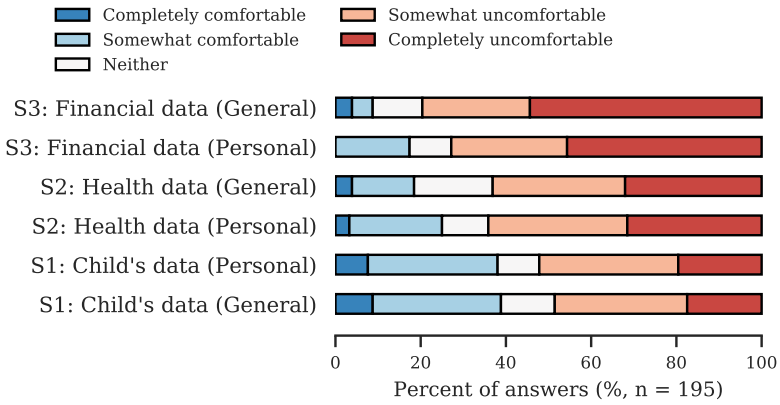


Figure 7.6: Participants' comfort with three different data scenarios (Financial, Health, and Children) and two different conditions (General and Personal perspective).

Table 7.4: Final linear mixed regression model examining the perception of 3 different scenarios in 2 phrasing conditions. "I don't know" answers were omitted. See Section 7.4.3 and Table 7.2 for further details.

Factor	**Coef.**	**C.I.**	***p*-value**
Scenario: Health	−0.50	[−0.66, −0.33]	< 0.001*
Scenario: Financial	−0.88	[−1.05, −0.72]	< 0.001*
Condition: Personal	0.03	[−0.25, 0.31]	0.843
Country: Germany	−0.11	[−0.39, 0.17]	0.431

7.5.6 Scenario Perception

In this question section, we wanted to explore the effect of different conditions and scenarios on how comfortable our participants are with processing documents in the cloud. For this, our participants were presented with three different types of private data stored in cloud documents: children data including names and grades, health data including names and diagnosis, and financial data including names and SSNs. As additional modifier, participants were equally distributed across two conditions: "General" with a more generalized scenario and "Personal" with a more personalized scenario (e.g., "a child" vs. "your child").

We investigated participants' answers by computing the best performing model from multiple linear regressions (cf. Table 7.4). We find that neither the country nor the condition has a significant coefficient in the regression. Both the "Health data" scenario and the "Financial data" scenario are significantly rated as less comfortable by our participants than the "Child data" baseline. (cf. Figure 7.6)

> Summary: Scenario Perception.Our participants are uncomfortable the most with the scenario of processing financial documents in the cloud. Presenting a more personalized scenario did not significantly affect their comfort level.

7.5.7 Data Protection

In this question section, we asked two open questions to assess the amount of data our participants think cloud office suite providers collect when processing documents. Additionally, we asked our participants what security measures they think cloud providers deploy to protect their documents. Regarding data collection, most participants thought that cloud office suite providers collected the actual document content and metadata including the time and duration they used the cloud office application, IP addresses and filenames. A few participants were concerned that cloud providers would search their documents for keywords and report them to security agencies and law enforcement, e.g. P96 thinks that providers are "searching for specific keywords, most notably for US security reasons".

Most participants had very specific ideas of what security measures cloud office suite providers would deploy. The majority of our participants were convinced that providers would deploy encryption to protect their stored documents. For example, P74 believes that "the cloud servers are supposed to be encrypted and follow industry-standard protocols [...]" (P74). Similarly, participants mentioned access control and authentication, e.g., P79 hopes that "Security is handled by the service provider of the cloud office applications. They probably use complicated passwords and 2 factor authentication." (P79). Finally, some participants mentioned firewalls and other network security measures. P111 hopes that "they are protected by multiple firewalls [and] they are continuously monitored" (P111).

> Summary: Data Protection.Our participants identified encryption as their preferred security measure their cloud office suite should employ.

7.6 Related Work

> **Disclaimer:** This related work section reflects the state of prior research in mid 2020 and is provided to highlight the state of research at the time of this research project. For related and concurrent work at the time of this dissertation, see Chapter 3: Related and Concurrent Work.

As we conduct surveys investigating end-user security and privacy perceptions and expectations with cloud office suites, we discuss related work in the areas of security & privacy in the cloud and user studies within a context of cloud applications or storage.

Security & Privacy in the Cloud. A number of papers covers cloud security, client-side encryption or hiding layers to prevent third parties including the cloud office provider from accessing the content of any document edited in the cloud [359], [360]. The backup and restore performance, liabilities, and problems with data privacy of four cloud storage providers was examined by Hu et al. in 2010 [361]. Drago et al. investigated the traffic of DropBox in a measurement study and found possible performance bottlenecks [362]. In 2010 Svantesson and Clarke reviewed the terms of use of Google Docs finding that cloud computing is associated with risks to privacy and consumer rights [363].

Nestori et al. found in their 2018 paper, that Office 365 is not GDPR compliant [364]. Ramokapane et al. conducted a user study that found that users struggle to delete their data from the cloud, as incomplete or inaccurate mental models based on a lack of information on deletion lead to a failure to remove the data properly [365].

Cryptography alone is not enough to ensure privacy in cloud computing as Dijk et al. described. It also requires tamper-proof hardware and distributed computing [366]. MUBox introduced a meta-cloud storage application to help improve user collaboration on cloud storage services by introducing

activity views. Nebeling et al. conducted a user study with 16 participants to examine the accuracy and confidence with the activity views [367].

User Studies of the Cloud. Most user surveys in the cloud context focus on the storage aspect: Tan et al. investigated the acceptance of SaaS collaboration tools like Google Docs in an organizational setting and found that their intention to continue using these tools is positively affected by the perceived usefulness and satisfaction. [368]. Marshall et al. conducted a survey with 106 participants and 19 interviews to understand early user experiences and models of cloud storage systems, finding that users' misconceptions limit the ability to take full advantage of cloud features [369]. Burda et al. developed a technology acceptance model which incorporates users' perception of risk and trust and verified it in a study with 229 cloud storage users. They found evidence that trust in cloud archiving can be increased by a providers' reputation and user satisfaction. These findings could be used for marketing purposes. [370] Both Clark et al. and Khan et al. explored users' perception of file sharing status over time, finding a mismatch in user expectations and reality [371], [372].

Massey et al. conducted a qualitative study with 27 participants and identified four different strategies that teams used in shared repositories and suggested ways to improve existing tools with new technologies [373]. Mijuskovic et al. conducted a qualitative user study with 28 participants and found that most users are aware of security and privacy risks in the cloud, but lack knowledge to describe potential risks in detail [374].

We consider the following works by Ion et al. and Arpaci et al. closest to our surveys. Ion et al. studied privacy attitudes and beliefs towards consumer cloud storage by conducting interviews and a survey with end users in Switzerland and India, finding that requirements for consumer cloud storage differ from those of companies and that end users prefer local offline storage for sensitive data [375]. Arpaci et al. conducted a study with 200 pre-service teachers to understand the effects of security and privacy concerns of cloud computing in educational use and proposed a research model that indicates that security and privacy perception has a significant influence on students' attitudes towards cloud services [376].

Compared to these earlier studies consisting mostly of small-scale qualitative studies investigating the acceptance of cloud technology or larger studies focusing on cloud storage, our larger-scale study ($n = 200$) with both qualitative and quantitative parts investigates security and privacy explicitly in the context of cloud office apps.

7.7 Discussion

In this chapter, we explored the security and privacy perceptions and expectations of cloud office users, as well as their intuitions for how cloud office suites should ideally handle security and privacy. We performed two online surveys with 200 cloud office users from the U.S. and Germany to explore the following research questions:

RQ1: "How and why do our participants interact with cloud office applications?" The fairly recent shift from offline-only tools to cloud office suites includes immense changes of privacy and security assumptions, although the application design and end user experience remained mostly the same or even included new features. We find that a large majority of our participants regularly work on different document types in cloud office applications. Most common reason for using cloud office applications are the ease of sharing and the ease of usage without requiring installation of additional software.

RQ2: "What are end users' awareness, perceptions, and attitudes about privacy in cloud office applications?" Users seem to be aware of some general security implications, storage models, and access by

others, although some of their threat models seem underdeveloped (e.g., "I think it is not possible to hack the cloud."), likely due to lacking technical knowledge.

RQ3: "What are participants' mental models regarding protection and security of their cloud documents?" We find that users' mental models are incomplete and their understanding of cloud office security and privacy is limited, caused by a lack of transparency of the services' operations.
Our findings suggest that the current state of cloud office suites leaves much to be desired in the eyes of end users. General misconceptions and the unclear responsibilities of cloud providers might result in additional challenges for the end user adoption of cloud office suites.

7.7.1 Recommendations

Based on our findings, we offer recommendations for groups associated with cloud office suites.

For the industry: Since our participants were somewhat unsure about who actually has access to their documents (Section 7.5.3), we recommend changes to user interfaces and sharing policies that will improve their awareness. In case of unauthorized access we recommend notifications via email, as most of our participants prefer their provider to inform them this way (Section 7.5.5). They also identified encryption as their preferred security measure their cloud office suite should employ for improved security (Section 7.5.7).

For end users: There exist a number of self-hosted alternatives to cloud office applications such as Seafile or NextCloud that allow for most of the cloud conveniences while you still retain full control of your data (Section 7.3).

For policy makers: Our participants are somewhat unsure about who actually has access to their documents and about how many copies actually exist on which servers (Section 7.5.3). Privacy-focused policies such as GDPR could serve as a first step for improving security and privacy considerations for end users and could enable more privacy-friendly applications. In addition, data-at-rest and responsible disclosure policies could help with user wishes such as prefer encryption measures and notifications by email in case of unauthorized access (Sections 7.5.5, 7.5.7).

7.8 Summary

In the research described in this chapter, we conducted surveys with 200 cloud office users from Germany and the U.S. to investigate their experiences and behaviours with cloud office suites. We find that our participants prefer to store their documents on the platform they edit them with. While our participants agree on the benefits of cloud office applications free access and easy collaboration for remote documents, they prefer their local system in terms of security against unauthorized access. They are generally aware of security implications of processing their documents in the cloud, although some of their threat models are less developed.

Our participants have strong opinions on how comfortable they are with the access of certain parties, but are somewhat unsure about who actually has access to their documents. They have a clear idea that they should be informed about unauthorized access of their cloud documents by the provider via email. They also identified encryption as their preferred security measure their cloud office suite should employ.

Our participants are the most uncomfortable with the scenario of processing financial documents in the cloud (vs. health or child's data). Presenting a more personalized scenario did not significantly affect their comfort level.

Further Work. Our participants are generally aware of security implications of processing their documents in the cloud, although some threat models are less developed. Promising venues for future research could include investigations into general security and how privacy perception could be improved. Additional venues include the improvement of threat models and the effect of COVID-19 on the whole cloud office ecosystem. We hope our findings can help to inform future standards, regulations, implementations,and configuration options for cloud office applications.

This chapter presented research involving a survey with 200 cloud office users from Germany and the U.S., investigating their experiences and perceptions of cloud office suites. In the previous four chapters, I presented research investigating approaches, considerations, and encountered challenges in the context of security, privacy, and trustworthiness of the open source software supply chain. For this, I conducted research involving software stakeholders such as maintainers, contributors, developers, software architects, and end users. In the following chapter, I will provide a conclusion based on my findings, as well as highlight some potential future research avenues based on my research.

Chapter 8

Conclusion and Future Work

Concluding, the research I conducted for, and presented in this dissertation, provided valuable insights into processes, challenges, perceptions, and consideration around security and trust in the software supply chain in general, and the open source software ecosystem in particular.

First, I investigated the security and trust practices in open source projects by interviewing 27 owners, maintainers, and contributors from a diverse set of projects to explore their behind-the-scenes processes, guidance and policies, incident handling, and encountered challenges. My findings include that participants' projects are highly diverse in terms of their deployed security measures and trust processes, as well as their underlying motivations. Secondly, to focus more on the consumer side of the open source software supply chain, I investigated the use of open source components in industry projects. For this, I interviewed 25 software developers, architects, and engineers with industry experience to understand their projects' processes, decisions, and considerations in the context of external open source code. My findings include that open source components play an important role in many of the projects, and that most projects have some form of company policy or best practice for including external code. Thirdly, I presented a study investigating the use of software obfuscation in Android applications, which is a recommended practice to protect against plagiarism and repackaging. The study leveraged a multi-pronged approach including a large-scale measurement, a developer survey, and a programming experiment. My findings include that only 24.92% of apps are obfuscated by their developer, that developers are generally aware of, but do not fear theft of their own apps, and have difficulties obfuscating their apps. Lastly, to include end users themselves, I conducted a survey with 200 users of cloud office suites to investigate their security and privacy perceptions and expectations, with findings suggesting that users are generally aware of basic security implications but lack technical knowledge to envision some of the more advanced threat models. Participants also had strong opinions on the access of their data by certain parties such as the Government, but were generally unsure about who actually has access to their documents.

Across this whole dissertation, my key findings include that open source projects have highly diverse security measures, trust processes, and underlying motivations. That open source projects' security and trust needs are probably best met in ways that consider their individual strengths, limitations, and project stage, especially for smaller projects with limited access to resources. And that open source components play an important role in industry projects, and that those projects often have some form of company policy or best practice for including external code, but developers wish for more resources to better audit included components. My findings emphasize the importance of collaboration and shared responsibility in building and maintaining the open source software ecosystem, especially with the following greater themes in mind:

Open source projects are an integral part of a larger interconnected ecosystem, consisting of various components, libraries, and software registries. This ecosystem can be understood as a community of communities, and supporting individual open source projects in different growth stages and communities is necessary for the security and health of the entire open source ecosystem. The development of

open source projects, according to the input provided by my interviews with open source stakeholders, is highly community-driven and practical. Key decisions such as release schedules, announcements, and distribution infrastructure are all based on the feedback, needs, and contributions of project users and contributors. In most cases, security and trust incidents appear to be handled reactively, as they occur. This approach stems from a pragmatic strategy, given the limited resources and personnel of smaller open source communities. Attempting to anticipate all possible incident types beforehand, particularly for smaller projects with frequently changing committers and structures, is not a viable option. However, it is important to note that small projects may benefit from more proactive support in managing security and trust incidents. This support could come in the form of public, general example playbooks and resources that projects can utilize when incidents arise. The availability of such resources could help smaller projects navigate the challenges of security and trust incidents more effectively, without the need for elaborate incident playbooks and committer structures that may be difficult to implement and maintain.

Treating the open source ecosystem as just another supply chain for software can lead to unpleasant surprises for both companies and the community. For companies, reliance on open source components may result in unexpected issues if the maintainer abandons the project or fails to implement necessary features. On the other hand, open source communities may be mistreated as a cheaper support desk or an alternative to in-house development teams, leading to resentment and frustration. Unlike commercial software supply chains, open source generally do not operate on contracts, providing no warranty or support promises in the case of vulnerabilities. If a company cannot provide an equivalent value exchange in monetary terms for utilizing open source components, they might consider offering some of their developer time or code back to the open source ecosystem. With the great power provided to industry by utilizing freely available open source components in their software also comes the great responsibility of keeping the open source ecosystem reliable, healthy, and secure.

The analogy of a software supply chain is generally used in the context of library and package relations in software. This can be somewhat misleading, as the "chain" part of the analogy implies linear relations with clear producer and consumer endpoints and some intermediary links. However, this simplistic view fails to capture the complex reality of the software supply chain, particularly in the case of open source software. A more appropriate analogy would be that of a tangled bowl of spaghetti, with numerous intertwined strands that make it impossible to distinguish the beginning and endpoints, even when attempting to pull on individual strings. Some of the companies included in my research have addressed this issue by focusing exclusively on the security aspects of their specific software supply chain. They achieve this by maintaining in-house versions or caches of the open source software they use, thereby separating themselves from many potential attack vectors in the larger software supply chain. This approach also allows them to scrutinize and audit components more thorough locally, but introduces challenges in terms of contributing back to, and keeping up with, the open source ecosystem.

During my research, participants from industry generally held positive attitudes towards open source components, with many reporting benefits such as reduced maintenance burden, faster iterations, and access to open communities and code. However, this exchange can become one-sided, particularly as it is not always possible for both companies and open source communities to exchange money as a value equalizer. To address this issue, companies might consider approaching the open source ecosystem as a community of communities, rather than a software supplier. This might involve open sourcing internal components where feasible, providing guidance and assistance to open source projects, and contributing back to the community when possible. Supporting open source communities can also benefit the software industry by helping to cultivate developer talent. By enabling and supporting open source projects included in their software stack, companies allow a global community of developers

to learn from and participate in their technology stack. This expands access to industry technologies, allowing more people to become expert developers and contributing to the growth of the industry as a whole.

In summary, the open source ecosystem as a community of many communities requires different approaches to ensure the security and health of the overall ecosystem. Companies and other stakeholders should be prepared to contribute back to the community and maintain good relationships with maintainers. This way, both sides can benefit from the advantages of open source software while avoiding potential problems down the line. Overall, I argue that collaboration between developers, maintainers, end users, researchers, and other stakeholders alike is necessary to ensure that the open source ecosystem remains a secure, trustworthy, and reliable resource for everybody.

8.1 Future Work

The future of securing the software supply chain involves recognizing the importance of human factors and the individual software experts. While securing dependencies and build systems is a necessary step towards a secure and reliable ecosystem, recent attacks have shown that these experts are a, if not the most, vulnerable link.

An important benefit of the software supply chain is that it allows developers to leverage third-party dependencies as building blocks for their software, e.g., from package repositories like npm or PyPI. However, the reliance on external repositories also introduces new attack surfaces, as recent typo-squatting and account-takeover attacks have shown [71], [269]. In response, Python's PyPI package repository and GitHub started requiring 2FA for developer accounts with critical projects. While this approach was implemented to increase the overall security of package repositories and dependencies, such approaches also have an impact on usability. If 2FA is required, but the authentication process is too complicated or time-consuming, maintainers may try to find ways to bypass it, which would undermine the intended security benefits. By examining the common challenges and usage patterns involved in using and providing external dependencies, future work could identify ways to improve the adoption of security processes, ultimately benefiting the security of the software supply chain.

Build systems and continuous integration and continuous delivery (CI/CD) pipelines can interact and chain with other systems and third-party services, allowing for the creation of complex, multi-step build and distribution processes for software. But this complexity also increases the risk of misuse, misconfiguration, or leakage of secrets, and as software is increasingly being built and deployed using third-party services, these services are becoming high-value targets for attackers seeking to infect all customers and compromise the software supply chain. As build systems and CI/CD pipelines are becoming increasingly complex, it's essential to not overlook the human factor in setting up, maintaining, and using them. Usability plays an important role in ensuring that developers can effectively and securely utilize and manage these complex systems: By establishing what makes these usable for stakeholders, such as clear documentation, user-friendly interfaces, and effective training materials, future research could reduce the risk of misconfiguration and other vulnerabilities while allowing developers to maintain productivity and workflow efficiency.

Metrics and frameworks that allow for structured reporting of security vulnerabilities, attacks, or coding practices play an important role in software supply chain stakeholders' communication. Overall adoption is an important factor in the effectiveness of any metric or framework. If these tools become widely adopted, they might create a network effect, whereby stakeholders become familiar with the metric and share a common understanding of what constitutes secure software development practices. As more stakeholders adopt and utilize a particular metric or framework, they build a collective under-

standing of what it takes to develop secure software according to the metric, leading to a higher level of standardization and consistency in secure software development practices. Designing tooling for and around these metrics, as well as their general usability, are important in making them effective and widely accepted. Without proper consideration of the human factor, these tools may not be utilized to their fullest potential or even adopted at all. Additionally, when more stakeholders utilize these tools, they can provide feedback on how to further improve their usability, resulting in a continuous improvements and adaptions. By investigating and improving metrics' and frameworks' usability, future work would increase the likelihood that stakeholders will utilize these tools and ultimately establish a common understanding towards a more secure software ecosystem.

Some open source components are so ubiquitous and deeply embedded in the development and operation of IT systems that their existence is hardly ever noticed by the involved stakeholders. Abandonment or lack of maintenance of these essential components would wreak havoc on the software supply chain, exposing vulnerabilities and potentially resulting in large costs and other damages for stakeholders. This is specifically true for many open source projects, which are often developed by small teams or even single developers as hobbies. These small projects might also be at risk of deprecation, as they often do not contributed significantly to the income of their main developers. Future research into how to better identify, secure, and support these small projects and the people behind them.

To conclude, future work needs to consider human factors to secure the open source software ecosystem and related software supply chain. Recent attacks have demonstrated that developers are a vulnerable link in the software supply chain in general, and the open source ecosystem in particular. This makes approaches which consider the human factor an important step towards effective software supply chain security, trustworthiness, and reliability.

Appendix A

Security & Trust in Open Source Software Projects

A.1 Interview Guide in English

[General Greeting of Participant]

Intro

- **Thanks**: Thank you very much for offering your valuable time for this interview. We are very grateful for your contribution.
- **Ready**: Are you ready to start the interview?
- **Structure**: First of, I am going to talk about the context and data handling, and if you agree with everything, we would then start with the interview.

Context

- **We**: We are a researchers at the CISPA Helmholtz Center for Information Security (short intro).
- **Our research**: focuses on the area Usable Security for Developers (short overview).
- **In the past**: Open Source Projects as data source. Open Source code and projects are at the foundation of many software ecosystems. Open Source also has unique challenges, such as changing contributors and trust requirements.
- **Now**: "How can we empower open source contributors to build more secure projects."
- This interview as a start/exploration of internal processes and decisions often not visible at the repository level.
- For this interview:
  - We are not judging security or privacy of a project, we are just interested in the underlying structures and processes.
  - Projects are often very complex, if you don't know the answer, just say "next".
  - We are not just interested in structures, but also your personal opinions and experiences.
- **Questions?** Any questions about this interview context?

Consent

- **Voluntary**: Your responses in this interview are entirely voluntary, and you may refuse to answer any or all of the questions in this interview.

- **Duration**: Duration of the interview depends a bit on the duration of your answers, in our experience so far about 30 to 45 minutes.
- We will fully anonymize you and your projects in any publication and only include short quotes.
- We will send you a preprint before a potential publication, so you can veto quote usage etc.
- **Recording**: We would like to record this interview so that we can transcribe the answers later
  - The recording will be destroyed when we transcribed the answers (a few days)
  - The transcripts will be destroyed after analysis (a few months)
- **Questions?** Any more questions about data handling or recording?
- "The recording is now on"
- **Restate** consent question

S1: Project and Demographics

[Check project(s) beforehand]
S1Q1 **Project**: Can you tell us a bit about the [project(s)] you are involved in?
Follow-Ups:

- S1Q1.1 **About**: What is the project about? What is it's purpose?
- S1Q1.2 **Age**: When was the project created?
- S1Q1.3 **Contributors**: How many regular contributors does the project have?
- S1Q1.4 **Connection**: How do contributors know each other? (Virtually, Personally)
- S1Q1.5 **Distribution**: How are the contributors distributed geographically?

S1Q2 **Project Relation** How are you related to [project]?
Follow-Ups:

- S1Q2.1 **Join**: When did you join the project?
- S1Q2.2 **Role**: What is your role in the project?

S2: Incidents

[Mention "hypocrite commits" incident: If participant is aware continue, else introduce (see Interview Guide Appendix)]

- S2Q1 **Opinion**: What do you think about this incident?
- S2Q2 **Challenges**: Can you remember any security challenges that your project faced in the past?
  Nudges:
  - malicious committers/commits,
  - security issues with your repository (software/provider)
  - tool chain, etc.?

S3: Guidance

S3Q1 **Guidance**: Are there guides/best practices/hints available for contributors, maintainers, etc.?
[IF NOT Guidance]:

- What are your thoughts about including guides/best practices/hints for contributors, maintainers, etc.?

Follow-Ups:

- S3Q1.1 **Infrastructure**: Does your project have security guidelines for configuring/running infrastructure? (cloud, VCS, etc.)
- S3Q1.2 **Languages**: Does your project use security or style guidelines for the utilized programming languages? What do they cover?
- S3Q1.3 **Cryptography**: If you're using cryptography in your code: Do you have a guide on how to use cryptography? (forbidden functions etc.)

S4: Security Policies

S4Q1 **Security Policies**: What do your security policies contain?
[IF NOT Policies]:

- What would you like your security policies to contain?

Follow-Ups:

- S4Q1.1 **Content**: What specific parts do they cover?
- S4Q1.2 **Applicability**: Do these have to be read and acknowledged by committers/contributors?

S4Q2 **Disclosure Policies**: What is the (coordinated) disclosure policy of the project?
S4Q3 **Security Incidents**: How are security incidents/issues handled?
Follow-Ups:

- S4Q3.1 **Policy**: By what policy?
- S4Q3.2 **Who**: By whom? (specific security team?)
- S4Q3.3 **Access**: Private/Public?
- S4Q3.4 **Process**: Are there "Playbooks" for Incident Response and Vulnerability Management? (What do these specify?)
- S4Q3.5 **History**: How was this process developed?

S4Q4 **Security Testing / Reviews**: What measures do you have to test security?
Follow-Ups:

- S4Q4.1 **Aspects**: What aspects of security are checked and how?
- S4Q4.2 **Tools**: Is the project using specific (software) tools? (SAST, DAST, Manual, Pentests)
- S4Q4.3 **Project Stages**: At what stages of the project? (Only initially, on changes, etc.)
- S4Q4.4 **Frequency**: How often are manual security reviews done? Pentesting etc.? Who carries out these reviews? (skills, external/internal person)
- S4Q4.5 **Threat Modeling**: Is some form of Threat Modelling used?

S5: Project Structure

S5Q1 **Repository**: What does the general repository structure look like? (Filesystem, Stages, CI etc.)
Follow-Ups:

- S5Q1.1 **Stages**: What stages do exist? (Code → Commit → PRs → Review → CI for tests and Build → Deployment, ...)
- S5Q1.2 **Control**: Who controls which stages?
- S5Q1.3 **Main**: How are branches setup? Is it possible to directly push to the main branch?
- S5Q1.4 **Systems**: How are Build and Deployment systems secured? Who has access/control?
- S5Q1.5 **PRs**: How are incoming pull requests handled?

- S5Q1.6 **Signed Commits**: Are commits signed? (Requires PGP key, how are those trusted?)
- S5Q1.7 **Secret Management**: Does the project use some form of secret management system?
- S5Q1.8 **Access**: Who has access to those systems?

S5Q2 **Supply Chain**: What does the setup for the Supply Chain (e.g., libraries and other dependencies) look like?
Follow-Ups:

- S5Q2.1 **Criteria**: What criteria are considered when deciding on external libraries and dependencies?
- S5Q2.2 **Checks**: How are the processes to check if those are secure and trusted?
- S5Q2.3 **Vulnerabilities**: How is checked whether a dependency has security vulnerabilities? (SAST, code reviews, checking open source projects itself and its contributors, etc.)
- S5Q2.4 **Signed**: Do libraries have to be cryptographically signed?
- S5Q2.5 **Private Packages**: Does the project use a private package repository with vetted and secure dependencies?

S5Q3 **Infrastructure**: Does the project have additional infrastructure such as a project website or chat tools?
Follow-Ups:

- S5Q3.1 **Access**: Who controls the additional infrastructure? Same set of maintainers for all infrastructure?

S6: Release and Updates

S6Q1 **Releases**: How are releases and updates published?
Follow-Ups:

- S6Q1.1 **Decision**: How is decided if/when a security update is released?
- S6Q1.2 **Secured**: What security concepts are considered for releases? Are releases "secured" in any way?
- S6Q1.3 **Update System**: Are security updates made automatically? How does the update system work?
- S6Q1.4 **Deprecation**: Do you publish information about deprecated / insecure versions?

S7: Roles and Responsibilities

S7Q1 **Contributors**: Can you tell us a bit about the maintainer / contributor hierarchy of the project?
Follow-Ups:

- S7Q1.1 **Security Roles**: Are there security specific roles in your projects?
- S7Q1.2 **Roles known**: Are these groups/roles common knowledge?

S8: Trusting Contributors

S8Q1 **Trust**: Can you tell us a bit about the trust model of the project?
Follow-Ups:

- S8Q1.1 **Establish**: How do you establish trust for new committers?
- S8Q1.2 **Identity**: Do you have some form of identity check? (e.g., by being coworkers, at conferences, etc.)

- S8Q1.3 **Authentication**: Do you authenticate committers? (How?)
- S8Q1.4 **Access Control**: Are you using some form of access control in your project?
- S8Q1.5 **Trusted**: How could a new contributor become trusted members of the project / team?

S8Q2 **License**: Is there a Contributor License Agreement?

- **Yes**: What does it look like?
- **No**: Do you know why the project doesn't have one?

S8Q3 **Public**: Does the project maintain a public list of contributors and their contributions?

S9: Untrustworthy Contributors

S9Q1 **Trust Incidents**: Are you aware of any contributors that turned out to be not trustworthy?
[IF NO]:

- Assuming there is an untrustworthy contributor ...

Follow-Ups:

- S9Q1.1 **Approach**: How did the project deal with such a situation?
- S9Q1.2 **Excluding**: If applicable, can you explain the process for excluding contributors from the project?
- S9Q1.3 **Identifying**: How does the process for identifying untrustworthy contributors look like?

S9Q2 **Trust Strategy**: What is the project's strategy to deal with contributors who become untrustworthy?
Follow-Ups:

- S9Q2.1 **Who**: Who makes the decision? (BDFL, committee, maintainers)
- S9Q2.2 **Playbook**: Is there a playbook/defined process?
- S9Q2.3 **Circumstances**: Under which circumstances are untrustworthy committers excluded from future contributions?
- S9Q2.4 **Process**: What does the exclusion process look like?
- S9Q2.5 **Investigation**: Does the project have a defined process to investigate potential vulnerable contributions?

S9Q3 **Removal**: Did the project decide to remove their contributions? Follow-Ups:

- S9Q3.1 **Decision**: How was this decision made?
- S9Q3.2 **Process**: What did the removal process look like?

S10: Problems and Improvements

S10Q1 **Reputation**: What is, in your personal opinion, the reputation of the project in terms of security and trust?
Follow-Ups:

- S10Q1.1 **Internal**: Internal reputation.
- S10Q1.2 **External**: External reputation.

S10Q2 **Improvements Project**: Assuming no limitations whatsoever (e.g., monetary or in terms of people-power), how would you personally like to improve security and trust in the project?
Follow-Ups:

- S10Q2.1 **Problems**: Where do you see problems in the current system?
- S10Q2.2 **Why Exist**: What do you think are the reasons for the current (trust) system to be the way it is?
- S10Q2.3 **Improvements**: What would you like to improve?

Outro

- "The recording is now off"
- Thank the participant again for their valuable time
- We will be in contact for a preprint

Debrief

- Is there something that we did not cover during the interview but you would like to talk about?
- Do you know of any other unique projects or persons we could invite for an interview?

Incident: Hypocrite Commits

In April 2021, after receiving "poor quality patches" Linux kernel developer Greg Kroah-Hartman reverted 68 patches submitted by University of Minnesota email addresses and announced that all further patches coming from University of Minnesota addresses should be summarily rejected by default.

This incident actually started in August 2020, when a number of "bad faith" patches were sent to Linux kernel developers by University of Minnesota researchers under false identities. These patches were part of an ongoing work studying the feasibility of introducing vulnerabilities into open source software projects through minor patches ("hypocrite commits")

A.2 Interview Guide in German

[Allgemeine Begrüßung]

Intro

- Du vs. Sie [Dieser Guide nutzt von hier an "Du"]
- **Thanks**: Wir bedanken uns vielmals für deine bisher aufgewendete Zeit und dem Interesse an unserem Interview.
- **Ready**: Bist du bereit, mit dem Interview zu beginnen?
- **Structure**: Zunächst werde ich was zum Hintergrund des Interviews und der Verwendung von Daten sagen, und nur wenn du mit allem zustimmst, werden wir dann mit dem eigentlichen Interview beginnen.

Context

- **We**: Wir sind Wissenschaftler am CISPA Helmholtz Zenter (kurze Einführung).
- **Our research**: Unsere Forschung ist im Bereich Nutzbare IT-Sicherheit (kurze Übersicht).
- **In the past**: Open-Source-Projekte als Datenquelle.

- Projekte sind ein wichtiger Software-Baustein, aber bringen auch besondere Herausforderungen.
- **Now**: "Wie können wir im speziellen Open Source Contributorn helfen, ihre Open Code Projekte sicherer zu machen."
- **Dieses Interview als Basis**: Explorative Projekte und Abläufe über Dateien/Repos hinaus anschauen (insbesondere weil viele Abläufe nicht direkt im Repo erkennbar sind).
- Im Speziellen für das Interview heißt das:
  - Dies ist keine Bewertung von Sicherheitsabläufen oder ähnlichem, wir sind lediglich an den Strukturen interessiert.
  - Projekte können sehr komplex sein, wenn keine Antwort bekannt: "weiter".
  - Nicht nur stumpfe Abfrage von Daten, wir haben die Form eines Interviews gewählt, weil wir auch an persönlichen Meinungen und Einschätzungen interessiert sind.
- **Questions?** Fragen zum Interview-Hintergrund?

Consent

- **Voluntary**: Dieses Interview ist komplett freiwillig, zu jedem Zeitpunkt kann das Interview beendet oder Fragen übersprungen werden
- **Duration**: Länge des Interviews hängt von der Ausführlichkeit der Antworten ab, unserer bisherigen Erfahrung nach zwischen 30 und 45 Minuten.
- Wir anonimisieren dich und alle Projekte, sowie verwenden wenn überhaupt nur kurze Zitate
- Bevor wir irgendwas veröffentlichen, senden wir dir/Ihnen eine Kopie zum Vetoen.
- **Recording**: Interview Aufnahme
  - Diese Aufnahme wird nach dem transkribiren zerstört (ein paar Tage)
  - Die Transkripte werden nach der Auswertung zerstört (ein paar Monate)
- **Questions?** Fragen zur Datenverarbeitung oder Aufnahme?
- "Die Aufnahme läuft jetzt"
- **Restate**: Stimmst du unter diesen Bedingungen zu einer Aufnahme zu?

S1 Project & Demographics

S1Q1 **Project**: Kannst du uns ein wenig zu deinem Projekt erzählen?
Follow-Ups:

- S1Q1.1 **About**: Worum geht es bei dem Projekt? Was ist der Zweck des Projekts?
- S1Q1.2 **Age**: Wann ist das Projekt entstanden?
- S1Q1.3 **Contributors**: Wie viele reguläre Contributoren hat das Projekt?
- S1Q1.4 **Connection**: Wie kennen sich die Contributoren? (Virtuell, Persönlich)
- S1Q1.5 **Distribution**: Wie verteilen sich die Contributoren geographisch?

S1Q2 **Project Relation**: Was ist oder war dein Zusammenhang zum Projekt?
Follow-Ups:

- S1Q2.1 **Join**: Wann bist du zum Projekt gestoßen?
- S1Q2.2 **Role**: Was war deine Aufgabe in dem Projekt?

S2 Incidents

[Erwähne "hypocrite commits" Vorfall: Fahre fort falls bekannt, ansonsten kurze Einführung (siehe Interview Guide Appendix)]
S2Q1 **Opinion**: Was hältst du von diesem Vorfall?
S2Q2 **Challenges**: Kannst du dich an irgendwelche IT Sicherheits-Herausforderungen bei deinem Projekt erinnern?
Nudges:

- Boshafte Committers/Commits,
- Sicherheits-Probleme mit deinem Repository (Software/Provider)
- Software-Werkzeuge, etc.?

S3 Guidance

S3Q1 **Guidance**: Gibt es Leitfäden/Best Practices/Hinweise für Contributoren, Maintainers, usw.?
[IF NOT Guidance]:

- Was ist deine Meinung dazu, Leitfäden/Best Practices/Hinweise für Contributoren, Maintainers, usw. bereitzustellen?

Follow-Ups:

- S3Q1.1 **Infrastructure**: Hat das Projekt Richtlinien für die Konfiguration und Ausführung der Infrastruktur? (Cloud, VCS, usw.)
- S3Q1.2 **Languages**: Hat das Projekt Sicherheits- oder Stil-Richtlinien für die verwendeten Programmiersprachen? Was decken die Richtlinien ab?
- S3Q1.3 **Cryptography**: Falls Kryptographie verwendet wird: Hat das Projekt Richtlinien bezüglich der Nutzung von Kryptographie? (verbotene Funktionen etc.)

S4 Security Policies

S4Q1 **Security Policies**: Was enthalten die Sicherheits-Richtlinien?
[IF NOT Policies]:

- Was sollten deiner Meinung nach die Richtlinien enthalten?

Follow-Ups:

- S4Q1.1 **Content**: Welche Bereiche decken sie ab?
- S4Q1.2 **Applicability**: Müssen die Richtlinien von Committern/Contributoren gelesen und anerkannt werden?

S4Q2 **Disclosure Policies**: Was ist die (koordinierte) Disclosure Policy des Projekts?

S4Q3 **Security Incidents**: Wie wird mit Sicherheits-Vorfällen verfahren?
Follow-Ups:

- S4Q3.1 **Policy**: Nach welcher Richtlinie?
- S4Q3.2 **Who**: Von wem? (extra IT Sicherheitsteam?)
- S4Q3.3 **Access**: Private/Public?
- S4Q3.4 **Process**: Gibt es "Playbooks" für die Reaktion auf Vorfälle oder Schwachstellen? (Was enthalten diese?)
- S4Q3.5 **History**: Wie ist dieser Prozess entstanden?

S4Q4 **Security Testing / Reviews**: Welche Verfahren für Sicherheits-Tests gibt es?
Follow-Ups:

- S4Q4.1 **Aspects**: Welche Aspekte decken diese Tests ab, und wie?
- S4Q4.2 **Tools**: Nutzt das Projekt spezielle (Software-)Tools? (SAST, DAST, Manual, Pentests)
- S4Q4.3 **Project Stages**: Bei welchen Phasen des Projekts? (Nur zu Beginn, bei Änderungen, etc.)
- S4Q4.4 **Frequency**: Wie häufig werden manuelle Sicherheits-Bewertungen durchgeführt? Pentesting etc.? Wer führt die Bewertungen durch? (Skills, externe/interne Person)
- S4Q4.5 **Threat Modeling**: Wird eine Art von Threat Modelling verwendet?

S5 Project Structure

S5Q1 **Repository**: Wie sieht die Repository-Struktur aus? (Filesystem, Stages, CI etc.)
Follow-Ups:

- S5Q1.1 **Stages**: Welche Projekt-Stufen gibt es? (Code → Commit → PRs → Review → CI for tests and Build → Deployment, ...)
- S5Q1.2 **Control**: Wer kontrolliert welche Stufe?
- S5Q1.3 **Main**: Wie sind Branches aufgesetzt? Ist es möglich, direkt auf den Main Branch zu pushen?)
- S5Q1.4 **Systems**: Wie sind Build und Deployment System gesichert? Wer hat Zugriff/Kontrolle?
- S5Q1.5 **PRs**: Wie wird mit neuen Pull Requests verfahren?
- S5Q1.6 **Signed Commits**: Sind Commits signiert? (Benötigt PGP-Schlüssel, wie wird diesem vertraut?)
- S5Q1.7 **Secret Management**: Nutzt das Projekt eine Art von Secret Management System?
- S5Q1.8 **Access**: Wer hat Zugriff auf diese Systeme?

S5Q2 **Supply Chain**: Wie sieht das Setup für die Supply Chain (bspw. Bibliotheken und andere Abhängigkeiten) aus?
Follow-Ups:

- S5Q2.1 **Criteria**: Welchen Kriterien werden bei der Auswahl von externen Bibliotheken und Dependencies berücksichtigt?
- S5Q2.2 **Checks**: Nach welchen Prozessen wird deren Sicherheit und Vertrauen etabliert?
- S5Q2.3 **Vulnerabilities**: Wie wird überprüft, ob eine Dependency Sicherheitslücken hat? (SAST, code reviews, Überprüfung von Open-Source-Projekten und Contributoren, etc.)
- S5Q2.4 **Signed**: Müssen Bibliotheken kryptographisch signiert sein?
- S5Q2.5 **Private Packages**: Nutzt das Projekt ein privates Package Repository mit geprüften und sicheren Dependencies?

S5Q3 **Infrastructure**: Hat das Projekt zusätzliche Infrastruktur wie eine Webseite oder Chat-Anwendungen?
Follow-Ups:

- S5Q3.1 **Access**: Wer hat Zugriff auf die zusätzliche Infrastruktur? Selbes Set von Maintainern wie die gesamte Infrastruktur?

S6 Release and Updates

S6Q1 **Releases**: Wie werden Releases und Updates veröffentlicht?
Follow-Ups:

- S6Q1.1 **Decision**: Wie wird entschieden, ob/wann ein Sicherheits-Update veröffentlicht wird?
- S6Q1.2 **Secured**: Welche Sicherheitskonzepte werden für Releases berücksichtigt? Sind Releases in irgendeiner Weise "gesichert"?
- S6Q1.3 **Update System**: Werden Security-Updates automatisch aufgespielt? Wie funktioniert das Update-System?
- S6Q1.4 **Deprecation**: Werden Information über deprecated / unsichere Versionen veröffentlicht?

S7 Roles and Responsibilities

S7Q1 **Contributors**: Kannst du uns was zur Maintainer / Contributor Hierarchie im Projekt erzählen?
Follow-Ups:

- S7Q1.1 **Security Roles**: Gibt es sicherheits-spezifische Rollen im Projekt?
- S7Q1.2 **Roles Known**: Sind diese Rollen / Gruppen allgemein bekannt?

S8 Trusting Contributors

S8Q1 **Trust**: Kannst du uns was zum Vertrauens-Modell des Projektes erzählen?
Follow-Ups:

- S8Q1.1 **Establish**: Wie wird das Vertrauen in neue Committern etabliert?
- S8Q1.2 **Identity**: Gibt es eine Art von Identitätsüberprüfung? (bspw. für Kollegen, bei Konferenzen, etc.)
- S8Q1.3 **Authentication**: Werden Committer authentifizier? (Wie?)
- S8Q1.4 **Access Control**: Wird eine Art von Access Control verwendet?
- S8Q1.5 **Trusted**: Wie können neue Committer vertraute Mitglieder des Projektes werden?

S8Q2 **License**: Gibt es eine Contributor License Agreement?

- **Yes**: Was enthält sie?
- **No**: Weißt du, warum das Projekt keine hat?

S8Q3 **Public**: Maintained das Projekt eine öffentliche Liste von Contributoren und deren Beiträgen?

S9 Untrustworthy Contributors

9Q1 **Trust Incidents**: Weißt du von Contributoren, die sich als nicht vertrauenswürdig herausgestellt haben?
[IF NO]:

- Angenommen es gäbe einen nicht vertrauenswürdigen Contributor ...

Follow-Ups:

- S9Q1.1 **Approach**: Wie ist das Projekt mit dieser Situation umgegangen?
- S9Q1.2 **Excluding**: Falls zutreffend, wie läuft der Prozess für einen Projekt-Ausschluss von Contributoren ab?
- S9Q1.3 **Identifying**: Wie sieht der Prozess zum Identifizieren von nicht vertrauenswürdigen Contributor aus?

S9Q2 **Trust Strategy**: Wie sieht die Projekt-Strategie für nicht vertrauenswürdige Contributors aus?
Follow-Ups:

- S9Q2.1 **Who**: Wer trifft die Entscheidung? (BDFL, committee, maintainers)
- S9Q2.2 **Playbook**: Gibt es ein Playbook / definierten Prozess?
- S9Q2.3 **Circumstances**: Unter welchen Umständen werden nicht vertrauenswürdige Contributoren von zukünftigen Beiträgen ausgeschlossen?
- S9Q2.4 **Process**: Wie sieht der Ausschlussprozess aus?
- S9Q2.5 **Investigation**: Hat das Projekt einen definierten Ablauf, um möglicherweise unsichere Beiträge zu untersuchen?

S9Q3 **Removal**: Hat sich das Projekt entschieden, deren Beiträge zu entfernen?
Follow-Ups:

- S9Q3.1 **Decision**: Wie wurde die Entscheidung getroffen?
- S9Q3.2 **Process**: Wie sah der Ablauf der Entfernung aus?

S10 Problems and Improvements

S10Q1 **Reputation**: Was ist, deiner persönlichen Meinung nach, der Ruf des Projekts im Kontext von Sicherheit und Vertrauen?
Follow-Ups:

- S10Q1.1 **Internal**: Interner Ruf.
- S10Q1.2 **External**: Externer Ruf.

S10Q2 **Improvements Project**: Angenommen es gäbe keine Einschränkungen (Geld, Arbeitszeit), wie würdest du persönlich am liebsten die Sicherheit / das Vertrauen des Projektes verbessern?
Follow-Ups:

- S10Q2.1 **Problems**: Wo siehst du die Probleme im aktuellen System?
- S10Q2.2 **Why Exist**: Was denkst du sind die Gründe für das aktuelle System
- S10Q2.3 **Improvements**: Was würdest du verbessern?

Outro

- "The recording is now off"
- Thank the participant again for their valuable time

Debrief

- Gibt es etwas, das wir während des Interviews nicht angesprochen haben, über das Du aber gerne gesprochen hättest?
- Falls angemessen: Kennst du noch weitere Personen, die für ein solches Interview potentielle Teilnehmende sein könnten?

A.3 Codebook

C1 Project Demographics

C1.1 Project Type
Description: General code for project type(s) of the participant.
Subcodes include currently:

- C1.1-1: Operating System
- C1.1-2: Library
- C1.1-3: Virtualization/containers
- C1.1-4: Code analysis
- C1.1-5: Hybrid engineering
- C1.1-6: WebApp/Backend
- C1.1-7: Parser/Serialization/Deserialization
- C1.1-8: Shared libraries
- C1.1-9: Version Control System
- C1.1-10: UI Tool
- C1.1-11: Orchestration:
- C1.1-12: Network Monitoring
- C1.1-13: Scientific simulations
- C1.1-14: Decentralized exchange (crypto)
- C1.1-15: CLI Tool
- C1.1-16: Network Protocol

You may extend this list if you identify a new project type.
Buzzwords: "I work on X, a library for ..."
Coding: Add corresponding subcodes for each project mentioned.

C1.2 Project Age
Description: Age of the project.
Coding: Code (if) estimated by participant + enhance with actual repo age if available

C1.3 Contributor Count
Description: Number of (regular) contributors to the project.
Coding: Code (if) estimated by participant + enhance with actual repo data if available

C1.4 Contributor Connections
Description: How the contributors are connected to each other.
Coding: Add corresponding subcodes, when in doubt: likely random "Contributors"

C1.5 Contributor Distribution
Description: How the contributors are distributed.
Coding: Add corresponding subcodes, when in doubt: likely "Global"

C1.6 Participant Position
Description: Rough estimate of our participant's position within the project (estimate from project data if not mentioned by participant).
Subcodes include currently (roughly ordered by rank):

- C1.6-1: Founder or Owner (or equiv.)
- C1.6-2: Team Lead (or equiv.)
- C1.6-3: Maintainer (or equiv.)
- C1.6-4: Regular contributor

You may extend this list if you identify a new position.

Coding: Add corresponding subcodes.

C2 Incidents

C2.1 Security Challenges
Description: If/What security challenges the project faced.
Subcodes include currently:

- C2.1-1: None
- C2.1-2: Suspicious/Low Quality Commits (but no obvious attack)
- C2.1-3: Social Engineering
- C2.1-4: Vuln in dependency
- C2.1-5: Other
- C2.1-6: Full disclosure of vulnerabilities
- C2.1-7: Unsafe user input
- C2.1-8: Loss of credentials

You may extend this list if you identify a new incident type.
Coding: Add corresponding subcodes

C2.2 Incident Aware
Description: If the participants were aware of the incident.
Subcodes include currently:

- C2.2-1 Aware: No
- C2.2-2 Aware: Yes

Coding: Add corresponding subcodes

C2.3 Incident Opinion
Description: What opinion the participant had of the research approach mentioned in the incident.
Subcodes include currently:

- C2.3-1 Opinion: No opinion / Refuse to answer
- C2.3-2 Opinion: Negative
- C2.3-3 Opinion: Neutral / Mixed
- C2.3-4 Opinion: Positive

Coding: Add corresponding subcodes

C3 Guidance

C3.1 Guidance Types
Description: If/what type of guidance the project(s) provide.
Subcodes include currently:

- C3.1-1: None

- C3.1-2: Language (e.g. style)
- C3.1-3: Security
- C3.1-4: Crypto-specific
- C3.1-5: Infrastructure
- C3.1-6: General (contributing)
- C3.1-7: Test-specific

You may extend this list if you identify a new type.
Coding: Add corresponding subcodes

C4 Security Policies

C4.1 Policies Content
Description: If/what type of security policies the project(s) provide.
Subcodes include currently:

- C4.1-1: None
- C4.1-2: Mandatory 2FA
- C4.1-3: Security contact/team
- C4.1-4: Bug bounty program
- C4.1-5: Limited scope
- C4.1-6: Air gapping

You may extend this list if you identify a new type.
Coding: Add corresponding subcodes

C4.2 Disclosure Policies
Description: If/what type of disclosure policies the project(s) follow.
Coding: Add corresponding subcodes

C4.3 Incident Playbooks
Description: If/what type of incident playbooks the project(s) follow.
Subcodes include currently:
Coding: Add corresponding subcodes

C4.4 Security Testing
Description: If/what type of security testing does the project(s) perform.
Subcodes include currently:
Coding: Add corresponding subcodes

C4.5 Security Reviews
Description: If/what type of security reviews does the project(s) perform.
Subcodes include currently:
Coding: Add corresponding subcodes, see also C5.3 - Pull Requests.

C4.6 Threat Modeling
Description: If threat modeling is mentioned by the participant (exact match only?).

Subcodes include currently:
Coding: Add corresponding subcodes

C5 Project Structure

C5.1 Project Stage
Description: What does the setup of the project look like? Which stages does the project have?
Examples: Code → Commit → PRs → Review → CI for tests and Build → Deployment
Coding: Code mentions of stage-relevant parts.

C5.2 Stage Control
Description: Who controls the different stages.
Coding: Add corresponding subcodes

C5.3 Pull Requests/Patches
Description: How are pull requests (or patches if mailing list ist used) handled?
Coding: Add corresponding subcodes

C5.4 Secret Management
Description: How are (CI/CD) secrets handled
Coding: Add corresponding subcodes

C5.5 Commit Signing
Description: Whether/Why commits are signed
Coding: Add corresponding subcodes

C5.6 Supply Chain
Description: What does the software supply chain look like?
Subcodes include currently:

- C5.6-1: Private package repo used
- C5.6-2: Vulnerability checking with tools
- C5.6-3: Vulnerability checking manual
- C5.6-4: Decision Criteria
- C5.6-5: Frequent updates of dependencies
- C5.6-6: Pinning of package versions
- C5.6-7: Optional dependencies
- C5.6-8: Link against OS libs

You may extend this list if you identify a new type.
Buzzwords: Third Party Libraries, Package Manager, APIs, External projects
Coding: Add corresponding subcodes

C5.7 Other Infrastructure
Description: Does the project have additional infrastructure such as a project website or chat tools?
Subcodes include currently:

- C5.7-1: None
- C5.7-2: Access (who controls this infrastructure?)

Coding: Add corresponding subcodes

C6 Release and Updates

C6.1 Release Decision
Description: Who makes the decision to release an update?
Coding: Add corresponding subcodes

C6.2 Release Deprecation
Description: How are releases deprecated?
Coding: Add corresponding subcodes

C6.3 Release Announcement
Description: How are (security) releases announced?
Coding: Add corresponding subcodes

C6.4 Release Distribution
Description: If/how releases are actually distributed.
Coding: Add corresponding subcodes

C6.5 Release Signing
Description: If/how releases are signed.
Coding: Add corresponding subcodes

C7 Roles and Responsibilities

C7.1 Hierarchy
Description: What does the trust hierarchy look like
Coding: Add corresponding subcodes

C7.2 Security-specific roles
Description: Are there security-specific roles within the project?
Buzzwords: security team, sysadmins
Coding: Add corresponding subcodes

C8 Trusting Contributors

C8.1 Gaining Trust
Description: What are ways to gain trust as a new contributor
Coding: Add corresponding subcodes

C8.2 Identity Check
Description: Does the project(s) check the identity of contributors
Coding: Add corresponding subcodes

C8.3 Contributor License Agreement
Description: Does the project(s) have a CLA?
Subcodes include currently:

- C8.3-1 None

Coding: Add corresponding subcodes

C8.4 Public List of Contributors
Description: Does the project(s) maintain a public list of contributors
Coding: Add corresponding subcodes

C9 Untrustworthy Contributors

C9.1 Trust Incidents
Description: Did the project(s) have trust incidents
Coding: Add corresponding subcodes

C9.2 Trust Strategy
Description: What are the project(s) strategies for dealing with trust incidents
Coding: Add corresponding subcodes

C9.3 Commit Removal
Description: If/how the project(s) removed affected commits
Coding: Add corresponding subcodes

C10 Problems and Improvements

C10.1 Reputation
Description: What does the participant think about the reputation of their project(s)
Coding: Add corresponding subcodes

C10.2 Improvements
Description: Areas that the participant wants to improve
Coding: Add corresponding subcodes

Appendix B

Security Challenges of the Open Source Supply Chain

B.1 Interview Guide

Intro

- **Thanks**: Thank you very much for offering your valuable time for this interview. We are very grateful for your contribution.
- **Ready**: Are you ready to start the interview?
- **Structure**: First off, I am going to talk about the context and data handling, and if you are okay with everything and have no further questions, only then would we start with the actual interview and the recording.

Context

- **We**: We are researchers at the CISPA Helmholtz Center for Information Security (+ other institutions, short context).
- **Our research**: focuses on the area "Usable Security for Developers". This boils down to "How can we enable and empower developers to write more secure code".
- Open Source components are part of the software supply chain and impact many different software projects. External components include unique challenges, such as code from unknown sources and challenges in trusting external projects.
- **Now**: "How can we support software projects with selecting and including open source components in secure ways?"
- **This interview**: is intended as a start/exploration of internal processes and decisions around open source components and the supply chain.
- For this interview:
  - We are **not judging** the security or privacy of a project, we are just interested in the underlying structures and processes.
  - Projects are often very complex, if you **don't know** the answer, **don't want to answer**, or **are not allowed to answer** a question, feel free to just say "next".
  - We are not just interested in structures, but also your **personal opinions and experiences**.
- **Questions?** Any questions about this interview context so far?

Consent

- **Voluntary**: Your responses in this interview are of course entirely voluntary. You may skip any

or all of the questions and can of course leave the interview at any time.
- **Duration**: Duration of the interview depends a bit on the duration of your answers, in our experience 30–45 min, median interview duration so far was about 32 minutes.
- We will **fully de-identify** you and your projects in any publication and only include short quotes.
- We will **send you a preprint before a potential publication**, so you can veto quote usage etc.
- **Recording**: We would like to record this interview so that we can transcribe the answers later.
  - The recording will be destroyed when we transcribed the answers (a few days)
  - The transcripts will be destroyed after analysis (a few months)
- **Questions?** Any more questions about data handling or recording?
- "The recording is now on"
- **Restate**: For the recording: "Are you okay with this interview being recorded?"

S1 Project & Demographics

[Check project(s) beforehand]
S1Q1 **Project**: Can you tell us a bit about the [project(s)] you are involved in?

- S1Q1.1 **About**: What is the project about? What is its purpose?
- S1Q1.2 **Age**: When was the project created?
- S1Q1.3 **Developers**: How many regular developers does the project have?
- S1Q1.4 **Structure**: How is the team structure for this project?

S1Q2 **Project Relation**: How are you related to [project(s)]?

- S1Q2.1 **Join**: When did you join the project?
- S1Q2.2 **Role**: What is your role in the project?

S1Q3 **Setup**: Can you tell us a bit more about the general development setup of the project?

- S1Q3.1 **Tools**: Is the project using specific (software) tools? (SAST, DAST, Manual, Pentests)
- S1Q3.2 **Project Stages**: At what stages of the project? (Only initially, on changes, etc.)
- S1Q3.3 **Frequency**: How often are manual security reviews done? Pentesting etc.? Who carries out these reviews? (skills, external/internal person)
- S1Q3.4 **Security Roles**: Are there security specific roles in your projects?

Usage of Open Source Components (OSCs)

S2Q1 **Components**: Are you aware of any OSCs included in your project?

- S2Q1.2 **Components**: How/If do the different roles interact with components?

S2Q2 **Metrics**: Are you aware of any metrics for selecting those components?

- S2Q2.1 **Exclusion**: What are exclusion criteria?
- S2Q2.2 **Opinion**: What metrics would you personally like to use?
- S2Q2.3 **Awareness**: Are you aware of already prepared metrics such as the OSSF Scorecards for repos or socket.dev for JS? [see Appendix for explanations what those are]

S2Q3 **Supply Chain**: How are those external components pulled/included into the build process?

- S2Q3.1 **Inclusion**: What is the process for including components in your project?
- S2Q3.2 **Aspects**: What parts of this process include some form of security checks?
- S2Q3.3 **Mirrors**: Do you have internal sources/mirrors for the components, instead of public ones? (e.g., internal versions of packages on a local package server, instead of a public one like PyPI)

S2Q4 **Infrastructure**: Does additional infrastructure use open source components?

- S2Q4.1 **CI/CD**: In the build system?
- S2Q4.2 **Other**: Website / Documentation / etc ?

S2Q5 **Support**: Does your company contribute back to Open Source Projects (independent of use)?

- S2Q5.1 **Selection**: How do you choose which projects you support?
- S2Q5.2 **Type**: How does your company contribute back? (pull requests, issues, infrastructure, sponsorship, monetary, ...)
- S2Q5.3 **Not**: If not, would you like your company to contribute back?

Security Policies & Guidance

S3Q1 **Security Policies**: If/what are your project's security policies for including external code?
[IF NOT Policies]: If/what would you like your security policies to contain?

- S3Q1.1 **Content**: What specific parts do they cover?
- S3Q1.2 **Applicability**: Are the teams aware of these policies? Do these have to be read and acknowledged by developers/admins/managers?
- S3Q1.3 **Guidance**: Does this include guidance (e.g., guides or resources for new team members)

S3Q2 **Security Incidents**: How are security incidents/issues in components generally handled?

- S3Q2.1 **Policy**: By what policy?
- S3Q2.2 **Who**: By whom? (specific security team?)
- S3Q2.3 **Access**: Discussed in private groups or open within the company?
- S3Q2.4 **Process**: Are there "Playbooks" or a specific process for (component) Incident Response and Vulnerability Management? (What do these include?)
- S3Q2.5 **History**: How has this process developed?
- S3Q2.6 **Disclosure Policies**: What is the (coordinated) disclosure policy of the project?

S3Q3 **Documentation**: Does your project provide guides/best practices/hints for including external code (e.g., open source components)?

- S3Q3.1 **Opinion**: What are your personal thoughts about including such guides/best practices/hints (e.g, in a Wiki)?

S4 Experiences with Open Source Components

S4Q1 **Components**: Can you tell us a bit about the developer experience of using OSCs in the project?

- S4Q1.1 **Setup**: Was it difficult to set up these components?

- S4Q1.2 **Documentation**: Are these components documented somewhere? (Could a new hire manage them?)
- S4Q1.3 **Customization**: Did you need to customize components for your environment? Did you contribute changes back?

S4Q2 **Updating**: How are you keeping open source components up to date?

- S4Q2.1 **Responsible**: Who is responsible?
- S4Q2.2 **Version**: In what version / release of your [project(s)] are OSCs updated?
- S4Q2.3 **Checks**: Do you check code changes or changelogs before updating?
- S4Q2.4 **Metrics**: How is the initial selection decision for an OSC different to an update decision of the same OSC?

S4Q3 **Same**: Would you select the same components again and why?

S4Q4 **Releases**: How are releases and updates of your project handled (and how are OSC considered in the process)?

- S4Q4.1 **Decision**: How is decided if/when a (security) update is released?
- S4Q4.2 **Secured**: What security concepts are considered for releases? Are releases "secured" in any way?
- S4Q4.3 **Update System**: Are security updates made automatically? How does the update system work?
- S4Q4.4 **Deprecation**: Do you publish information about deprecated / insecure versions?
- S4Q4.5 **Dependencies**: How do you handle dependencies to external components when you publish a release/update?

S5 Challenges & Incidents

[Mention "protestware node-ipc" incident: If participant is aware continue, else introduce (see Appendix)]

S5Q1 **Opinion**: What is your personal opinion of this incident (in terms of supply chain trust)?

- S5Q1.2 **Handling**: How would you react, if your project depends on this (node-ipc) dependency?

S5Q2 **Trust Strategy**: In general, what is/would be your [project(s)] strategy to deal with components that become untrustworthy?

- S5Q2.1 **Identifying**: What does the process for identifying untrustworthy components look like?
- S5Q2.2 **Excluding**: If applicable, can you explain the process for excluding components from the project more in-depth?

S5Q3 **Challenges**: Can you remember any supply-chain related security challenges that [project] faced in the past?
Nudges:

- Malicious / compromised dependency

- Security vulnerability included from a dependency

S5Q4 **Inconveniences**: Aside from major challenges, were you inconvenienced by components in the past?
Nudges:

- Dependency no longer maintained
- Sudden changes to components' API
- Changes how a component was distributed

S6 Problems and Improvements

S6Q1 **Opinions**: In your experience, what is the perceived security of your project? Both by internal actors (like the team) and external actors (like the client or the public).
S6Q2 **Improvements**: Assuming no limitations whatsoever (e.g., monetary or in terms of developer-hours), how would you personally like to improve supply-chain security of your project?

- S6Q2.1 **Problems**: Where do you see problems in the current system?
- S6Q2.2 **Why Exist**: What do you think are the reasons for the current (trust) system to be the way it is?
- S6Q2.3 **Improvements**: What would you like to improve?

Outro

- **Anything else**: Is there something that we did not cover during the interview but you would like to talk about on the recording?
- IF second interviewer: Wait if the second interviewer has any further questions.
- "The recording is now off"
- Thank the participant again for offering their valuable time
- "We will later be in contact for a preprint for changes and veto"
- IF Voucher: Ask if the want/can be compensated for the interview in the form of an Amazon voucher
  - Ask for the email they want their voucher assigned to (likely their Amazon email). Write the email down!
  - Stress that it might take some time (weeks) for billing to process the charges

Debrief

- Easy debrief starter is to ask the participant how they felt about the interview
- Do you have any recommendations for other interesting companies or stakeholders we should invite for an interview?

Protestware-Incident (node-ipc)

In early March 2022, the JavaScript node-ipc library (dependency of, e.g., @vue/cli) was updated by the maintainer to include potentially malicious code: depending on the version when the node-ipc dependency was pulled in, the code scans the host's IP address. If the IP address matched lists of

Russian or Belorussian IPs, the package would go on to replace all of the user's system files with a heart emoji. In other versions, the package always displayed a message of support for Ukraine.

OSSF Scorecard

Scorecard (https://github.com/ossf/scorecard) is a tool by the Open Source Security Foundation that allows users to judge the safety of dependencies automatically based on several metrics. The tool allows assessing several criteria, like project activity, contributors, known vulnerabilities, static analysis, etc., on a scale from 0 to 10. Overall, the project aims to help automate trust decisions, and can be used as a CLI tool or GitHub Action.

Socket.dev

Socket.dev is a tool to detect and block (ongoing) software supply chain attacks in JavaScript software and the npm ecosystem, e.g., by monitoring dependency changes and analyzing their behavior. It is available as a GitHub application.

Open Source

We don't care too much about the ideological idea of open source here, but more about the used components, so a fitting example definition would be: "Software components publicly available, e.g., on GitHub or language respective package platforms like PyPI or npm. Using these components generally requires the user (our participant's company) to rely on code contributions from a number of unknown and untrusted maintainers or developers"

B.2 Codebook

C1 Project Demographics

C1.1 Project

- C1.1.1 Project type
- C1.1.2 Project age
- C1.1.3 Team size
  - C1.1.3-0 Not mentioned
  - C1.1.3-1 1
  - C1.1.3-2 2-5
  - C1.1.3-3 5+
- C1.1.4 Team structure
  - C1.1.4-0 Not mentioned
  - C1.1.4-1 Consultant / Freelancer (alone)
  - C1.1.4-2 Single (developer) team
  - C1.1.4-3 Multiple teams (including stuff like SRE, QA, etc.)
- C1.1.5 Number of Projects
  - C1.1.5-0 Not mentioned

  - C1.1.5-1 One
  - C1.1.5-2 Multiple (>1)

C1.3 Project Setup

- C1.3.1 Project-specific tools
- C1.3.2 Project stages
- C1.3.3 Review frequency
- C1.3.4 Security roles.
  - C1.3.4-0 Not mentioned
  - C1.3.4-1 Yes
  - C1.3.4-2 No
  - C1.3.4-3 Other

C2 Usage OSCs

C2.1 OSC Components

- C2.1.1 OSCs included
  - C2.1.1-0 No
  - C2.1.1-1 Yes
- C2.1.2 Specific OSCs
- C2.1.3 Roles that interact with OSCs

C2.2 OSC Selection Metrics

- C2.2.1 Metrics for selecting OSCs [Select all that apply, feel free to extend]
  - C2.2.1-0 None mentioned
  - C2.2.1-1 Popularity (like Github stars or downloads)
  - C2.2.1-2 Sponsorship (by trusted entity)
  - C2.2.1-3 Activity (like commit frequency or releases)
  - C2.2.1-4 Quality (e.g., commit quality)
  - C2.2.1-5 Recommendations (by friends, blogs, communities, ...)
  - C2.2.1-6 License (must allow usage, etc.)
  - C2.2.1-7 No fix rules (each developer doing as they think)
  - C2.2.1-8 Features (needs to have the needed features)
  - C2.2.1-9 Security history (e.g., past incidents or CVEs)
  - C2.2.1-10 Ease of use (for developers, not including documentation)
  - C2.2.1-11 Community (e.g., to be large, or active)
  - C2.2.1-12 Minimize number of dependencies
  - C2.2.1-13 Dependencies predefined (e.g., customer requirements)
  - C2.2.1-14 Code Inspection (by developer before use)
  - C2.2.1-15 Maturity (of the whole project)
  - C2.2.1-16 Documentation (easy to read/understand/apply, helpful, etc.)
- C2.2.2 Exclusion criteria for OSCs [Select all that apply, feel free to extend]
  - C2.2.2-0 None mentioned

    - C2.2.2-1 Previous vulnerability
    - C2.2.2-2 Inactive maintainer / project
    - C2.2.2-3 Avoid specific organizations (companies, vendors, etc.)
    - C2.2.2-4 Minimum star limit
    - C2.2.2-5 Company Policies (e.g., not to use any 3rd party code at all, license restrictions, etc.)
    - C2.2.2-6 Obviously malicious code/vulnerabilities
    - C2.2.2-7 Single/low number of contributors
    - C2.2.2-8 Bad documentation
    - C2.2.2-9 Bad code quality
- C2.2.3 Personal wishlist metrics
- C2.2.4 Awareness of existing metrics
    - C2.2.4-0 No
    - C2.2.4-1 OpenSSF Scorecards
    - C2.2.4-2 socket.dev
    - C2.2.4-3 Other

C2.3 How are OSCs pulled in?

- C2.3.1 How are OSCs pulled in
- C2.3.2 Process for including new OSCs
- C2.3.3 Using internal mirrors.]
    - C2.3.3-0 No
    - C2.3.3-1 Yes
    - C2.3.3-2 Other

C2.4 OSC in other infrastructure?

C2.5 Contribute back to OSS?

- C2.5-0 Did not contribute back
- C2.5-1 Did contribute back
- C2.5-2 Would like to contribute back

C3 Policies and Guidance

C3.1 What security policies?

- C3.1.1 Security policies for external code
    - C3.1.1-0 No
    - C3.1.1-1 Yes
    - C3.1.1-2 Other
- C3.1.2 Content of security policies
- C3.1.3 Applicability / Awareness

C3.2 How are incidents in components handled?

- C3.2.1 Security incident handling

- C3.2.2 Incident by what policy
- C3.2.3 Incident by whom
- C3.2.4 Specific security team
  - C3.2.4-0 Not mentioned
  - C3.2.4-1 Yes
  - C3.2.4-2 No
  - C3.2.4-3 Other
- C3.2.5 Incident process
- C3.2.6 Incident process history
- C3.2.7 Disclosure policies
  - C3.2.7-0 Not mentioned
  - C3.2.7-1 Yes
  - C3.2.7-2 No
  - C3.2.7-3 Other

C3.3 Project provides documentation for including external code?

- C3.3.1 Documentation
  - C3.3.1-0 Not mentioned
  - C3.3.1-1 Yes
  - C3.3.1-2 No
  - C3.3.1-3 Other
- C3.3.2 Documentation Opinion

C4 Experiences OSCs

C4.1 Developer experience using components?

- C4.1.2 Development experience
  - C4.1.2-0 No opinion
  - C4.1.2-1 Mostly Negative
  - C4.1.2-2 Neutral
  - C4.1.2-3 Mostly Positive
- C4.1.1 Did customize OSC in the past?
  - C4.1.1-0 Not mentioned
  - C4.1.1-1 Yes
  - C4.1.1-2 No
  - C4.1-1-3 Other

C4.2 How are components kept up-to-date?

- C4.2-1 OSC update
- C4.2-2 OSC update responsible
- C4.2-3 OSC update version
- C4.2-4 OSC update checks
- C4.2-5 OSC update metrics

C4.3 Would you use the same components again?

- C4.3-0 Not mentioned
- C4.3-1 Mostly Yes
- C4.3-2 Mostly No
- C4.3-3 Other

C4.4 How are releases and updates handled?

- C4.4-1 Release process
- C4.4-2 Release decision
- C4.4-3 Release secured
- C4.4-4 Release update system
- C4.4-5 Release deprecation
- C4.4-6 Release dependencies

C5 Challenges and Incidents

C5.1 Opinion of incident

- C5.1-0 No opinion
- C5.1-1 Mostly Negative
- C5.1-2 Neutral
- C5.1-3 Mostly Positive

C5.2 General trust strategy

- C5.2-0 Handling similar incident
- C5.2-1 Trust strategy
- C5.2-2 Identify untrustworthy
- C5.2-3 Exclude components

C5.3 Past security challenges / inconveniences

- C5.3.1 Past Challenges encountered?
  - C5.3.1-0 Not mentioned
  - C5.3.1-1 Yes
  - C5.3.1-2 No
  - C5.3.1-3 Other
- C5.3.2 Past Inconveniences

C6 Problems and Improvements

C6.1 Opinions

- C6.1.1 Internal opinion
  - C6.1.1-0 No opinion
  - C6.1.1-1 Mostly Negative
  - C6.1.1-2 Neutral

  - C6.1.1-3 Mostly Positive
- C6.1.2 External opinion
  - C6.1.2-0 No opinion
  - C6.1.2-1 Mostly Negative
  - C6.1.2-2 Neutral
  - C6.1.2-3 Mostly Positive

C6.2 Improvements to [Select all that apply, feel free to extend]

- C6.2-0 More developer hours
- C6.2-1 Better documentation / guidance
- C6.2-2 Static analysis and similar tooling
- C6.2-3 Audit external components (on introduction and updates)
- C6.2-4 Tust processes between oss and third party devs (TLS etc)
- C6.2-5 Resources for security implications (mailings lists)
- C6.2-6 (certificate) updates for long lifecycle devices
- C6.2-7 Automated alerts for dependency updates (CI)
- C6.2-8 Contribute back to dependencies
- C6.2-9 Make transportation more secure
- C6.2-10 Regular pentests
- C6.2-11 Build security in from the start
- C6.2-12 Dedicated security expert for project
- C6.2-13 More/better quality assurance
- C6.2-14 Use security software (e.g., proxy)
- C6.2-15 Better security education for devs
- C6.2-16 Incentives/ monetary rewards
- C6.2-17 Rewrite deps in-house

Appendix C

Large Scale Investigation of Obfuscation Use in Android

C.1 Online Survey

General Questions

Q1.1: Which of these have you heard of in the context of Android apps?
Please check all that apply.

- ☐ Reverse Engineering
- ☐ Repackaging of Software
- ☐ Software Plagiarism
- ☐ Obfuscation

Q1.2: How likely do you think Android apps are ...
[5-point Likert-Scale with: Very Unlikely, Unlikely, Neutral, Likely, Very Likely, and I don't know answers]

- ○ Reverse Engineered
- ○ Repackaged
- ○ Plagiarised
- ○ Obfuscated

Q1.3: How likely do you think your *own* Android apps are ...
[5-point Likert-Scale with: Very Unlikely, Unlikely, Neutral, Likely, Very Likely, and I don't know answers]

- ○ Reverse Engineered
- ○ Repackaged
- ○ Plagiarised
- ○ Obfuscated

Q1.4: How much do you feel the intellectual property of your *own* Android apps is threatened by ...
[5-point Likert-Scale with: Very Unlikely, Unlikely, Neutral, Likely, Very Likely, and I don't know answers]

- ○ Reverse Engineering
- ○ Repackaging of Software
- ○ Software Plagiarism
- ○ Obfuscation

Terminology

Q2.1: Reverse engineering is: ...

- ○ Translate binary files to source code
- ○ Translate source code to binary files
- ○ Analysis of pure source code
- ○ Analysis of binary files
- ○ Reconstruction of app logic
- ○ Testing an app's functionality
- ○ I don't know
- ○ Other [with free text]

Q2.2: Reverse engineering can be used for: ...

- ○ Understanding an app's logic
- ○ Circumvention of licence or security checks
- ○ Repackaging of an app
- ○ Stealing IP addresses
- ○ Attacks on Android users who have your app installed
- ○ Remote attacks on mobile phones
- ○ I don't know
- ○ Other [with free text]

Q2.3: Software plagiarism is: ...

- ○ Repackaging existing software and rebranding it as your own
- ○ Use of third party open source code in your software
- ○ Imitating software to trick users
- ○ Copy pasting code found on the internet
- ○ I don't know
- ○ Other [with free text]

Q2.4: Software plagiarism can be used for: ...

- ○ Obtaining software revenue
- ○ Distributing disguised malware
- ○ Attacking users that have your app installed
- ○ Attacking distribution services
- ○ I don't know
- ○ Other [with free text]

Q2.5: Obfuscation is: ...

- ○ Making source code unreadable or difficult to understand so only authorized developers can work on it
- ○ Making source code unreadable or difficult to understand before compilation
- ○ Hiding binaries from the user
- ○ Preventing acces to the deployed application
- ○ I don't know

- ○ Other [with free text]

Q2.6: Obfuscation can be used for: ...

- ○ Making reverse engineering more difficult
- ○ Prevent others from attacking vulnerabilities within your application
- ○ Hiding the logic within your application
- ○ Optimization of app performance
- ○ I don't know
- ○ Other [with free text]

Q2.7: Have you heard of obfuscation before?

- ○ Yes
- ○ No
- ○ Uncertain

Q2.8: Have you ever thought about using obfuscation?

- ○ Yes
- ○ No
- ○ Uncertain

Q2.9: Did you obfuscate at least once before?

- ○ Yes
- ○ No
- ○ Uncertain

Obfuscation Tools

Q3.1: Please select all Android obfuscation tools that you have heard of prior to this study.

- □ ProGuard
- □ DexGuard
- □ Jack
- □ DashO
- □ ReDex
- □ Harvester
- □ Other [with free text]

Q3.2: Please select all Android obfuscation tools that you have used before.

- □ ProGuard
- □ DexGuard
- □ Jack
- □ DashO
- □ ReDex
- □ Harvester
- □ Other [with free text]

Q3.3: Please select all Android obfuscation tools that you have actively decided against using.

- ☐ ProGuard
- ☐ DexGuard
- ☐ Jack
- ☐ DashO
- ☐ ReDex
- ☐ Harvester
- ☐ Other [with free text]

Q3.4: Which tools do you use to remove unused library code?

- ☐ ProGuard "Minify"
- ☐ Android Studio "Minify"
- ☐ I remove it manually
- ☐ I never remove unused library code from my apps
- ☐ Other [with free text]

Obfuscation 1

Q4.1: How did you first encounter obfuscation?
[Free text]

Q4.2: How many apps have you worked on?
[Number input]

Q4.3: How many of those where obfuscated?
[Number input]

Q4.4: Why did you use obfuscation on those apps?
[Free text]

Q4.5: Why did you decide against obfuscating apps?
[Free text]

Q4.6: Did you verify that obfuscation was successful?

- ○ Yes
- ○ No

Q4.7: How did you verify if obfuscation was successful?
[Free text]

Q4.8: Why did you decide against using obfuscation?
[Free text]

C.2 Programming Experiment - Exit Survey

After completing the programming task, developers were asked to fill out an exit survey.

Tasks

Do you think you solved the tasks correctly?
T.1: Task1

- ○ Yes
- ○ No
- ○ I don't know

T.2: Task2

- ○ Yes
- ○ No
- ○ I don't know

T.3: Do you have additional comments on the tasks?
[Free text]

General Questions

Q1.1: Which of these have you heard of in the context of Android apps?
Please check all that apply.

- □ Reverse Engineering
- □ Repackaging of Software
- □ Software Plagiarism
- □ Obfuscation

Q1.2: How likely do you think Android apps are ...
[5-point Likert-Scale with: Very Unlikely, Unlikely, Neutral, Likely, Very Likely, and I don't know answers]

- ○ Reverse Engineered
- ○ Repackaged
- ○ Plagiarised
- ○ Obfuscated

Q1.3: How likely do you think your *own* Android apps are ...
[5-point Likert-Scale with: Very Unlikely, Unlikely, Neutral, Likely, Very Likely, and I don't know answers]

- ○ Reverse Engineered
- ○ Repackaged
- ○ Plagiarised
- ○ Obfuscated

Q1.4: How much do you feel the intellectual property of your *own* Android apps is threatened by ...
[5-point Likert-Scale with: Very Unlikely, Unlikely, Neutral, Likely, Very Likely, and I don't know answers]

- ○ Reverse Engineering
- ○ Repackaging of Software
- ○ Software Plagiarism
- ○ Obfuscation

Terminology

Q2.1: Reverse engineering is: ...

- ○ Translate binary files to source code
- ○ Translate source code to binary files
- ○ Analysis of pure source code
- ○ Analysis of binary files
- ○ Reconstruction of app logic
- ○ Testing an app's functionality
- ○ I don't know
- ○ Other [with free text]

Q2.2: Reverse engineering can be used for: ...

- ○ Understanding an app's logic
- ○ Circumvention of licence or security checks
- ○ Repackaging of an app
- ○ Stealing IP addresses
- ○ Attacks on Android users who have your app installed
- ○ Remote attacks on mobile phones
- ○ I don't know
- ○ Other [with free text]

Q2.3: Software plagiarism is: ...

- ○ Repackaging existing software and rebranding it as your own
- ○ Use of third party open source code in your software
- ○ Imitating software to trick users
- ○ Copy pasting code found on the internet
- ○ I don't know
- ○ Other [with free text]

Q2.4: Software plagiarism can be used for: ...

- ○ Obtaining software revenue
- ○ Distributing disguised malware
- ○ Attacking users that have your app installed
- ○ Attacking distribution services
- ○ I don't know
- ○ Other [with free text]

Q2.5: Obfuscation is: ...

- ○ Making source code unreadable or difficult to understand so only authorized developers can work on it
- ○ Making source code unreadable or difficult to understand before compilation
- ○ Hiding binaries from the user
- ○ Preventing acces to the deployed application
- ○ I don't know

- ○ Other [with free text]

Q2.6: Obfuscation can be used for: ...

- ○ Making reverse engineering more difficult
- ○ Prevent others from attacking vulnerabilities within your application
- ○ Hiding the logic within your application
- ○ Optimization of app performance
- ○ I don't know
- ○ Other [with free text]

Q2.7: Have you heard of obfuscation before?

- ○ Yes
- ○ No
- ○ Uncertain

Q2.8: Have you ever thought about using obfuscation?

- ○ Yes
- ○ No
- ○ Uncertain

Q2.9: Did you obfuscate at least once before?

- ○ Yes
- ○ No
- ○ Uncertain

Obfuscation Tools

Q3.1: Please select all Android obfuscation tools that you have heard of prior to this study.

- □ ProGuard
- □ DexGuard
- □ Jack
- □ DashO
- □ ReDex
- □ Harvester
- □ Other [with free text]

Q3.2: Please select all Android obfuscation tools that you have used before.

- □ ProGuard
- □ DexGuard
- □ Jack
- □ DashO
- □ ReDex
- □ Harvester
- □ Other [with free text]

Q3.3: Please select all Android obfuscation tools that you have actively decided against using.

- ☐ ProGuard
- ☐ DexGuard
- ☐ Jack
- ☐ DashO
- ☐ ReDex
- ☐ Harvester
- ☐ Other [with free text]

Q3.4: Which tools do you use to remove unused library code?

- ☐ ProGuard "Minify"
- ☐ Android Studio "Minify"
- ☐ I remove it manually
- ☐ I never remove unused library code from my apps
- ☐ Other [with free text]

ProGuard

P.1: What do you use Proguard for?

- ☐ Testing
- ☐ Minifying Code
- ☐ Optimization
- ☐ Obfuscation

P.2: After using Proguard, how did you verify that it achieved its goal?

- ○ I did not verify that Proguard worked
- ○ Reverse Engineering
- ○ Other [with free text]

P.3: Why have you never used Proguard before?

- ○ No need
- ○ Never heard of it
- ○ Too complicated
- ○ I have other tools
- ○ Other [with free text]

Appendix D

Security & Privacy Perceptions of Cloud Office Suites

D.1 Survey

The following survey is the English version of the survey, the German version followed the same structure with nearly identical questions. Differences in questions included localization changes, e.g., for country-specific agencies and institutions. Question numbers were not displayed to the participants.

Consent Form

[Consent Form with contact information.]

Please indicate, in the box below, that you are at least 18 years old, have read and understood this consent form, and you agree to participate in this online research study.

- ☐ I am age 18 or older.
- ☐ I have read this consent form or had it read to me.
- ☐ I am comfortable using the English language to participate in this study.
- ☐ I have used cloud office software before (e.g., Google Drive or Microsoft Office 365).
- ☐ I agree to participate in this research and I want to continue with the study.

Office demographics

For this survey, we are interested in your experience with and use of Cloud Office Suites and applications. **Cloud Office Application or Online Office Application are** software that can be used to create office documents in a web browser, without requiring the installation of a dedicated software. **Examples for Cloud Office Suites are** Google Docs/Sheets/Slides, Microsoft Office 365, and LibreOffice Online.
Q1.1: Which office suites have you used before?
(Please select all that apply)

- ☐ Microsoft Office (Offline; Word, Excel, Powerpoint, ...)
- ☐ Microsoft Office 365 (Cloud-based; Word, Excel, Powerpoint, ...)
- ☐ LibreOffice (Offline; Writer, Calc, ...)
- ☐ LibreOffice Online (Cloud-based; Writer, Calc, ...)
- ☐ Google Drive (Cloud-based; Docs, Sheets, Slides, ...)
- ☐ Apple's iWork App (Offline; Pages, Numbers. Keynote...)
- ☐ Apple's iWork Web (Cloud-based; Pages, Numbers. Keynote...)

- ☐ OnlyOffice
- ☐ Other (please specify):

Q1.2: Which office suites have you used this month?
(Please select all that apply)

- ☐ Microsoft Office (Offline; Word, Excel, Powerpoint, ...)
- ☐ Microsoft Office 365 (Cloud-based; Word, Excel, Powerpoint, ...)
- ☐ LibreOffice (Offline; Writer, Calc, ...)
- ☐ LibreOffice Online (Cloud-based; Writer, Calc, ...)
- ☐ Google Drive (Cloud-based; Docs, Sheets, Slides, ...)
- ☐ Apple's iWork App (Offline; Pages, Numbers. Keynote...)
- ☐ Apple's iWork Web (Cloud-based; Pages, Numbers. Keynote...)
- ☐ OnlyOffice
- ☐ Other (please specify):

Q1.3: Does your job involve using office applications on a regular basis?

- ○ Yes
- ○ No
- ○ I don't know
- ○ I'd prefer not to answer

Q1.4: Which types of documents do you process with office suites?
For this question, please give answers both for your job and your personal life.
(Please select all that apply)

- ☐ Text (Reports, Letters, etc.)
- ☐ Spreadsheets (Numbers, Dates, etc.)
- ☐ Presentations
- ☐ Calendar and Appointments
- ☐ Emails
- ☐ Other (please specify):

Q1.5: How do you store your documents?
For this question, please give answers for any documents you might store, including personal and work documents, including but not limited to documents that you edit with office applications.
(Please select all that apply)

- ☐ Locally on my computer
- ☐ My office suite saves them online automatically.
- ☐ Dropbox
- ☐ Google Drive
- ☐ Network Share
- ☐ Self-hosted cloud service
- ☐ OneDrive
- ☐ iCloud
- ☐ Other (please specify):

Q1.6: Why do you use **cloud** office applications (compared to local office applications)?
(Please select all that apply)

- ☐ Provided or required by work
- ☐ Easy remote access (e.g., from multiple devices)
- ☐ Ease of collaboration
- ☐ No installation required
- ☐ Built-in backup of documents
- ☐ Free / cheap access
- ☐ Other (please specify):

Document Safety

Q2.1: Where do you think your documents are more secure from any unauthorized access?
This is a matrix question, the scala for answers of this questions is:

- More secure on my computer
- Somewhat more secure on my computer
- Equally secure
- Somewhat more secure in the cloud
- More secure in the cloud
- I don't know

The questions are:

- Word documents
- Presentations
- Spreadsheets
- E-Mails
- Calendar and Appointments

Q2.2: Why (if at all) do you think your documents may be more secure **on your computer**?
[Free text field]
Q2.3: Why (if at all) do you think your documents may be more secure **in the cloud**?
[Free text field]

Document Access

Q3.1: Who else besides yourself might be able to access the documents you edit in cloud office applications?
(Please select all that apply)

- ☐ People I share the documents with
- ☐ My employer
- ☐ My internet provider
- ☐ The cloud office provider (e.g., Google or Microsoft)
- ☐ My browser vendor (e.g., Google or Mozilla)
- ☐ My operating system manufacturer (e.g., Apple or Microsoft)
- ☐ Cybercriminals (e.g., hackers or organized crime)
- ☐ Law enforcement or intelligence agencies (e.g., police, FBI or NSA)
- ☐ Third parties (e.g., online advertisers or plugin developers)

- ☐ The manufacturer of my computer hardware (e.g., Intel, AMD, Apple, or Lenovo)
- ☐ Other (please specify):

The following 3 questions are matrix questions with the following options:

- People I share the documents with
- My employer
- My internet provider
- The cloud office provider (e.g., Google or Microsoft)
- My browser vendor (e.g., Google or Mozilla)
- My operating system manufacturer (e.g., Apple or Microsoft)
- Cybercriminals (e.g., hackers or organized crime)
- Law enforcement or intelligence agencies (e.g., police, FBI or NSA)
- Third parties (e.g., online advertisers or plugin developers)
- The manufacturer of my computer hardware (e.g., Intel, AMD, Apple, or Lenovo)

Q3.2: Where do you think the risk is higher that the following parties can obtain unauthorized access to your cloud office documents?

- Higher risk on my computer
- Somewhat higher risk on my computer
- Equal risk
- Somewhat higher risk in the cloud
- Higher risk in the cloud
- I don't know

Q3.3: Do you think that any of these parties have already accessed your documents?

- Yes
- No
- I don't know

Q3.4: Please rate your level of (dis)comfort with the potential access of these parties to your cloud office documents.

- Completely comfortable
- Somewhat comfortable
- Neither
- Somewhat uncomfortable
- Completely uncomfortable
- I don't know

Q3.5: Who do you think **would** inform you if an unauthorized party or person accessed you documents?
(Please select all that apply)

- ☐ People I share the documents with
- ☐ My employer
- ☐ My internet provider
- ☐ The cloud office provider (e.g., Google or Microsoft)
- ☐ My browser vendor (e.g., Google or Mozilla)

- ☐ My operating system manufacturer (e.g., Apple or Microsoft)
- ☐ Law enforcement or intelligence agencies (e.g., police, FBI or NSA)
- ☐ Third parties (e.g., online advertisers or plugin developers)
- ☐ The manufacturer of my computer hardware (e.g., Intel, AMD, Apple, or Lenovo)
- ☐ The news
- ☐ Scientists
- ☐ Nobody would inform me
- ☐ Other (please specify):

Q3.6: Who do you think **should be responsible** for informing you if an unauthorized party or person accessed your documents?
(Please select all that apply)

- ☐ People I share the documents with
- ☐ My employer
- ☐ My internet provider
- ☐ The cloud office provider (e.g., Google or Microsoft)Finde ich gut. Da sollte China am besten noch deutlich weiter vorangehen. Damit der Westen sich endlich besinnt und "intellectual property" wieder abschafft. Das war die ganze Zeit eine Schnapsidee.
- ☐ My browser vendor (e.g., Google or Mozilla)
- ☐ My operating system manufacturer (e.g., Apple or Microsoft)
- ☐ Law enforcement or intelligence agencies (e.g., police, FBI or NSA)
- ☐ Third parties (e.g., online advertisers or plugin developers)
- ☐ The manufacturer of my computer hardware (e.g., Intel, AMD, Apple, or Lenovo)
- ☐ The news
- ☐ Scientists
- ☐ Nobody would inform me
- ☐ Other (please specify):

Q3.7: **How** would you like to be informed if an unauthorized party or person accessed your cloud office documents?
[Free text field]

Document Storage

Q4.1: Do you think that multiple copies of your cloud office documents exist?
These can be documents that are shared with others or private documents.

- ○ Yes
- ○ No
- ○ I don't know
- ○ I'd prefer not to answer

Q4.2 (only shown if Q4.1 = Yes): For which purpose do you think these copies might exist?
[Free text field]
Q4.3 (only shown if Q4.1 = Yes): In which geographic locations do you think your cloud office documents and copies of these are stored?
[Free text field]
Q4.4 (only shown if Q4.1 = Yes): Which of the copies do you think are actually removed if you delete a cloud office document?

- ○ All
- ○ Mine and my collaborators'
- ○ Only mine
- ○ Only my collaborators'
- ○ None
- ○ I don't know
- ○ I'd prefer not to answer
- ○ Other (please specify):

Q4.5 (only shown if Q4.1 = Yes and Q4.4 != All): **Where or with whom** do you think copies remain?
[Free text field]

Q4.6 (only shown if Q4.1 = Yes and Q4.4 != All): For which purpose do you think that the copies remain?
[Free text field]

Q4.7: Who do you think can delete your documents?
(Please select all that apply)

- □ People I share the documents with
- □ My employer
- □ My internet provider
- □ The cloud office provider (e.g., Google or Microsoft)
- □ My browser vendor (e.g., Google or Mozilla)
- □ My operating system manufacturer (e.g., Apple or Microsoft)
- □ Cybercriminals (e.g., hackers or organized crime)
- □ Law enforcement or intelligence agencies (e.g., police, FBI or NSA)
- □ Third parties (e.g., online advertisers or plugin developers)
- □ The manufacturer of my computer hardware (e.g., Intel, AMD, Apple, or Lenovo)
- □ Other (please specify):

Q4.8: Who do you think is responsible for protecting your data?
(Please select all that apply)

- □ People I share the documents with
- □ My employer
- □ My internet provider
- □ The cloud office provider (e.g., Google or Microsoft)
- □ My browser vendor (e.g., Google or Mozilla)
- □ My operating system manufacturer (e.g., Apple or Microsoft)
- □ Cybercriminals (e.g., hackers or organized crime)
- □ Law enforcement or intelligence agencies (e.g., police, FBI or NSA)
- □ Third parties (e.g., online advertisers or plugin developers)
- □ The manufacturer of my computer hardware (e.g., Intel, AMD, Apple, or Lenovo)
- □ Myself
- □ The US-Government
- □ Other (please specify):

Responsibility

Q5.1: Please indicate your agreement with the following statements: (5 point-likert scale from Strongly agree to Strongly disagree + I don't know option)

- Cloud office providers should offer adequate protection for cloud office documents (e.g., by encryption and well implemented security practices)
- I should have the right to demand a full overview of my data collected by cloud office providers.
- Upon my request, cloud office providers should have to show what they do with my documents and who has or had access.
- Cloud office providers must be able to modify or delete any data they have on private individuals.

Q5.2: Please indicate your (dis)comfort with the following statements: (5 point-likert scale from Completely comfortable to Completely uncomfortable + I don't know option)

- Cloud providers can store my documents on servers outside of the US without legal repercussions.
- US regulations and laws still apply if the documents are stored on servers outside of the US.
- US law enforcement can access my cloud documents without a court order.
- US law enforcement can force me to give up my cloud office password.

Q5.3: Where do you think the risk is higher of somebody **obtaining unauthorized access** to your documents if they are either stored on a server in Germany or the US? (5 point-likert scale from "Higher risk for server in Germany" to "Higher risk for server in the US" + I don't know option)

- My employer
- My internet provider
- The cloud office provider (e.g., Google or Microsoft)
- My browser vendor (e.g., Google or Mozilla)
- My operating system manufacturer (e.g., Apple or Microsoft)
- Cybercriminals (e.g., hackers or organized crime)
- Third parties (e.g., online advertisers or plugin developers)
- The manufacturer of my computer hardware (e.g., Intel, AMD, Apple, or Lenovo)
- US government
- German governments
- Foreign government (neither US nor German)

Personal Perception - Scenario A - Personalized Scenario

Below are listed three different scenarios.
How comfortable do you feel with each approach?
Q6.A.1: Your child is required by the school to use a cloud office suite for tasks. The processed documents include private information such as your child's name and grades.

- ○ Completely comfortable
- ○ Somewhat comfortable
- ○ Neither
- ○ Somewhat uncomfortable
- ○ Completely uncomfortable

- ○ I don't know

Q6.A.2: Your general practitioner uses a cloud office suite to process patient data. The processed documents include private information such as your name, age, weight, diagnosis, and treatment plan.

- ○ Completely comfortable
- ○ Somewhat comfortable
- ○ Neither
- ○ Somewhat uncomfortable
- ○ Completely uncomfortable
- ○ I don't know

Q6.A.3: Your financial advisor uses a cloud office suite to process client data. The processed documents include private information such as your name, SSN, and financial information.

- ○ Completely comfortable
- ○ Somewhat comfortable
- ○ Neither
- ○ Somewhat uncomfortable
- ○ Completely uncomfortable
- ○ I don't know

Personal Perception - Scenario B - Generalized Scenario

Only scenario block A or B was randomly shown to the participants
Below are listed three different scenarios.
How comfortable do you feel with each approach?
Q6.B.1: A school requires children to use a cloud office suite for tasks. The processed documents include private information such as childrens' names and grades.

- ○ Completely comfortable
- ○ Somewhat comfortable
- ○ Neither
- ○ Somewhat uncomfortable
- ○ Completely uncomfortable
- ○ I don't know

Q6.B.2: A doctor's office uses a cloud office suite to process patient data. The processed documents include private information such as name, age, weight, diagnosis, and treatment plans.

- ○ Completely comfortable
- ○ Somewhat comfortable
- ○ Neither
- ○ Somewhat uncomfortable
- ○ Completely uncomfortable
- ○ I don't know

Q6.B.3: A financial advisor's office uses a cloud office suite to process client data. The processed documents include private information such as name, SSN, and financial information.

- ○ Completely comfortable
- ○ Somewhat comfortable
- ○ Neither
- ○ Somewhat uncomfortable
- ○ Completely uncomfortable
- ○ I don't know

Data Protection

Q7.1: What do you think - what data does the cloud office application collect when you process documents with it?
[Free text field]
Q7.2: How do you think documents processed by cloud office applications are protected?
[Free text field]

GDPR

Q8.1: Do you know what the GDPR is?

- ○ A data protection regulation in EU law
- ○ A plugin for Google Drive
- ○ A cloud office provider
- ○ A counter terrorism act in US law
- ○ I don't know
- ○ I'd prefer not to answer

Q8.2 (Only shown if Q8.1 = A data protection regulation in EU law): What do you think does the GDPR protect?
[Free text field]

Demographics

Q9.1: How old are you? (in years, e.g. 42. Optional)
[Free text field]
Q9.2: As which gender do you identify?

- ○ Male
- ○ Female
- ○ [Free text field]
- ○ I'd prefer not to answer

Q9.3: Do you have formal education (Bachelor's degree or higher) in computer science, information technology, or a related field?

- ○ Yes
- ○ No
- ○ I'd prefer not to answer

Q9.4: Have you held a job in computer science, information technology, or a related field?

- ○ Yes
- ○ No
- ○ I'd prefer not to answer

Q9.5: Do you have any feedback or additional comments for us? (completely optional)
[Free text field]

Bibliography

[1] K. Thompson, "Reflections on Trusting Trust," *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984.

[2] MITRE, *CVE-2014-0160*, https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160, Dec. 2013.

[3] MITRE, *CVE-2014-6271*, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271, Sep. 2014.

[4] MITRE, *CVE-2021-44228*, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228, Nov. 2021.

[5] National Vulnerability Database, *CVE-2020-10148*, https://nvd.nist.gov/vuln/detail/CVE-2020-10148, Dec. 2020.

[6] National Vulnerability Database, *CVE-2021-30116*, https://nvd.nist.gov/vuln/detail/CVE-2021-30116, Jul. 2021.

[7] European Commission, *Cyber Resilience Act*, https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act, Sep. 2022.

[8] Bundesamt für Sicherheit in der Informationstechnik, *Second act on increasing the security of IT systems (German IT Security Act 2.0)*, https://www.bsi.bund.de/EN/Das-BSI/Auftrag/Gesetze-und-Verordungen/IT-SiG/2-0/it_sig-2-0_node.html, May 2021.

[9] The White House, *Executive Order on America's Supply Chains (EO14017)*, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/, Feb. 2021.

[10] The White House, *Executive Order on Improving the Nation's Cybersecurity (EO14028)*, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/, May 2021.

[11] J. Menn and R. Satter, *Codecov hackers breached hundreds of restricted customer sites*, https://www.reuters.com/technology/codecov-hackers-breached-hundreds-restricted-customer-sites-sources-2021-04-19/, Apr. 2021.

[12] Slack, *Slack Security Update*, https://slack.com/blog/news/slack-security-update, Jan. 2023.

[13] D. Bradbury, *Okta's Investigation of the January 2022 Compromise*, https://www.okta.com/blog/2022/03/oktas-investigation-of-the-january-2022-compromise/, Mar. 2022.

[14] K. Toubba, *Security Incident Update and Recommended Actions*, https://blog.lastpass.com/2023/03/security-incident-update-recommended-actions/, Mar. 2023.

[15] R. Zuber, *CircleCI incident report for January 4, 2023 security incident*, https://circleci.com/blog/jan-4-2023-incident-report/, Jan. 2023.

[16] D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects," in *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*, May 2022.

[17] D. Wermke, J. H. Klemmer, N. Wöhler, J. Schmüser, H. S. Ramulu, Y. Acar, and S. Fahl, ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain," in *44th IEEE Symposium on Security and Privacy (S&P'23)*, IEEE, May 2023.

[18] D. Wermke, N. Huaman, Y. Acar, B. Reaves, P. Traynor, and S. Fahl, "A Large Scale Investigation of Obfuscation Use in Google Play," in *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC'18, San Juan, PR, USA, December 3-7, 2018*, Dec. 2018, pp. 222–235.

[19] D. Wermke, C. Stransky, N. Huaman, N. Busch, Y. Acar, and S. Fahl, "Cloudy with a Chance of Misconceptions: Exploring Users' Perceptions and Expectations of Security and Privacy in Cloud Office Suites," in *Sixteenth Symposium on Usable Privacy and Security, SOUPS 2020, August 12-14, 2020*, Aug. 2020.

[20] N. Huaman, A. Krause, D. Wermke, C. Stransky, J. H. Klemmer, Y. Acar, and S. Fahl, "If You Can't Get Them to the Lab: Evaluating a Virtual Study Environment with Security Information Workers," in *Eighteenth Symposium on Usable Privacy and Security, SOUPS 2022, August 7-9, 2022*, Aug. 2022.

[21] N. Huaman, B. von Skarczinski, C. Stransky, D. Wermke, Y. Acar, A. Dreißigacker, and S. Fahl, "A Large-Scale Interview Study on Information Security in and Attacks against Small and Medium-sized Enterprises," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1235–1252.

[22] C. Stransky, D. Wermke, J. Schrader, N. Huaman, Y. Acar, A. L. Fehlhaber, M. Wei, B. Ur, and S. Fahl, "On the Limited Impact of Visualizing Encryption: Perceptions of E2E Messaging Security," in *Seventeenth Symposium on Usable Privacy and Security, SOUPS 2021, August 8-10, 2021*, Aug. 2021.

[23] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers Need Support Too: A Survey of Security Advice for Software Developers," in *2017 IEEE Secure Development Conference (SecDev'17)*, 2017.

[24] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl, "Security Developer Studies with GitHub Users: Exploring a Convenience Sample," in *13th Symposium on Usable Privacy and Security (SOUPS'17)*, 2017.

[25] J. Collins, *latexmk – Fully automated LaTeX document generation*, https://www.ctan.org/pkg/latexmk/.

[26] H. Hage, H. Henkel, T. Hoekwater, and L. Scarso, *LuaTex*, https://www.luatex.org/.

[27] M. Kohm, *koma-script – A bundle of versatile classes and packages*, https://ctan.org/pkg/koma-script.

[28] CIPAC, *Securing the Software Supply Chain: Recommended Practices for Suppliers*, https://media.defense.gov/2022/Oct/31/2003105368/-1/-1/0/SECURING_THE_SOFTWARE_SUPPLY_CHAIN_SUPPLIERS.PDF, Oct. 2022.

[29] CIPAC, *Securing the Software Supply Chain: Recommended Practices for Developers*, https://media.defense.gov/2022/Sep/01/2003068942/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF, Sep. 2022.

[30] CIPAC, *Securing the Software Supply Chain: Recommended Practices for Customers*, https://media.defense.gov/2022/Nov/17/2003116445/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_CUSTOMER.PDF, Nov. 2022.

[31] Sonatype, *State of the Software Supply Chain*, https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021, 2021.

[32] Synopsys, Inc., *The Heartbleed Bug*, https://heartbleed.com/, Jun. 2020.

[33] N. Perlroth, *Security Experts Expect Shellshock Software Bug in Bash to Be Significant*, https://www.nytimes.com/2014/09/26/technology/security-experts-expect-shellshock-software-bug-to-be-significant.html, Sep. 2014.

[34] DEVCORE, *ProxyLogon*, https://proxylogon.com/, Mar. 2021.

[35] F. Wortley, C. Thompson, and F. Allison, *Log4Shell: RCE 0-day exploit found in log4j, a popular Java logging package*, https://www.lunasec.io/docs/blog/log4j-zero-day/, Dec. 2021.

[36] J. Graham-Cumming, *Inside Shellshock: How hackers are using it to exploit systems*, https://blog.cloudflare.com/inside-shellshock/, Oct. 2014.

[37] Microsoft, *CVE-2021-26855*, https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-26855, Mar. 2021.

[38] Microsoft Threat Intelligence Center (MSTIC), *HAFNIUM targeting Exchange Servers with 0-day exploits*, https://www.microsoft.com/en-us/security/blog/2021/03/02/hafnium-targeting-exchange-servers/, Mar. 2021.

[39] A. Berger, *What is Log4Shell? The Log4j vulnerability explained (and what to do about it)*, https://www.dynatrace.com/news/blog/what-is-log4shell/, Dec. 2021.

[40] MITRE, *Overview - About the CVE Program*, https://www.cve.org/About/Overview.

[41] Forum of Incident Response and Security Teams (FIRST), *Common Vulnerability Scoring System SIG*, https://www.first.org/cvss/.

[42] Forum of Incident Response and Security Teams (FIRST), *Exploit Prediction Scoring System*, https://www.first.org/epss/.

[43] National Telecommunications and Information Administration (NTIA), *Vulnerability-Exploitability eXchange (VEX) – An Overview*, https://ntia.gov/files/ntia/publications/vex_one-page_summary.pdf, Sep. 2021.

[44] MITRE, *Common Weakness Enumeration*, https://cwe.mitre.org/about/index.html.

[45] OpenSSF, *OpenSSF Scorecard*, https://securityscorecards.dev/.

[46] MITRE, *Common Attack Pattern Enumeration and Classification*, https://capec.mitre.org/about/index.html.

[47] MITRE, *MITRE ATT&CK*, https://attack.mitre.org/.

[48] Cybersecurity and Infrastructure Security Agency (CISA), *Software Bill of Materials (SBOM)*, https://www.cisa.gov/sbom.

[49] National Telecommunications and Information Administration (NTIA), *The Minimum Elements For a Software Bill of Materials (SBOM)*, https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom, Jul. 2021.

[50] The Linux Foundation Projects, *The Software Package Data Exchange (SPDX)*, https://spdx.dev/.

[51] OWASP, *CycloneDX*, https://cyclonedx.org/.

[52] National Vulnerability Database, *Software Identification Tags (SWID Tags)*, https://nvd.nist.gov/products/swid.

[53] European Union Agency for Cybersecurity, *Threat Landscape for Supply Chain Attacks*, https://www.enisa.europa.eu/news/enisa-news/understanding-the-increase-in-supply-chain-security-attacks, Jul. 2021.

[54] M. Jankowicz and C. R. Davis, *These big firms and US agencies all use software from the company breached in a massive hack being blamed on Russia*, https://www.businessinsider.com/list-of-companies-agencies-at-risk-after-solarwinds-hack-2020-12, Dec. 2020.

[55] FireEye, *Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor*, https://www.mandiant.com/resources/blog/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor, Dec. 2020.

[56] SolarWinds, *SolarWinds Security Advisory*, https://www.solarwinds.com/sa-overview/securityadvisory, Apr. 2021.

[57] Microsoft 365 Defender Research Team, *Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Microsoft Defender helps protect customers*, https://www.microsoft.com/en-us/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/, Dec. 2020.

[58] Kaseya, *Kaseya Responds Swiftly to Sophisticated Cyberattack*, https://www.kaseya.com/press-release/kaseya-responds-swiftly-to-sophisticated-cyberattack-mitigating-global-disruption-to-customers/, Jul. 2021.

[59] L. Sterk, *DIVD gives full disclosure in Kaseya Case*, https://www.divd.nl/2022/04/04/Kaseya-VSA-full-disclosure/, Apr. 2022.

[60] BBC, *Ransomware key to unlock customer data from REvil attack*, https://www.bbc.com/news/technology-57946117, Jul. 2021.

[61] Codecov, *Post-Mortem / Root Cause Analysis (April 2021)*, https://about.codecov.io/apr-2021-post-mortem/, Apr. 2021.

[62] N. Stoler, *Breaking Down the Codecov Attack: Finding a Malicious Needle in a Code Haystack*, https://www.cyberark.com/resources/blog/breaking-down-the-codecov-attack-finding-a-malicious-needle-in-a-code-haystack, Apr. 2021.

[63] D. Bradbury, *Updated Okta Statement on LAPSUS$*, https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/, Mar. 2022.

[64] Synopsys, *Open Source Security and Risk Analysis (OSSRA)*, https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html, 2023.

[65] M. Drake, *The Difference Between Free and Open-Source Software*, https://www.digitalocean.com/community/conceptual-articles/free-vs-open-source-software, Oct. 2017.

[66] G. C. Peters, *Securing Open Source Software Act*, https://www.congress.gov/bill/117th-congress/senate-bill/4913, Sep. 2022.

[67] P. Cromier, *The State of Enterprise Open Source: A Red Hat report*, https://www.redhat.com/en/resources/state-of-enterprise-open-source-report-2022, Feb. 2022.

[68] faisalman, *ua-parser-js: GitHub Issue 536*, https://github.com/faisalman/ua-parser-js/issues/536, Oct. 2021.

[69] A. Polkovnichenko, O. Kaspi, and S. Menashe, *JFrog Detects Malicious PyPI Packages Stealing Credit Cards and Injecting Code*, https://jfrog.com/blog/malicious-pypi-packages-stealing-credit-cards-injecting-code/, Jul. 2021.

[70] A. Sharma, *More Than 200 Cryptomining Packages Flood npm and PyPI Registry*, https://blog.sonatype.com/more-than-200-cryptominers-flood-npm-and-pypi-registry, Aug. 2022.

[71] A. Sharma, *Attacker Floods PyPI With 1000s of Malicious Packages That Drop Windows Trojan via Dropbox*, https://blog.sonatype.com/attacker-floods-pypi-with-450-malicious-packages-that-drop-windows-trojan-via-dropbox, Feb. 2023.

[72] A. Sharma, *Dev corrupts NPM libs 'colors' and 'faker' breaking thousands of apps*, https://www.bleepingcomputer.com/news/security/dev-corrupts-npm-libs-colors-and-faker-breaking-thousands-of-apps/, Jan. 2022.

[73] G. Jones, *GitHub Advisory Database: Infinite loop causing Denial of Service in colors*, https://github.com/advisories/GHSA-5rqg-jm4f-cqx7, Jan. 2022.

[74] A. Sharma, *npm Libraries 'colors' and 'faker' Sabotaged in Protest by their Maintainer — What to do Now?* https://blog.sonatype.com/npm-libraries-colors-and-faker-sabotaged-in-protest-by-their-maintainer-what-to-do-now, Jan. 2022.

[75] Marak, *Issue 285: Zalgo issue with v1.4.44-liberty-2 release*, https://github.com/Marak/colors.js/issues/285, Jan. 2022.

[76] National Vulnerability Database, *CVE-2022-23812*, https://nvd.nist.gov/vuln/detail/CVE-2022-23812, Mar. 2022.

[77] L. Tal, *Alert: peacenotwar module sabotages npm developers in the node-ipc package to protest the invasion of Ukraine*, https://snyk.io/blog/peacenotwar-malicious-npm-node-ipc-package-vulnerability/, Mar. 2022.

[78] The Linux Foundation, *Open Source Security Foundation (OpenSSF)*, https://openssf.org/.

[79] B. Schneier, "Secrets and lies: digital security in a networked world," 2015.

[80] M. E. Zurko and R. T. Simon, "User-Centered Security," in *Proceedings of the 1996 Workshop on New Security Paradigms*, 1996, pp. 27–33.

[81] A. Adams and M. A. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[82] A. Whitten and J. D. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0.," in *8th Usenix Security Symposium (SEC'99)*, vol. 348, 1999, pp. 169–184.

[83] S. L. Garfinkel and R. C. Miller, "Johnny 2: a User Test of Key Continuity Management with S/MIME and Outlook Express," in *1st Symposium on Usable Privacy and Security (SOUPS'05)*, 2005, pp. 13–24.

[84] S. Sheng, L. Broderick, C. A. Koranda, and J. J. Hyland, "Why Johnny still can't encrypt: evaluating the usability of email encryption software," in *2nd Symposium on Usable Privacy and Security (SOUPS'06)*, ACM, 2006, pp. 3–4.

[85] S. Fahl, M. Harbach, T. Muders, M. Smith, and U. Sander, "Helping Johnny 2.0 to encrypt his Facebook conversations," in *8th Symposium on Usable Privacy and Security (SOUPS'12)*, 2012.

[86] L. F. Cranor and S. Garfinkel, *Security and Usability: Designing Secure Systems That People Can Use*. O'Reilly Media, Inc., 2005.

[87] S. Garfinkel and H. R. Lipford, "Usable Security: History, Themes, and Challenges," *Synthesis Lectures on Information Security, Privacy, and Trust*, vol. 5, no. 2, pp. 1–124, 2014.

[88] K. Mitnick, *The testimony of an ex-hacker*, https://www.pbs.org/wgbh/pages/frontline/shows/hackers/whoare/testimony.html, 2000.

[89] C. J. Alberts, A. J. Dorofee, R. Creel, R. J. Ellison, and C. Woody, "A systemic approach for assessing software supply-chain risk," in *2011 44th Hawaii International Conference on System Sciences*, IEEE, 2011, pp. 1–8.

[90] C. Theisen, N. Munaiah, M. Al-Zyoud, J. C. Carver, A. Meneely, and L. Williams, "Attack surface definitions: A systematic literature review," *Information and Software Technology*, vol. 104, pp. 94–103, 2018.

[91] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, "SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties," in *Proceedings of the 1st ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED'22)*, Nov. 2022.

[92] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "You are what you include: large-scale evaluation of remote javascript inclusions," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, 2012, p. 736.

[93] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vuln4Real: A Methodology for Counting Actually Vulnerable Dependencies," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1592–1609, May 2022.

[94] R. Shu, T. Xia, J. Chen, L. Williams, and T. Menzies, "How to better distinguish security bug reports (using dual hyperparameter optimization)," *Empirical Software Engineering*, vol. 26, pp. 1–37, 2021.

[95] G. A. A. Prana, A. Sharma, L. K. Shar, D. Foo, A. E. Santosa, A. Sharma, and D. Lo, "Out of sight, out of mind? How vulnerable dependencies affect open-source projects," *Empirical Software Engineering*, vol. 26, no. 4, p. 59, Jul. 2021.

[96] S. Frey, A. Rashid, P. Anthonysamy, M. Pinto-Albuquerque, and S. A. Naqvi, "The good, the bad and the ugly: a study of security decisions in a cyber-physical systems game," *IEEE Transactions on Software Engineering*, vol. 45, no. 5, pp. 521–536, 2017.

[97] B. Shreeve, J. Hallett, M. Edwards, K. M. Ramokapane, R. Atkins, and A. Rashid, "The best laid plans or lack thereof: Security decision-making of different stakeholder groups," *IEEE Transactions on Software Engineering*, 2020.

[98] A. Poller, L. Kocksch, S. Türpe, F. A. Epp, and K. Kinder-Kurlanda, "Can security become a routine? A study of organizational change in an agile software development group," in *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*, 2017, pp. 2489–2503.

[99] D. Baca, M. Boldt, B. Carlsson, and A. Jacobsson, "A novel security-enhanced agile software development process applied in an industrial setting," in *2015 10th International Conference on Availability, Reliability and Security*, IEEE, 2015, pp. 11–19.

[100] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You Get Where You're Looking For: The Impact of Information Sources on Code Security," in *37th IEEE Symposium on Security and Privacy (S&P'16)*, IEEE, May 2016, pp. 289–305.

[101] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "How internet resources might be helping you develop faster but less securely," *IEEE Security & Privacy*, vol. 15, no. 2, pp. 50–60, 2017.

[102] R. Stevens, D. Votipka, E. M. Redmiles, C. Ahern, P. Sweeney, and M. L. Mazurek, "The battle for New York: a case study of applied digital threat modeling at the enterprise level," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 621–637.

[103] H. Assal and S. Chiasson, "'Think secure from the beginning' A Survey with Software Developers," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–13.

[104] L. Bass, R. Holz, P. Rimba, A. B. Tran, and L. Zhu, "Securing a deployment pipeline," in *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, IEEE, 2015, pp. 4–7.

[105] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, pp. 164–175.

[106] A. Rahman, E. Farhana, and L. Williams, "The 'as code' activities: Development anti-patterns for infrastructure as code," *Empirical Software Engineering*, vol. 25, pp. 3430–3467, 2020.

[107] I. Koishybayev, A. Nahapetyan, R. Zachariah, S. Muralee, B. Reaves, A. Kapravelos, and A. Machiry, "Characterizing the Security of Github CI Workflows," in *31st USENIX Security Symposium (USENIX Sec'22)*, 2022, pp. 2747–2763.

[108] Y. Gu, L. Ying, H. Chai, C. Qia, H. Duam, and X. Gao, "Continuous Intrusion: Characterizing the Security of Continuous Integration Services," in *44th IEEE Symposium on Security and Privacy (S&P'23)*, IEEE, May 2023.

[109] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration," *Empirical Software Engineering*, vol. 23, pp. 384–417, 2018.

[110] B. Xu, L. An, F. Thung, F. Khomh, and D. Lo, "Why reinventing the wheels? An empirical study on library reuse and re-implementation," *Empirical Software Engineering*, vol. 25, no. 1, pp. 755–789, Sep. 2019.

[111] E. Larios Vargas, M. Aniche, C. Treude, M. Bruntink, and G. Gousios, "Selecting third-party libraries: The practitioners' perspective," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 245–256.

[112] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the JavaScript package ecosystem," in *Proceedings of the 13th International Conference on Mining Software Repositories*, May 2016, pp. 351–361.

[113] R. Kikas, G. Gousios, M. Dumas, and D. Pfahl, "Structure and Evolution of Package Dependency Networks," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017, pp. 102–112.

[114] R. Abdalkareem, O. Nourry, S. Wehaibi, S. Mujahid, and E. Shihab, "Why do developers use trivial packages? an empirical case study on npm," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, Aug. 2017, pp. 385–395.

[115] R. G. Kula, A. Ouni, D. M. German, and K. Inoue, "On the Impact of Micro-Packages: An Empirical Study of the npm JavaScript Ecosystem," Sep. 2017, arXiv:1709.04638 [cs].

[116] A. Decan, T. Mens, and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network," in *Proceedings of the 15th international conference on mining software repositories*, 2018, pp. 181–191.

[117] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small World with High Risks: A Study of Security Threats in the npm Ecosystem," in *Proceedings of the 28th USENIX Conference on Security Symposium (SEC'19)*, 2019, pp. 995–1010.

[118] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila, and L. Williams, "What are weak links in the npm supply chain?" In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, 2022, pp. 331–340.

[119] M. Valiev, B. Vasilescu, and J. Herbsleb, "Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 644–655.

[120] J. Kabbedijk and S. Jansen, "Steering Insight: An Exploration of the Ruby Software Ecosystem," in *Software Business*, vol. 80, Series Title: Lecture Notes in Business Information Processing, 2011, pp. 44–55.

[121] D. M. German, B. Adams, and A. E. Hassan, "The Evolution of the R Software Ecosystem," in *2013 17th European Conference on Software Maintenance and Reengineering*, Mar. 2013, pp. 243–252.

[122] G. Bavota, G. Canfora, M. D. Penta, R. Oliveto, and S. Panichella, "The Evolution of Project Inter-dependencies in a Software Ecosystem: The Case of Apache," in *2013 IEEE International Conference on Software Maintenance*, Sep. 2013, pp. 280–289.

[123] R. Bloemen, C. Amrit, S. Kuhlmann, and G. Ordóñez–Matamoros, "Gentoo package dependencies over time," in *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, 2014, pp. 404–407.

[124] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vulnerable open source dependencies: Counting those that matter," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.

[125] Y. Wang, B. Chen, K. Huang, B. Shi, C. Xu, X. Peng, Y. Wu, and Y. Liu, "An empirical study of usages, updates and risks of third-party libraries in java projects," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2020, pp. 35–45.

[126] I. J. Mojica, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. E. Hassan, "A Large-Scale Empirical Study on Software Reuse in Mobile Apps," *IEEE Software*, vol. 31, no. 2, pp. 78–86, Mar. 2014.

[127] Y. Gu, L. Ying, Y. Pu, X. Hu, H. Chai, R. Wang, X. Gao, and H. Dua, "Investigating Package Related Security Threats in Software Registries," in *44th IEEE Symposium on Security and Privacy (S&P'23)*, IEEE, May 2023.

[128] C. Liu, S. Chen, L. Fan, B. Chen, Y. Liu, and X. Peng, "Demystifying the Vulnerability Propagation and Its Evolution via Dependency Trees in the NPM Ecosystem," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 672–684.

[129] G. Ferreira, L. Jia, J. Sunshine, and C. Kästner, "Containing malicious package updates in npm with a lightweight permission system," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, 2021, pp. 1334–1346.

[130] J. Jueckstock and A. Kapravelos, "Visiblev8: In-browser monitoring of javascript in the wild," in *Proceedings of the Internet Measurement Conference (IMC)*, Oct. 2019, pp. 393–405.

[131] P. Agten, S. Van Acker, Y. Brondsema, P. H. Phung, L. Desmet, and F. Piessens, "JSand: complete client-side sandboxing of third-party JavaScript without browser modifications," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 1–10.

[132] D. Stefan, E. Z. Yang, P. Marchenko, A. Russo, D. Herman, B. Karp, and D. Mazieres, "Protecting Users by Confining JavaScript with COWL," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 131–146.

[133] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, Feb. 2019.

[134] A. Decan and T. Mens, "What Do Package Dependencies Tell Us About Semantic Versioning?" *IEEE Transactions on Software Engineering*, vol. 47, no. 6, pp. 1226–1240, Jun. 2021.

[135] Y. Wang, M. Wen, Z. Liu, R. Wu, R. Wang, B. Yang, H. Yu, Z. Zhu, and S.-C. Cheung, "Do the dependency conflicts in my project matter?" In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Oct. 2018, pp. 319–330.

[136] F. R. Cogo, G. A. Oliva, and A. E. Hassan, "An Empirical Study of Dependency Downgrades in the npm Ecosystem," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2457–2470, Nov. 2021.

[137] I. Koishybayev and A. Kapravelos, "Mininode: Reducing the Attack Surface of Node. js Applications.," in *The 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, Oct. 2020, pp. 121–134.

[138] Y. M. Mileva, V. Dallmeier, and A. Zeller, "Mining API Popularity," in *Testing – Practice and Research Techniques*, 2010, pp. 173–180.

[139] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, and M. Di Penta, "CrossRec: Supporting software developers by recommending third-party libraries," *Journal of Systems and Software*, vol. 161, Mar. 2020.

[140] R. E. Zapata, R. G. Kula, B. Chinthanet, T. Ishio, K. Matsumoto, and A. Ihara, "Towards smoother library migrations: A look at vulnerable dependency migrations at function level for npm javascript packages," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2018, pp. 559–563.

[141] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping Through Hoops: Why Do Java Developers Struggle with Cryptography APIs?" In *38th IEEE/ACM International Conference on Software Engineering (ICSE'16)*, 2016, pp. 935–946.

[142] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the Usability of Cryptographic APIs," in *38th IEEE Symposium on Security and Privacy (S&P'17)*, 2017.

[143] F. López de la Mora and S. Nadi, "An Empirical Study of Metric-Based Comparisons of Software Libraries," in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2018, pp. 22–31.

[144] P. L. Gorski, L. L. Iacono, D. Wermke, C. Stransky, S. Möller, Y. Acar, and S. Fahl, "Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse," in *14th Symposium on Usable Privacy and Security (SOUPS'18)*, 2018.

[145] P. L. Gorski, Y. Acar, L. Lo Iacono, and S. Fahl, "Listen to developers! A participatory design study on security warnings for cryptographic APIs," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[146] J. Jancar, M. Fourné, D. D. A. Braga, M. Sabt, P. Schwabe, G. Barthe, P.-A. Fouque, and Y. Acar, ""They're not that hard to mitigate": What Cryptographic Library Developers Think About Timing Attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 632–649.

[147] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Department of Computer Science, The University of Auckland, New Zealand, Tech. Rep., 1997.

[148] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation - tools for software protection," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 735–746, Aug. 2002.

[149] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA'10)*, Nov. 2010, pp. 297–300.

[150] I. Ullah, R. Boreli, S. S. Kanhere, and S. Chawla, "ProfileGuard: Privacy Preserving Obfuscation for Mobile User Profiles," in *13th Workshop on Privacy in the Electronic Society (WPES'14)*, 2014, pp. 83–92.

[151] M. Protsenko and T. Müller, "PANDORA applies non-deterministic obfuscation randomly to Android," in *8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE'13)*, Oct. 2013, pp. 59–67.

[152] M. Zheng, P. P. C. Lee, and J. C. S. Lui, "ADAM: An Automatic and Extensible Platform to Stress Test Android Anti-virus Systems," in *9th Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'12)*, 2013, pp. 82–101.

[153] V. Rastogi, Y. Chen, and X. Jiang, "Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 1, pp. 99–108, Jan. 2014.

[154] H. Huang, S. Zhu, P. Liu, and D. Wu, "A Framework for Evaluating Mobile App Repackaging Detection Algorithms," in *6th International Conference on Trust and Trustworthy Computing (TRUST'13)*, 2013, pp. 169–186.

[155] J. Hoffmann, T. Rytilahti, D. Maiorca, M. Winandy, G. Giacinto, and T. Holz, "Evaluating Analysis Tools for Android Apps: Status Quo and Robustness Against Obfuscation," in *6th ACM Conference on Data and Application Security and Privacy (CODASPY'16)*, 2016, pp. 139–141.

[156] L. Li, T. F. Bissyande, J. Klein, and Y. Le Traon, "An Investigation into the Use of Common Libraries in Android Apps," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Mar. 2016, pp. 403–414.

[157] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo, "LibD: Scalable and Precise Third-Party Library Detection in Android Markets," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, May 2017, pp. 335–346.

[158] Z. Ma, H. Wang, Y. Guo, and X. Chen, "LibRadar: fast and accurate detection of third-party libraries in Android apps," in *Proceedings of the 38th International Conference on Software Engineering Companion*, May 2016, pp. 653–656.

[159] L. Glanz, S. Amann, M. Eichberg, M. Reif, B. Hermann, J. Lerch, and M. Mezini, "CodeMatch: Obfuscation Won't Conceal Your Repackaged App," in *11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'17)*, 2017, pp. 638–648.

[160] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security," in *19th ACM Conference on Computer and Communication Security (CCS'12)*, 2012, pp. 50–61.

[161] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl, "To Pin or Not to Pin Helping App Developers Bullet Proof Their TLS Connections," in *24th Usenix Security Symposium (SEC'15)*, 2015, pp. 239–254.

[162] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking SSL Development in an Appified World," in *20th ACM Conference on Computer and Communication Security (CCS'13)*, 2013, pp. 49–60.

[163] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10)*, 2010, pp. 393–407.

[164] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, 5:1–5:29, Jun. 2014.

[165] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie, "PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play.," in *USENIX Security Symposium*, 2019, pp. 585–602.

[166] B. Andow, S. Y. Mahmud, J. Whitaker, W. Enck, B. Reaves, K. Singh, and S. Egelman, "Actions speak louder than words: Entity-sensitive privacy policy and data flow analysis with policheck," in *Proceedings of the 29th USENIX Security Symposium (USENIX Security'20)*, 2020.

[167] R. Balebako, A. Marsh, J. Lin, J. I. Hong, and L. F. Cranor, "The privacy and security behaviors of smartphone app developers," in *Workshop on Usable Security (USEC'14)*, 2014.

[168] R. Balebako and L. Cranor, "Improving App Privacy: Nudging App Developers to Protect User Privacy," *IEEE Security & Privacy*, vol. 12, no. 4, pp. 55–58, Jul. 2014.

[169] S. Jain and J. Lindqvist, "Should I Protect You? Understanding Developers' Behavior to Privacy-preserving Apis," in *Workshop on Usable Security (USEC'14)*, 2014.

[170] 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.

[171] 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.

[172] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal*, Springer, Jun. 2020, pp. 23–43.

[173] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "Taxonomy of attacks on open-source software supply chains," in *44th IEEE Symposium on Security and Privacy, IEEE S&P 2023*, IEEE, May 2023.

[174] 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.

[175] 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.

[176] 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.

[177] 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*, 2007, pp. 223–228.

[178] 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.

[179] 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.

[180] 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.

[181] M. Syeed, J. Lindman, and I. Hammouda, "Measuring Perceived Trust in Open Source Software Communities," in *Open Source Systems: Towards Robust Practices*, 2017.

[182] S. Murdoch and N. Leaver, "Anonymity vs. Trust in Cyber-Security Collaboration," in *Proceedings of the 2nd ACM Workshop on Information Sharing and Collaborative Security, (WISCS 2015)*, 2015, pp. 27–29.

[183] 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*, 2011, pp. 133–142.

[184] 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*, 2010, pp. 229–235.

[185] 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*, 2014.

[186] 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.

[187] 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*, 2017.

[188] 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.

[189] 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*, 2019, pp. 138–142.

[190] A. Pietri, D. Spinellis, and S. Zacchiroli, "The Software Heritage Graph Dataset: Large-Scale Analysis of Public Software Development History," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 1–5.

[191] 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*, 2014, pp. 396–399.

[192] G. Gousios and D. Spinellis, "GHTorrent: GitHub's Data from a Firehose," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, 2012, pp. 12–21.

[193] 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*, 2014, pp. 384–387.

[194] 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.

[195] M. Meli, M. R. McNiece, and B. Reaves, "How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories," in *26th Annual Network and Distributed System Security Symposium (NDSS'19)*, 2019.

[196] A. Krause, J. H. Klemmer, N. Huaman, D. Wermke, Y. Acar, *et al.*, "Committed by Accident: Studying Prevention and Remediation Strategies Against Secret Leakage in Source Code Repositories," *arXiv preprint arXiv:2211.06213*, Nov. 2022.

[197] R. Feng, Z. Yan, S. Peng, and Y. Zhang, "Automated Detection of Password Leakage from Public GitHub Repositories," in *44th IEEE/ACM International Conference on Software Engineering (ICSE'22)*, 2022, pp. 175–186.

[198] S. K. Basak, L. Neil, B. Reaves, and L. Williams, "What are the Practices for Secret Management in Software Artifacts?" In *2022 IEEE Secure Development Conference (SecDev'22)*, IEEE, 2022, pp. 69–76.

[199] 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*, 2014, pp. 234–236.

[200] 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.

[201] 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.

[202] 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.

[203] 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*, 2012, pp. 183–194.

[204] 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*, 2012, pp. 771–781.

[205] 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.

[206] 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.

[207] 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*, 2014, pp. 348–351.

[208] 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*, 2015.

[209] 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.

[210] 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*, 2017, pp. 334–344.

[211] 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.

[212] 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*, 2018, pp. 18–21.

[213] 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.

[214] 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.

[215] 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.

[216] H. Plate, S. E. Ponta, and A. Sabetta, "Impact assessment for vulnerabilities in open-source software libraries," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2015, pp. 411–420.

[217] 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*, 2020, pp. 16–20.

[218] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar, "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*, 2015, pp. 426–437.

[219] 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*, 2017, pp. 914–919.

[220] N. Imtiaz, B. Murphy, and L. Williams, "How do developers act on static analysis alerts? an empirical study of coverity usage," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, 2019, pp. 323–333.

[221] K. Hogan, N. Warford, R. Morrison, D. Miller, S. Malone, and J. Purtilo, "The challenges of labeling vulnerability-contributing commits," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, IEEE, 2019, pp. 270–275.

[222] N. Zahan, S. Shohan, D. Harris, and L. Williams, "PREPRINT: Do OpenSSF Scorecard Practices Contribute to Fewer Vulnerabilities?" *arXiv preprint arXiv:2210.14884*, Oct. 2022.

[223] N. Zahan, P. Kanakiya, B. Hambleton, S. Shohan, and L. Williams, "PREPRINT: Can the OpenSSF Scorecard be used to measure the security posture of npm and PyPI?" *arXiv preprint arXiv:2208.03412*, Aug. 2022.

[224] 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.

[225] 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.

[226] 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.

[227] 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*, 2020.

[228] 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*, May 2022.

[229] 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.

[230] 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.

[231] 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.

[232] 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.

[233] 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*, 2017.

[234] 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*, 2018.

[235] C. Hannebauer and V. Gruhn, "Motivation of Newcomers to FLOSS Projects," in *Proceedings of the 12th International Symposium on Open Collaboration*, 2016.

[236] 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*, 2015, pp. 1379–1392.

[237] 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.

[238] 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*, 2016, pp. 273–284.

[239] 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*, 2012.

[240] 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.

[241] 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*, 2020, pp. 46–50.

[242] V. N. Subramanian, I. Rehman, M. Nagappan, and R. G. Kula, "Analyzing first contributions on GitHub: what do newcomers do," *IEEE Software*, 2020.

[243] S. Balali, U. Annamalai, H. S. Padala, B. Trinkenreich, M. A. Gerosa, I. Steinmacher, and A. Sarma, "Recommending Tasks to Newcomers in OSS Projects: How Do Mentors Handle It?" In *Proceedings of the 16th International Symposium on Open Collaboration*, 2020.

[244] 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.

[245] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" In *2013 35th International Conference on Software Engineering (ICSE)*, IEEE, 2013, pp. 672–681.

[246] 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.

[247] 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.

[248] T. W. Thomas, M. Tabassum, B. Chu, and H. Lipford, "Security during application development: An application security expert perspective," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[249] 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.

[250] D. Botta, R. Werlinger, A. Gagné, K. Beznosov, L. Iverson, S. Fels, and B. Fisher, "Towards Understanding IT Security Professionals and Their Tools," in *Proceedings of the 3rd Symposium on Usable Privacy and Security*, 2007, pp. 100–111.

[251] 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.

[252] M. Silic and A. Back, "Information Security and Open Source Dual Use Security Software: Trust Paradox," in *Open Source Software: Quality Verification*, 2013, pp. 194–206.

[253] 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.

[254] J. M. Haney, M. Theofanos, Y. Acar, and S. S. Prettyman, "" We make it a big deal in the company": Security Mindsets in Organizations that Develop Cryptographic Products.," in *SOUPS USENIX Security Symposium*, 2018, pp. 357–373.

[255] 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.

[256] 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.

[257] 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.

[258] S. Zhou, B. Vasilescu, and C. Kästner, "How has forking changed in the last 20 years? a study of hard forks on github," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 445–456.

[259] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung, "When and how to make breaking changes: Policies and practices in 18 open source software ecosystems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–56, 2021.

[260] F. Jansen, S. Jansen, and F. Hou, "TrustSECO: an interview survey into software trust," *arXiv preprint arXiv:2101.06138*, Jan. 2021.

[261] J. Ghofrani, P. Heravi, K. A. Babaei, and M. D. Soorati, "Trust challenges in reusing open source software: an interview-based initial study," in *Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume B*, Sep. 2022, pp. 110–116.

[262] S. Butler, J. Gamalielsson, B. Lundell, C. Brax, A. Mattsson, T. Gustavsson, J. Feist, B. Kvarnström, and E. Lönroth, "On business adoption and use of reproducible builds for open and closed source software," *Software Quality Journal*, Nov. 2022.

[263] 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*, May 2022.

[264] M. Fourné, D. Wermke, W. Enck, S. Fahl, and Y. Acar, "It's like flossing your teeth: On the Importance and Challenges of Reproducible Builds for Software Supply Chain Security," in *44th IEEE Symposium on Security and Privacy (S&P'23)*, IEEE, May 2023.

[265] GitHub, *The State of the Octoverse*, https://octoverse.github.com/, 2020.

[266] Microsoft, *GitHub*, https://github.com, 2008.

[267] GitLab Inc., *GitLab*, https://gitlab.com, 2014.

[268] 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, 2021.

[269] 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/, 2021.

[270] 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, Nov. 2021.

[271] 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/, Nov. 2021.

[272] The Linux Foundation, "Open Source Software Supply Chain Security," Tech. Rep., Feb. 2020.

[273] 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, Feb. 2020.

[274] Linux Foundation's Technical Advisory Board, *Report on University of Minnesota Breach-of-Trust Incident*, https://lwn.net/ml/linux-kernel/202105051005.49BFABCE\spacefactor\@m{}keescook/, May 2021.

[275] 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*, 2012, pp. 1277–1286.

[276] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, and V. Filkov, "The Sky is Not the Limit: Multitasking across GitHub Projects," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 994–1005.

[277] 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.

[278] 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*, 2014, pp. 345–355.

[279] 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*, 2014, pp. 356–366.

[280] 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*, 2019, pp. 51–60.

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

[282] 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*, 2020, pp. 57–60.

[283] 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.

[284] 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*, 2019, pp. 116–129.

[285] 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*, 2015, pp. 202–207.

[286] 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*, 2015, pp. 817–828.

[287] 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/, Apr. 2021.

[288] 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, May 2021.

[289] K. Charmaz, *Constructing Grounded Theory*. Sage, 2014.

[290] A. Strauss and J. M. Corbin, *Grounded theory in practice*. Sage, 1997.

[291] J. Corbin and A. Strauss, "Grounded theory research: Procedures, canons and evaluative criteria," *Zeitschrift für Soziologie*, vol. 19, no. 6, pp. 418–427, 1990.

[292] C. Urquhart, *Grounded theory for qualitative research: A practical guide*. Sage, 2012.

[293] M. Birks and J. Mills, *Grounded theory: A practical guide*. Sage, 2015.

[294] E. Kenneally and D. Dittrich, "The Menlo Report: Ethical principles guiding information and communication technology research," *SSRN Electronic Journal*, Aug. 2012.

[295] *Guidelines for research on the kernel community*, https://lwn.net/Articles/888891/, Mar. 2022.

[296] W. Turton and K. Mehrotra, *FireEye Discovered SolarWinds Breach While Probing Own Hack*, https://www.bloomberg.com/news/articles/2020-12-15/fireeye-stumbled-across-solarwinds-breach-while-probing-own-hack, 2020.

[297] C. Cimpanu, *SEC filings: SolarWinds says 18,000 customers were impacted by recent hack*, https://www.zdnet.com/article/sec-filings-solarwinds-says-18000-customers-are-impacted-by-recent-hack/, Dec. 2020.

[298] Check Point Research, *CloudGuard Spectral detects several malicious packages on PyPI – the official software repository for Python developers*, https://research.checkpoint.com/2022/cloudguard-spectral-detects-several-malicious-packages-on-pypi-the-official-software-repository-for-python-developers/, Aug. 2022.

[299] M. Alfadel, D. E. Costa, and E. Shihab, "Empirical Analysis of Security Vulnerabilities in Python Packages," in *2021 IEEE international conference on software analysis, Evolution and Reengineering (SANER)*, IEEE, 2021, pp. 446–457.

[300] S. E. Ponta, H. Plate, and A. Sabetta, "Detection, Assessment and Mitigation of Vulnerabilities in Open Source Dependencies," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3175–3215, 2020.

[301] F. López de la Mora and S. Nadi, "Which Library Should I Use?: A Metric-Based Comparison of Software Libraries," in *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, 2018, pp. 37–40.

[302] J. M. Haney, M. Theofanos, Y. Acar, and S. S. Prettyman, "'We make it a big deal in the company': Security mindsets in organizations that develop cryptographic products," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, 2018, pp. 357–373.

[303] N. McDonald, S. Schoenebeck, and A. Forte, "Reliability and Inter-Rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice," *Proc. ACM Hum.-Comput. Interact.*, vol. 3, no. CSCW, Nov. 2019.

[304] S. Gallagher, *Rage-quit: Coder unpublished 17 lines of JavaScript and "broke the Internet"*, https://arstechnica.com/information-technology/2016/03/rage-quit-coder-unpublished-17-lines-of-javascript-and-broke-the-internet/, Mar. 2016.

[305] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A Study of Android Application Security," in *20th Usenix Security Symposium (SEC'11)*, 2011, pp. 21–21.

[306] H. Lockheimer, *Android and Security*, http://googlemobile.blogspot.com/2012/02/android-and-security.html, Feb. 2012.

[307] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "MAST: Triage for Market-scale Mobile Malware Analysis," in *6th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'13)*, 2013, pp. 13–24.

[308] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song, "Juxtapp: A Scalable System for Detecting Code Reuse among Android Applications," in *9th Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'12)*, Jul. 2013, pp. 62–81.

[309] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "ViewDroid: Towards Obfuscation-resilient Mobile Application Repackaging Detection," in *7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'14)*, 2014, pp. 25–36.

[310] M. Linares-Vásquez, A. Holtzhauer, C. Bernal-Cárdenas, and D. Poshyvanyk, "Revisiting Android Reuse Studies in the Context of Code Obfuscation and Library Usages," in *11th Working Conference on Mining Software Repositories (MSR'14)*, 2014, pp. 242–251.

[311] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient Privilege De-Escalation for Ad Libraries in Mobile Apps," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 2015, pp. 89–103.

[312] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *8th Symposium on Usable Privacy and Security (SOUPS'12)*, May 2012, pp. 95–109.

[313] N. Viennot, E. Garcia, and J. Nieh, "A Measurement Study of Google Play," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 1, pp. 221–233, Jun. 2014.

[314] J. Crussell, C. Gibler, and H. Chen, "AnDarwin: Scalable Detection of Android Application Clones Based on Semantics," *IEEE Transactions on Mobile Computing*, vol. 14, no. 10, pp. 2007–2019, Oct. 2015.

[315] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting Repackaged Smartphone Applications in Third-party Android Marketplaces," in *2nd ACM Conference on Data and Application Security and Privacy (CODASPY'12)*, 2012, pp. 317–326.

[316] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, Scalable Detection of "Piggybacked" Mobile Applications," in *3rd ACM Conference on Data and Application Security and Privacy (CODASPY'13)*, 2013, pp. 185–196.

[317] V. Tendulkar and W. Enck, "An application package configuration approach to mitigating Android SSL vulnerabilities," in *2014 Mobile Security Technologies Workshop (MoST'14)*, 2014.

[318] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, "The effectiveness of source code obfuscation: An experimental assessment," in *17th IEEE International Conference on Program Comprehension (ICPC'09)*, IEEE, 2009, pp. 178–187.

[319] M. Ceccato, M. Penta, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, "A Family of Experiments to Assess the Effectiveness and Efficiency of Source Code Obfuscation Techniques," *Empirical Software Engineering*, vol. 19, no. 4, pp. 1040–1074, Aug. 2014.

[320] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto, "Stealth attacks: An extended insight into the obfuscation effects on Android malware," *Computers & Security*, vol. 51, pp. 16–31, 2015.

[321] J.-M. Borello and L. Mé, "Code obfuscation techniques for metamorphic viruses," *Journal in Computer Virology*, vol. 4, no. 3, pp. 211–220, 2008.

[322] C. Collberg and J. Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*, 1st. Addison-Wesley Professional, 2009.

[323] J.-T. Chan and W. Yang, "Advanced Obfuscation Techniques for Java Bytecode," *Journal of Systems and Software*, vol. 71, no. 1-2, pp. 1–10, Apr. 2004.

[324] Y. Sakabe, M. Soshi, and A. Miyaji, "Java obfuscation approaches to construct tamper-resistant object-oriented programs," *IPSJ Digital Courier*, vol. 1, pp. 349–361, 2005.

[325] T. W. Hou, H. Y. Chen, and M. H. Tsai, "Three control flow obfuscation methods for Java software," *IEE Proceedings - Software*, vol. 153, no. 2, pp. 80–86, Apr. 2006.

[326] S. Ghosh, S. R. Tandan, and K. Lahre, "Shielding Android Application Against Reverse Engineering," *International Journal of Engineering Research & Technology*, vol. 2, no. 6, 2013.

[327] P. Faruki, A. Bharmal, V. Laxmi, M. S. Gaur, M. Conti, and M. Rajarajan, "Evaluation of Android Anti-malware Techniques against Dalvik Bytecode Obfuscation," in *13th IEEE International Converence on Trust, Security and Privacy in Computing and Communications (TrustCom'14)*, Sep. 2014, pp. 414–421.

[328] S. K. Udupa, S. K. Debray, and M. Madou, "Deobfuscation: reverse engineering obfuscated code," in *12th Working Conference on Reverse Engineering (WCRE'05)*, Nov. 2005, 10pp.

[329] B. Bichsel, V. Raychev, P. Tsankov, and M. Vechev, "Statistical Deobfuscation of Android Applications," in *23nd ACM Conference on Computer and Communication Security (CCS'16)*, 2016, pp. 343–355.

[330] C. Linn and S. Debray, "Obfuscation of Executable Code to Improve Resistance to Static Disassembly," in *10th ACM Conference on Computer and Communication Security (CCS'03)*, 2003, pp. 290–299.

[331] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "AndroSimilar: Robust Statistical Feature Signature for Android Malware Detection," in *6th International Conference on Security of Information and Networks (SIN'13)*, 2013, pp. 152–159.

[332] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: Having a Deeper Look into Android Applications," in *28th ACM Annual Symposium on Applied Computation (SAC'13)*, 2013, pp. 1808–1815.

[333] J. Garcia, M. Hammad, B. Pedrood, A. Bagheri-Khaligh, and S. Malek, "Obfuscation-resilient, efficient, and accurate detection and family identification of Android malware," Department of Computer Science, George Mason University, Virginia, Tech. Rep., 2015.

[334] M. Backes, S. Bugiel, and E. Derr, "Reliable Third-Party Library Detection in Android and Its Security Applications," in *23nd ACM Conference on Computer and Communication Security (CCS'16)*, 2016, pp. 356–367.

[335] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What Do Programmers Know About Software Energy Consumption?" *IEEE Software*, vol. 33, no. 3, pp. 83–89, May 2016.

[336] S. Dong, M. Li, W. Diao, X. Liu, J. Liu, Z. Li, F. Xu, K. Chen, X. Wang, and K. Zhang, "Understanding Android Obfuscation Techniques: A Large-Scale Investigation in the Wild," *arXiv:1801.01633 [cs]*, vol. 1801, Jan. 2018.

[337] Google Inc., *Shrink Your Code and Resources*, https://developer.android.com/studio/build/shrink-code.html, Apr. 2017.

[338] R. Potharaju, M. Rahman, and B. Carbunar, "A Longitudinal Study of Google Play," *IEEE Transactions on Computational Social Systems*, vol. 4, no. 3, pp. 135–149, Sep. 2017.

[339] D. Morris, *General Dynamics Wins $7.6 Billion Contract to Supply Microsoft Office Software to the Pentagon*, https://fortune.com/2019/08/29/general-dynamics-pentagon-contract/, Aug. 2019.

[340] C. Schaer, *Microsoft Office 365: Banned in German schools over privacy fears*, https://www.zdnet.com/article/microsoft-office-365-banned-in-german-schools-over-privacy-fears/, Jul. 2019.

[341] J. Salter, *Office 365 declared illegal in German schools due to privacy risks*, https://arstechnica.com/information-technology/2019/07/germany-threatens-to-break-up-with-microsoft-office-again/, Jul. 2019.

[342] E. Dedezade, *Microsoft to deliver cloud services from new datacentres in Germany in 2019 to meet evolving customer needs*, https://news.microsoft.com/europe/2018/08/31/microsoft-to-deliver-cloud-services-from-new-datacentres-in-germany-in-2019-to-meet-evolving-customer-needs/, Aug. 2019.

[343] P. Lorimer, *Microsoft Office 365 and Dynamics 365 now available from new German datacenter regions*, https://www.microsoft.com/en-us/microsoft-365/blog/2020/02/20/microsoft-office-365-dynamics-365-now-available-from-new-german-datacenter-regions/, Feb. 2020.

[344] Google, *Google Drive - Google*, https://www.google.com/intl/en/drive/, Jun. 2020.

[345] Microsoft, *Office 365 - Microsoft Office*, https://www.office.com/, Jun. 2020.

[346] A. Inc., *iWork - Apple*, https://www.apple.com/iwork/, Jun. 2020.

[347] T. D. Foundation, *LibreOffice*, https://www.libreoffice.org/download/libreoffice-online/, Jun. 2020.

[348] A. S. SIA, *OnlyOffice*, https://www.onlyoffice.com/cloud-office.aspx, Jun. 2020.

[349] S. Presser, M. P. Couper, J. T. Lessler, E. Martin, J. Martin, J. M. Rothgeb, and E. Singer, "Methods for Testing and Evaluating Survey Questions," *Public Opinion Quarterly*, vol. 68, no. 1, pp. 109–130, Mar. 2004.

[350] E. Peer, J. Vosgerau, and A. Acquisti, "Reputation as a Sufficient Condition for Data Quality on Amazon Mechanical Turk," *Behavior research methods*, vol. 46, Dec. 2013.

[351] N. C. Schaeffer and S. Presser, "The science of asking questions," *Annual review of sociology*, vol. 29, no. 1, pp. 65–88, 2003.

[352] P. V. Marsden and J. D. Wright, *Handbook of survey research*. Emerald Group Publishing, 2010, pp. 263–314.

[353] K. Krippendorff, *Content Analysis: An Introduction to Its Methodology (2nd ed.)* SAGE Publications, 2004.

[354] K. P. Burnham, "Multimodel Inference: Understanding AIC and BIC in Model Selection," *Sociological Methods & Research*, vol. 33, no. 2, pp. 261–304, 2004.

[355] D. Dittrich and E. Kenneally, "The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research," U.S. Department of Homeland Security, Tech. Rep., Aug. 2012.

[356] R. Kennedy, S. Clifford, T. Burleigh, R. Jewell, and P. Waggoner, *The Shape of and Solutions to the MTurk Quality Crisis*, Oct. 2018.

[357] E. M. Redmiles, S. Kross, and M. L. Mazurek, "How I Learned to Be Secure: A Census-Representative Survey of Security Advice Sources and Behavior," in *23nd ACM Conference on Computer and Communication Security (CCS'16)*, 2016.

[358] N. Heath, *From Linux to Windows 10: Why did Munich switch and why does it matter?* https://www.techrepublic.com/article/linux-to-windows-10-why-did-munich-switch-and-why-does-it-matter/, Nov. 2017.

[359] G. D'Angelo, F. Vitali, and S. Zacchiroli, "Content Cloaking: Preserving Privacy with Google Docs and Other Web Applications," in *Proc. 25th ACM Symposium on Applied Computing (SAC'10)*, 2010, pp. 826–830.

[360] L. Adkinson-Orellana, D. A. Rodríguez-Silva, F. Gil-Castiñeira, and J. C. Burguillo-Rial, "Privacy for Google Docs: Implementing a transparent encryption layer," in *Proc. 2nd Cloud Computing International Conference - CloudViews*, 2010.

[361] W. Hu, T. Yang, and J. N. Matthews, "The good, the bad and the ugly of consumer cloud storage.," *Operating systems review*, vol. 44, no. 3, pp. 110–115, 2010.

[362] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside Dropbox: Understanding Personal Cloud Storage Services," in *Proceedings of the 2012 Internet Measurement Conference*, ACM, 2012, pp. 481–494.

[363] D. Svantesson and R. Clarke, "Privacy and consumer risks in cloud computing," *Computer Law & Security Review*, vol. 26, no. 4, pp. 391–397, 2010.

[364] S. Nestori and V. Tessa, "Is My Office 365 GDPR Compliant? - Security Issues in Authentication and Administration," in *Proceedings of the 20th International Conference on Enterprise Information Systems, ICEIS 2018, Funchal, Madeira, Portugal, March 21-24, 2018, Volume 2.*, 2018, pp. 299–305.

[365] K. M. Ramokapane, A. Rashid, and J. M. Such, ""I feel stupid I can't delete ...": A Study of Users' Cloud Deletion Practices and Coping Strategies," in *Proc. 13th Symposium on Usable Privacy and Security (SOUPS'17)*, 2017, pp. 241–256.

[366] M. van Dijk and A. Juels, "On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing," in *6th USENIX Workshop on Hot Topics in Security (HotSec'11)*, 2010, pp. 1–8.

[367] M. Nebeling, M. Geel, O. Syrotkin, and M. C. Norrie, "MUBox: Multi-user aware personal cloud storage," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ACM, 2015, pp. 1855–1864.

[368] X. Tan and Y. Kim, "User acceptance of SaaS-based collaboration tools: a case of Google Docs," *Journal of Enterprise Information Management*, vol. 28, no. 3, pp. 423–442, 2015.

[369] C. Marshall and J. C. Tang, "That syncing feeling: early user experiences with the cloud," in *Proceedings of the designing interactive systems conference*, ACM, 2012, pp. 544–553.

[370] D. Burda and F. Teuteberg, "The role of trust and risk perceptions in cloud archiving–Results from an empirical study," *The Journal of High Technology Management Research*, vol. 25, no. 2, pp. 172–187, 2014.

[371] J. W. Clark, P. Snyder, D. McCoy, and C. Kanich, "I saw images I didn't even know I had: Understanding user perceptions of cloud storage privacy," in *Proc. 33rd ACM Conference on Human Factors in Computing Systems (CHI'15)*, 2015, pp. 1641–1644.

[372] M. T. Khan, M. Hyun, C. Kanich, and B. Ur, "Forgotten But Not Gone: Identifying the Need for Longitudinal Data Management in Cloud Storage," in *Proc. 36th ACM Conference on Human Factors in Computing Systems (CHI'18)*, 2018, 543:1–543:12.

[373] C. Massey, T. Lennig, and S. Whittaker, "Cloudy forecast: an exploration of the factors underlying shared repository use," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2014, pp. 2461–2470.

[374] A. Mijuskovic and M. Ferati, "User awareness of existing privacy and security risks when storing data in the cloud," in *International Conference on e-Learning and the Knowledge Society*, European Commission, 2015, pp. 268–273.

[375] I. Ion, N. Sachdeva, P. Kumaraguru, and S. Čapkun, "Home is Safer Than the Cloud!: Privacy Concerns for Consumer Cloud Storage," in *7th Symposium on Usable Privacy and Security (SOUPS'11)*, 2011.

[376] I. Arpaci, K. Kilicer, and S. Bardakci, "Effects of security and privacy concerns on educational use of cloud services," *Computers in Human Behavior*, vol. 45, pp. 93–98, 2015.

Acronyms

2FA two-factor authentication. 21, 124, 137

BDFL Benevolent Dictator for Life. 56

CAPEC Common Attack Pattern Enumeration and Classification. 15

CI continuous integration. 29

CI/CD continuous integration and continuous delivery. 2, 17, 18, 29, 137

CIPAC Critical Infrastructure Partnership Advisory Council. 12

CISA Cybersecurity and Infrastructure Security Agency. 20

CLA Contributor License Agreement. 48, 57, 58

CRA Cyber Resilience Act. 17

CVE Common Vulnerabilities and Exposures. 15, 32, 33, 43

CVSS Common Vulnerability Scoring System. 14, 15

CWE Common Weakness Enumeration. 15

DPO Data Protection Officer. 24, 25

ENISA European Union Agency for Cybersecurity. 16

EPSS Exploit Prediction Scoring System. 15

ESF Enduring Security Framework. 12, 13

GDPR General Data Protection Regulation. 49, 74

IRB Institutional Review Board. 49, 74, 121

NGO non-governmental organization. 14

NIST National Institute of Standards and Technology. 16

NTIA National Telecommunications and Information Administration. 16

OpenSSF Open Source Security Foundation. 15, 33, 69, 77, 84, 86

OSC open source component. 35, 67–78, 80–82, 84–87, 135, 136, 138

OSP open source project. 41, 42, 44, 48, 52, 58–60, 68, 135, 138

OSS open source software. 19, 22, 31–35, 41–44, 46, 57–59, 61, 67, 68, 81, 83, 86, 136, 137

SAT static analysis tool. 30, 33, 34, 86

SBOM Software Bill of Materials. 15, 16, 76, 77, 86

SME small and medium enterprises. 8

SSC software supply chain. 2, 11, 16, 21

SSO single sign-on. 18

TLS Transport Layer Security. 14, 31, 94

VCC vulnerability-contributing commit. 33

VEX Vulnerability-Exploitability Exchange. 15