Understanding the Social Aspects of Software Security

by

Soroush Yousefi
B.Sc., Shahid Beheshti University, 2015

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Understanding the Social Aspects of Software Security

by

Soroush Yousefi
B.Sc., Shahid Beheshti University, 2015

Supervisory Committee

Dr. Margaret-Anne Storey, Supervisor
(Department of Computer Science)

Dr. Daniel M. German, Departmental Member
(Department of Computer Science)

ABSTRACT

Context: Security is a critical non-functional requirement that remains to be adequately addressed in software development. Security breaches occur with surprising regularity and many of them are due to inadequate understanding of development processes and tools. Researchers and practitioners have conducted studies and proposed new interventions to understand and improve security, but understanding how this research is related and how much progress has been made can be difficult. Moreover, some of the existing research has been criticized for focusing too much on technical aspects and not enough on understanding the human and social aspects that are a critical aspect of ensuring higher security.

Objective: My thesis addresses two objectives. First of all, I aim to develop a map of the research done to date with a focus on how much of the literature considers social and human aspects of security. Secondly, I aim to investigate how developers in industry approach security, what motivates them, and what activities they follow to improve security of the projects they work on.

Method: To meet my first objective, I conducted a systematic mapping study, finding 36 papers that study security issues using projects hosted on GitHub. I classified these papers into nine different problem areas. I also investigated how these papers address social aspects of security by applying a socio-technical research framework to capture the beneficiary of the reported research, the type of contribution produced by that research, and the research strategy used by the research in the papers. To meet the second objective and to address the gap identified in the mapping study, I conducted an interview study with 28 practitioners to understand their behavior towards security and what motivates them. I used a behavioral model from psychology to design the interviews and to understand developer behaviours and motivations towards security.

Results: The mapping study shows that much of the research to date has focused on advancing technical aspects with much less attention placed on studying how developers and practitioners might adopt and use the available security tools. My review suggests that human and social aspects are neglected in security research. I summarize gaps in the literature and suggest future areas for research. Also, I synthesized suggestions for how to mitigate security threats in projects hosted on GitHub. From the cross-sectional interview study, motivated by the mapping study to focus more on social aspects, I found practitioners need more support from their

organizations, peers, and managers to achieve higher security practice adoption. I also observed how different factors affect the adoption of security in software development, such as industry type and organization characteristics.

Conclusion: I provide an overview of security issues and how researchers approach improving security. I identify gaps and challenges in the literature and suggest future research areas. The gaps from the mapping study show a dearth of research on social and human aspects which motivated an interview study with developers from industry to understand their attitudes towards security and how they can be encouraged to adopt more secure practices. I conclude with insights that could potentially help organizations design new strategies to increase the adoption of more secure practices by software developers.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deep and sincere gratitude to my supervisor, Dr. Margaret-Anne (Peggy) Storey, for her mentorship, enthusiasm, and encouragement throughout my study and research stages. Without her invaluable guidance and inspiration, this thesis would not have been possible.

I would also like to thank Dr. Enrique Larios Vargas for his support, insightful feedback, and mentorship, which have contributed to the improvement of this thesis.

I would also like to thank Dr. Daniel M. German for his insightful feedback which have contributed to the improvement of this thesis.

To all present and former members of the CHISEL lab, I am grateful for your suggestions on this thesis, our friendship, and all the fun we have had: Arman Yousef Zadeh Shooshtari, Andreas Koenzen, Eirini Kalliamvakou, Jorin Weatherston, Leon Li, Omar Elazhary, Trishala Bhasin, Matt Termuende, Cassandra Cupryk, Alessandra Maciel Paz Milani, Neil Ernst, and Cassandra Petrachenko.

Lastly, I would like to thank my parents, sister,and friends for their love, understanding, and support along the way.

Soroush Yousefi

DEDICATION

For everyone interested in improving security in software development.

Chapter 1

Introduction

With the rapid growth of the internet and rise of technology since the early 2000s, software products have become an essential part of our lives. We use software in a wide variety of areas, from personal chores and entertainment, to medical use and banking services. This exponential growth of demand for software products and software development has brought new challenges, such as software security.

While many papers have studied the challenges in software security in order to mitigate issues [16] [38] [48], security complications have continued to increase every year. The National Vulnerability Database[1](NVD) has reported 184% increase in the number of vulnerabilities since 2016 [8] that cause many exploit incidents happening each year. This growth in the number vulnerabilities, shows there are still areas in software security that have not been addressed properly and require more attention.

The security problems that persist in software products range from secret leakage and exposing API keys [59] to vulnerabilities in third-party libraries [13] [43]. For these security issues, researchers have proposed solutions to help reduce or eliminate the vulnerabilities. However, it seems there are still some aspects in software security that have not been thoroughly investigated since we see more security bugs every year.

Security in software development has two sides: **social** and **technical**. Security in the context of software development is not just about performing static analysis, dynamic analysis, and security testing. It is also about how to arm developers with tools and training to check code for security bugs, conduct threat assessments to find potential threats, and perform various security testing before release.

[1]National Vulnerability Database (NVD), https://nvd.nist.gov

The **social aspects** of software security involve psychological and environmental factors that affect software developers' ability to apply security best practices in the software development process. These factors determine how developers perceive, adopt, and use security tools and practices. I use this definition of the social aspects of software security in this thesis to study the issues software developers face when they adopt security practices.

The **technical aspects** consist of tools, technologies, and security knowledge. These aspects have seen far more attention in the literature compared to the social aspects [69]. Therefore, to understand why security issues continue to increase every year, we need to have a clear understanding of the issues experienced and explore how we can potentially improve the current state of developers' security adoption.

I argue that software security is never just a technology problem, it is also a socio-technical problem. Having adequate tools to implement security in the Software Development Life Cycle (SDLC) is not a silver bullet if developers are not adopting security practices.

Ideally, software security would be improved by building security into software projects while they are being developed. This means providing developers with the proper tools and training, and facilitating a supportive environment to encourage the adoption of security practices. Current approaches have depended heavily on vulnerability management and static analysis but overlooked the importance of understanding developers' challenges and behavior towards security adoption.

In this thesis, I present a mapping study on papers that investigate security issues of projects hosted on GitHub[2]. With this mapping study, I aim to depict a clear picture of the existing security issues and better understand gaps in the literature. I categorized the security issues into nine different topics and used a socio-technical framework to analyze the papers. My analysis showed that most studies focus on the technical aspects of software security while few investigated security from the developers' perspective, meaning how developers perceive security.

After I discovered that software security research needs more attention on the social aspects of security, I collaborated in a research study to investigate developer behaviors towards adopting security. I conducted this study with a team of researchers from the University of Victoria. In this research study, we recruited practitioners and conducted semi-structured interviews to gain knowledge on their decision making process for security matters. As a member of this team, I contributed to designing

[2]https://github.com/

the study, conducting the interviews, and coding the transcripts of participants. We extracted capabilities, opportunities, and motivations of the practitioners, in the context of software security, using a behavioral model. Through this study, we aimed to create a framework for organizations so they can refine their strategies to increase developers' security adoption.

1.1 Research Questions

My thesis consists of two studies that address two main goals: a systematic mapping study to understand the papers that aim to improve the security of projects hosted on GitHub and a cross-sectional interview study to investigate the behavioral aspects of adopting security in software development.

1.1.1 Systematic Mapping Study

For the mapping study, I constructed the following research questions to identify studies focusing on security issues in projects hosted on GitHub:

RQ1: What topics are discussed in the literature regarding security in GitHub?

First, we aimed to extract all the topics discussed in the literature regarding security that used GitHub's data. Using a card sorting method, we identified nine topics such as, *Managing Vulnerabilities in Third-Party Libraries* and *Developing and Enhancing Static Analysis Tools*. For each topic, we presented a summary and highlighted important takeaways from the papers. We hope describing the topics can help future researchers better understand the literature.

RQ2: Inspired by the Who-What-How framework:

- Who is the main claimed beneficiary of the research study?
- What is the main type of research contribution?
- What is the main research approach (descriptive or solution)?

We analyzed the papers to identify the audience of the research study, research strategies used, and whether the study described a security issue or provided a solution. By answering these research questions, we wanted to observe how social aspects are studied in the literature and what gaps may need to be addressed in future research.

RQ3: What recommendations are proposed in the literature to enhance security in open source projects hosted on GitHub?

Based on the suggestions mentioned in the papers, we provided insights that could help future researchers have an overview of the issues and proposed solutions. For instance, from the papers that address *Managing Vulnerabilities in Third-Party Libraries* topic, we observed that security is a secondary concern for many developers. This is a helpful insight for researchers to investigate why security is not a priority.

Answering these research questions helped inform us of the main research on this topic, bring awareness of what solutions might be helpful to practitioners, and pave a path for future research to enhance security in an open source platform such as GitHub. Additionally, we discovered one of the gaps in the literature is investigating security from the developers' perspective. This is why we decided to conduct a qualitative study to observe the developer behaviors towards security that we discuss in the next section.

1.1.2 Cross-Sectional Interview Study

To understand developer behaviors, I decided to conduct a qualitative study collaborating with a team of researchers from the University of Victoria—In Section 5.2.4 I describe the team. . We used a behavioral model to design our interviews that aimed to answer the following question:

RQ: What factors influence developers' security adoption?

With this study, we aimed to help organizations understand the shortcomings in their security adoption strategies and how they can improve them. We looked at areas that can be developed, discovered potential abilities to improve security adoption, and understood developers motives towards adopting security.

1.2 Thesis Contributions

This study makes four major contributions that can help researchers and practitioners understand the current state of software security issues and how they can be addressed. This study depicts a clear picture of the literature and the proposed solutions. Additionally, we analyzed the literature that led to identifying the gaps that

need more attention in future research. After we noticed social aspects of software security need more attention, we conducted a cross-sectional interview study. The results from the interviews will potentially help organizations better implement security in their software development life cycle. The main contributions of this thesis towards understanding and improving software security are as follows:

1. A systematic mapping study that results in identifying 36 primary studies that describe the papers that investigated security issues using projects hosted on GitHub.

2. Through my study, I categorize the literature into nine topics, using a card sorting method, and demonstrate the trends existing in the literature. For each paper, I presented key takeaways and a summary that can potentially be useful for future researchers to get an overview of the previous studies. Additionally, for every topic I provide insights that demonstrate the gaps in the literature and highlight the important suggestions mentioned by the papers.

3. To gain deeper knowledge on the literature, I used a maturity model and a socio-technical framework to analyze the papers. The maturity model helped me analyze how the topics are distributed across security practices in different stages of software development. I used a socio-technical framework to investigate how social aspects of software security have been addressed in the literature. With these two analyses, I found that not many studies focus on security from the developer's perspective, and mostly the research is mostly focused on technical issues.

4. I conducted a cross-sectional interview study that investigates the factors affecting developers' security adoption. I provide an overview of the challenges faced by developers when they deal with security issues and how these challenges could be mitigated. Furthermore, I provide suggestions for organizations to improve their current state of security adoption. For instance, I describe how an organization's culture can have a direct influence on developers' security adoption and how the culture can be improved to address this shortcoming.

1.3 Thesis Overview

This thesis is structured as follows.

In Chapter 2 Background and Related Work I review the background and related work of my thesis. This chapter includes works related to the mapping study presented in this thesis. I mention previous systematic literature reviews similar to my mapping study that investigated security matters. I also describe previous work that studied social aspects of software security.

In Chapter 3 Research Method I describe my research methods for the systematic mapping study. I elaborate the details of how I collected and analyzed data.

In Chapter 4 A Systematic Mapping Study to Understand and Improve the Security of Projects Hosted on GitHub I present the mapping study I conducted. In this chapter, I provide the insights and summaries of the primary studies. I present the results of the analysis using the maturity model and the socio-technical framework. I provide suggestions on how to improve security and summarize trends identified in the literature.

In Chapter 5 Investigating the Behavioral Aspects of Adopting Security in Software Development I describe the cross-sectional interview study conducted with software practitioners. I collaborated with a team of researchers from the University of Victoria to gain insights on the factors affecting developers' security adoption. I describe how organizations can benefit from the suggestions provided.

In Chapter 6 Conclusions I conclude my thesis, synthesize findings from the two studies I conducted and discuss future work.

Chapter 2

Background

There have been relatively few systematic mapping studies regarding software security. I categorized the studies I found into three topics: *Security in Agile Methodologies*, *Security from the End Users' Standpoint*, and *Social Aspects of Developer-Centric Software Security*. First, I describe the studies that performed a literature review on papers investigating security in agile practices and workflows. The papers in this category, discuss the challenges, opportunities, and possibilities in agile software development that impact software security.

2.1 Security in Agile Methodologies

In 2016, Khaim et al. [32] performed an extensive literature review to summarize security issues that have emerged in agile methodologies. They investigate the types of approaches suggested in literature for the purpose of incorporating security in agile practice. They found that the majority of the challenges reported in the literature, typically occur due to the absence of a security expert.

In 2018, Riisom et al. [57] conducted a literature review on the adoption of security in agile software development. They identified the challenges mentioned in the literature and what solutions have been proposed. For instance, the most common challenge faced is "ensuring agile conformity of activities". This challenge shows that sometimes the integration of security is sometimes in conflict with the agile methodology and this can be difficult for the developers to overcome without abandoning agile principles. To overcome this challenge, a proposed solution is to assign a weight to the security activity and only apply the ones that have a certain

amount of weight to limit the influence of security. This way, it is easier to detect the security activities that have high impact on agile practices.

In 2018, Villamizar et al. [65] performed a systematic mapping study on security in agile requirements engineering and identified 21 approaches for dealing with security challenges in projects that use agile methodology. For instance, one of the identified approaches was "Proposes a conceptual agile security framework using a hybrid technique for requirements elicitation".

These literature reviews provide us with insightful information about how we can incorporate security into agile practices. They say how some of the current approaches in agile practices might be in contrast with the adoption of security or incorporating a security role in agile methodologies is essential. However, they do not uncover many aspects that affect software security, such as social aspects. For instance, an important social aspect of security is usability, which I will talk about more by describing literature reviews that investigated this aspect in the next section.

2.2 Security from the End Users' Standpoint

A recent systematic literature review performed by Lennartsson et al. in 2021 [39], described applicable factors that influence the perceived usability of security mechanisms. They identified 14 themes and 30 associated sub-themes. A few of the most significant themes they found include: simplicity, information and support, and task completion time.

In 2018, Iacono et al. [30] presented a literature review of usable security principles and patterns. They extracted 47 usable security patterns and 23 usable security principles, and developed an interactive tool to enable users to search and explore these principles and patterns. They mention the literature has focused only on a subset of important principles. For instance, one presented principle was the importance of consistency of security-related controls in of graphical user interfaces.

Nwokedi et al. [46] conducted a systematic literature review in 2016 to close the gap between usability and security in the software development process. They aimed to address the importance of balancing between usability and the security of the system through a review of existing literature. They argue that a common misconception is that security and usability are incompatible. They say in many cases a simple interface is considered more secure because it does not require complicated authentication mechanism. They note that to find usability issues regarding security,

more research can be done with real-life use cases using other research methods such as interviews or questionnaires.

The main actors in developing secure and usable products are developers. Until now, we read about literature reviews that considered usability of security from end users' perspective and security in agile methodologies. These literature reviews do not explicitly mention how developers should adopt security. In the next section, I describe literature that investigated other social aspects of security, from the developers' perspective.

2.3 Social Aspects of Developer-Centric Software Security

Tahaei and Vaniea's [61] 2019 study provided an overview of developer-centered security studies. They selected 49 studies with software developer participants and derived eight themes: organizations and context, structuring software development, privacy and data, third-party updates, security tool adoption, application programming interfaces, programming languages, and testing assumptions. They extracted the research strategies used in the papers and presented information about the participants, such as whether they were developers, software professionals, or students.

Wen's [69] 2017 Systematic Literature Review (SLR) on security studies within the context of open source development categorized papers according to the Open Web Application Security Project (OWASP) software assurance maturity model (SAMM) [12]. They also analyzed their final set of primary studies with a socio-technical method presented by Kowalski [35]. One of their findings was that socio-technical aspects need more attention from researchers. Inspired by this work, I also adopted SAMM to categorize the topics I found in the literature.

Until now, we read about literature reviews reporting on studies focusing on security in agile methodologies, security from the end users' standpoint, and social aspects of developers' security adoption. We learned many factors affect security, such as having a security role in the development workflow. However, we did not find literature reviews investigating other important social factors of security. For instance, how new security tools help developers to adopt more secure practices.

Since I did not find many literature reviews on the social aspects of security I decided to focus on investigating these aspects in the literature. I believe one way to

investigate the social aspects is to critique the literature on their research methods and objectives to observe how they address the developers' security adoption. This idea led me to find another literature review. In 2016, Carver et al. [11] critique the literature on software security. They aimed to ascertain if security research is conducted in a way that facilitates replication, theory building, and meta-analysis. Their results show studies often lack fundamental information such as threats to validity, the process of choosing the cases in the studies, and lack clear research objectives.

After reading all the literature reviews, I decided to look at my final set of primary studies with a critical lens using a socio-technical framework [60]. In Chapter 4, I elaborate on the results of my mapping study and describe my analysis of the final set of primary studies. I discovered most of the studies focused on technological advancements and did not look at whether developers adopted the technologies.

In the next Chapter, I explain the methodology I followed to conduct my mapping study which includes the data gathering strategies and data analysis methods.

Chapter 3

Systematic Mapping Study Research Method

As part of my thesis research, I conducted a mapping study by performing a systematic literature review of papers that investigate security issues with projects hosted on GitHub. The goal of this mapping study was to analyze existing research to understand the security issues, the proposed solutions, and see what needs more attention in future research. I performed the mapping study according to the guidelines presented by Kitchenham and Charters [34] and Petersen [50].

I define software security to be clear about my intentions. By software security, I mean the necessary instruments and best practices used to implement software, from the design phase to the deployment phase, to keep it safe against hackers and malicious attacks.

3.1 Search Protocol

To develop my search string, I followed PICO (Population, Intervention, Comparison, and Outcomes) presented by Kitchenham and Charters [34].

Population: A population is any type of software engineering role, an application area, a category of software engineering, or an industry group. In my research, the projects hosted on GitHub are the population.

Intervention: The intervention is the software tool or technology that focuses on a specific issue. I selected GitHub, the most popular technology in the open source community according to its number of projects and contributors.

Comparison: This is the software engineering tool or technology with which the intervention is being compared. In my mapping study, I investigated and compared different approaches that aimed to mitigate security threats in GitHub.

Outcomes: Outcomes relate to aspects of importance to practitioners. By providing an overview of the security risk mitigation solutions using GitHub projects, I may improve security in projects hosted on this platform and depict a clearer picture for future researchers in this field. Our outcomes will hopefully help researchers address any gaps identified in the literature.

First, I performed an exploratory search on Google Scholar to have a better idea about what keywords and search strings would fit my research goals. Using The PICO criteria, my research questions, and the initial searches I performed, I chose the following keywords: GitHub Security, GitHub Vulnerabilities, GitHub Developers, and Secure Dependencies. I then generated search strings for four databases specified by Kitchenham and Charters [34]: IEEE, ACM, Science Direct, and Engineering Village.

3.2 Study Selection

From the exploratory search in Google Scholar, I selected 10 papers based on the title, abstract, and conclusion, that satisfied my initial inclusion and exclusion criteria. Subsequently, I decided to investigate other databases to see if there are more studies that fit my criteria. I performed trial searches using various combinations of the mentioned keywords for each database. After assessing the results of the trial searches, I created my search strings. Table 3.1 shows the search strings I created for the databases used in my study. To perform an exhaustive search, Kitchenham and Charters [34] note electronic sources of relevance to computer science community. From these databases, I selected four of them based on popularity in computer science research domain.

I used the following inclusion and exclusion criteria:

Inclusion: To include the paper in my study, I made sure the research used either GitHub data, GitHub repositories, or conducted studies on GitHub users to mitigate open source software security risks in GitHub.

Database	**Search String**	**# Search Results**	**# After Reading the Title and Abstract**	**# After Reading the Full Text**
Google Scholar	Initial exploratory search		10	7
ACM	(("GitHub developer" OR "GitHub user" OR "GitHub Platform" OR "GitHub website") AND security) OR ("GitHub Security" OR "GitHub Vulnerabilities" OR "GitHub Vulnerability")	33	6	3
IEEE	(("GitHub developer" OR "GitHub user") AND security) AND ("GitHub Security" OR "GitHub Vulnerabilities" OR "GitHub Vulnerability")	55	17	13
Science Direct	(("GitHub developer" OR "GitHub user" OR "GitHub Platform" OR "GitHub website") AND security) OR ("GitHub Security" OR "GitHub Vulnerabilities" OR "GitHub Vulnerability")	46	4	1
Engineering Village	((package OR packages OR dependency OR dependencies) AND security AND GitHub) OR (("GitHub developer" OR "GitHub user" OR "GitHub Platform" OR "GitHub website") AND security) OR ("GitHub Security" OR "GitHub Vulnerabilities" OR "GitHub Vulnerability")	57	6	2
Snowballing First Round				7
Snowballing Second Round				3
Total				36

Table 3.1: Search strings in four selected databases.

Inclusion: The papers had to be in the field of computer science in order to be included in my mapping study.

Exclusion: I excluded non-English studies.

Exclusion: I did not include duplicate studies published in different venues.

Exclusion: If the full text of the study was not accessible (via University of Victoria library), I did not consider it for my mapping study.

Once my queries for each database returned results, I checked the title, abstract, and conclusion of each paper to see if it fit my inclusion and exclusion criteria. In this phase, I accepted 43 papers out of 191 papers. Then, I applied inclusion and exclusion criteria taking into account the full content of 43 papers. I accepted 26 papers out

of 43 after reading the full text. On the 26 papers selected from the databases I conducted backward[1] and forward[2] snowballing [72] to make sure I did not miss any papers (either they were not in my chosen databases or were not included in the search results). After three rounds of snowball sampling, I added another 10 studies to the list of primary studies, bringing my total to 36 studies. Figure 3.1 shows the number of papers in every step and Table A.1 (Appendix A) shows the list of selected studies.

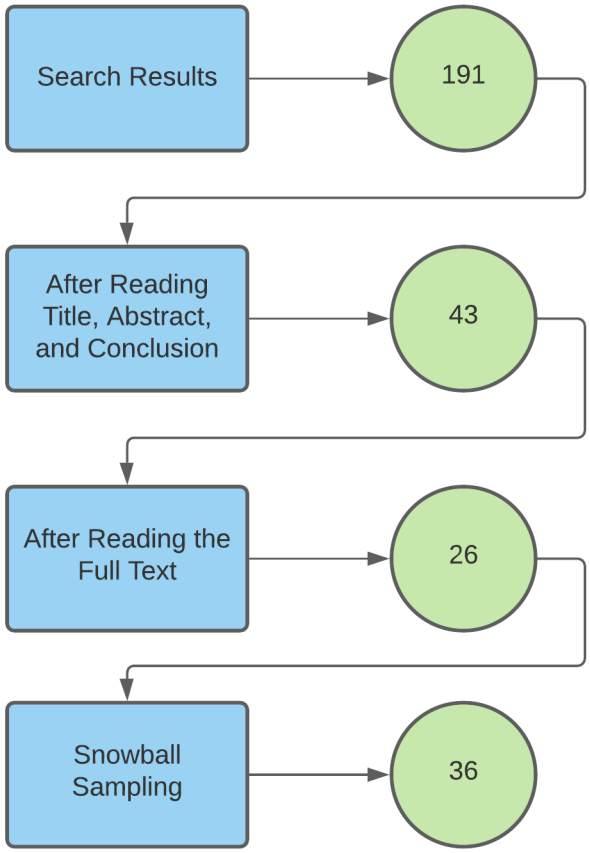

Figure 3.1: Number of papers selected in every step of the mapping study.

[1]Backward snowballing refers to identifying new papers based on the references of a paper.
[2]Forward snowballing refers to identifying new papers, citing the paper being examined.

3.3 Data Collection

To extract insights from the primary studies, I used two approaches: (a) I used a card sorting method to identify topics in the literature and (b) I used the Who-What-How framework [60] to investigate who are the beneficiaries of the research (Who), which methods are used and whether those methods consider human and social aspects (How), and what is the the nature of the research contribution from the papers (What).

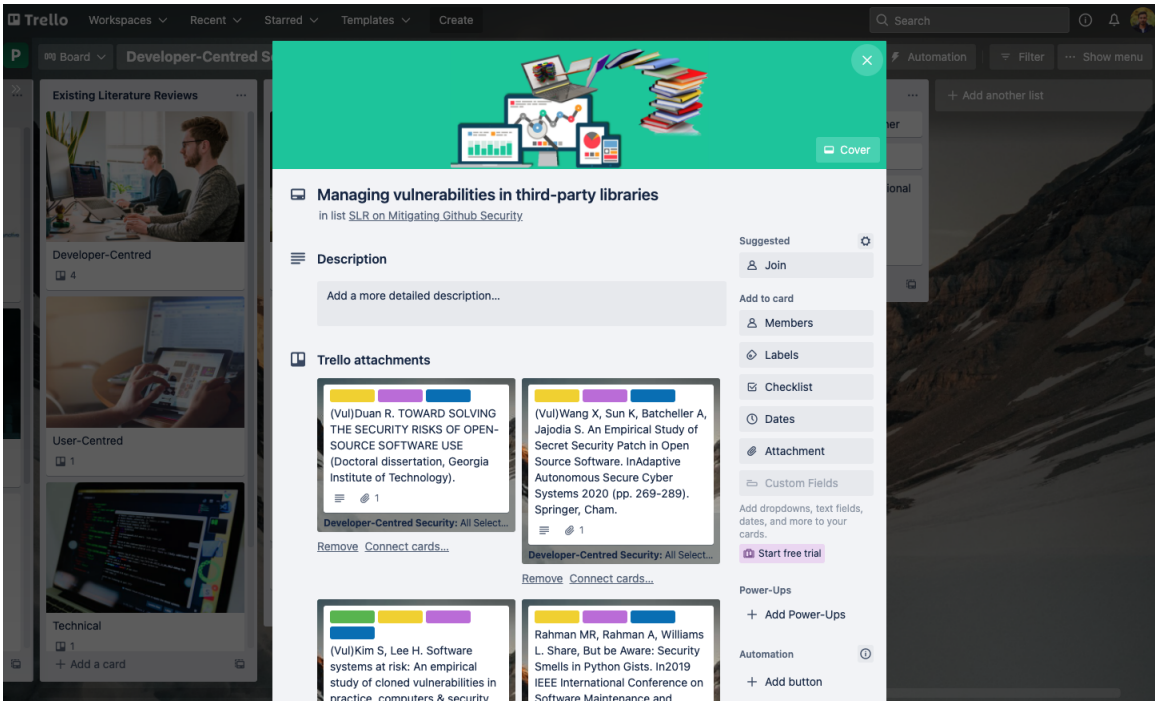

Figure 3.2: Card sorting method performed with the Trello web application.

To perform the card sorting method, I used the Trello[3] web application. I assigned each paper to a card based on the problem they described or addressed. If a paper described issues with vulnerabilities in third-party libraries, I assigned it to a different card than a paper that presented a new Static Analysis Tool (SAT). After assigning the papers to the cards for the first time, I gave each card a label. For instance, I gave the *Managing vulnerabilities in third-party libraries* label to the papers that investigated issues or presented solutions to mitigate vulnerabilities in third-party libraries. I performed multiple iterations to make sure all the papers were assigned to an appropriate card. At the end of the iterations, I sought feedback from another

[3]https://trello.com/

researcher who was a member of the CHISEL research group[4]. The researcher examined the papers associated with each card. We discussed the categorization and we were able to reach an agreement on the results of my card sorting method. Figure 3.2 shows a screenshot of the papers categorized as *Managing vulnerabilities in third-party libraries*.

Dimension	**Value**
Who	Human (developers and practitioners), System (tools and applications), and Researcher
What	Descriptive (describes a problem) and Solution (provides a solution to a specific problem)
How	Sample Survey, Data Study, Formal Theory, Lab Experiment, Field Study, Field Experiment, Judgment Study, Experimental Simulation, and Meta (e.g., systematic literature review)

Table 3.2: Description for the three main components of the Who-What-How framework [60].

I used the information described in Table 3.2 to analyze the primary studies. This information was extracted from the Who-What-How paper [60]. For all the primary studies, I answered all the questions about Who, What, and How by reading the papers and documented the findings in a google spreadsheet[5]. After I performed the analysis, another researcher performed the same steps on the primary studies. We held an agreement session to discuss our analysis. Notably, we saw only a few conflicts and they were resolved in the agreement session.

In the next Chapter, I discuss the details of the mapping study. I provide a summary of the literature, draw insights of important takeaways extracted from the primary studies, and present my findings from analyzing the papers with a socio-technical framework.

[4]https://thechiselgroup.org/team/
[5]https://www.google.ca/sheets/about/

Chapter 4

A Systematic Mapping Study to Understand and Improve the Security of Projects Hosted on GitHub

The main focus of this study is to describe the nature of previous research that has looked at the security of open source projects hosted on GitHub. From this systematic review, I identified 36 papers that focus on the topic of security issues. I categorized all the papers into nine topics, which I sorted by similarity. In the following sections, I synthesize suggestions on how to mitigate security threats in projects hosted on GitHub, I use a maturity model to extract trends from the literature, and analyze the final set of primary studies with a socio-technical framework.

4.1 Categorizing the Primary Studies

In this section, I answer my first research question and provide a summary of the primary studies that either used GitHub data, studied GitHub repositories to describe different aspects of package vulnerability issues, or conducted user studies with GitHub users to understand how they adopt security.

As mentioned in Chapter 3 (research method), I used a card sorting method to identify themes in the literature. I was able to classify the studies into nine different topics: *Managing vulnerabilities in third-party libraries*, *Developing and enhanc-*

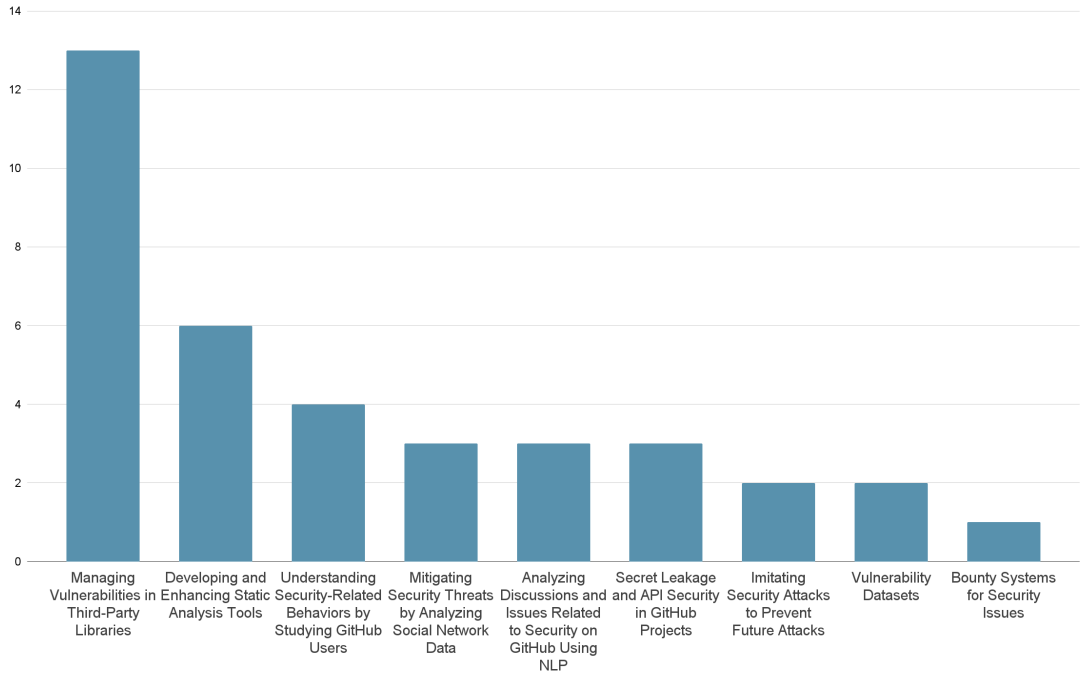

Figure 4.1: Number of papers in each topic.

ing static analysis tools, *Understanding security-related behaviors by studying GitHub users*, *Mitigating security threats by analyzing social network data*, *Analyzing discussions and issues related to security on GitHub using NLP*, *Secret leakage and API security in GitHub projects*, *Imitating security attacks to prevent future attacks*, *Vulnerability datasets*, and *Bounty systems for security issues*. These topics were based on common themes I found in my sample set. For instance, the topic "Analyzing discussions and issues related to security on GitHub using NLP" emerged from the papers that mentioned using NLP in their methodology to analyze discussions, comments, and issues in GitHub. While the theme "Understanding Security-Related Behaviors by Studying GitHub Users" emerged from papers that either conducted studies with GitHub users or analyzed trace data describing user behaviours.

The distribution of the papers according to each topic is presented in Figure 4.1. Most of the papers were about the topics of "Managing vulnerabilities in third-party libraries" or "Developing and enhancing static analysis tools". I describe all the identified topics in the following section.

4.1.1 Managing Vulnerabilities in Third-Party Libraries

I identified 13 papers in the literature that address this theme. The papers either analyze the data regarding the vulnerabilities or propose a solution to mitigate vulnerabilities.

Citation: Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In 2017 87 32nd IEEE/ACM International Conference on Automated Software Engineer- ing (ASE), pages 84–94. IEEE, 2017.

Mirhosseini and Parnin [43] investigated why developers neglect software dependency updates. They analyzed 7,470 GitHub projects that contained a notification mechanism, either an automated pull request system such as Greenkeeper, a project badge[1] similar to David-DM, or a baseline that used neither, to alert the repository owners about a vulnerability issue in their project. They also surveyed 62 developers to see the effect of these tools. Their study shows that automated pull requests led to a higher number of updates compared to badges, and at the same time pull requests had significantly more downgrades to their previous versions compared to badges and the baseline group that did not use either of the notification systems. 54% of those surveyed preferred passive notifications such as badges, while the others preferred automated pull requests.

Paper Takeaway [43]: ***Active notification mechanisms to inform project owners about a vulnerability, such as pull requests, have higher adoption compared to passive styles, such as notification badges. Changes made because of active notifications are downgraded more often compared to changes made in passive style notifications.***

Citation: Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. Do developers update their library dependencies? Empirical Software Engineering, 23(1):384–417, 2018.

Kula et al. [36] investigated to what extent developers update their dependencies when notified of a vulnerability. They conducted an empirical study on library migrations over 4,600 GitHub projects and 2,700 libraries. They mined and reconstructed historic library migrations for these projects. The results show that 81.5% of

[1] GitHub Badges are a way to boost the readability of the GitHub repositories by providing users with a fast way to capture repository metrics.

the projects remained with the more popular older version, which had vulnerabilities. They observed developers would not typically update a library if it involved migration efforts, and 69% of the developers were unaware of their vulnerable dependencies.

Paper Takeaway [36]: ***More than 80% of the studied projects remain with the older version of their dependencies and 69% of the developers are not aware of their vulnerable dependencies.***

Citation: Brandon Carlson, Kevin Leach, Darko Marinov, Meiyappan Nagappan, and Atul Prakash. Open source vulnerability notification. In Francis Bordeleau, Alberto Sillitti, Paulo Meirelles, and Valentina Lenarduzzi, editors, Open Source Systems, pages 12–23, Cham, 2019. Springer International Publishing.

Carlson et al. [10] analyzed 600 projects to investigate how many projects hosted on GitHub contained a vulnerable dependency. They also investigated whether the projects used a communication medium that could enable users of the library to contact the project owner regarding security issues. They found that 64.2% (385 out of 600) of the Java projects contained at least one vulnerable dependency and only 13 of these 385 had a reporting process for security issues; 19.8% had no public contact information. In the 600 projects they investigated, they found an average of 7.8 direct or transitive vulnerabilities in every project and only 3.2% (19) of the projects had a process for reporting vulnerabilities. Finally, they suggested adding a standardized security.md file to the projects to specify a disclosure process and contact information for vulnerabilities.

Paper Takeaway [10]: ***Most studied projects do not have a communication medium for vulnerability reporting. Adding a standardized security.md file containing a disclosure process is suggested.***

Citation: Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. In 24th Annual Network and Distributed System Security Symposium, NDSS '17', February 2017.

To see whether websites keep their dependencies updated with the latest publicly disclosed vulnerabilities, Lauinger et al. [37] analyzed 133,000 websites. As there was no vulnerability database or security mailing list, they had to manually select

72 libraries and extract versioning information from GitHub to conduct their study. They found that 37.8% of the websites contained at least one vulnerable library and 9.7% contained two or more. Eminent websites were less likely to contain vulnerabilities but they also did not include any of the 72 selected libraries for this study on their websites and only 21% of the top 100 websites had a known vulnerable library. Interestingly, financial and government websites ranked last among other categories (malicious, parked, and adult websites) with over 50% vulnerable websites. Another one of their findings was that a very small portion (2.8% in ALEXA) of the websites could remove the vulnerabilities with patch updates.

Paper Takeaway [37]: ***About 38% out of the 133,000 analyzed websites have at least one vulnerability and financial and government websites are the most vulnerable websites when grouping by Alexa ranking according to the McAfee Smart-Filter web categorization.***

Citation: Alex Tompkins, Josh Bernasconi, Andrew Davidson, and James Toohey. On security vulnerabilities stemming from the usage of open-source dependencies. SENG401 Conference, ACM ISBN, 2019.

To study security issues related to package managers, Tompkins et al. [62] investigated whether the current state of package managers such as NPM, Python's PIP, and Java's Maven add any specific security risk to a project. They analyzed 500 of the most popular packages on the above mentioned package managers, most of which were hosted on GitHub. They also surveyed 21 4th-year engineering students in New Zealand. The survey aimed to find developers' attitudes towards using open source dependencies, how often they use them, and how they perceive the importance of security vulnerabilities from these dependencies.

Tompkins et al. also found that 8.34% of all packages had at least one vulnerability, PyPI packages had a higher rate of vulnerabilities, and both Maven and PyPI repositories had a high number of severe vulnerabilities. In their survey 47.7% of the participants thought the security of open source packages was their responsibility, but many of them did not consider security when adding packages. 33% of the participants in the survey study mentioned their projects had at least 51 dependencies and 75% of them said they use more than 10. They suggested that cryptography libraries should not only provide a convenient interface but also ensure support for a wide range of cryptography tasks and provide accessible documentation about security.

Paper Takeaway [62]: ***PyPI packages, compared to NPM and Maven, have a higher rate of vulnerabilities and among software developers many of them do not consider security when adding packages.***

Citation: Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In Proceedings of the 15th International Conference on Mining Software Repositories, pages 181–191, 2018.

Decan et al. [14] investigated how vulnerabilities affect other packages in Node Package Manager (NPM) ecosystem and analyzed how and when these vulnerabilities are discovered and fixed. They present an empirical study of nearly 400 security reports on over 610k JavaScript packages that were distributed over six years in the NPM dependency tree. Their findings show it takes a long time to discover vulnerabilities with low severity. Many of the vulnerabilities found are older than 28 months and most of them are fixed before they become public, but there is a non-trivial number of vulnerabilities that take time to resolve. Another finding was that even if the upstream fix for a package is available, a considerable portion of dependant packages are not updated. They also mentioned that the main reasons for vulnerabilities in dependant packages include dependency constraints that are too restrictive and unmaintained packages.

Decan et al. also provided some suggestions for package maintainers and developers to keep their packages updated and fix vulnerabilities before they are publicly disclosed. They also suggest there is a need for a single common vulnerability database that the community can contribute to. For package users, they recommended watching for packages with vulnerabilities and they suggested the use of monitoring and security report systems for these packages.

Paper Takeaway [14]: ***In NPM ecosystem, many vulnerabilities are older than 28 months; there is a need for a common vulnerability database and the main causes of vulnerabilities are restrictive dependency constraints and unmaintained packages.***

Citation: Bingchang Liu, Guozhu Meng, Wei Zou, Qi Gong, Feng Li, Min Lin, Dandan Sun, Wei Huo, and Chao Zhang. A large-scale empirical study on vulnerabil- ity distribution within projects and the lessons learned. In 2020 IEEE/ACM 42nd International Conference on Software

Engineering (ICSE), pages 1547– 1559, 2020.

Liu et al. [40] analyzed the vulnerability distributions associated with five open source projects across four dimensions: files, functions, vulnerability types, and responsible developers. By using crawlers and manual analysis, they collected a large vulnerability dataset. Using the data, they present twelve practical insights that helped them successfully find ten zero-day vulnerabilities[2]. They found only 30% of the files accounted for 66% of the vulnerabilities and that higher code complexity does not necessarily mean more vulnerabilities. For vulnerable functions, 30% of the functions accounted for 50% of the vulnerabilities. More than 60% of vulnerable functions were related to another vulnerable function with different vulnerabilities. For one of the investigated projects, 70% of the vulnerabilities were categorized by 20% of the vulnerability types[3]. Popular vulnerability types in a project will not be extremely affected by time, so security analysts can focus more on these types compared to the other ones. They noted that 10% of the developers of a project that contribute from 60% to 100% of the code are responsible for almost all of the vulnerabilities. They point out that incomplete fixes of the vulnerabilities are the main cause for security breaches.

Paper Takeaway [40]: ***The 10% of developers that contribute most of the code in the projects studied were responsible for contributing almost all the vulnerabilities. Incomplete fixes of vulnerabilities cause them to occur repeatedly. Finally, in the projects studied, higher code complexity does not necessarily mean more vulnerabilities.***

Citation: Antonios Gkortzis, Daniel Feitosa, and Diomidis Spinellis. Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities. In Journal of Systems and Software, page 110653. Elsevier, 2020.

To investigate the difference between vulnerability distribution in the code created by a development team (native) and reused code from a third-party library, Gkortzis et al. [21] investigated 1,244 popular open source Java projects hosted on GitHub. They extracted vulnerability data using using static analyzer SpotBugs and the OWASP dependency checking tool. In both native and reused code, the bigger

[2]A zero-day vulnerability is a vulnerability which is unknown or not addressed [71].

[3]Types of vulnerabilities are labeled by the National Vulnerability Database (NVD), following common weakness enumerations (CWE) categorization [40].

the project, the more vulnerabilities they found. Also, the number of dependencies had a positive correlation with the number of vulnerabilities. In their results, there found no proof that reuse code through dependencies affected the number of vulnerabilities in a project and they statistically showed the median vulnerability density is higher in native code compared to reused code.

Paper Takeaway [21]: ***In the 1,244 projects studied, the number of dependencies is correlated with the number of vulnerabilities, and the median vulnerability density is higher in native code compared to reused code in dependencies.***

Citation: Pei Xia, Makoto Matsushita, Norihiro Yoshida, and Katsuro Inoue. Studying reuse of out-dated third-party code in open source projects. In Information and Media Technologies, volume 9, pages 155–161. Information and Media Technolo- gies Editorial Board, 2014.

Xia et al. [73] studied the use of several popular open source libraries that had become outdated (zlib, libcurl, libpng) to observe if vulnerabilities were caused by the reuse of old code. They used mining techniques and clone detection to identify different versions of the code reused in the projects and spot vulnerabilities from outdated libraries. Their results showed that for projects using zlib, 31.1% of them incorporated an outdated version of the library with possible vulnerabilities. For libcurl projects, 85.7% of them used an outdated version, with only 1 out of 28 projects using the most recent version. For libpng, 92% of the projects used an outdated version, with only 2 of 50 projects using the most recent library. Some of the potential vulnerabilities when reusing outdated versions of these libraries include denial of service and execution of arbitrator code, but further details can be found in NVD [45].

To investigate the developers' perspective, Xia et al. also contacted developers of teams that used outdated versions. After receiving a response from developers on the teams that used outdated code, only one immediately updated the code, but other developers avoided the update. Some of the reasons mentioned by the developers to avoid the update were: updating the outdated versions might introduce new bugs and due to the compatibility of multiple dependencies updates can not be done.

Paper Takeaway [73]: ***Participants of this study avoid updating third-party libraries until absolutely needed because doing so might introduce bugs.***

Citation: Md Rayhanur Rahman, Akond Rahman, and Laurie Williams. Share, but be aware: Security smells in python gists. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 536–540. IEEE, 2019.

For developers sharing code snippets, GitHub provides a service called GitHub Gist. While this may help developers support each other by providing faster solutions, it can introduce security flaws that could create vulnerabilities. Rahman et al. [55] called these flaws security smells and conducted an empirical study on 5,822 public Python Gists. They categorized the security smells into 13 types (e.g., Command injection and Use of HTTP without TLS) and found 31% of them have at least one security smell. They mentioned they did not find any connection between the reputation of the Gist author and the existence of a security smell in the Gist.

Paper Takeaway [55]: ***There are 13 possible different categories of security smells identified in Python GitHub Gists, and 31% of the Gists studied have at least one security smell.***

Citation: Ruian Duan. TOWARD SOLVING THE SECURITY RISKS OF OPEN-SOURCE SOFTWARE USE. PhD thesis, Georgia Institute of Technology, 2019.

Duan [15], as part of their Ph.D. thesis, studied vulnerabilities and licensing issues in open source software. The thesis is composed of three main sections. They introduced a tool called OSSPolice that detects licensing misuse and vulnerabilities in open source software. To mitigate vulnerabilities, Duan proposed OSSPatcher to automatically create new patches to fix vulnerabilities and they conducted a study on supply chain attacks against an open source ecosystem where many breaches occurred in package managers.

OSSPolice analyzes open source software with a hierarchical indexing scheme that can efficiently and accurately compare app binaries with billions of lines of code. Out of 1.6 million Google Play Store apps analyzed, 40k potentially violated licensing terms. To test OSSPatcher, Duan used 1,000 vulnerable apps and 39 open source software projects. Duan found that 675 patches were suitable for the project and the fixes used a trivial amount of memory and performance overhead.

Finally, Duan presented the MALOSS framework to study supply chain attacks in a package manager ecosystem. They demonstrated the causes of the attacks and

gave a rundown on malicious behaviors. By using MALOSS, Duan identified seven malwares in PyPI, 41 in Npm, and 291 in RubyGems, and reported them to package manager maintainers. Package manager maintainers successfully removed 278 of them while reporting three of them to the CVE database.

Paper Takeaway [15]: ***These tools mitigate vulnerabilities and licensing issues in open source software: OSSPolice, a tool developed for detecting licensing misuses; OSSPatcher, a tool developed to automatically create patches to fix vulnerabilities; and MALOSS, a framework to study supply chain attacks in a package manager ecosystem.***

Citation: Seulbae Kim and Heejo Lee. Software systems at risk: An empirical study of cloned vulnerabilities in practice. In Computers & Security, volume 77, pages 720–736. Elsevier, 2018.

Kim and Lee [33] proposed a new scalable approach called VUDDY to detect code clones in large software projects with high accuracy and efficiency. They showed VUDDY can process a billion lines of code in 14 hours and 17 minutes. Using an abstraction scheme optimized for identifying unknown vulnerabilities, VUDDY detected up to 24% more vulnerable code clones, which are a variation of existing vulnerabilities. Kim and Lee analyzed cloned vulnerabilities in popular software projects hosted on GitHub and SourceForge, and illustrated there is a dangerous feedback loop of vulnerable code clones propagating in the projects. An extensive number of vulnerabilities in the projects that contained code clones remained unfixed for 0.5-1.5 years, with some up to 3 years.

Paper Takeaway [33]: ***Vuddy is a scalable approach for detecting code clones and it can detect up to 24% more vulnerable code clones than other approaches.***

Citation: Xinda Wang, Kun Sun, Archer Batcheller, and Sushil Jajodia. An empirical study of secret security patch in open source software. In Adaptive Autonomous Secure Cyber Systems, pages 269–289. Springer, 2020.

Some software developers may secretly patch vulnerabilities in their projects but do not disclose the fixes to the public [68]. They may think that too many vulnerability fixes may weaken the software's reputation, and so until it is safe to do so, they block the publication of the related CVE. However, some attackers might find

out about the security issue fixed in the patch and exploit it in the used projects. To overcome this issue, developers need an approach to track security patches, not only in the projects they are directly involved in, but also in other similar projects so that they also update their code in a timely manner.

To fix this issue, Wang et al. [68] developed a machine learning-based tool to automatically identify security patches. They also performed a case study on three famous open source SSL libraries in where they found 12 secret security patches. They claimed they generated the first large-scale open source security patch dataset by querying CVE entries from 1999 to 2018. They mentioned some tips to differentiate security patches from non-security patches, including: security patches will more likely alter less code where non-security patches bring new functionalities and features; lines before and after changes in security patches are very much similar since the modification is small; security patches mostly have the same modification multiple times; and security patches always have changes in conditional statements.
Paper Takeaway [68]: ***To identify security patches in an open source environment, a machine learning tool was presented that could automatically identify security patches. This research also presented the first large-scale open source security patch dataset.***

In summary, some common points made in these papers include: developers have a tendency to postpone vulnerability fixes; security is mostly a secondary concern; and there is no common vulnerability database that everyone can refer to. To address this issues, many powerful tools are being developed, but they are developed without studying if developers need new and more accurate tools. Creating new tools might not mitigate vulnerabilities when developers still have a tendency to think of security as a secondary concern.

4.1.2 Developing and Enhancing Static Analysis Tools

Static analysis tools (SAT) are popular interventions to mitigate security threats in software applications. In this section, I review six papers proposing solutions that either present new SATs or enhance existing tools. Some of the papers performed evaluations using existing data from projects hosted on GitHub, and some conducted user studies to observe how developers would benefit from using static analysis tools.

Citation: Akond Rahman, Chris Parnin, and Laurie Williams. The seven sins: security smells

in infrastructure as code scripts. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pages 164–175. IEEE, 2019

Developers use infrastructure as code (IaC) scripts in their projects to benefit from different development environments. However, these IaC scripts may contain security smells that could increase the possibility of a breach. To help practitioners develop safer IaC scripts and adopt more security practices, Rahman et al. [54] conducted an empirical study on security smells in IaC scripts, and implemented a static analysis tool called Security Linter for Infrastructure as Code scripts (SLIC) to study the existence of these security smells. They identified seven security smells by performing a qualitative analysis of 1,726 IaC scripts: 29.3% of the scripts extracted from GitHub contained at least one of the known security smells, with a total of 13,221 of security smells identified. They found that, security smells occur more often in GitHub projects compared to Mozilla, Openstack, and Wikimedia projects. They illustrate that out of 16,952 occurrences of hard-coded secrets, 68.3% are hard-coded keys, 23.9% are usernames, and 7.8% are passwords. The lifetime of these hard-coded secrets can be as long as 92 months.

Paper Takeaway [54]: ***To reduce code smells in infrastructure as code (IaC) scripts, a SAT tool called Security Linter was developed to help practitioners develop safer IaC scripts. In addition, seven common security smells in IaC scripts were identified by performing a qualitative analysis on a sample of 1,726 scripts.***

Citation: Ivan Pashchenko. Decision Support of Security Assessment of Software Vulnerabilities in Industrial Practice. PhD thesis, University of Trento, 2019.

Pashchenko's Ph.D. thesis [48] about the security assessment of software vulnerabilities includes three main components: a novel approach called Delta-Bench to automatically build benchmarks for static analysis security testing (SAST) using previous vulnerabilities in existing free open source software; Vuln4Real, a methodology for measuring vulnerabilities in free open source software; and a qualitative study (15 semi-structured interviews) to investigate developer decision-making regarding updating dependencies and security concerns. Their results show that 27% of the alerts from direct and 21% from transitive dependencies that cannot be exploited were removed. They also show that 21% of the vulnerabilities exist within direct dependencies so developers can take action towards eliminating these security issues.

According to previous state-of-the-art approaches such as FindBugs and Fortify SCA, developers have control over only 37% of the vulnerabilities, but in reality, they can fix about 80% of their known vulnerabilities. Also, 13.2% of the dependencies do not get any updates so developers should rely on them with caution.

In Pashchenko's semi-structured interviews, developers using different programming languages (C/C++, JAVA, JavaScript, or Python) mentioned four problems with dependency management: 1) the potential to introduce bugs and vulnerabilities; 2) lack of resources for proper dependency management; 3) high number of transitive dependencies; and 4) lack of proper dependency information sources. Most C/C++ developers were concerned about licensing issues. From their interviews, they found that most developers noted bugs to be the most crucial aspect to update a dependency. If developers were working on a project for internal purposes, security was not a concern. The main reason developers did not update their dependencies was the possibility of introducing new bugs and a lack of resources. Most of them were satisfied with existing static analysis tools. Experienced developers sometimes contribute to fixing dependency vulnerabilities, although the less skilled developers do not feel comfortable enough to do it. In the end, when it came to selecting new dependencies for a project, developers emphasized these five aspects: 1) popularity of the dependency; 2) checking the repository of the library; 3) checking the frequency of resolved issues for the library; 4) checking the release notes; and 5) checking if the library is easy to understand.

Paper Takeaway [48]: ***To automatically build benchmarks for SAT testing, DeltaBench was introduced to detect previous vulnerabilities in existing free open source software, and the Vuln4Real tool helps measure the extent of vulnerabilities in open source projects. From the interviews, they found that developers tend not to update dependencies because 1) it may increase the possibility of introducing new bugs, 2) they lack resources to properly manage dependencies, 3) there is a high number of transitive dependencies, and 4) there is a lack of proper dependency information sources.***

Citation: Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin 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, pages 426–437, 2015.

The use of analysis tools can help developers notice potential vulnerabilities during the development process. However, the high number of false positives these tools return makes this task a tedious process, thus causing some developers to avoid fixing the issues. Perl et al. [49] introduced a new method called VCCFinder, a support vector machine-based detection model that can find potential vulnerabilities with a lower false-positive rate compared to existing techniques such as Flawfinder [70]. They reduced the false positives by over 99% compared to Flawfinder. They also constructed the first large-scale database mapping of CVEs to GitHub commits. They evaluated their method of gathering this data to confirm its quality and to use it as a benchmark for future studies.

Paper Takeaway [49]: ***A support vector machine-based detection model, VCCFinder, helps find potential vulnerabilities with lower false-positive rates compared to existing methods. This work led to the first large-scale database mapping CVEs to GitHub commits to create a vulnerable commit database.***

Citation: Joseph P Near and Daniel Jackson. Finding security bugs in web applications using a catalog of access control patterns. In Proceedings of the 38th International Conference on Software Engineering, pages 947–958. ACM, 2016.

Near and Jackson [44] proposed a technique called SPACE to find missing security checks in web applications. Their technique checks to see if all the data exposures allowed in a web application comply with their catalog of access control policies, and if they do not, they report the data exposure as a vulnerability. Their catalog is constructed on the assumption that most developers allow a certain code pattern for all uses of a given resource type. They came up with this assumption based on their experience with real-world applications. After examining 50 open source Rails applications hosted on GitHub, they found 23 unique bugs where the analysis took no more than 64 seconds to complete.

Paper Takeaway [44]: ***To find missing security checks in web applications, SPACE was developed to check all the data exposures in an application.***

Citation: Rijnard van Tonder and Claire Le Goues. Defending against the attack of the micro-

clones. In 2016 IEEE 24th International Conference on Program Compre- hension (ICPC), pages 1–4. IEEE, 2016.

In software projects, redundant statements, subexpressions, and small fragments of code can lead to security breaches. van Tonder and Le Goues [63] presented a solution for automatic detection and removal of these micro-clones. They used the Boa mining infrastructure [17] in 380,125 Java repositories to find redundant micro-clones that would benefit the project if removed. After finding the micro-clones, they created 43 pull requests to the relevant GitHub repositories: 95% of their patches were merged within 15 hours, and 76% of their accepted patches did not require a lot of modification and could simply be merged with the project.

Paper Takeaway [63]: ***To find redundant micro-clones that could lead to security breaches, the Boa tool can be used for mining infrastructure.***

In summary, one aspect neglected from the proposed SAT solutions for identifying security issues is evaluating the tools with human stakeholders. These tools are eventually going to be used by developers so their usability would be better determined through user studies. If developers do not recognize security as a priority, new SAT solutions might not be useful to them.

4.1.3 Understanding Security-Related Behaviors by Studying GitHub Users

Four papers I found in my review aimed to understand user behaviors towards security or aimed to detect malicious users on GitHub. I provide an overview of these papers in this section.

Citation: Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. Comparing the usability of cryp- tographic apis. In 2017 IEEE Symposium on Security and Privacy (SP), pages 154–171, 2017.

Acar et al. [1] investigated how and why the design of different cryptography libraries affect security. The goal of this study was to see how to build improved libraries. They recruited 256 Python developers from GitHub to implement common tasks involving cryptography using one of five different APIs. They examined the participants' results both for functional correctness and security. The results of this study demonstrated that in 20% of the tasks that were implemented functionally correct, participants believed their code was secure even when it was not. Simpler

APIs did boost security, as reducing the number of choices for developers decreased the chance they do make mistakes. These researchers observed that Python experience level was not correlated with security issues, but developers with more security experience had a higher chance to produce secure code.

Paper Takeaway [1]: ***In a user study, they found that simple APIs are easier to use, and thus they end up having fewer security issues. Providing clear instructions for using APIs can also reduce security issues.***

Citation: Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L. Mazurek, and Sascha Fahl. Security developer studies with github users: Exploring a convenience sample. In Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017), pages 81–95, Santa Clara, CA, July 2017. USENIX Association.

One important aspect of any security study is to recruit security professionals. This is a challenging task since these people are limited in number, geographically concentrated, and have higher demands to join a user study placed on them. So an alternative is to recruit students for studies, but the results might not be the same as if a set of professionals participated.

Acar et al. [3] recruited 307 users from GitHub and conducted an experiment to look for differences in how students and professionals address security issues. They compared the results in terms of functional correctness and security of the code produced. To differentiate students from professionals, they used the participants' self-reported data (years of experience, whether they are a student or a professional). In this study, professionals implemented 68.5% of the tasks meeting the required level of security, while students implemented 67.6% of the tasks with the required security. From a functionality perspective, professionals completed 89.6% of the tasks correctly, while students succeeded with implementing 84.5% of the tasks in a functionally correct manner. Each added year of experience correlated with an average 10% more functionally correct code and 5% more secure code. But their results illustrate there are no major differences between students and professionals when it comes to security tasks. In the end, they concluded that at least when it comes to GitHub users, students and professionals can be equally successful.

Paper Takeaway [3]: ***According to a study of 307 GitHub users, there are no major differences to recruiting students or professionals for security studies in GitHub.***

Citation: Wei Bai, Omer Akgul, and Michelle L. Mazurek. A qualitative investigation of insecure code propagation from online forums. In 2019 IEEE Cybersecurity Development (SecDev), pages 34–48, 2019.

Some well-known vulnerabilities, such as user input validation that leads to SQL injection are common. One of the main reasons shown by previous works [2], [19] is that developers using Stack Overflow tend to produce less secure code. To investigate why developers propagate vulnerabilities from Stack Overflow, Bai et al. [6] identified clones of insecure code snippets from Stack Overflow in GitHub repositories, then conducted surveys and interviews with the authors of the repositories. They found that some of the developers did not have enough security knowledge to validate their code or were over-confident in their opinions that the code was secure. And for some developers, adding functionality was higher of priority to writing secure code. While others thought security was not their job and should be performed by another team or be outsourced. In the end, they provided some suggestions for online forums (e.g., Stack Overflow) such as linking the material to educational content, archiving problematic posts so they can not be accessed as easily, and promoting a feedback system for security in these forums to encourage community feedback.

Paper Takeaway [6]: ***According to surveys and interviews done in this study, developers either do not have adequate security knowledge, think functionality has a higher priority over security, or assume security is not their job.***

Citation: Qingyuan Gong, Jiayun Zhang, Yang Chen, Qi Li, Yu Xiao, Xin Wang, and Pan Hui. Detecting malicious accounts in online developer communities using deep learning. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pages 1251–1260, 2019.

Open development platforms such as GitHub are open to a variety of members to contribute to projects. Attackers and malicious users also exist among genuine users and try to achieve their goals by concealing their true identities. To identify malicious users, Gong et al. [24] proposed a deep-learning tool called GitSec, which is based on a recent variation of LSTM (long-short term memory) called phased LSTM. They found, malicious users create events much faster than regular users, they have

more characters in their usernames, and they follow fewer users and have fewer followers. Most of the malicious users tend not to complete the optional fields on their profile pages, while legitimate users are more likely to show more information about themselves. Their study showed that 21.5% of the crawled users were classified as malicious, a significant number.

Paper Takeaway [24]: ***The GitSec deep learning tool can be used to detect malicious users on GitHub. By analyzing samples of the collected data, the found that malicious users create events faster, have more characters in their usernames, and follow fewer people/have fewer followers.***

In sumarry, by studying GitHub users we can identify and understand behaviours that lead to insecure solutions. Findings from the literature, such as, for developers functionality has more priority compared to security, conducting experiments with students would output the same results as if it was ran with developers, and the characteristics of malicious users on GitHub are valuable and useful information for constructing the path to a more secure environment.

4.1.4 Mitigating Security Threats by Analyzing Social Network Data

Threat assessment is a crucial security component in software development that often relies on social network analysis to gather insights and information about vulnerabilities, security breaches, and developer activities. In this section, I describe three papers in my review that discuss using social networks in security-related research of projects hosted on GitHub.

Citation: Sameera Horawalavithana, Abhishek Bhattacharjee, Renhao Liu, Nazim Choudhury, Lawrence O. Hall, and Adriana Iamnitchi. Mentions of security vulnerabili- ties on reddit, twitter and github. In IEEE/WIC/ACM International Conference on Web Intelligence, pages 200–207, 2019.

Horawalavithana et al. [28] compared user conversations on three different platforms (Reddit, Twitter, and GitHub) that discussed security vulnerabilities. They selected several vulnerabilities between January 2015 and May 2018 described by their Common Vulnerability Exposure (CVE) identifiers documented in the NVD (National Vulnerability Database). Using these vulnerabilities, the authors filtered tweets and conversations from Reddit between March 2016 and August 2017, and

they extracted repositories from GitHub that were involved with the selected CVEs between January and August 2017. The results of this study shows that some CVEs appeared on Twitter and GitHub a year before they were publicly disclosed. Most tweets regarding the CVEs were after the public disclosure, while most of the discussions on Reddit were before they were publicly disclosed. 80% of the posts on Reddit start and end before the publicly disclosed date of the vulnerability. The authors point out that the peak of activity concerning a vulnerability is eight months after the public disclosure date. They speculate the lag until the disclosure date happens because of the passive development process life-cycle where vulnerabilities are not addressed until a considerable exploit occurs. 40% of the CVEs that appeared on GitHub were also mentioned on Twitter, but only 3% were discussed on Reddit. Finally, the majority of activity relating to a vulnerability occurs after public disclosure. They suggest using monitoring and predicting tools based on Twitter and Reddit to mitigate security in GitHub.

Paper Takeaway [28]: ***According to a study on vulnerabilities between 2015 and 2018, social networks such as Twitter and Reddit contain many posts about vulnerabilities before they are publicly disclosed.***

Citation: Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. Toxic code snippets on stack overflow. IEEE Transactions on Software Engineering, 47(3):560–581, 2021.

Since there is no checking mechanism to validate code snippets copied in answers or questions posted on online forums such as Stack Overflow, there can be a substantial amount of code that contains security vulnerabilities. Ragkhitwetsagul et al. [53] surveyed 201 highly reputed Stack Overflow members and 87 visitors to the platform to investigate developers' awareness towards using code snippets that might contain security vulnerabilities. 19% of the developers reported that they do not update their code snippets when they know they are outdated. 66% of the visitors noted running into problems using the code snippets. 51.5% of the highly reputed members were rarely notified about their outdated code and 35.5% never received any notification. Outdated code has a high potential to introduce vulnerabilities to a project.

Paper Takeaway [53]: ***According to a survey of 201 highly reputed Stack Overflow members, most developers on Stack Overflow are not aware of their outdated snippets.***

Citation: Morteza Verdi, Ashkan Sami, Jafar Akhondali, Foutse Khomh, Gias Uddin, and Alireza Karami Motlagh. An empirical study of c++ vulnerabilities in crowd-sourced code examples. IEEE Transactions on Software Engineering, pages 1–1, 2020.

To study the impact of code snippets and how they flow through social networks hosted on Stack Overflow, Verdi et al. [64] investigated the reason why vulnerabilities are commonly found on these platforms. They specifically examined C++ code snippets posted on Stack Overflow over a period of ten years. They studied the prevalence of the C++ vulnerabilities and how they were reused in GitHub repositories. They reviewed 72,483 code snippets and found 69 vulnerable code snippets categorized into 29 types of CWE. 2,859 GitHub repositories contained these 69 vulnerabilities.

Verdi et al. also found 287 GitHub files from the repositories that linked to a potential vulnerability, in which 34 of them were fixed, but 253 of them still had vulnerabilities. To study how developers react when they are notified about a vulnerability, they created issues in the repositories with the vulnerabilities. They received 15 responses out of 174 posted issues. From the responses, they saw that 40% of the developers felt that although the code snippet was vulnerable, it would not affect their security because they were not using dynamic inputs. To allow users to detect vulnerabilities posted on Stack Overflow, a browser extension was developed as part of this work to check for vulnerabilities in code snippets.

Paper Takeaway [64]: ***Vulnerabilities in code snippets on Stack Overflow were categorized into 29 CWE types. A study to see how developers react when they are informed about a vulnerability showed that 40% of the developers thought the vulnerabilities would not affect their security.***

In summary, social network data can notify us of on-going vulnerability issues in the open source communities. This useful data can help developers and security analysts be aware of their security defects and help them avoid repeating insecure behaviors.

4.1.5 Analyzing Discussions and Issues Related to Security on GitHub Using NLP

Developer discussions, comments, and issues can be analyzed to understand the security topics that developers may focus on. One technology that can be used to analyze this data is Natural Language Processing (NLP). In this section, I describe papers

that used NLP to understand security concerns with projects hosted on GitHub.

Citation: Mansooreh Zahedi, Muhammad Ali Babar, and Christoph Treude. An empirical study of security issues posted in open source projects. In Proceedings of the 51st Hawaii International Conference on System Sciences, 2018.

Zahedi et al. [74] empirically identified security issues posted on 200 randomly chosen GitHub repositories. Their method of approach was combined with topic modeling (a Natural Language Processing technique) and qualitative analysis. They found 3% (1,938 out of 6,4963) were issues related to security and the most frequent keywords in these issues were login, hash, password, inject, authentic, crypt, cookie, credential, and certificate— the keyword "login" appeared the most frequently across the issues. Zahedi et al. identified 26 security-related topics among these issues that were primarily about cryptography and identity management. Finally, they presented seven high-level themes demonstrating the problems developers encounter when implementing security traits. Some of the themes include authentication or authorization issues, cryptography issues, and handling protocols.

Paper Takeaway [74]: ***3% of all the issues posted for 200 repositories on GitHub were related to security, and among the security issues, most of them were about cryptography and identity management.***

Citation: Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: Sentiment analysis of security discussions on github. In Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, page 348–351, New York, NY, USA, 2014. Association for Computing Machinery.

Pletea et al. [51] analyzed 60,658 commits and 54,892 pull requests from repositories hosted on GitHub. They performed sentiment analysis on their mined data with a Natural Language Text Processing (NLTP) tool. They discovered that approximately 10% of all discussions on GitHub were related to security. Another interesting finding was that negative emotions were expressed more in security discussions than others. They mention defects in their work, such as how some discussions were mistakenly tagged as security-related because the usernames of developers contributing to the discussion were included the keywords of the discussion. They used a pre-processing stage to eliminate irrelevant data but the tool was trained for movie reviews, so the accuracy of their results is debatable.

Paper Takeaway [51]: ***10% of all discussions of a sample of projects hosted on GitHub were security related and negative emotions were expressed more often in security discussions compared to other discussions.***

Citation: David N Palacio, Daniel McCrystal, Kevin Moran, Carlos Bernal-C ardenas, Denys Poshyvanyk, and Chris Shenefiel. Learning to identify security-related issues using convolutional neural networks. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 140–144. IEEE, 2019.

Different tools have been integrated into software development workflows to automate the process and eliminate repetitive and time-consuming tasks for developers. The application of security practices in an automated process has also received a lot of attention from leading organizations such as Microsoft by introducing the Secure Software Development Lifecycle [29].

To automatically identify security issues among other types of issues posted for a project, Palacio et al. [47] proposed an approach called SecureReqNEt that involves a combination of supervised and unsupervised deep learning techniques. They evaluated their approach using their CVE database, which included issues posted on GitHub and GitLab, Wikipedia articles, and a set of security-related requirements extracted from a project developed by a large telecommunications company.

Paper Takeaway [47]: ***To automatically detect security-related issues, SecureReqNEt was created by using a combination of supervised and unsupervised deep learning techniques.***

In summary, analyzing different artifacts in GitHub, such as discussions and issues, can show how developers communicate with each other to solve security issues and help us understand the existing security culture in the open source environment. These papers showed us how NLP can help to understand how widely security is discussed and identify security issues in GitHub.

4.1.6 Secret Leakage and API Security in GitHub Projects

Many projects hosted on GitHub use third-party services to speed up their development processes or to make them safer—they use functionality features from leading organizations or they apply ready-to-use cryptography services in their software to reduce security threats. Safer use of APIs typically requires using the protocols men-

tioned in their documentation. Unfortunately, some projects do not follow these protocols. In the following, I describe three papers that study projects hosted on GitHub that encounter issues with secret leakage or with API security.

Cryptography APIs are commonly used in open source projects. In a recent study of 2,324 projects hosted on GitHub that used the Java Cryptography Architecture, 72% misused an API [26]. One of the reasons given for this misuse is that these APIs are difficult to use so developers follow the practices that are available online, many of which are not safe.

Citation: Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. Cryptoexplorer: An interactive web platform supporting secure use of cryptography apis. In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 632–636. IEEE, 2020.

To help developers use cryptography APIs appropriately, Hazhirpasand et al. [27] developed a web platform called CryptoExplorer containing real-world examples for developers to browse and learn how to use the APIs securely. They provided 3,263 secure and 5,897 insecure uses mined from 2,324 Java projects on GitHub that used the Java Cryptography Architecture. To evaluate the user experience for this platform, they conducted interviews with 4 participants. The participants all acknowledged this application would help people better use cryptography APIs by illustrating real-world examples.

Paper Takeaway [27]: ***To facilitate learning cryptography APIs, CryptoExplorer, a web platform with real-world examples, was presented.***

Many applications use services provided by other companies through web APIs. For the user to make a secure connection and for the service provider to become aware that the person is a legitimate user, authentication using API keys is necessary. Some of the projects that use these APIs are hosted on open source platforms such as GitHub. Surprisingly, the API keys are saved in the projects, which are then exposed to malicious users who take advantage of the APIs.

Citation: Vibha Singhal Sinha, Diptikalyan Saha, Pankaj Dhoolia, Rohan Padhye, and Senthil Mani. Detecting and mitigating secret-key leaks in source code repositories. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pages 396–400. IEEE, 2015.

To detect and fix the problem with embedded API keys, Sinha et al. [59] de-

scribed several methods that can be used to detect API keys placed in projects. They showed the efficacy of the methods by testing them on a set of GitHub projects. They also proposed some remedies for developers to avoid this issue. The leak detection methods include: sample selection using keyword search; pattern-based search; heuristics-driven filtering; and source-based program slicing. The remedies for fixing the leaks include: preventing leaks explicitly (e.g., adding the files to .gitignore); remedying leaked keys (modifying the file containing sensitive information in the old version of the project); and preventing leaks automatically (warnings pushed to users by the open source platforms to fix the issue).

Paper Takeaway [59]: ***This paper suggests methods to detect and fix embedded API keys in open source projects, such as adding the files containing API keys to .gitignore and send warnings to users to fix the issue.***

Citation: Michael Meli, Matthew R McNiece, and Bradley Reaves. How bad can it git? characterizing secret leakage in public github repositories. In Proceedings of the 26th Annual Network and Distributed System Security Symposium, (NDSS'19), 2019.

GitHub promotes public code sharing to increase collaboration and enhance communication. In some projects, not only is the code publicly available, but also the authentication secrets such as API keys and other credentials, which may be a threat to the projects' or people's security and privacy. To investigate this issue, Meli et al. [41] performed a large-scale and longitudinal analysis of secret leakage in GitHub projects. They collected all the files that potentially contained a secret with a GitHub API search and searched a snapshot of GitHub and maintained it as a public dataset in Google BigQuery [25]. Then they scanned the collected files with regular expressions to identify files with secrets. Their findings showed that the majority of leaked secrets remain available for weeks or longer, and committing cryptographic API keys embedded in code and files containing secrets are the main cause of leaks. They discussed some mitigation practices, such as automatic leakage detectors, requiring multiple secret values, and limiting queries on GitHub.

Paper Takeaway [41]: ***API keys and cryptography secrets are leaked at the rate of thousands per day. There are various mitigation solutions for addressing secret leakage, such as automatic leakage detectors.***

In summary, the above studies analyzed projects hosted on GitHub that use different API services. These studies show that issues can be introduced by both the consumer and the provider of an API. Providing clear and usable documentation can help developers avoid security threats when using different APIs and avoid secret leaks in public repositories. Consumers of the APIs can avoid secrete leakage by not including the secrets in the repository.

4.1.7 Imitating Security Attacks to Prevent Future Attacks

An acceptable practice for finding security flaws is to imitate security attacks that might occur in the future. In this section, I mention two papers that present attacks to help anticipate future exploits.

Citation: Alberto Calvi and Luca Vigano. An automated approach for testing the security of web applications against chained attacks. In Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC '16, page 2095–2102, New York, NY, USA, 2016. Association for Computing Machinery.

To test the security of web applications, Calvi and Vigano [9] presented an automated chained attack approach based on Model-Based Testing. They applied their chained attacks to a real-life case study of an exploitation disclosed by a bug bounty hunter, combined with five low-severity vulnerabilities to access a private repository on GitHub.

Paper Takeaway [9]: ***To test web application security, the authors introduced an automated chained attack approach based on Model-Based Testing.***

Citation: Liang Wang, Paul Grubbs, Jiahui Lu, Vincent Bindschaedler, David Cash, and Thomas Ristenpart. Side-channel attacks on shared search indexes. In 2017 IEEE Symposium on Security and Privacy (SP), pages 673–692. IEEE, 2017.

Modern search systems such as Elasticsearch [18] and Apache Solr [4] provide document retrieval capabilities by using keywords and queries. According to Wang et al. [67], the indexes used in these search systems might contain private data that is exposed to other users who should not have the privilege. They investigated ways to mitigate this side-channel attack on modern multi-tenant search systems. They developed the Search Text Relevance Score Side channel (STRESS) attack, a methodology

to exploit DF side channels. They formed efficient attacks on GitHub using their multi-step methodology, and they estimated the time and cost to extract information such as phone and credit numbers from private files in these services.

Paper Takeaway [67]: ***To mitigate side-channel attacks on modern multi-tenant search systems, a methodology called STRESS attacks was created to exploit document frequencies side channels.***

In summary, anticipating future attacks is a key element in security breach prevention. Papers in this topic introduce approaches to imitate security attacks in order to help threat assessment by foreseeing future breaches.

4.1.8 Vulnerability Datasets

Having a curated database of vulnerabilities and a history of security issues in the open source community can help in a variety of ways. The database can be used as training data for machine-learning applications, can support future research to ameliorate security practices in software development, and can bring awareness about security issues to developers.

Citation: Antonios Gkortzis, Dimitris Mitropoulos, and Diomidis Spinellis. Vulinoss: A dataset of security vulnerabilities in open-source systems. In Proceedings of the 15th International Conference on Mining Software Repositories, MSR '18, page 18–21, New York, NY, USA, 2018. Association for Computing Machinery.

Gkortzis et al. [22] analyzed NVD advisories and mapped every vulnerability to 153 open source projects. To build their database, they completed four steps: they processed various vulnerability reports from NVD to identify a set of open source projects, they cloned selected open source projects, they gathered version references of the projects, and they checked-out specific versions from the repository. Their database contains information about CVEs, CWEs, software categories, projects, project releases, vulnerable cases, continuous integration providers, programming languages, and code metrics.

Paper Takeaway [22]: ***A database of NVD vulnerabilities mapped to 153 open source projects was created to support security research.***

Citation: Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cedric

Dangremont. A manually-curated dataset of fixes to vulnerabilities of open-source software. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pages 383–387. IEEE, 2019.

A database curated by Ponta et al. [52] has a finer granularity compared to the previous work. Instead of projects, they mapped vulnerabilities to the commits that fix them. They followed three steps to construct their database: they chose an identifier for the vulnerability (such as CVE), they identified the repository containing the vulnerability, and they identified the commit that fixed the vulnerability. Their database contains 1,282 commits from 205 different open source Java projects in SAP[4] software. They claimed that industry has widely adopted the projects in their dataset. They pointed out the advantage of their database compared to Snyk is that they provide the fix for the vulnerability. This fix is not always included in Snyk and the Snyk database is not free to download.

Paper Takeaway [52]: ***A database of vulnerabilities mapped to commits with suggested fixes for the vulnerabilities was made freely available.***

In summary, having a curated database of vulnerabilities that leads to security fixes in open source projects is highly valuable to package managers and developers who use the packages. Using these databases as test and training datasets is also another benefit.

4.1.9 Bounty Systems for Security Issues

Since participation in open source platforms is voluntary, security vulnerabilities with high priority can be neglected. Monetary reward systems such as bounties can be an appropriate incentive to attract developers' attention towards these issues.

Citation: Jiayuan Zhou, Shaowei Wang, Cor-Paul Bezemer, Ying Zou, and Ahmed E. Hassan. Studying the association between bountysource bounties and the issue-addressing likelihood of github issue reports. IEEE Transactions on Software Engineering, 47(12):2919–2933, 2021.

Zhou et al. [75] studied the effect of bounties in solving issues posted for projects hosted on GitHub. They investigated 5,445 bounties for projects on GitHub posted on the Bountysource platform with a value of $406,425. They used logistic regression to investigate the relationships between 26 factors, e.g., the time the bounty was proposed and the bounty usage frequency of the project. They found that for a bounty

[4]https://www.sap.com/index.html

to be solved, the timing of posting it is the most influential factor. Bounty issue reports are more likely to be used in a project that uses bounties more often, and bounties are more likely to be addressed if they are posted earlier. The total value of the bounty is the most important factor in the projects using this system for the first time. 62.7% of all bounty reports are closed, although one-third of these closed issues are unpaid with a value of $41,856. 30% of all the bounties were posted 180 days after the issues for them were created, while 35% of the bounties were posted within seven days.

Paper Takeaway [75]: ***The time for submitting a bounty is an important factor, bounties are more likely to work in projects that use bounties more often, and the total value of the bounty is the most important factor for a project that is using it for the first time to receive more attention from bounty hunters.***

In summary, putting security issues in the spotlight by providing incentives is an effective approach for solving them. Applying important factors such as the time for bounty submission, frequency of bounties for a project, and the value of the bounty can help solve security issues faster.

In this section, I provided an overview and summary of all the studies I found in the mapping study. I classified these papers into nine topics to provide a high-level overview of the main topics addressed in this research. In the next section, I use a maturity model to analyze the topics and find which aspects in the topics are missing from the literature.

4.2 A Deeper Analysis of the Primary Studies

For my mapping study, I first identified 36 papers that address security concerns in projects hosted on the GitHub platform. I categorized the papers based on their research goals and used a card sorting method to identify the main topics these papers addressed. I found most of the papers focused on tools and interventions, while some of them examined how these tools can be improved or how they compare with other tools. Other researchers analyzed vulnerabilities in the projects and described how these vulnerabilities may propagate in open source projects. In the following, I consider how the paper topics I found in my review align with topics described by a

software assurance model. I do this to validate the topics I found in the card sorting but also to check if there are gaps in the papers that consider the security of projects hosted on GitHub.

During the course of my review, I further observed, that only seven studies addressed the developer perspective to see whether they adopt security and the reasons behind their choices. To confirm this observation, I use a framework, called the Who-What-How framework, to examine how these papers address social and technical aspects.

4.2.1 Mapping to the Software Assurance Maturity Model

Chandra et al. built SAMM, a software assurance maturity model, to describe how security can be analyzed in software development [12]. At the highest level, SAMM is comprised of four different business functions: *Governance*, *Construction*, *Verification*, and *Deployment*. Each business function captures a category of activities related to the mechanics of software development, and every business function is connected to three security practices. The following lists the practices for each of these four business functions:

> **Governance:** Strategy & Metrics, Policy & Compliance, and Education & Guidance.
>
> **Construction:** Threat Assessment, Security Requirements, and Secure Architecture.
>
> **Verification:** Design Review, Code Review, and Security Testing.
>
> **Deployment:** Vulnerability Management, Environment Hardening, and Operational Enablement.

Since I categorized the papers into nine topics using a bottom-up process, I next used SAMM to map the topics I found according to the security practices in the four different software development business functions mentioned above. Figure 4.2 shows a visual representation of the SAMM model. I assigned each of the topics found to a security practice in this model, and by doing so, I was able to identify gaps in the topics covered in the papers. I linked these mappings based on the definitions of the security practices presented by Chandra [12]. Figure 4.3 illustrates the result of my

mappings. In the following, I list all the categories I discussed in Section 4.1 and specify which security practice in SAMM they are mapped to.

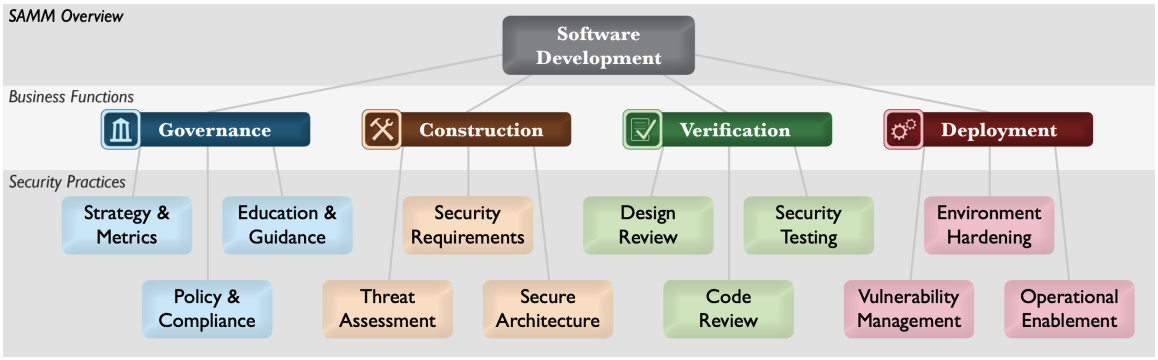

Figure 4.2: Software assurance maturity model presented by Chandra [12].

Managing Vulnerabilities in Third-Party Libraries: The papers summarized in this category analyzed vulnerabilities in projects hosted on GitHub and discussed how developers work to solve them. I observed that 13 papers discussed how projects are affected by a vulnerable package. The papers' authors mentioned that developers who use the dependencies with vulnerabilities do not take adequate action to mitigate them. The studies generally mine repositories hosted on GitHub and illustrate how vulnerabilities are included in many projects by tracking their dependency tree. Although, not many studies focused on why these vulnerabilities exist and did not discuss why developers ignore them until a breach occurs. Consequently, I believe there is room to investigate why developers ignore mitigating vulnerabilities. This topic maps closely to the "Vulnerability Management" security practice in SAMM.

Developing and Enhancing Static Analysis Tools: Static analysis tools are used to review code in an automated fashion. Some of the papers introduced novel methods and some improved existing ones. Nevertheless, not many papers mentioned why these tools are not widely adopted or why developers ignore the warnings provided by static analysis tools. More accurate tools might not be helpful if developers still choose not to pay attention to them. Thus, I think investigating the adoption of SAT tools by developers might give us a better understanding on how to improve utilizing the tools. Since code review refers to the inspection of software at the source-code level, I mapped this topic to the "Code Review" security practice topic in SAMM.

Understanding Security-Related Behaviors by Studying GitHub Users: In SAMM, I did not find a security practice to map to this topic. However, by

conducting studies on developers who contribute to an open source platform such as GitHub, we may better understand how to help developers adopt more secure practices.

Mitigating Security Threats by Analyzing Social Network Data: Studying existing data in social networks might help us mitigate vulnerabilities quickly, warn developers about certain vulnerabilities, and help them recover from security incidents faster. The papers in this category demonstrate the importance of the insights that can be discerned from data in social networks and how they can be used for bolstering security. I matched this topic to "Threat Assessment" in SAMM because the information from social networks may help reduce security risks and anticipate security breaches.

Analyzing Discussions and Issues Related to Security on GitHub Using NLP: I also mapped this topic to the "Threat Assessment" topic in SAMM. We can study the discussions around security issues existing on GitHub projects to find out the security trends. Finding the trends can help practitioners to see what issues need more attention.

Secret Leakage and API Security in GitHub Projects: I mapped this topic to the "Operational Enablement" security practice in SAMM because these papers point out how documenting usable instructions for working with an API plays an important role in secure API usage. However if the documentation of the APIs are vague or hard to use, developers will not adopt the guidelines mentioned in the documenation and might end up introducing a vulnerability. Also many API secrets and keys are pushed to the repositories. This issue can be avoided by providing more education on how to include these secrets and keys in the project. Overall, this topic points out how communication matters in software development and how this social aspect plays a prominent role in security.

Imitating Security Attacks to Prevent Future Attacks: Since this topic is about preventing future attacks, I mapped this topic to the "Threat Assessment" topic in SAMM.

Vulnerability Datasets: I could not find a match for this topic in SAMM. The papers in this topic mostly capture techniques and associated datasets for addressing security concerns by developers, researchers, and operators.

Bounty Systems for Security Issues: Again, none of the SAMM security practices were a perfect fit for papers in this topic.

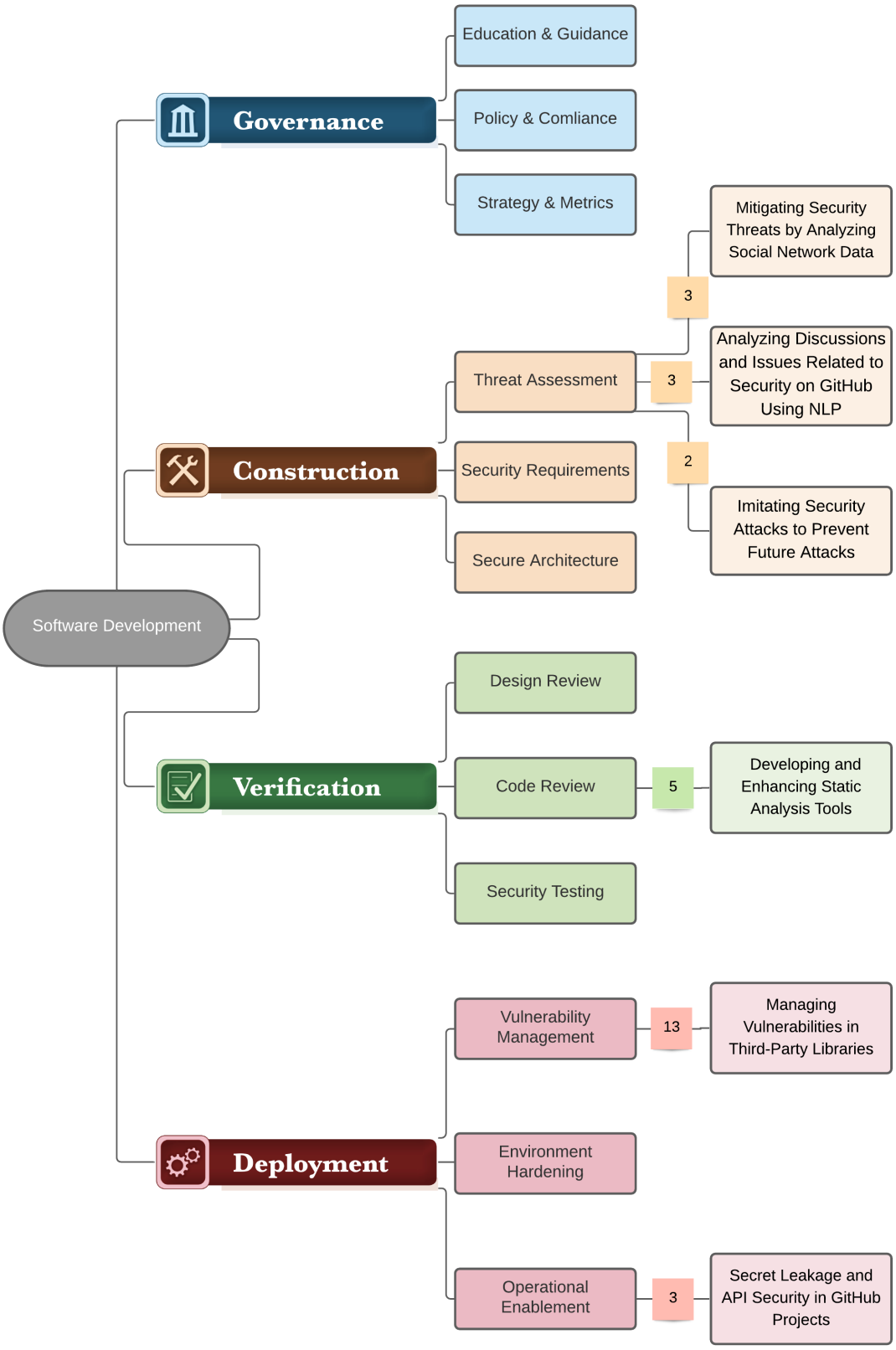

Figure 4.3: Mapping emerged topics from my mapping study to SAMM security practices.

After I mapped the topics from my mapping study to the security practices in SAMM, I discovered that most of the papers were associated with the *Vulnerability Management* and *Code Review* security practices. We know vulnerability management and code review have two sides; technical and social. An example of a technical aspect of vulnerability management is when tools give accurate information about the dependency tree for the project and an example of a social aspect is when developers adopt the tool to track down vulnerabilities in used packages and mitigate them. Most papers in my mapping study did not focus on social aspects and most researchers mined repositories to give insights on the security status of existing projects hosted on GitHub. Not many pointed out how developers ignore the fact that there is a possible vulnerability in their third-party packages. For code review, an example of a technical aspect is the use of static analysis tools to find security issues. An example of a social aspect is the adoption of those static analysis tools by developers. Notably, most of the studies categorized in this topic aimed to improve static analysis tools and not many focused on the adoption of these tools.

Figure 4.3 shows that none of the identified topics in the literature were associated with the *Governance* business function. In this business function, three important security practices are mentioned: *Education & Guidance*, *Policy & Compliance*, and *Strategy & Metrics*. These security practices are mainly about how organizations manage the overall software development activities. Without these practices, organizations cannot ensure the security of their software. Similar to this finding, Wen [69] concluded that not many papers focused on the *Governance* business function and mentioned this may be due to many open source projects do not have management teams to organize, lead, and monitor the software development process. Wen [69] also found that most papers (98%) in their SLR were about technical aspects compared to other aspects such as cultural, legal, and managerial aspects.

In the next section, I present how I further analyzed the papers using a socio-technical framework to examine the social aspects of the final set of primary studies.

4.3 How Human and Social Aspects Are Considered in the Security Research for GitHub

From the literature I reviewed in my mapping study, I observed that most of these studies primarily focus on technical issues, neglecting much needed consideration

of human and social aspects. As mentioned, Wen [69] also pointed out that human aspects in this research area are neglected and require more attention. To confirm my observation, I revisited the papers I reviewed and performed an analysis of the papers using the Who-What-How framework [60] that considers **who** are the beneficiaries of the research, **which methods** are used and whether those methods consider human and social aspects, and what is the the nature of the **research contribution** from the papers.

To find the claimed beneficiary of the studies in my mapping review, I studied the authors' texts to find quotes that explicitly mentioned who would benefit from the study or I relied on other statements made by the authors to determine the beneficiary (in only a few cases, the papers lacked explicit indication of the research beneficiary). Based on the Who-What-How framework (see Table 3.2), I categorized the claimed beneficiaries into three groups: **Human** (e.g., developers and practitioners), **System** (e.g., tools and applications), and **Researcher**.

I also investigated how the authors of the papers in my review approached their research and identified which research strategies they used (see Table 3.2). Some papers used qualitative methods such as interviews and surveys with direct involvement of human subjects, while others used quantitative methods such as mining repositories hosted on GitHub.

For the research contribution, I categorized the studies according to two different categories: 1) **Solution** papers provide a contribution concerning an intervention (a tool or a process) and 2) **Descriptive** papers provide a descriptive or predictive theoretical contribution. For the papers that described a problem and provided a solution to mitigate it, I categorized them as both **Descriptive** and **Solution**.

In the next section, I describe the results of my analysis on finding the claimed beneficiaries in the primary studies.

4.3.1 Who is the Main Beneficiary of the Research

One of the claimed beneficiaries of 89% of the primary studies (32 out of 36) was human stakeholders (developers, managers, and experts). The rest of the studies mentioned their research only benefits researchers. Studies contained statements such as "We report our findings and provide guidelines for package maintainers and tool developers to improve the process of dealing with security issues" [14] and "The goal of this paper is to help practitioners avoid insecure coding practices" [54].

Who	**Number of Studies**	**Paper**
Human	32	[1] [6] [9] [10] [14] [15] [21] [24] [27] [28] [33] [36] [37] [40] [43] [44] [47] [48] [49] [51] [52] [53] [54] [55] [59] [62] [63] [64] [67] [68] [73] [75]
Researcher	19	[1] [3] [10] [15] [21] [22] [27] [28] [40] [41] [43] [48] [49] [52] [53] [54] [68] [73] [74]
System	15	[9] [15] [24] [27] [33] [40] [44] [47] [48] [49] [54] [63] [64] [67] [68]
Human & Researcher	8	[1] [10] [21] [28] [43] [52] [53] [73]
Human & System	8	[9] [24] [33] [44] [47] [63] [64] [67]
Human, Researcher & System	7	[15] [27] [40] [48] [49] [54] [68]
What	**Number of Studies**	**Paper**
Descriptive	26	[1] [3] [6] [10] [14] [15] [21] [28] [33] [36] [37] [40] [41] [43] [48] [51] [53] [54] [55] [59] [62] [64] [67] [73] [74]
Solution	16	[9] [15] [21] [22] [24] [27] [33] [44] [47] [48] [49] [52] [63] [64] [67] [68]
Descriptive & Solution	6	[15] [21] [33] [48] [64] [67]
How	**Number of Studies**	**Paper**
Data Study	33	[6] [10] [14] [15] [21] [22] [24] [27] [28] [33] [36] [37] [40] [41] [43] [44] [47] [48] [49] [51] [52] [53] [54] [55] [59] [62] [63] [64] [67] [68] [73] [74] [75]
Sample Survey	7	[6] [36] [43] [48] [53] [62] [64]
Field Study	2	[54] [63]
Laboratory Experiment	3	[1] [3] [27]
Formal Theory	2	[24] [67]
Experimental Simulation	1	[9]

Table 4.1: Results of the socio-technical analysis.

In Figure 4.4, I show the distribution of the beneficiaries claimed in the literature by the topics I identified in my mapping study. For example, in the "Managing Vulnerabilities in Third-Party Libraries" topic, 13 papers mentioned they are aiming to benefit humans with their work, four papers indicated they aimed to enhance systems, and eight papers said their work can help future researchers. To clarify, 23 papers mentioned two or more beneficiaries for their studies, for instance, 8 papers (Table 4.1) claimed their research benefits both practitioners and researchers.

Overall, 15 studies specified their research will potentially improve current software systems. All of these 15 studies also claimed **Human** as one of the beneficiaries; however, not all of them used a research strategy that directly involved human participants. In the next section, I describe the research strategies used in the studies.

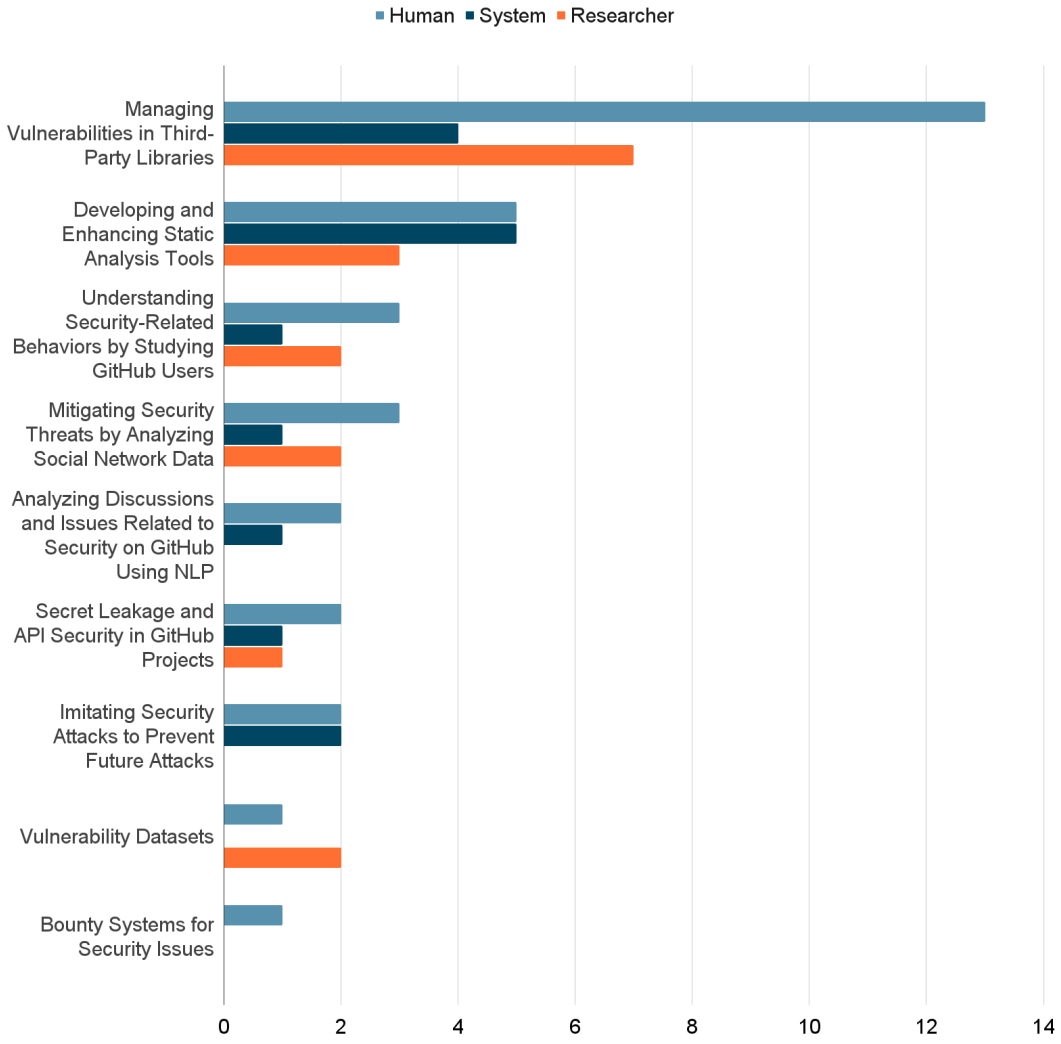

Figure 4.4: Distribution of the claimed beneficiary in each topic (Who).

4.3.2 How Was the Research Approached

33 studies (92% of the primary studies) in my review relied on mining trace data such as repositories, pull requests, and discussions on projects hosted on GitHub to analyze security issues. However not all of them only relied on data strategies. Seven used *sample survey* where researchers either conducted interviews or surveyed participants about a set of security tasks to understand their behaviors. Three studies conducted laboratory experiments, two studies performed field studies, and two used

formal theories to uncover new information using mathematical terms. I only found one paper that conducted experimental simulation.

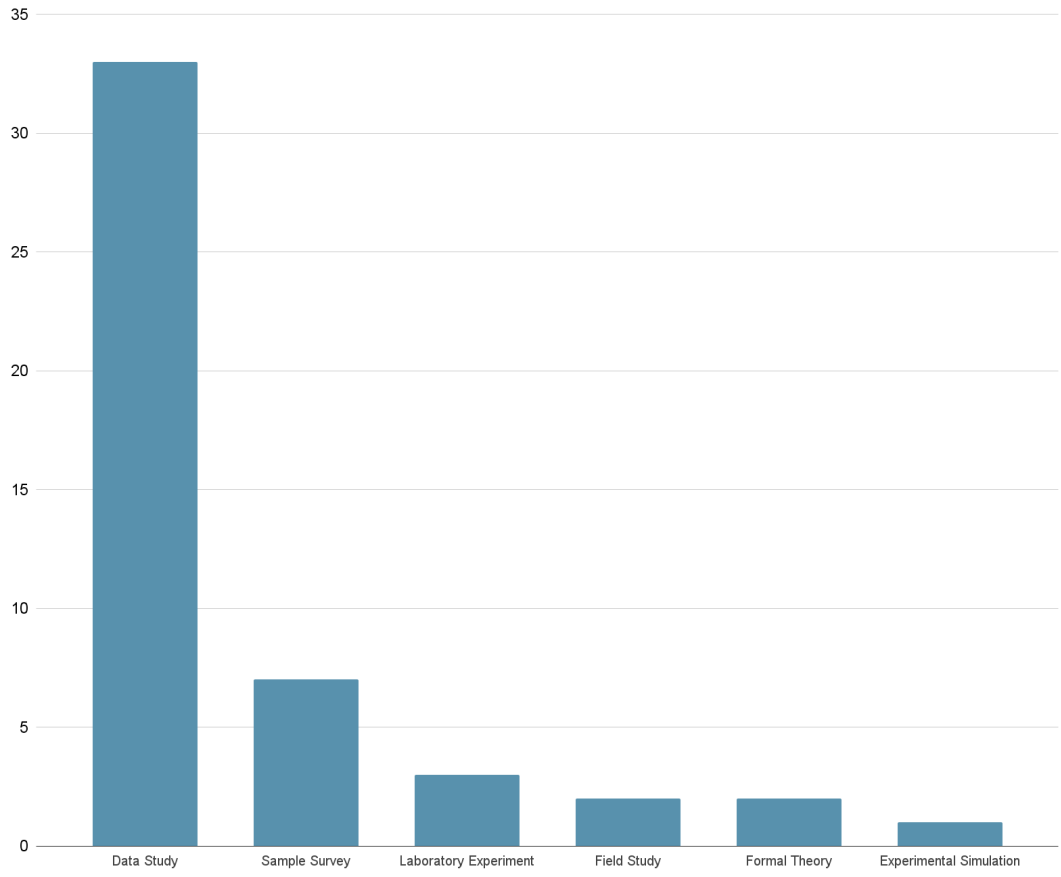

Figure 4.5: Number of studies that used each research strategy (How).

Figure 4.5 shows a summary of the research strategies used. We can see from the Figure the huge difference between number of studies that used other research methods (sample survey, field study, laboratory experiment, formal theory, and experimental simulation) and those that used data mining strategies. 12 (33%) studies used multiple research strategies, e.g., after performing an analysis on a set of GitHub repositories, a study [48] also conducted interviews with participants to further investigate the usability of their presented tools. All seven (18%) studies that used sample survey as one of their research strategies also used data mining strategies.

Among the 12 primary studies (39%) that used *sample survey, field study* or *laboratory experiment* as one of their research strategies, only three [1] [3] [6] recruited

participants that were GitHub users. Others either found participants from forums such as Reddit, contacted practitioners from different companies, or they did not mention explicitly how they found the participants for their studies.

Recruiting participants to conduct interviews and surveys is a difficult task. However, I found 12 studies (33%) that used research methods that directly involved human participants. To understand how papers used the research methods in order to show how their research contribution benefits human stakeholders, I elaborate on two of them in the following.

Pashchenko [48] performed semi-structured interviews with 15 software developers. Their study aimed to find out how security concerns influence developer strategies for selecting and updating dependencies, what methods they apply for managing dependencies, and how they mitigate vulnerabilities that do not have a fixed version. To find developer strategies when it comes to updating dependencies, it is more effective to use qualitative approaches such as interviews and surveys compared to data strategies.

Hazhirpasand et al. [27] created a tool called CryptoExplorer to help developers use cryptographic APIs properly. They evaluated their tool by conducting a laboratory experiment with four participants. Observing the participants in a laboratory setting, helped them understand it is easier for them to use the tool rather than searching for cryptography API misuses on the internet.

Now that we have an understanding on the claimed beneficiaries of the primary studies and the research methods used in the papers, in the next section I describe what was the type of research contribution of the papers.

4.3.3 What is the Main Type of Research Contribution

26 papers (72%) in my mapping study were categorized as descriptive papers. These studies explored and described either the data, users, or system characteristics concerning security issues. 16 studies presented a solution to either mitigate vulnerabilities, reduce false positives of security concerns, help developers patch security issues, or performed other fixes to address security issues.

Overall, six studies (18%) ([21], [64], [48], [15], [67], and [63]) contained both a descriptive analysis and a proposed solution. Figure 4.6 illustrates how most studies (26) took a descriptive approach. For topics, such as “Vulnerability datasets” and “Imitating security attacks to prevent future attacks”, the studies only presented a

solution. Also, most papers in "Managing vulnerabilities in third-party libraries" described an event, problem, or a situation where vulnerabilities exist. Fewer papers focused on providing a solution from this topic.

Across four topics, I found seven studies that provided a descriptive contribution and presented a solution for a problem. The four topics that included these studies are: "Managing Vulnerabilities in Third-Party Libraries", "Developing and enhancing static analysis tools", "Mitigating Security Threats by Analyzing Social Network Data", and "Imitating security attacks to prevent future attacks".

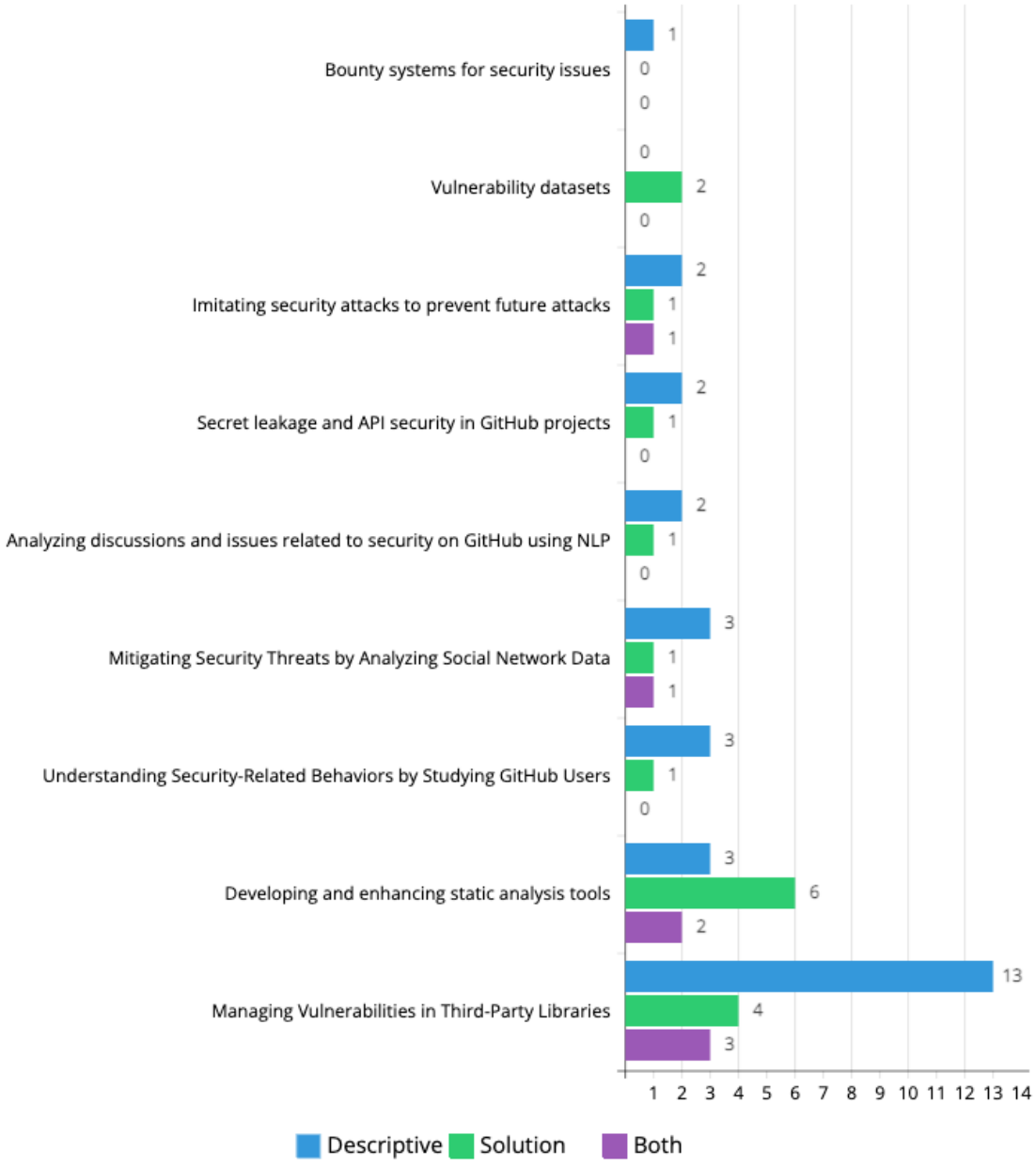

Figure 4.6: Distribution of research contributions in different topics (What).

4.3.4 Final Remarks on the Who-What-How Analysis

Although 32 papers (89%) aimed to benefit human stakeholders, most of them used strategies that did not involve human participants to validate their claims. 23 papers (64%) used data strategies as their research method and did not use strategies that included humans. Our analysis also shows most studies that claimed to benefit human stakeholders were descriptive. In these descriptive studies, 10 of them included

humans in their research design. From the studies that presented a solution, only six out of 16 included humans in their research design. Out of these 16 studies, only one of them did not claim to benefit humans. This means not all the solutions were evaluated by humans even though 15 out 16 of them claimed their work would benefit human stakeholders in some way. In Table 4.1, I demonstrate the distribution of the studies according to the Who-What-How framework.

Figure 4.7 shows that most studies chose data strategies over qualitative research methods. On the left vertical axis, we see most studies either claimed humans or humans and systems will benefit from their research study. However, most of them only used data strategies (the middle axis). On the right axis, we see somewhat an equal split between papers that provided a solution or described a problem. The right axis shows the papers that claimed to benefit humans and systems mostly provided a solution and the papers that claimed to only benefit humans mostly provided a descriptive contribution.

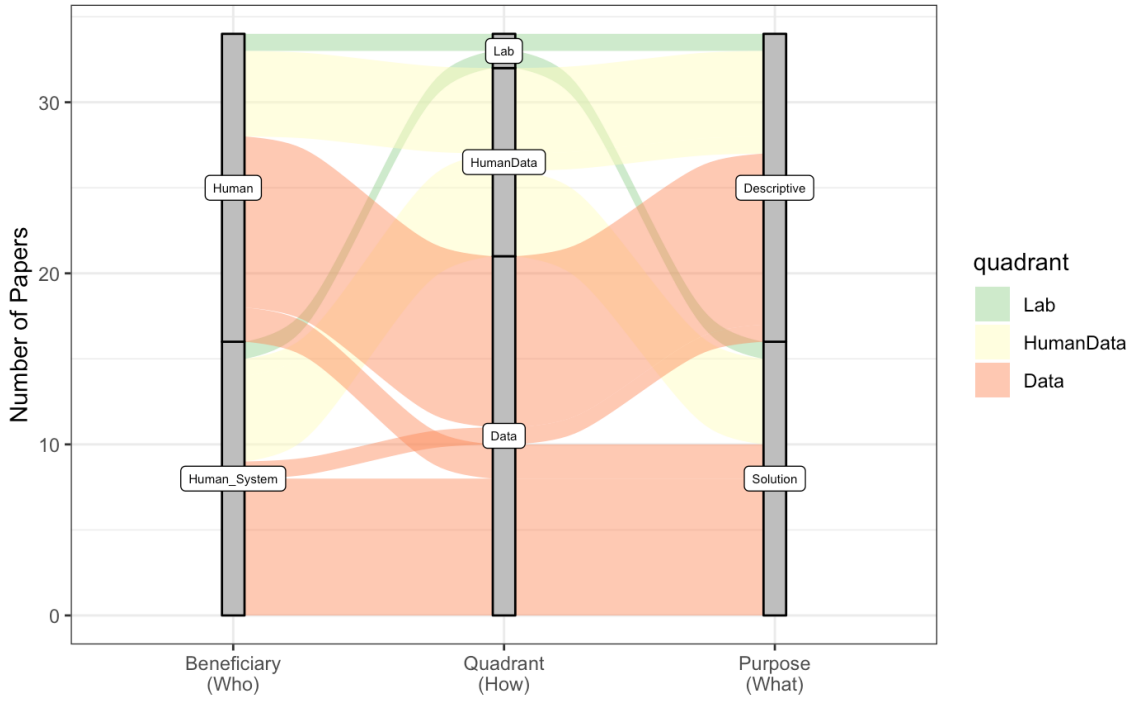

Figure 4.7: This alluvial diagram shows the flow from the claimed beneficiaries, to the research methods used, to contribution types of the primary studies.

4.4 Limitations of the Mapping Study

I conducted the mapping study using the methodology presented by Kitchenham and Charters [34]. After generating research questions and search strings, the research steps were evaluated by my supervisor using Kitchenham and Charters' guidelines. My supervisor also assisted in the evaluation of the research questions and choice of search strings extracted from the questions. Finally, she reviewed my categorization of the final set of primary studies, the result of mapping the nine topics to SAMM security practices, and the outcomes of Who-What-How analysis.

After conducting my mapping study, I categorized the papers into nine different topics according to the claims made in the studies. I used a card sorting method to group all similar papers together and labeled them according to their shared characteristics. I recognize that if other researchers performed this bottom-up clustering, there is a chance they might categorize the studies into different groups.

In the card sorting process, I mapped each paper only to one topic. However, some of the papers could be associated with more than one topic. For instance, the paper [36] was associated with the topic "Managing Vulnerabilities in Third-Party Libraries". The authors of this paper investigated to what extent developers update their dependencies when notified of a vulnerability. Although the main theme of this paper is about vulnerabilities, the authors also investigated why developers tend to postpone updating packages. Therefore, this paper could also be categorized as "Understanding Security-Related Behaviours by Studying GitHub Users". Assigning papers to multiple topics could have made the analysis more complicated. As a result, I decided to categorize the papers based on their main theme instead of assigning them to multiple topics.

I performed the Who-What-How analysis on the final set of primary studies. To gain feedback, another researcher—a graduate student in the CHISEL lab with experience in empirical software engineering and in conducting Systematic Literature Reviews—performed the analysis on all the studies for comparison of this step. We conducted agreement sessions to check the results and resolve any conflicts. Notably, only a few conflicts were spotted and they were resolved in the agreement sessions. I considered verifying this classification with the authors of the papers but found in other studies that authors normally don't take the time to answer such questions. But doing so may have improved the accuracy of my analysis.

The authors of the papers in my systematic mapping study investigated security

issues in open source projects hosted on GitHub. Some of these projects were developed by software practitioners who do not work in teams and contribute to these projects individually based on self-interest. One aspect that could be missing from these projects is the social interactions that developers usually have when building software products inside teams with a collaborative effort. There is a chance that not all social aspects have been fully captured in this systematic mapping study.

There are two approaches to incorporate security in a software product: proactive and reactive. A reactive approach involves responding to incidents such as hacks and data breaches after they occur. In contrast, a proactive approach to adopting security includes preemptively identifying security weaknesses and adding processes to identify threats before they occur. In this systematic mapping study, by investigating papers that looked into security issues in projects hosted on GitHub, I primarily focused on reactive approaches. For instance, many papers discussed how to mitigate vulnerabilities when they occur. Including proactive approaches would have made my results more comprehensive, but it was not in the scope of this study.

4.5 Future Work

I found many insightful suggestions such as imitating attacks for security breach prevention and providing clear and usable documentation for tools. Also, to put security in the spotlight, bounty systems require more attention to highlight security issues for bounty hunters. Currently, bounty systems are generalized for different types of issues posted for open source software. By emphasizing security-related issues in these systems, developers would more likely focus on eliminating them. The incentives provided for the bounties and the reputation developers get by resolving them motivates developers to focus on the bounties. One of the challenges of using bounty systems is to put adequate monetary incentives on the bounty so it attracts attention. However, not all project owners might have a great amount of budget to put high incentives on the bounties.

From my mapping study, I observed six studies focused on either improving or creating new Static Analysis Tools. While this is an important topic, many other topics such as using social-network data to understand and prevent vulnerabilities might be a good complement to the existing methods to mitigate security threats in projects hosted on GitHub. Social media has shown an immediate response in many exploit incidents. This data can be used much more effectively than waiting

for vulnerability databases to be updated such as, NVD, which usually takes a lot of time. However, data from social networks may not always be reliable. It needs to be validated before it is provided to use. Also, there might not be enough details in information to take action and fix the vulnerability.

To investigate how the authors of the papers in my systematic mapping study addressed different aspects of software security, I mapped the topics I identified from the literature to the security practices in SAMM [12]. From this mapping, I discovered that none of the topics identified in the literature were related to the security practices in the *Governance* business function. Future researchers can look into the issues that exist in the security practices associated with this business function and investigate how they can be mitigated. For instance, for the *Education & Guidance* security practice, researchers can study how to educate developers on open-development platforms to handle vulnerabilities.

The Who-What-How framework helped me to see how the literature has studied social aspects of software security. I observed that many researchers fail to directly study the developers they claim to support when they report how to address security issues in their research. One solution to reduce this problem, is to study the developers using open development platforms and figure out why security issues are generated in the first place. Even with advanced technologies, developers have yet to adopt security practices and tend to skip security checks. Many researchers focus on pushing the technology further but what would be the point if developers have not yet adopted the culture of using these technologies? Although, there are some challenges in studying developers such as recruiting participants, designing studies that aim to answer the intended research questions, and analyzing the collected data with appropriate tools.

I presented a few suggestions which I believe will help researchers better understand the cause of security issues. By recognizing the issues, we can design guidelines and best practices that would eventually help developers to better adopt security practices.

4.6 Chapter Summary

I conducted a mapping study on how security is studied on projects hosted on GitHub. I extracted nine topics that researchers focused on in a sample of 36 papers. I mapped my topics using SAMM to have a better understanding of the business functions and security practices in software development.

Based on mapping my topics to SAMM, I observed most of the papers focused on technical aspects. So to examine this thoroughly, I performed an analysis based on a socio-technical framework. From this analysis, I found that the technical aspects have seen far more attention in the literature compared to the social aspects. Many papers (22 out of 36) did not include humans in their research design; however they claimed to benefit human stakeholders.

By studying the literature and performing this mapping study, I provided some suggestions that I hope can help future researchers fill the gaps I identified and enhance security in open source platforms. Additionally, my work can guide researchers for future studies and provide an overview for practitioners on how we can improve security in open source environments.

In the next chapter, I will discuss the cross-sectional interview study performed to understand developers' behavior towards security adoption.

Chapter 5

Investigating the Behavioral Aspects of Adopting Security in Software Development

In Chapter 4 (mapping study), we observed that only a few studies have focused on investigating the social aspects of why developers either adopt or ignore security in their software development. As a result, I decided to collaborate with a team of researchers from the University of Victoria—I elaborate more on the team later in Section 5.2.4—to conduct a cross-sectional interview study based on a framework for behavior change in psychology to understand developers' capabilities, opportunities, and motivations towards security. We found that just having the technical skills to incorporate security in the SDLC is not sufficient as factors such as strong communication skills to facilitate performing security practices play a crucial role in software security.

This study aims to help organizations develop a strategy to adopt security practices. Furthermore, we aim to shed light on the aspects that have not yet been thoroughly investigated in order to understand the issues better and potentially find ways to approach them. In the following, we describe how based on the literature we found the need to investigate developer behaviors regarding security.

5.1 Background

As we found in our mapping study (Chapter 4), only a few papers have studied human and social aspects of software security. Assal et al. [5] investigated human behavior and motivation of developers towards software security. They surveyed developers and found a lack of organizational or process support to integrate security in development tasks. Balebako et al. [7], and Kaur et al. [31] surveyed and interviewed developers and found most of the small to medium-sized companies do not entirely adopt security. Sajwan et al. [58] investigated developers' security attitudes and behavior using Grounded Theory [23]. They discovered that factors influencing security could be presented in three themes: programmer culture, organizational factors, and industry trends.

Rauf et al. [56] conducted an interdisciplinary study using theory from cognitive and social psychology. They demonstrate different factors that hinder developers in achieving their security goals. One of the mentioned impediments for developers was not having the technical knowledge to perform security tasks. They show the current landscape of security interventions can be useful to address some of the impediments but many do not take the socio-technical context into account.

Votipka et al. [66] presented a validation on measuring secure software development self-efficacy for a variety of purposes. They review secure development frameworks and survey 22 security specialists to identify 58 tasks. Moreover, they asked 311 developers to rate their skill at each task. Their final 15-item scale measures two distinct factors: security communication, and vulnerability identification and mitigation. Finally, they make some suggestions on how the self-efficacy scale can be used for secure software development. For instance, one application of the self-efficacy scale is measuring security culture in organizations.

From the mapping study, we learned how many factors influence developers' security adoption, such as, organizational factors, industry trends, and technical knowledge about security. However, we did not observe a thorough investigation about developers' behaviors and decision-making process when it came to security. As a result, we decided to interview practitioners to gain a deeper understanding on their motivations, challenges, and attitudes regarding adopting security. We designed the interview using a behavioral model from psychology, which I elaborate more in the next section.

5.2 Research Method

For this study, I collaborated with a team of researchers from the University of Victoria—I elaborate more on the team later in this section. We designed a cross-sectional interview study to explore developers' security decision making process. We recruited 19 developers and nine security specialists by contacting practitioners on LinkedIn[1] and asking them if they were interested in participating in the study. The participants had a wide range of experience level, from one year to more than 20 years working in startups, medium-sized companies, and big corporations. All the participants joined the study solely based on self-interest. Each interview lasted about one hour, and was recorded and transcribed for further analysis. In the following section, we describe the study's design process.

5.2.1 Designing the Cross-sectional Interview Study

The goal of this study was to conduct a behavioral diagnosis about developers' security adoption. To achieve this goal, we followed a qualitative approach. Thus, we designed semi-structured interviews, performed five pilot interviews, and conducted 28 semi-structured interviews. For analyzing these interviews, we performed qualitative coding [20] on the transcripts. To create our master codebook, we held three agreement sessions based on coding three transcripts. Two researchers in our research group performed the agreement sessions. After reaching 80% agreement in the agreement sessions, each researcher coded the rest of the transcribed interviews. In addition to coding the transcripts, we held one thematic analysis session with two researchers[2] using the three coded interviews and the three components of the COM-B model [42] (described in the next section) to identify behavior patterns for security adoption. In the thematic analysis, we used information from three participants that we refer to E1, E9, and E16 in the preliminary results section. Figure 5.1 shows the steps performed in this study.

To understand developer behaviors towards security, we had to know the context where security is being adopted, developers technical capabilities for performing security tasks, information regarding their organizations, and developers' motivations and attitudes towards security practices.

To understand the context, we asked participants about their experience level,

[1] https://www.linkedin.com/

[2] The thematic analysis session was performed by the principal researcher and a PhD student.

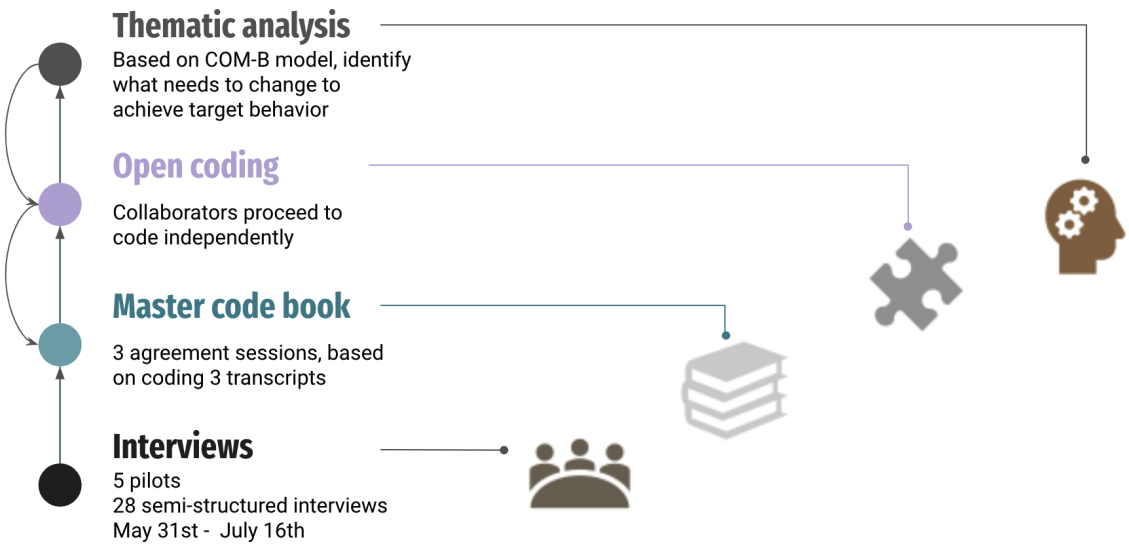

Figure 5.1: All the steps performed for the cross-sectional interview study.

workflows, and tools and technologies they use in their work. We wanted to gain insight into their experience level with security, so we asked them about security practices they use and how much those practices affect their day-to-day work. We were also curious about how security is being adopted in their organization. For instance, we asked them how their organization helps them to adopt security practices.

To understand developers' behaviors towards adopting security, an expert in social psychology suggested using behavioral models from psychology to understand how developers perceive security. We used a behavioral model to investigate the participants' perception of security and their motivation to adopt security practices. I elaborate more on the model in the next section.

We performed several iterations to design the interview questions until we reached a mature version of the interview protocol. Then, we grouped the questions based on similarity, and we revised the wording to be easier for the participants to comprehend. The final set of questions included four main sections: (1) interviewee experience and background, (2) interviewee's company information, (3) interviewee's information on security in software engineering, security interventions, and motivations for adopting security, and (4) questions based on a behavioral model. The list of questions we asked from the participants is provided in Appendices B.

After designing the questions, we performed pilot interviews to ensure that the questions were not ambiguous. We found that we should prioritize some of the ques-

tions according to the participant's experience and knowledge about software security. For instance, if a participant was not aware of security practices, we did not ask them about the advantages of adopting security practices, but instead we asked them about the challenges of adopting security. Furthermore, we shortened the interviews' length to one hour since initially they were too long.

The final set of questions was an excellent guide for collecting relevant information for our study. However, we wanted to know their stories and experiences with software security practices. So, we conducted semi-structured interviews to ask follow-up questions and gain more insight from diverse participants' contexts.

Designing and conducting the interviews was a collaborative effort between the principal researcher of this study and me. We held multiple meetings to discuss the interview questions, decide on the study's steps, and share knowledge on how to conduct the interviews. In addition, we both attended the pilot interviews and the semi-structured interviews. I acted as the primary interviewer in some cases, and the principal researcher took notes during the interviews. In other cases, the principal researcher led the interviews, and I took notes and asked follow-up questions if needed.

In the analysis phase, I performed open coding on two of the transcribed interviews. I had two agreement sessions with the principal researcher of this study to discuss the outcomes of the open coding. The principal researcher and a PhD student from our research group coded the rest of the interviews and held the thematic analysis session. After the thematic analysis session (based on the responses of three participants), I had multiple meetings with the principal researcher to discuss the results of the thematic analysis.

In this section, I described how we designed the interview questions for our study. In the next section, I elaborate on the behavioral model and how we used it in software security.

5.2.2 Behavioral Model

We designed the interviews using a behavioral model called the COM-B model [42] to investigate the behavioral aspects of developers' security adoption. The COM-B model suggests any behavior is comprised of three components; capability, opportunity, and motivation. Capability means that in order for someone to perform a behavior they must feel physiologically and physically able to do so. Opportunity means to possess the social and physical opportunity of the behavior. Lastly, moti-

vation is the need to perform the behavior more than other competing behaviors.

The COM-B model is a part of a behavioral framework called the Behavior Change Wheel (BCW) [42]. BCW has three main parts: source of behavior, intervention functions, and policy categories. The COM-B model is the center of this framework, the source of behavior, which we used in our study to investigate developers' security adoption. We used this behavioral framework because it is the first systematic analysis of behavior intervention frameworks that apply usefulness criteria, having been constructed from 19 existing frameworks to overcome their limitations [42].

Each of the components in the COM-B model has two sub-categories:

- **Capability-Psychological**: Our knowledge strength, skills or stamina.
- **Capability-Physical**: Our physical strength, skill or stamina.
- **Opportunity-Social**: Opportunities as a result of social factors, such as cultural norms and social cues.
- **Opportunity-Physical**: Opportunities provided by the environment, such as time, location and resource.
- **Motivation-Reflective**: Reflective processes, such as making plans and evaluating things that have already happened.
- **Motivation-Automatic**: Automatic processes, such as our desires, impulses and inhibitions.

To design our cross-sectional interview study, we interpreted the physical term mentioned in the subcategories as technical, non-technical, or social so it would be coherent in the context of software security. To elaborate, the physical ability to perform a task in software engineering is to have the technical knowledge to execute it. Also, software engineering is a group effort and softs skills are essential in order to perform tasks. Our new set of subcategories in the context of software security are: Capabilities-Psychological, Capabilities-Technical, Capabilities-Non-Technical, Opportunities-Technical, Opportunities-Social, Motivations-Reflective, and Motivations-Automatic. In Table 5.1, we list all the subcategories that we considered in our study.

Table 5.1: COM-B model components used in this study.

Capabilities	Technical	Having the technical skills to perform security practices
	Non-Technical	Having the non-technical skills to perform security practices
	Psychological	Having the knowledge of the potential negative consequences of not adopting security practices
Opportunities	Technical	Having the right tools, technologies to perform security practices (Opportunities afforded by the environment involving time, resources, locations, etc.)
	Social	Having people around to facilitate performing security practices (opportunities afforded by interpersonal influences, social and cultural norms, everything that influence the way developers think)
Motivations	Reflective	Self conscious intentions, reflective processes involving plans and evaluations (beliefs about what is good and bad)
	Automatic	Automatic processes involving emotional reactions, desires (wants and needs)

5.2.3 Ethical Considerations

As a part of the design process of the interviews, we submitted a human research ethics board application[3] seeking the approval of our research protocol. This included considerations regarding participant anonymity, data storage and disposal, and publication of the findings. The reference number of the approved ethics application is 21-0122. I submitted the email invitation script for semi-structured interviews (Appendices B), an implied consent form, and a copy of the semi-structured interview questions. This ethics package for the semi-structured interviews is attached in Appendices B and C.

5.2.4 Research Team

This research study was a collaborative effort with four members of the CHISEL research group[4] performing the following steps: conducting the interviews, creating the master code book, coding interviews' transcripts, and data analysis. All of the steps were a team effort and the roles of the members are as follows:

- **Soroush Yousefi** - Researcher, Masters student. My contribution to this research was limited to designing the semi-structured interviews, performing the

[3]https://www.uvic.ca/userais/index.php
[4]https://thechiselgroup.org/team/

interviews with the principal researcher, and finally analyzing and coding the transcripts of two interviews.

- **Dr. Enrique Larios Vargas** - Principal researcher, Postdoctoral fellow. He was the main contributor of this research project. He led all the steps from designing the project to analyzing the interviews and assisted me in synthesizing the results.
- **Omar Elazhary** - Researcher, PhD student. He was involved in the analysis phase. He coded a number of transcribed interviews and held the thematic analysis sessions with the principal researcher.
- **Dr. Margaret-Anne Storey** - Masters supervisor, Professor. She was involved in all phases of the research project as my supervisor and acted as a consulting researcher who helped make key decisions during the research process.
- **CHISEL Group Members** - Several other individuals were both formally and informally involved in the study. Formally as the pilot group for the semi-structured interviews, and informally through discussions about the contents and ideas in this study.

5.3 Preliminary Results

As part of the first thematic analysis session that involved the analysis of the responses from three software developers (as mentioned in Section 5.2), we used the COM-B model as a diagnosis tool to understand what influences software practitioners to adopt software security practices. We categorized the codes that emerged from the open coding of responses from three participants (E1, E9, and E16), based on the three components defined by the COM-B model: *capabilities*, *opportunities*, and *motivations*. This categorization led us to find 15 factors that affect the adoption of software security by software developers, which we group together in six categories based on their similarities: (a) *Building an organizational security culture*, (b) *Building a proactive security mindset*, (c) *Fostering a positive developer perception and attitude towards security*, (d) *Providing cognitive support to developers for writing secure code*, (e) *Providing contextual information to motivate developers to write secure code*, and (f) *Facilitating/Incentivizing the acquisition of developers' security-specific*

skill set. In Table 5.2, we detailed the entire landscape of factors we found from our analysis.

In the following, we describe each of the six categories in detail using the components of the COM-B model associated with them, as well as the role of these factors in the overall adoption of software security practices. These factors were based on analyzing the responses from three participants in our study.

Table 5.2: What needs to change or happen so the adoption of security practices occurs

a. Building an organizational security culture	
F1	Organization promoting/mandating security
F2	Having a security-specific role filled
b. Building a proactive developer security mindset (cost-benefit analysis)	
F3	Awareness of potential risks and security incidents
F4	Learning from actual incidents
F5	Fear of the consequences of not adopting security practices
c. Fostering a positive developer perception and attitude towards security	
F6	Shaping developer's attitude towards security
F7	Awareness of the social perception of security adoption in the organization and professional network (bandwagon)
d. Providing cognitive support to developers for writing secure code	
F8	Reducing the effort required to learn or apply security
e. Providing contextual information to motivate developers to write secure code	
F9	Awareness of the influential role of the industry type in developers' disposition towards security
F10	Awareness of developers' perceptions of the need for software security (based on the application type / characteristics)
f. Facilitating/Incentivizing the acquisition of developers' security-sepecific skill set	
F11	Accessibility to learning resources
F12	Using security practices as learning tools
F13	Providing security education (At the university and trainings at the organization)
F14	Creating and participating in Communities of Practice
F15	Having non-technical skills

5.3.1 Building an organizational security culture

The practitioners we interviewed believe organizational culture plays an essential role in nudging developers to adopt security practices. They told us that organizations that promote the adoption of security in their development workflow often incorporate a specialized role such as security champion in their software development lifecycle.

F1: Organization promoting or mandating security Organizations that are aware of their influential role in software security adopt serious measures to promote

security practices. We use the COM-B model diagnosis tool to highlight the *social opportunities* and *reflective motivations* of the security adoption behavior. In this factor, we found how organizations' social norm and protocols influence developers' attitudes towards security. We also found how these social norms motivate developers to adopt security practices.

Social opportunities: The practitioners we interviewed believe their attitude towards security is influenced by their organizational culture. For instance, E9 highlighted that "*Adopting security requires support from the organization by facilitating the solution of a security issue*". Organizations that support developers in adopting security practices ensure the developers have appropriate amount of time and resource, a security role is present in the company, and the management team is supporting developers to incorporate security tasks in their workflow. However, in some situations, organizations treat security as a secondary concern. E16 emphasized this aspect, stating that "*Making it work is more important in my organization than doing it securely*".

Reflective motivations: Strategies used by organizations to update developers about security guidelines and practices affect the developers' motivation towards security. Developers believe these strategies help practitioners reflect on their security adoption and have a deeper understanding about why security is needed. As E16 reported "*The company is pushing for security. Sending in the monthly newsletter what new security rules have been put in place*".

F2: Having a security-specific role in the organization Security specialists in organizations can promote, maintain, and enforce security practices. Pertaining to the COM-B model, we identified that having a security role provides *social opportunities* and *reflective motivations*. For this factor, *social opportunities* is about how a security specialist creates opportunities for developers to adopt security practices. *Reflective motivations* describes how the presence of a security specialist encourages developers to think about the security implications of their technical decisions.

Social opportunities: All three participants mentioned the importance of having a security role in the organization. An organization that promotes a software security culture designates a specific position for security matters. In addition, developers are willing to adopt software security practices if organizations allow them to shadow security specialists to learn and understand how security issues are handled and patched. For instance, E1 stated "*I will learn from them, see how people fix the problem, how they prioritize the problem, and that's one thing the company is really*

good at, letting you see what happens behind the scenes so you can get a good understanding of it. And eventually, that will translate into me working on more and more security-related tickets".

Reflective motivations: Security specialists in organizations can facilitate discussions about security practices, consequences of security issues, and potential threats to a system. Thus, developers will be aware of implications of security issues and, as a result, will be motivated to follow the guidelines for incorporating security practices in their software development workflow. E9 mentioned that "*communication between security team and development team is very direct. Security team exposes the consequences or issues we will get by deploying a new feature*". Also, having experience with security incidents can affect developers' alertness regarding security issues. For instance, E16 said "*On the operational side, people have been exposed to sort of the effect of security incidents, so we tend to think more about security than developers*".

5.3.2 Building a proactive security mindset

Developers perceive that bringing awareness about security risks, providing information about the consequences of not adopting security practices, and discussing the benefits of building security into the development pipeline will incentivize developers to adopt security. Furthermore, being aware of the trade-offs of adopting software security practices will help developers maintain a positive attitude towards security and always look out for security defects in their products.

F3: Awareness of potential risks and security incidents Being aware of existing threats and vulnerabilities that software products face is one of the critical motivations for developers to adopt security practices. In addition, by observing what risks other organizations face, developers can identify and prioritize their applications' security road map. Here we observe how awareness of potential security risks elevates *Psychological/Technical/Non-Technical capabilities* and improves *Reflective/Automatic motivations*.

Psychological capabilities: Developers stay up to date with security threats and incidents by reading news and articles on the internet. They believe this is a practical way to learn about security and mitigate the threats that could possibly harm their organization. For instance, E1 stated "*Just digging around, seeing what information I could learn about on the Internet and see what was going on. You read many security news articles about people getting hacked, and I was just curious to see*

how they did it".

Technical capabilities: One of the essential skills for addressing security issues is having the technical knowledge and skill to mitigate them. From to our analysis, participants were familiar with the tools and technology required for implementing security in their workflow. However, E9 and E16 noted that they were unaware of security standards. This reveals that some developers lack deep understanding of the required technical skills to implement security. To improve our technical knowledge on security, it is helpful to learn from others' experiences handling security issues. For instance, E16 mentioned how knowing about security incidents helps practitioners apply security practices: "*Learning from security tech news make me competent for conducting security code reviews*".

Non-Technical capabilities: Our three participants indicated that communication is the most important non-technical skill to bring awareness about potential risks of not adopting security practices. A critical task for developers is to communicate with customers to explain the consequences of ignoring security. For instance, E1 stated "*If you can't explain why you want to do something for security reasons, they probably won't want to do it, if you can't demonstrate that to the client. Usually, you tell them that, yeah, you could lose money if we don't fix this, and that sort of communication is better than just explaining to them that they could be hacked or something like that*".

Reflective motivations: One effective way to keep practitioners motivated towards security adoption is to let them know about the consequences of security defects in their products. For example, if they are aware that a security breach could expose confidential information about their users and damage their reputation, they will most likely be incentivized to adopt security. E16 said "*If your services get compromised, and you cannot meet your customers' goals at all, then you all lose. It's a cost you have to balance on the business level*".

Automatic motivations: One factor that influences practitioners' motivation towards adopting security is to feel empathy for their customers by default. If they see another company undergo a security breach that causes drastic effects on their customers, they will try to avoid that happening to more people. In other words, an excellent motivation for developers to adopt software security practices is to know what might happen to their customers if they do not follow security practices. When security is adopted, it provides tremendous satisfaction to developers making their customers safe. For example, E1 said "*Adopting security, make yourself feel happy,*

make yourself feel safe. If the software is secure, then you make your customer happy too".

F4: Learning from actual incidents A great learning resource for practitioners is to study security exploits that happened in the past. Investigating how they occurred, what the attackers were aiming for, and finding mitigation for the vulnerability will build more confidence for the practitioners about the state of their product and keep them engaged in actively incorporating security into their software development workflow.

Psychological capabilities: To perform security tasks, as highlighted by the COM-B model, practitioners should feel confident about executing them. Developers can gain this confidence by examining security incidents and breaches and understanding why the incidents happened in the first place. By knowing the reason and studying a way to mitigate security breaches, developers will be more confident in assessing their product's security. For instance, E16 mentioned that "*Academic knowledge can get you close in ensuring a secure software but having experience or having someone on your team who has experienced that will allow you to have more confidence to say you have a secure product*".

Reflective motivations: Maintaining motivation towards adopting software security practices is a complicated task. However, learning about security incidents empower developers to reflect on their practices and be motivated to adopt more secure practices. Thus, they become aware that the chances of having a security breach are not small if they do not incorporate security into their product. Our study shows so far that to build a secure product, developers need experience in security, which is acquired by being involved in security incidents. Accordingly, E16 shared how experience with security incidents can maintain practitioners' alertness about security issues: "*How can we prevent this from happening again? Restoring that sense of confidence in the service that we sold them was one of my first experiences. Experience is one of the things that I want to help build a secure product. So, you gain experience by, unfortunately, being involved in security incidents*".

F5: Fear of the consequences of not adopting security practices

Psychological capabilities: The developers we interviewed acknowledged that the negative consequences of not adopting security in software development is an influential factor in adopting software security practices. Here we discuss the capabilities and the motivations that fear of consequences of not adopting security practices can

bring.

Psychological capabilities: The developers we interviewed believe that being aware of the dire consequences of not adopting security in the development workflow influences their mindset about adopting security. They mentioned three disadvantages of not adopting security in the following way: spending a lot of time on fixing vulnerabilities, losing money by allocating resources to patch and reinforce security in the software product, and losing reputation which could result in the loss of customers, and severe financial issues in the organization. E1 highlighted this issue: "*The negative consequences of not adopting security is that you can lose time, money and potentially your company or your reputation*".

Reflective motivations: The negative consequences of ignoring software security practices make developers consider to what extent their software product is "*secure enough*" and this motivates them to minimize any possibility of exposure to exploits. For instance, E16 expressed his concern "*you don't know if you are secure enough until you get hacked and find out that you aren't; otherwise, you are. But there might potentially be a false positive state where you feel secure*".

Automatic motivations: One of the main motives for practitioners thinking about security in their software development workflow is to avoid experiencing a security incident. However, the fear of exposing confidential information is always a good reminder for developers to be careful about security implementations. For instance, E1 emphasized that not taking into account security features exposes their product to malicious attacks: "*The lack of security features leaves ourselves open and vulnerable*".

5.3.3 Fostering a positive developer attitude towards security

Security is not a topic that most developers prioritize. They primarily focus on delivering features quickly rather than working on non-functional requirements, such as security. Developers also tend to not see the immediate benefit of adopting security practices but instead focus on the negative effects of introducing security practices that affect their workflows, such as delaying their development pipeline. In this section, we describe how organizations promote a more positive attitude towards adopting security practices.

F6: Shaping developers' attitudes towards security Organizational security

culture has a direct effect on developers' attitudes towards security. Developers believe that organizations that provide both the right opportunities and motivate practitioners make it effortless for them to adopt software security practices

Social opportunities: Organizations have a crucial role in fostering opportunities to shape developers' attitudes towards security. Organizations that involve developers in fixing security incidents influence a more positive attitude towards security. Practitioners perceive this as an opportunity to learn new technical skills as part of their regular development work. For example, E1 said "*That's one of the things the company is good at, letting you see what happens behind the scenes so you can get a good understanding of it. And eventually, that will translate into me working on more and more security-related tickets*".

Automatic motivations: Developers working in organizations that promote software security practices tend to perceive security as challenging but rewarding work. In addition, developers with a positive attitude towards security feel highly motivated to adopt security practices to protect customers and the company. For instance, E1 highlighted "*I think of security as a chess game. I play one side; the attackers play the other side. It can be quite challenging, it's hard work, but it's rewarding in the end, so my motivation is to protect people and protect the company.*".

F7: Awareness of the social perception of security adoption in the organization and professional network Developers believe collaboration between all teams in the organization is necessary for adopting security practices; most importantly between the management and engineering teams. Therefore, being aware of the overall state of security adoption in a company highly impacts developers' willingness to adopt security practices.

Reflective motivations: The overall adoption of security practices within the organization significantly influences developers' perception of software security. Not having people within the organization adopting security will have considerable negative effects on developers' motivation towards adopting security. For instance, E16 pointed out "*if nobody else takes it seriously, I'll never take it seriously. If it's not part of the culture, if it's just one guy saying security, security, security, then people will do the bare minimum to adopt security, which might be better than nothing. Still, it needs to be a part of everyone. Everybody has to care about it for you to feel like you're making secure software*".

5.3.4 Providing cognitive support to developers for writing secure code

Practitioners find it challenging to incorporate security practices into their software development workflow. This challenge might be due to a perception that security requires a lot of knowledge about various technologies and tools in order to be done correctly. Based on our analysis, I describe how developers' motivation towards adopting security practices can be increased by reducing their cognitive load.

F8: Reducing the effort required to learn or apply security We learned so far that some developers might lack the knowledge and experience to implement security properly. To tackle this challenge, they might face an overwhelming amount of learning resources and tools to facilitate the adoption of security practices. Based on the COM-B model, developers' automatic motivations are significantly improved if they are given the right resources.

Automatic motivations: Developers perceive that adopting software security practices demands a lot of cognitive effort. This effort can be a result of investing a significant amount of time on learning all the different techniques and staying up to date with the latest security news. These challenges highly influence developers' motivations to adopt security practices. For instance, E1 mentioned that "*Adopting security just requires a lot of knowledge. So sometimes companies don't have that time. You just need time to learn it, so, yeah, it's just the amount of knowledge you need is that slows down the adoption of security*".

5.3.5 Providing contextual information to motivate developers to write secure code

Developers recognize that an awareness of contextual aspects of their organization, such as industry type and application characteristics, influences their perception of security practices.

F9: Awareness of the influential role of the industry type in developers' disposition towards security The participants we interviewed noticed how some industries, have a level of security that is not as high as in others. If practitioners don't deal with sensitive information they might think spending resources and time on security is unnecessary.

Social opportunities: Some industries, such as gaming industry, tend to neglect

the adoption of security practices at the application level. Instead, their primary security-related concerns are at the infrastructure level. As a result, developers in this type of industry do not feel the need to adopt security in their development workflow because it is abstracted by the infrastructure. For example, E1 highlighted "*Security is not so much the game studios problem, it's the internet part of that*".

Reflective motivations: In contrast to the gaming industry, some industries are better known for their security requirements. For instance, developers that work with customers' payment information are more likely to be aware of the security implications of their product. Therefore, they feel the need to adopt software security practices. For example, E9 said "*Major customers are coming from the financial services, so security concern is very high*". Additionally, E16 points out "*In the video game industry, security is not taken seriously. Some industries require like PCI compliance but not for video game industry*".

F10: Awareness of developers' perceptions of the need for software security (based on the application type/characteristics) We learned that the application type remarkably influences practitioners' perceptions regarding the need to adopt software security practices. Based on the COM-B model, we identify developers' reflective motivations that are affected by this factor.

Reflective motivations: Web applications, mobile applications, and embedded systems have different characteristics. Therefore, there are multiple security considerations for each of them. Developers may perceive that adopting security for certain types of applications is unnecessary due to the reduced likelihood of being exposed to potential risks. For instance, developers, who write code for embedded systems feel security is not a significant issue since they may not be exposed to malicious users. E16 pointed out "*Developers who work in embedded systems, writing for a system on a chip, they don't necessarily care about security*".

5.3.6 Facilitating/incentivizing the acquisition of developers' security-specific skill set

The ability to perform security-related tasks does not necessarily depend on technical skills. Practitioners perceive that having support from their organizations helps them acquire the necessary technical and non-technical skills to understand security challenges. Organizations need to provide developers with the right tools and learning resources to facilitate the adoption of software security practices.

F11: Accessibility to learning resources There is a huge diversity of learning resources to boost developers' technical skills in security. For example, developers might access online courses, conferences, or have security specialists or communities of practice to address security concerns, which are excellent ways to learn about security.

Technical opportunities: The practitioners we interviewed believe they can elevate their technical skills in security by benefiting from many learning resources provided by organizations. An organization that provides developers with courses, training, and conferences fosters an environment for security learning. E9 elaborated how their company helps developers by providing learning resources that could potentially increase their technical knowledge: "*It's a requirement for every employee in a company to take the security course. To take the security course every year, which is mainly about what kind of attack your system is going to face, what technique they will use to make those attacks, and how you could prevent those attacks*".

F12: Using security practices as learning tools Being involved in security practices within the organization is perceived by developers as an effective way to learn about security. The COM-B model highlights psychological capabilities and social opportunities in this context.

Psychological capabilities: The developers we interviewed perceive that they gain more confidence by working on security practices. By experiencing how to handle a real security issue, developers build confidence in their capabilities to implement security practices. For instance, E1 thought security testing is a great way to learn and gain confidence to understand security issues over time: "*Applying security testing, that's half development. I guess I got some things to learn, but I have been doing security testing for quite a bit*".

Social opportunities: Organizations play an essential role in providing developers with experiential learning resources using security practices. Practices such as code review and pair programming may help practitioners have a deeper understanding of security issues. In addition, organizations that foster knowledge sharing bring more opportunities to their developers to adopt security. E1 highlighted "*Usually, the software architect will tag me in security issues that I'm not necessarily involved in, but I will review them and learn from them. Seeing how people fix the problem and prioritize the issue will eventually translate into me working on more and more security-related tickets*".

F13: Providing security education (training at the organization) Providing continuous learning and training for developers to keep up to date with security practices is an essential part of an organizations' security culture. Using the COM-B model, our analysis identifies social opportunities and reflective motivations.

Social opportunities: Organizations that provide workshops, courses, and training for developers proactively remind them how and why they should adopt security. Security is all about staying abreast of the latest practices and methods to avoid vulnerabilities that could potentially harm the organization. Organizations are lowering the chance of having a security breach by facilitating security training. For example, E9 mentioned the importance of having workshops: "*I would like to have some workshops. To show me, given this piece of code, how would a hacker attack it?*".

Reflective motivations: Security training has a positive motivational effect on developers. Being aware of the most common mistakes developers make in software development in terms of security allows them to reflect on their practices and learn how to avoid introducing security flaws into their code. For instance, E1 said "*Being familiar with the most common mistakes people make in terms of security, that's just something I enjoy and I study*".

F14: Creating and participating in Communities of Practice Not only are communities a great way to learn about a topic, they are also a practical way to find answers to unresolved questions. Organizations that cultivate communities inside the company and support practitioners to reach out to external communities make it easy for developers to adopt security.

Social opportunities: Developers use external communities to get help on security issues. Communities such as OWASP provide an excellent environment for practitioners to learn about security. Similarly, developers perceive that organizations that form communities inside the company with practitioners who show an interest security help build a supportive environment to facilitate learning. E9 mentioned how having a community can help practitioners adopt software security practices and gain experience: "*To gather knowledge and experience is important that organizations promote knowledge sharing sessions with the development team or collaboratively with the security team*".

F15: Having non-technical skills All three participants in our study mentioned the importance of soft skills (non-technical skills) in security adoption. For instance, it is essential to know how and what to communicate to the right stakeholders in the

organization when a security incident occurs.

Non-technical capabilities: All the participants (E1, E9, and E16) shared that communication is the most important non-technical skill. They believe communication is essential when describing the importance of security to other stakeholders, discussing security issues with developers, and handling security incidents. When an incident happens, developers need to clearly describe the situation to management and other stakeholders in order to determine further actions. For example, E1 pointed out "*Communication is essential to explain why we need to change something for security reasons*". These participants also mentioned the importance of other critical non-technical skills, such as collaboration, research, empathy, and curiosity.

5.4 Discussion

In this section, we discuss the implications of our work to practitioners and researchers.

5.4.1 Implications for Practitioners

Security in software development is a socio-technical activity. Organizations not only should consider the technical capabilities that developers need to implement security practices, they should also pay close attention to motivating developers towards adopting security and providing opportunities for practitioners so that they can learn and adopt security practices. For the adoption of security to occur, our work shows a comprehensive list of 15 factors that represent what needs to change or happen (Table 5.2). We believe these factors are the foundation for designing strategies and interventions in organizations to bolster the adoption of security. Therefore, we recommend practitioners to follow these suggestions:

- Organizations can examine the relevance of each factor mentioned in Table 5.2 according to their own needs. To find the relevant factors, organizations can conduct semi-structured interviews or focus groups with all the members of the company that are affected by security, from development to management teams. If it is not possible for organizations to conduct interviews and focus groups, they can also conduct surveys with security stakeholders. In the survey, they can ask employees to express their level of agreement or disagreement for each factor.

- Having a dedicated security specialist or a security team was a shared opinion from the participants that could possibly help increase the adoption of security practices. We identified two paths for organizations to have a dedicated security team. One is to hire security specialists from outside the company and the second is to train the developers who are interested in taking on this role. One of the main differences with these two options, which is a concern for many companies, is the cost. On one hand, startups and small businesses might not have the budget to invest in an external security hire. Instead, they can train developers who are already interested in this role to become security champions. On the other hand, big companies with enough budget can invest more in their security concerns by hiring external security specialists while also training their developers.

- From the interviews, we noticed the importance of staying up to date with the latest security incidents, news, and practices. To keep developers motivated to stay up to date, organizations should incorporate strategies in their company that continuously remind people about security. A few examples that can be implemented are: (a) sending developers regular newsletters about the latest security incidents, practices, and disclosed vulnerabilities; (b) providing developers a security checklist to follow when performing code reviews; (c) conducting random security tests with the employees (e.g., mock phishing tests) to remind and teach them that anyone can be targeted; (d) providing learning resources and security tools to reduce cognitive load; and (e) creating communities inside the company to encourage developers to discuss security topics and learn from each other.

5.4.2 Implications for Researchers

The factors we identified in our study are part of the missing step in the literature we analyzed as part of the mapping study. These factors describe how and what needs to be done so that developers adopt security practices. Finding these factors using a behavioral model led us to believe more future research can be done using behavior change techniques to understand developer behaviors in different domains.

By comparing the results of this study to the papers I analyzed in the mapping study in Chapter 4, I found that many factors that were not mentioned in the literature can affect developer's security adoption. In the mapping study, I saw papers

investigated how the design of different cryptography APIs can affect security [1] [27] and a study on how a new tool can detect vulnerabilities with lower false positive rate [49]. However, I did not find any studies about how organizations should support developers in security tasks. I suggest researchers consider analyzing software engineering challenges under the lens of behavioral science theories

In Chapter 2 (background and related work), I mentioned a few papers studying social aspects of security, such as investigating developer's motivations and deterrents for adopting security practices [5]. Understanding the motivations and deterrents might be a good starting point but we have to know how to mitigate impediments in order to achieve higher security adoption. This is why we designed a study using the COM-B model from psychology to understand behaviors and also help us find ways to improve the current state of security practice adoption.

For researchers who are aiming to conduct further studies using behavioral science theories, I discuss our experience using the COM-B model. The COM-B model helped us draw a better picture of social aspects of software security. It gave us a lens, in the context of security adoption, to look at areas that can be developed (opportunities), discover potential abilities to grow (capabilities), and understand developers' motives (motivation). In the beginning of this study, we aimed to understand security from the developer's perspective, and using this model helped us understand what we can change to increase security adoption. For instance, if an organization lacks a dedicated security role to promote and implement best practices, it is unlikely their developers will embrace security. For identifying capabilities, if the organization decides to invest in hiring dedicated security roles but does not have the budget they could train their developers to take on the role. Overall, this model did not only help us find the barriers for adopting security but also showed how to overcome them. For this reason, I recommend using this model for understanding developer behaviors in research studies.

5.5 Threats to Validity

As this is an empirical study that relies heavily on qualitative data gathered from practitioners; there are limitations on how this study can be applied and understood in the computer science research domain.

During the recruitment phase for the semi-structured interviews, we explicitly mentioned to participants that we were interested in how practitioners adopt security.

Subsequently, we informed them we were aiming to investigate practitioners' thought processes when handling security issues. This information may have influenced who participated in the semi-structured interviews. Therefore, there is a chance those practitioners who were not interested in security or did not adopt security were not interested in participating in our study.

The set of factors we gathered in our study to show different aspects impacting security adoption was based on the recruited participants. We aimed to have practitioners that represent all industries. However, due to limitations in finding participants with the intended diversity, we could not include all industry types. More factors could emerge if participants from other industries were involved in the study. For instance, we did not include open source developers in our study. Future researchers could investigate whether practitioners in the open source community consider the factors useful or not.

To avoid bias in our study, two researchers iteratively developed the factors and asked an expert to review them. Moreover, to create our master codebook, we held agreement sessions based on coding the transcripts. Two researchers in our research group performed the agreement sessions. After reaching 80% agreement in the agreement sessions, each researcher coded the rest of the transcribed interviews.

5.6 Conclusion and Future Work

Studying developers' security adoption from a behavioral perspective has not been thoroughly investigated. Despite all the work on secure software development, we wanted to understand why we still have many software exploit incidents every year. We designed a cross-sectional interview study using the COM-B model from psychology to gain insights on developers' behaviors towards adopting security. Our results highlight 15 different factors that represent what needs to change or happen in order to adopt software security practice. Each factor is associated with one of the three components of the COM-B model (capabilities, opportunities, and motivations). Our study will help organizations design strategies to nudge developers to adopt security practices.

Since there have only been a few studies investigating the social aspects of adopting security, we will continue with our analysis to produce a more holistic picture of security adoption in software development from a developer's perspective. With our research, we aim to bring insights on the issues of software security adoption and

suggest venues on how a behavioral science approach could help organizations solve them.

Chapter 6

Conclusions

Security is a critical non-functional requirement that remains to be adequately addressed in software development. Issues in software security have increased and become more complicated with the evolution of software applications. Also, building secure software products is not only about having the technical knowledge to incorporate security it is also about understanding how practitioners adopt security practices in their development process.

To understand the current state of security issues in software applications, I performed a systematic mapping study. I analyzed papers that studied security issues using projects hosted on GitHub and identified nine topics in the literature. To find gaps in the literature, I analyzed the papers with a socio-technical framework. With this analysis, I observed that not many papers studied how developers adopt security practices or how they benefit from security tools. For instance, papers introduced new security tools to benefit developers; however, the tools were not examined by developers to see if they are useful.

My mapping study showed that developer security adoption needs more investigation. With this, I collaborated with a research team at the University of Victoria to understand software developer behaviors and motivations towards security. We designed a cross-sectional interview study using a behavioral model from psychology and interviewed software developers. Our results highlight 15 different factors that point out what needs to change or happen that improves the adoption of software security practices. We found that besides technical capabilities, developers need support from their organization, peers, and managers to be encouraged to adopt more secure practices. Developers are more likely to adopt security when they have a mentor for security tasks to guide them on how to incorporate security practices into their

workflow.

In summary, I aimed to depict a clearer picture of security issues in software applications and investigate developers' perceptions towards security. I also hope my study helps organizations better understand their security issues and help them design new strategies to mitigate them.

Bibliography

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. Comparing the usability of cryptographic apis. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154–171, 2017.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, 2016.

[3] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L. Mazurek, and Sascha Fahl. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 81–95, Santa Clara, CA, July 2017. USENIX Association.

[4] Apache. Apache solr. `https://lucene.apache.org/solr/`, 2021. [Online; accessed 2021].

[5] Hala Assal and Sonia Chiasson. Think secure from the beginning: A survey with software developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Wei Bai, Omer Akgul, and Michelle L. Mazurek. A qualitative investigation of insecure code propagation from online forums. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 34–48, 2019.

[7] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I Hong, and Lorrie Faith Cranor. The privacy and security behaviors of smartphone app developers. In Workshop on Usable Security (USEC'14), 2014.

[8] Harold Booth, Doug Rike, Gregory A Witte, et al. The national vulnerability database (nvd): Overview. `Availableonline:https://nvd.nist.gov/`, 2013. [Online; accessed 2021].

[9] Alberto Calvi and Luca Viganò. An automated approach for testing the security of web applications against chained attacks. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, page 2095–2102, New York, NY, USA, 2016. Association for Computing Machinery.

[10] Brandon Carlson, Kevin Leach, Darko Marinov, Meiyappan Nagappan, and Atul Prakash. Open source vulnerability notification. In Francis Bordeleau, Alberto Sillitti, Paulo Meirelles, and Valentina Lenarduzzi, editors, *Open Source Systems*, pages 12–23, Cham, 2019. Springer International Publishing.

[11] Jeffrey C Carver, Morgan Burcham, Sedef Akinli Kocak, Ayse Bener, Michael Felderer, Matthias Gander, Jason King, Jouni Markkula, Markku Oivo, Clemens Sauerwein, et al. Establishing a baseline for measuring advancement in the science of security: an analysis of the 2015 ieee security & privacy proceedings. In *Proceedings of the Symposium and Bootcamp on the Science of Security*, pages 38–51, 2016.

[12] Pravir Chandra et al. Software assurance maturity model. `https://www.opensamm.org/`, 2008. [Online; accessed 2021].

[13] Bodin Chinthanet, Raula Gaikovina Kula, Takashi Ishio, Akinori Ihara, and Kenichi Matsumoto. On the lag of library vulnerability updates: An investigation into the repackage and delivery of security fixes within the npm javascript ecosystem. *CoRR*, abs/1907.03407, 2019.

[14] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 181–191, 2018.

[15] Ruian Duan. *TOWARD SOLVING THE SECURITY RISKS OF OPEN-SOURCE SOFTWARE USE*. PhD thesis, Georgia Institute of Technology, 2019.

[16] Saniora R Duclervil and Jing-Chiou Liou. The study of the effectiveness of the secure software development life-cycle models in it project management. In *16th*

International Conference on Information Technology-New Generations (ITNG 2019), pages 91–96. Springer, 2019.

[17] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 422–431. IEEE, 2013.

[18] Elastic. Elasticsearch. `https://www.elastic.co/elasticsearch/`, 2021. [Online; accessed 2021].

[19] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136. IEEE, 2017.

[20] Graham R Gibbs. Thematic coding and categorizing. volume 703, pages 38–56. Sage London, England, 2007.

[21] Antonios Gkortzis, Daniel Feitosa, and Diomidis Spinellis. Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities. In *Journal of Systems and Software*, page 110653. Elsevier, 2020.

[22] Antonios Gkortzis, Dimitris Mitropoulos, and Diomidis Spinellis. Vulinoss: A dataset of security vulnerabilities in open-source systems. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 18–21, New York, NY, USA, 2018. Association for Computing Machinery.

[23] Barney G Glaser and Anselm L Strauss. *Discovery of grounded theory: Strategies for qualitative research*. New York. Aldine Publishing. 1967. x, 271 p.

[24] Qingyuan Gong, Jiayun Zhang, Yang Chen, Qi Li, Yu Xiao, Xin Wang, and Pan Hui. Detecting malicious accounts in online developer communities using deep learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1251–1260, 2019.

[25] Google. Bigquery public datasets — google cloud. `https://cloud.google.com/bigquery/public-data/`, 2021. [Online; accessed 2021].

[26] Mohammadreza Hazhirpasand, Mohammad Ghafari, Stefan Krüger, Eric Bodden, and Oscar Nierstrasz. The impact of developer experience in using java cryptography. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–6. IEEE, 2019.

[27] Mohammadreza Hazhirpasand, Mohammad Ghafari, and Oscar Nierstrasz. Cryptoexplorer: An interactive web platform supporting secure use of cryptography apis. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 632–636. IEEE, 2020.

[28] Sameera Horawalavithana, Abhishek Bhattacharjee, Renhao Liu, Nazim Choudhury, Lawrence O. Hall, and Adriana Iamnitchi. Mentions of security vulnerabilities on reddit, twitter and github. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 200–207, 2019.

[29] Michael Howard and Steve Lipner. *The security development lifecycle*, volume 8. Microsoft Press Redmond, 2006.

[30] Luigi Lo Iacono, Matthew Smith, Emanuel von Zezschwitz, Peter Leo Gorski, and Peter Nehren. Consolidating principles and patterns for human-centred usable security research and development. In *European Workshop on Usable Security, London*, 2018.

[31] Daljit Kaur and Parminder Kaur. Software development life cycle security issues. In *AIP Conference Proceedings*, volume 1414, pages 237–239. American Institute of Physics, 2011.

[32] Raja Khaim, Saba Naz, Fakhar Abbas, Naila Iqbal, Memoona Hamayun, and Rawalpindi Pakistan. A review of security integration technique in agile software development. *Int. J. Softw. Eng. Appl*, 7(3):49–68, 2016.

[33] Seulbae Kim and Heejo Lee. Software systems at risk: An empirical study of cloned vulnerabilities in practice. In *Computers & Security*, volume 77, pages 720–736. Elsevier, 2018.

[34] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. In *Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University.*, 2007.

[35] S Kowalski. It insecurity: A multi-disciplinary inquiry. diss. the royal institute of technology. *Department of Computer and Systems Science Stockholm Univ. Report series*, (94-040), 1994.

[36] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. Do developers update their library dependencies? *Empirical Software Engineering*, 23(1):384–417, 2018.

[37] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. *In 24th Annual Network and Distributed System Security Symposium, NDSS '17'*, February 2017.

[38] Junho Lee, Jungwoong Woo, Cheongan Lee, and Kyungsoo Joo. A software development methodology for secure web application. *International Journal on Advanced Science, Engineering and Information Technology*, 9:336–341, 2019.

[39] Markus Lennartsson, Joakim Kävrestad, and Marcus Nohlberg. Exploring the meaning of usable security–a literature review. *Information & Computer Security*, 2021.

[40] Bingchang Liu, Guozhu Meng, Wei Zou, Qi Gong, Feng Li, Min Lin, Dandan Sun, Wei Huo, and Chao Zhang. A large-scale empirical study on vulnerability distribution within projects and the lessons learned. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1547–1559, 2020.

[41] Michael Meli, Matthew R McNiece, and Bradley Reaves. How bad can it git? characterizing secret leakage in public github repositories. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium, (NDSS'19)*, 2019.

[42] Susan Michie, Maartje M Van Stralen, and Robert West. The behaviour change wheel: a new method for characterising and designing behaviour change interventions. *Implementation science*, 6(1):1–12, 2011.

[43] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *2017*

32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 84–94. IEEE, 2017.

[44] Joseph P Near and Daniel Jackson. Finding security bugs in web applications using a catalog of access control patterns. In *Proceedings of the 38th International Conference on Software Engineering*, pages 947–958. ACM, 2016.

[45] NVD. National vulnerability database. `https://nvd.nist.gov/`, 2021. [Online; accessed 2021].

[46] Ugochi Oluwatosin Nwokedi, Beverly Amunga Onyimbo, and Babak Bashari Rad. Usability and security in user interface design: a systematic literature review. *International Journal of Information Technology and Computer Science (IJITCS)*, 8(5):72–80, 2016.

[47] David N Palacio, Daniel McCrystal, Kevin Moran, Carlos Bernal-Cárdenas, Denys Poshyvanyk, and Chris Shenefiel. Learning to identify security-related issues using convolutional neural networks. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 140–144. IEEE, 2019.

[48] Ivan Pashchenko. *Decision Support of Security Assessment of Software Vulnerabilities in Industrial Practice*. PhD thesis, University of Trento, 2019.

[49] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin 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*, pages 426–437, 2015.

[50] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[51] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: Sentiment analysis of security discussions on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 348–351, New York, NY, USA, 2014. Association for Computing Machinery.

[52] Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric Dangremont. A manually-curated dataset of fixes to vulnerabilities of open-source software. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 383–387. IEEE, 2019.

[53] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering*, 47(3):560–581, 2021.

[54] Akond Rahman, Chris Parnin, and Laurie Williams. The seven sins: security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 164–175. IEEE, 2019.

[55] Md Rayhanur Rahman, Akond Rahman, and Laurie Williams. Share, but be aware: Security smells in python gists. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 536–540. IEEE, 2019.

[56] Irum Rauf, Marian Petre, Thein Tun, Tamara Lopez, Paul Lunn, Dirk Van der Linden, John Towse, Helen Sharp, Mark Levine, Awais Rashid, et al. The case for adaptive security interventions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(1):1–52, 2021.

[57] Klaus Reche Riisom, Martin Slusarczyk Hubel, Hasan Mousa Alradhi, Niels Bonde Nielsen, Kati Kuusinen, and Ronald Jabangwe. Software security in agile software development: A literature review of challenges and solutions. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pages 1–5, 2018.

[58] Lavanya Sajwan, James Noble, Craig Anslow, and Robert Biddle. Why do programmers do what they do? a theory of influences on security practices. *Workshop on Usable Security and Privacy*, May 2021.

[59] Vibha Singhal Sinha, Diptikalyan Saha, Pankaj Dhoolia, Rohan Padhye, and Senthil Mani. Detecting and mitigating secret-key leaks in source code repositories. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 396–400. IEEE, 2015.

[60] Margaret-Anne Storey, Neil A Ernst, Courtney Williams, and Eirini Kalliamvakou. The who, what, how of software engineering research: a socio-technical framework. *Empirical Software Engineering*, 25(5):4097–4129, 2020.

[61] Mohammad Tahaei and Kami Vaniea. A survey on developer-centred security. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 129–138. IEEE, 2019.

[62] Alex Tompkins, Josh Bernasconi, Andrew Davidson, and James Toohey. On security vulnerabilities stemming from the usage of open-source dependencies. *SENG401 Conference, ACM ISBN*, 2019.

[63] Rijnard van Tonder and Claire Le Goues. Defending against the attack of the micro-clones. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–4. IEEE, 2016.

[64] Morteza Verdi, Ashkan Sami, Jafar Akhondali, Foutse Khomh, Gias Uddin, and Alireza Karami Motlagh. An empirical study of c++ vulnerabilities in crowd-sourced code examples. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.

[65] Hugo Villamizar, Marcos Kalinowski, Marx Viana, and Daniel Méndez Fernández. A systematic mapping study on security in agile requirements engineering. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 454–461. IEEE, 2018.

[66] Daniel Votipka, Desiree Abrokwa, and Michelle L Mazurek. Building and validating a scale for secure software development self-efficacy. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–20, 2020.

[67] Liang Wang, Paul Grubbs, Jiahui Lu, Vincent Bindschaedler, David Cash, and Thomas Ristenpart. Side-channel attacks on shared search indexes. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 673–692. IEEE, 2017.

[68] Xinda Wang, Kun Sun, Archer Batcheller, and Sushil Jajodia. An empirical study of secret security patch in open source software. In *Adaptive Autonomous Secure Cyber Systems*, pages 269–289. Springer, 2020.

[69] Shao-Fang Wen. Software security in open source development: A systematic literature review. In *2017 21st Conference of Open Innovations Association (FRUCT)*, pages 364–373. IEEE, 2017.

[70] David A. Wheeler. Flawfinder. `https://dwheeler.com/flawfinder/`, 2021. [Online; accessed 2021].

[71] Wikipedia. Zero-day (computing). `https://en.wikipedia.org/wiki/Zero-day_(computing)`, 2021. [Online; accessed 2021].

[72] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10, 2014.

[73] Pei Xia, Makoto Matsushita, Norihiro Yoshida, and Katsuro Inoue. Studying reuse of out-dated third-party code in open source projects. In *Information and Media Technologies*, volume 9, pages 155–161. Information and Media Technologies Editorial Board, 2014.

[74] Mansooreh Zahedi, Muhammad Ali Babar, and Christoph Treude. An empirical study of security issues posted in open source projects. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018.

[75] Jiayuan Zhou, Shaowei Wang, Cor-Paul Bezemer, Ying Zou, and Ahmed E. Hassan. Studying the association between bountysource bounties and the issue-addressing likelihood of github issue reports. *IEEE Transactions on Software Engineering*, 47(12):2919–2933, 2021.

Appendix A

List of Selected Primary Studies

In this section, I present the final set of selected primary studies based on the inclusion and exclusion criteria of the systematic mapping study, including ID, authors, year, and title corresponding to each primary study.

Author(s)	Title	Year	ID	Reference
Tompkins A, Bernasconi J, Davidson A, Toohey J	On security vulnerabilities stemming from the usage of open source dependencies	2019	P1	[62]
Lauinger T, Chaabane A, Arshad S, Robertson W, Wilson C, Kirda E.	Thou shalt not depend onme: Analysing the use of outdated javascript librarieson the web	2018	P2	[37]
Carlson B, Leach K, Marinov D, Nagappan M, Prakash A	Open source vulnerabilitynotification	2019	P4	[10]
lexandre Decan, Tom Mens, and Eleni Constantinou	On the impact of security vulnerabilities in thenpm package dependency network	2018	P5	[14]
Samim Mirhosseini and Chris Parnin	Can automated pull requests encourage software developers to upgrade out-of-date dependencies?	2017	P6	[43]
Kula RG, German DM, Ouni A, Ishio T, Inoue K	Do developers update their library dependencies?	2018	P7	[36]
Liu B, Meng G, Zou W, Gong Q, Li F, Lin M, Sun D, Huo W, Zhang C	Large-Scale Empirical Studyon Vulnerability Distribution within Projects and theLessons Learned	2020	P8	[40]
Antonios Gkortzis, Daniel Feitosa, and DiomidisSpinellis	Software reuse cuts both ways: An empir-ical analysis of its relationship with security vulnera-bilities	2020	P9	[21]
Xia P, Matsushita M, Yoshida N, Inoue K	Studying reuse of out-dated third-partycode in open source projects	2014	P10	[73]
Md Rayhanur Rahman, Akond Rahman, and LaurieWilliams	Share, But be Aware: Security Smells inPython Gists	2019	P11	[55]
Duan R	Toward Solving the Security Risks of open source Software Use	2019	P12	[15]
Kim S, Lee H	Software systems at risk: An empirical study of cloned vulnerabilities in practice	2018	P13	[33]
Wang X, Sun K, Batcheller A, Jajodia S	An Empirical Study of Secret Security Patch in Open Source Software	2020	P14	[68]
Rahman A, Parnin C, Williams L	The seven sins: security smells in infrastructure as code scripts	2019	P15	[54]
Pashchenko I	Decision Support of Security Assessment of Software Vulnerabilities in Industrial Practice	2019	P16	[48]
Perl H, Dechand S, Smith M, Arp D, Yamaguchi F, Rieck K, Fahl S, Acar Y	Vccfinder: Finding potential vulnerabilities in open source projects to assist code audits	2015	P18	[49]
Near JP, Jackson D	Finding security bugs in web applications using a catalog of access control patterns	2016	P19	[44]
van Tonder R, Le Goues C	Defending against the attack of the micro-clones	2016	P20	[63]
Acar Y, Backes M, Fahl S, Garfinkel S, Kim D, Mazurek ML, Stransky C	Comparing the usability of cryptographic apis	2017	P21	[1]
Acar Y, Stransky C, Wermke D, Mazurek ML, Fahl S	Security developer studies with github users: Exploring a convenience sample	2017	P22	[3]
Bai W, Akgul O, Mazurek ML	A Qualitative Investigation of Insecure Code Propagation from Online Forums	2019	P23	[6]

Continued on next page

Author(s)	Title	Year	ID	Reference
Gong Q, Zhang J, Chen Y, Li Q, Xiao Y, Wang X, Hui P	Detecting Malicious Accounts in Online Developer Communities Using Deep Learning	2019	P24	[24]
Horawalavithana S, Bhattacharjee A, Liu R, Choudhury N, O. Hall L, Iamnitchi A	Mentions of Security Vulnerabilities on Reddit, Twitter and GitHub	2019	P25	[28]
Ragkhitwetsagul C, Krinke J, Paixao M, Bianco G, Oliveto R	Toxic code snippets on stack overflow	2019	P26	[53]
Verdi M, Sami A, Akhondali J, Khomh F, Uddin G, Motlagh AK	An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples	2019	P27	[64]
Mansooreh Zahedi, Muhammad Ali Babar, andChristoph Treud	An empirical study of securityissues posted in open source project	2018	P28	[74]
Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik	Security and emotion: sentiment analysisof security discussions on GitHub	2014	P29	[51]
Palacio DN, McCrystal D, Moran K, Bernal-Cárdenas C, Poshyvanyk D, Shenefiel C	Learning to Identify Security-Related Issues Using Convolutional Neural Networks	2019	P30	[47]
Hazhirpasand M, Ghafari M, Nierstrasz O	CryptoExplorer: An Interactive Web Platform Supporting Secure Use of Cryptography APIs	2020	P31	[27]
Sinha VS, Saha D, Dhoolia P, Padhye R, Mani S	Detecting and mitigating secret-key leaks in source code repositories	2015	P32	[59]
Meli M, McNiece MR, Reaves B	How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories	2019	P33	[41]
Calvi A, Viganò L	An automated approach for testing the security of web applications against chained attacks	2016	P34	[9]
Wang L, Grubbs P, Lu J, Bindschaedler V, Cash D, Ristenpart T	Side-channel attacks on shared search indexes	2017	P35	[67]
Gkortzis A, Mitropoulos D, Spinellis D	VulinOSS: a dataset of security vulnerabilities in open source systems	2018	P36	[22]
Ponta SE, Plate H, Sabetta A, Bezzi M, Dangremont C	A manually-curated dataset of fixes to vulnerabilities of open source software	2019	P37	[52]
Zhou J, Wang S, Bezemer CP, Zou Y, Hassan AE	Studying the Association between Bountysource Bounties and the Issue-addressing Likelihood of GitHub Issue Reports	2020	P38	[75]

Table A.1: List of selected studies.

Appendix B

Semi-structured Interview Questions

In this appendix, I provide the questions asked in the semi-structured interviews from practitioners.

B.1 Interviewee Information

Q1: Interviewee presents themselves.

Q2: Do you have a degree in computer science/engineering?

Q3: How many years of professional experience do you have? In which roles?

Q4: What technologies do you use for programming? (what type of software development like backend and frontend)

Q5: What is the role you have in your team?(Test engineer, devops,...)

B.2 Interviewee Company Information

Q6: How do you describe your company? (a small startup/ medium/ big corporation)

Q7: What is the type of business/industry your company is involved with?

Q8: Is the software you work on an open-source or closed-source?

Q9: How important is it for the company to handle security for software development?

Q10: How can your company help you to handle security better?

B.3 Security Related Questions

B.3.1 Interviewee Information on Security

Q11: What was the last project you had a task related to security? (create a context for the next question)

Q12: To what extent does your work involve security tasks? Could you provide some examples?

Q13: On a regular working day, How much time do you invest working on security related tasks?

Q14: How many years of experience have you focused on security?(what role did you have while performing security tasks)

Q15: Is security a priority for you? If no/yes why?

Q16: Did you have any security training during undergraduate/college studies? Is it necessary? Why?

Q17: How difficult do you think adopting security practices to your regular development tasks is? Why? Rate from 1 - 10

Q18: How difficult is learning about security? Why?

B.3.2 Tools and Interventions

Q19: How do you approach security tasks?(How do you detect vulnerabilities?) Based on your recent experience answered in question 11.

Q20: To what extent is security a requirement in your development process? (do you have to meet any security standards in your development)(Do you perform code reviews?

If yes do you have security checklists) (What secure code practices do you usually follow?)

Q21: To whom do you approach to discuss security decisions? How are the discussions handled?

Q22: How does your development team collaborate with each other for handling security incidents?

Q23: Do you have any security training in your company?(if yes, what kind of training does your company provide?)

Q24: What kind of information would you need to better assess vulnerabilities?

Q25: What tools or interventions do you use to help with improving security in your application? (such as static analysis tools)(what could be improved in security tools? If they got improved would it affect your security related tasks?)(not only SATs, could be documents, reports, logs, external databases, or any resource of information that helps them handle security) (How artifact X helps you to ensure security in your development process?)

B.3.3 Motivations and Deterrents

Q26: What are your motivations to adopt/not to adopt security?

If yes: What will help you to maintain your motivations to adopt security?

If no: Why don't you use secure coding practices? What would help you to adopt them?

Q27: What factors deter you from working on security tasks?

Q28: How do you think the deterrents can be mitigated?

B.4 Behavioral Aspects of Adopting Security in Software Development

Q29: What do you see as the advantages of your adopting security practices when developing software?

Q30: What do you see as the disadvantages of your adopting security practices when developing software?

Q31: What positive feelings do you associate with adopting security practices when developing software?

Q32: What negative feelings do you associate with adopting security practices when developing software?

Q33: Please list the individuals or groups who would approve or think you should adopt security practices when developing software.

Q34: Please list the individuals or groups who would disapprove or think you should not adopt security practices when developing software.

Q35: Sometimes when we are not sure what to do, we look to see what others are doing, Please, list the individuals or groups who are most likely to adopt security practices when developing software.

Q36: Please, list the individuals or groups who are least likely to adopt security practices when developing software.

Q37: Please list any factors or circumstances that would make it easy or enable you to adopt security practices when developing software.

Q38: Please list any factors or circumstances that would make it difficult or prevent you from adopting security practices when developing software.

Appendix C

Semi-structured Interview Recruitment Documents

In this appendix, I provide documents for recruiting interviewees.

C.1 Invitation to Participants

Dear CONTACT NAME,

We are LIST OF CURRENT INVESTIGATORS researchers from the computer-human interaction and software engineering lab (chisel) in the department of computer science at the University of Victoria AND AFFILIATIONS OF CO-INVESTIGORS, writing to request your participation in a research project. We would be grateful if you could help us to understand how software developers approach security in software development and the current adoption of secure code practices by participating in an interview. The interview will be conducted using zoom or skype and should take about 1 hour. The interview will be recorded as well, and all data – identifiable will be anonymized and stored on secure servers.

We will openly publish the results so everyone can benefit from them. But we will anonymize everything before doing so; your responses will be handled confidentially. Please note that you are not obligated to participate in the interview or respond to this email. If you want to stop during the interview at some point, you are free to do so without any negative consequences.

If you know of any individual you feel would be interested in participating in this study and would have valuable insights to add, please feel free to forward this to

them.

We welcome your participation. Thank you for taking the time to read this email. Kind regards,
NAME OF PRINCIPAL INVESTIGATOR

C.2 Signed Consent Form

You are invited to participate in a study entitled "Understanding the adoption of Secure Software Development Practices" that is being conducted by Soroush Yousefi (MSc. Student), Enrique Larios Vargas, and Margaret-Anne Storey as part of Soroush Yousefi's master thesis. You can contact them by e-mail (soroushysf@uvic.ca, elariosvargas@uvic.ca, mstorey@uvic.ca) or by telephone at +1-778-922-6054.

Purpose and Objectives

This project will explore how software developers approach security when developing software. In particular, we explore three aspects: (1) the process, artifacts, actors, and tools involved in ensuring security, (2) motivations and challenges for handling security in software development, and (3) attitudes and behavior concerning the adoption of secure code practices.

Importance of this Research

There is still much work to be done in the field of Software Engineering research to understand how to ensure security in software development. The contribution of this research is to deepen our understanding of how the adoption of secure code practices can be motivated and improved among practitioners.

Participant Selection

You are being asked to participate in this study because you are an employee at ¡organization¿.

Involvement

If you agree to participate in this research voluntarily, your participation will include individual interviews and member checking meetings. Soroush Yousefi and Enrique Larios Vargas will be conducting the interviews. Member checking meetings will occur only when the interviews' analysis is completed to validate the information collected.

Risks

Participation in this study may cause some inconvenience to you, including brief distractions from your current activities (our interviews may last around 1 hour, but

we can schedule them for the duration of your choice). To avoid this inconvenience, the researcher will coordinate with you for the best time to participate in the research. Please note that this research does not collect information about individual performance in any way.

Benefits

The potential benefits of your participation in this research include a better understanding of how to ensure security in software development by understanding the current secure code practices adopted within your organization. The study will result in improved strategies and solutions that will benefit your team in your daily work regarding security.

Voluntary Participation

Your participation in this research must be entirely voluntary. If you decide to participate, you may withdraw at any time without consequences or any explanation. If you withdraw from the study, your data will be not used in any research articles and will be destroyed. The notes from interviews will also be destroyed.

Anonymity

Apart from identifying information collected by the researcher, your anonymity will be protected. The researchers will be the only persons that have access to this information. We will not inform your managers whether you chose to participate in the study or not. The researcher must maintain a way to identify each piece of the data collected to enable follow-up with participants in subsequent research phases. This identifying information will be protected in possession of the researcher.

Confidentiality

Your confidentiality and the data's confidentiality will be protected by data being kept electronically on a password-protected computer in the CHISEL offices at the University of Victoria. No one other than the researchers will have access to information that identifies you with the research data.

Dissemination of Results

It is anticipated that the results of this study will be shared with others in the following ways: presentations at scholarly meetings, published articles at conferences or journals, as well as directly to participants in the form of reports made available to your company, online in the form of a thesis. None of these reports will provide information to identify the participants or the company name. Publication drafts will be provided to the company, and the company reserves the right to obfuscate any company sensitive material from any resulting papers or publications. Data from

this study will be disposed of after 3 years.

Ethics Protocol Approval

In addition, you may verify the ethical approval of this study or raise any concerns you might have by contacting the Human Research Ethics Office at the University of Victoria (250-472-4545 or ethics@uvic.ca).

Your signature below indicates that you understand the above conditions of participation in this study. You have had the opportunity to have your questions answered by the researchers and consent to participate in this research project.

Name of Participant

Signature

Date

A copy of this consent will be left with you, and the researcher will take a copy.