Pushed by Accident: A Mixed-Methods Study on Strategies of Handling Secrets in Source Code Repositories

Alexander Krause [C] Jan H. Klemmer [*] Nicolas Huaman [*] Dominik Wermke [C]
Yasemin Acar [†, ‡] Sascha Fahl [C]

[C] *CISPA Helmholtz Center for Information Security, Germany,* {alexander.krause,dominik.wermke,sascha.fahl}@cispa.de
[*] *Leibniz University Hannover, Germany,* {klemmer,huaman}@sec.uni-hannover.de
[†] *Paderborn University, Germany,* yasemin.acar@uni-paderborn.de
[‡] *The George Washington University, USA*

Abstract

Version control systems for source code, such as Git, are key tools in modern software development. Many developers use services like GitHub or GitLab for collaborative software development. Many software projects include code secrets such as API keys or passwords that need to be managed securely. Previous research and blog posts found that developers struggle with secure code secret management and accidentally leaked code secrets to public Git repositories. Leaking code secrets to the public can have disastrous consequences, such as abusing services and systems or making sensitive user data available to attackers. In a mixed-methods study, we surveyed 109 developers with version control system experience. Additionally, we conducted 14 in-depth semi-structured interviews with developers who experienced secret leakage in the past. 30.3% of our participants encountered code secret leaks in the past. Most of them face several challenges with secret leakage prevention and remediation. Based on our findings, we discuss challenges, such as estimating the risks of leaked secrets, and the needs of developers in remediating and preventing code secret leaks, such as low adoption requirements. We conclude with recommendations for developers and source code platform providers to reduce the risk of secret leakage.

1 Introduction

Version control systems (VCSs) are an essential technology for collaborative software development. Git [1], a fundamental tool to orchestrate collaborative development, has been voted as the most common tool in the recent Stack Overflow Developer Survey [2] with 93.4% of participants specifying to use this tool in their development workflow. Git-based code repository platforms (e. g., GitHub [3] and GitLab [4]) aim to ease sharing, reviewing, and contributing to software projects. In modern development pipelines, software is commonly directly built, tested, and deployed within and from these code repositories. To deploy software on server infrastructure, automate interactions with third-party services, or handle authentication, developers need to provide secrets, e. g., credentials, authentication tokens, or secret encryption keys. However, these secrets must be protected from being leaked accidentally into the public codebase. Unfortunately, this is no straightforward task. Recent work by Meli et al. [5] found that on GitHub, the most popular code sharing platform[1], thousands of automatically detectable secrets are leaked daily.

A leaked secret can have a significant impact depending on the type of secret and how long it takes for the secret owner to revoke it after noticing its leak. In some cases, a leak can be highly critical, as in the case of Toyota: a hard-coded credential for accessing a data server was publicly pushed to GitHub in 2017. It allowed attackers to control the Toyota T-Connect accounts for 296,019 customers [7]. T-Connect provides features like remote starting, in-car Wi-Fi, digital key access, full control over dashboard-provided metrics, and a direct line to the My Toyota service app. After more than five years, Toyota invalidated the key. Such a long time could mean multiple malicious actors have already gained access. GitHub recently leaked a private SSH host key of their production Git servers in a public repository and replaced them to prevent abuse. This incident illustrates the complexity of secret management, even for large companies experienced with code secrets and public source code repositories [8].

There is anecdotal knowledge [9–12] on secret leaks through source code repositories. However, there has been little prior research trying to understand better the reasons for and experiences with code secret leakage in source code repositories. To address this gap, to the best of our knowledge we are the first to conduct a mixed-methods study, including an online survey and semi-structured in-depth interviews with experienced developers. We investigate the following research questions:

RQ1. *How widespread is code secret leakage among developers?* Leaking secrets and access tokens in source code poses a potentially serious security threat. We asked 109 developers how often they encountered secret leaks in the past.

[1] According to the Tranco list [6] generated on May 25, 2023, available at https://tranco-list.eu/list/JX43Y.

RQ2. *What are secret leakage prevention approaches, and what are developers experiences?* Depending on the scenario and context, prevention approaches can differ widely. We surveyed and interviewed developers to reveal prevention approaches and the experiences, challenges, and needs that developers have when using them.

RQ3. *What are developers' experiences with code secret leakage incidents?* Little is known about developers' experiences when remediating code secret leaks. We interviewed developers on their latest and most impactful secret leaks to learn from their experiences, how they recognized a leak, and their consequences.

RQ4. *What are developers' experiences with code secret remediation techniques and tools?* Remediating code secret leakage can be challenging. We examine deployed remediation approaches, developers' experiences with these approaches, and their requirements for approaches.

Overall, we conducted a survey with 50 freelancers from Upwork and 59 developers from GitHub. For in-depth insights in developers' experiences with code secret leakage incidents and the approaches they use to prevent and remediate leaks, we interviewed another 14 developers who experienced code secret leakage. We make the following key contributions:

Identifying 18 Secret Leakage Prevention and Remediation Approaches. We present survey results with freelancers and GitHub developers, investigating their approaches and experiences with code secret prevention and remediation (Section 3.1). We discovered 18 approaches to prevent and remediate code secret leakage (Table 3). 30.3% of our participants reported first-hand experience with secret leakage in their projects.

Identifying Challenges Developers Face with Secret Leakage Prevention and Remediation Approaches. In addition to the survey, we interviewed GitHub developers who experienced code secret leakage to gather more qualitative insights (Section 3.2). We report on how they detected the leaks, their experiences with code secret leak incidents, their approaches to preventing leaks, and their techniques and tools for secret leakage remediation. We identified several challenges with common remediation and prevention approaches and motivations to use them.

Providing Recommendations to Reduce the Risk of Secret Leakage. Based on our findings, we provide recommendations for future research, software developers, and collaborative source code platforms to prevent and remediate code secret leakage in Section 6.

Providing a Full Replication Package. To support future research, a full replication package is available in line with the effort to support replication of our work, containing all study materials in the *Availability* section (after Section 7).

2 Related Work

We present and discuss related work in two key areas: previous research on secret leakage in source code repositories and developers' secure development approaches and practices.

2.1 Secret Leakage in Code Repositories

In recent years, researchers made efforts to measure secret leakage in source code repositories. In addition, we discuss secret detection and methods to improve its accuracy.

Measurement Studies. Sinha et al. discussed different approaches on how to detect, prevent, and fix code secret leakage in source code repositories [13]. In 2019, Meli et al. presented a large-scale measurement study on secret leakage in public GitHub repositories, finding more than 100,000 repositories with leaked secrets. They demonstrated and evaluated approaches to detect secrets on GitHub. The authors examined potential root causes, including the developer experience and practice of storing secret information in repositories [5].

Secret Leakage Detection. Secret scanners produce a high rate of false-positive results, which is a major problem because developers have to review them manually. Meli et al. used past secret detection strategies; while they avoided high false-positive results. They combined multiple methods to detect potential secrets and secret evaluation [5]. However, automatic detection of secret leaks can be challenging. In 2020, Saha et al. applied machine learning to reduce the false-positive rate of secret scanners [14]. Similarly, Lounici et al. developed and evaluated machine learning classifiers to reduce false-positives [15]. Recently, Kall and Trabelsi proposed and evaluated an approach to improve the detection of leaked credentials in source code repositories [16]. In 2022, Feng et al. developed an automated approach to effectively detect password leakage from public repositories [17]. In 2022, Rahman et al. investigated human factors on code secret leakage detection tool warnings [18]. In 2022, Basak et al. conducted a gray literature review to identify developer and organizational practices [19].

The previous research on code secret leakage focused on assessing its prevalence and identifying code secret leaks after an incident occurred. We investigate developers' experiences with code secret leakage incidents and explore their strategies for preventing and remediating them by conducting two developer studies.

2.2 Exploring Secure Development Approaches and Practices

Researchers have extensively studied secure software development practices. This section describes these research efforts dedicated to improving secure development methodologies, offering insights and guidance to empower developers in achieving secure software.

Assal et al. interviewed 13 developers to investigate their real-life software security practices during each stage of the development lifecycle. Real-life security practices differ markedly from best practices identified in the literature [20]. Assal et al. continued by conducting an online survey with 123 software developers to explore the interplay between developers and software security processes. They found that security vulnerabilities often result from a lack of organizational or process support [21]. In 2017, Haney et al. surveyed 121 representatives from organizations that work in cryptographic development. They characterized the cryptographic practices, types of resources, and standards used by cryptographic developers. They found that participants used cryptography for a variety of purposes, with the majority relying on generally accepted, standards-based implementations as guides [22]. In 2018, the researchers conducted 21 in-depth interviews with highly experienced individuals from organizations that employ cryptographic implementations to gain a more profound understanding of their cryptographic development practices. They demonstrate a strong organizational security culture that guides the careful selection of resources and informs formal, rigorous development and testing practices [23]. In 2020, Votipka et al. qualitatively analyzed BIBIFI [24] submissions on how and why programmers make security errors. They found that most vulnerabilities result from misconceptions. Furthermore, they suggest APIs should be simpler and more precisely documented, including multiple use cases and edge cases [25]. In 2020, Palombo et al. presented an ethnographic study of secure software development processes in a software company finding that sometimes vulnerabilities were ignored or even consciously introduced to fix other issues [26]. In 2022, Wermke et al. interviewed 27 open source developers to investigate their security and trust practices. They found that open source projects are highly diverse in deployed security measures, trust processes, and their underlying motivations [27]. Naiakshina et al. qualitatively analyzed security problems when implementing secure password storage. The authors found conflicting advice to be an obstacle that developers struggle with [28]. Lopez et al. analyzed security-related conversations on Stack Overflow and found that developers use online environments to actively connect, exchange information, and provide assistance, despite concerns about their reliability as security information sources. [29]. Recently, Fischer et al. conducted a study analyzing the effect of Google Search on security in software development. Besides insecure resources among search results, they demonstrated that re-ranking search results significantly improved security.

While previous research explored secure development practices and approaches in general, in companies, and open source communities, focusing on security vulnerabilities and implementations, we extend this by focusing on the practices developers use to prevent and remediate code secret leakage at code repository platforms.

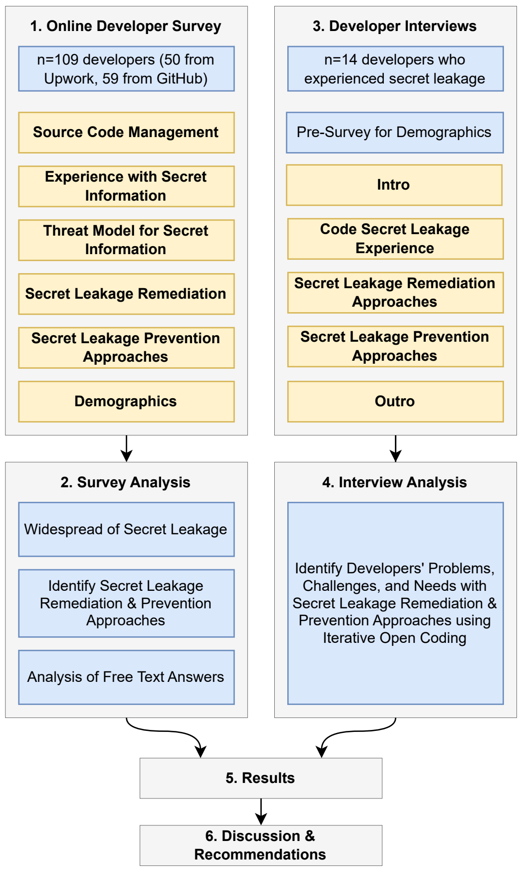
1. Online Developer Survey
n=109 developers (50 from Upwork, 59 from GitHub)
Source Code Management
Experience with Secret Information
Threat Model for Secret Information
Secret Leakage Remediation
Secret Leakage Prevention Approaches
Demographics
3. Developer Interviews
n=14 developers who experienced secret leakage
Pre-Survey for Demographics
Intro
Code Secret Leakage Experience
Secret Leakage Remediation Approaches
Secret Leakage Prevention Approaches
Outro
2. Survey Analysis
Widespread of Secret Leakage
Identify Secret Leakage Remediation & Prevention Approaches
Analysis of Free Text Answers
4. Interview Analysis
Identify Developers' Problems, Challenges, and Needs with Secret Leakage Remediation & Prevention Approaches using Iterative Open Coding
5. Results
6. Discussion & Recommendations

Figure 1: Study overview, showing methodology in blue and the content of the survey/interview sections in yellow.

3 Methodology

In this section, we explain the methodology of our studies. An overview is depicted in Figure 1. After a detailed description of the online survey with Upwork and GitHub developers, we describe our interview study with developers from GitHub, in which we gained further qualitative insights into code secret incidents and their remediation and prevention approaches.

3.1 Online Survey with Developers

Below, we provide details on the approach and structure of the survey which we conducted with 109 developers. We detail the analysis of both qualitative and quantitative data points.

3.1.1 Survey Procedure

Between September 6 and October 26, 2021, we conducted an online survey with 50 freelancers from Upwork and 59 developers from GitHub. We used Qualtrics [30] to provide the survey and collect the respondent's data.

Questionnaire Development. First, we conducted an exploratory analysis of online guides, reviewing 100 web pages for an impression of what kind of information for code secret leakage prevention and remediation is provided online (cf. Appendix A). We developed our questionnaire based on this exploratory analysis, previous research (cf. Section 2), and the research questions. In addition, we established additional areas of interest for our survey based on participants' input during pre-testing. In both our survey and interview studies, we provided explanations for the terms *code secret*, *code secret leakage*, and *code secret handling approaches* for a shared understanding of the terms with participants. These explanations emerged from the exploratory analysis and were validated for understanding in survey pilots.

Piloting. Before conducting the survey, we piloted our questionnaire with 11 participants. First, we conducted cognitive walkthroughs [31] with four usable security researchers. Subsequently, seven participants completed the survey unsupervised, with an additional feedback text box on each page of the questionnaire. The results were used to verify and improve survey clarity and flow.

3.1.2 Recruitment and Inclusion Criteria

We used two different recruitment strategies to obtain a diverse sample of developers: We recruited Upwork freelancers and GitHub developers. We include the recruitment material in our replication package (cf. *Availability* section).

Upwork. On the freelancer platform Upwork [32], we published several identical postings for developers from September 8 through October 23, 2021. Freelancers applied by writing a short application. This included answering screening questions, which we used to accept only participants that met our inclusion criteria. We accepted freelancers who had worked with VCSs and related platforms to manage their source code. They were also required to have collaborated with other people in the past. These criteria ensured they potentially had to handle secrets on source code platforms. We compensated the freelancers with \$25 to offer a competitive reward (\$1/minute), so that professionals would also have an incentive to participate [33].

GitHub. Following security research from 2021 and 2022 [34, 35], we implemented the following procedure to recruit developers from GitHub. From October 18 to October 20, 2021, we invited 5,310 GitHub users, which had an activity on September 1, 2021. We also verified that all recruited developers were also active contributors in general by manually analyzing their GitHub commits. We only invited developers who had published their contact email addresses on their profiles, and stopped inviting developers once we reached saturation. We refrained from contacting developers who did not want to be contacted, contacted developers only once, and offered to add them to an ignore list[2]. We received no complaints from any of the invited developers (cf. Section 3.4).

[2]1.4% of the users we invited made use of this option.

3.1.3 Survey Structure

Below, we outline topics and questions covered in the questionnaire (cf. Figure 1). The full survey including the consent form can be found in our replication package (cf. *Availability* section).

Source Code Management. We asked questions regarding what types of source code repositories participants used (e. g., local only, remote platform, self- or third-party-hosted), and about usage of different VCSs and platforms to gather a general understanding of respondents source code management.

Experience with Secret Information. This block of questions aims to understand what type of different secrets participants encountered. We asked these questions for all past projects and the most recent project.

Threat Model for Secret Information. To understand participants' threat model, we asked participants which other stakeholders had access to which secret information and who made this decision in their most recent project. We followed up with brief definitions of *code secret* and *code secret leakage* to ensure a shared understanding for the following questions. This question block concludes with two Likert scale questions regarding the perceived prevalence of code secret leakage and potential consequences.

Secret Leakage Remediation. Next, we asked questions regarding whether the participants experienced code secret leakage themselves or knew someone who did. To investigate current secret leakage remediation practices, participants were prompted to describe (based on the previous question's answer) how they remediated code secret leakage themselves, how others did, or how they hypothetically would.

Code Secret Handling Approaches. Another important aspect, covered in this question block, are approaches to prevent code secret leakage in general. Again, we defined *code secret handling approach* (all approaches to avoid code secret leakage). To gather prevalent practices, we asked participants in free text questions which approaches they have heard of, which ones they used, and why. Additionally, we asked participants for any issues they had, with which approaches, why they failed with an approach, and what they wanted to use an approach for. Finally, we asked whether and how the participants helped co-workers handle secrets.

Demographics. The concluding block was about demographics. This included standard questions (e. g., age, gender, country, education, employment status), but also specialized questions to investigate the development (e. g., years in development, number of projects) and security experience.

3.1.4 Analysis

Our analysis is a mix of quantitative and qualitative evaluation. We report various counts and percentages of single- and multiple-choice questions in text and figures in the survey results (cf. Section 4). For the free-text questions, two researchers used an iterative open-coding approach to extract the used approaches on code secret prevention and remediation (Q11–15) [36–38]. To prevent mislabeling, the researchers first coded ten answers together and discussed problematic codes for each question. Subsequently, two researchers coded the answers independently, followed by a third researcher independently reviewing the coding. Finally, they resolved any coding conflicts in a consensus discussion or introduced new codes if necessary [39]. All previous answers were re-coded if new codes were introduced.

3.2 Interviews with Developers

To enrich and deepen the survey insights, we decided to complement those with qualitative insights from semi-structured interviews ($n = 14$). The interview results cover the reasons, experiences, and processes for the prevention and remediation approaches we collected in our survey (cf. Section 5). We reached saturation within the high-level codes of our codebook. The average interview duration was 32 minutes (median: 32.5 minutes).

3.2.1 Interview Procedure

All 14 interviews were conducted in June and July 2022. We utilized a setup with two interviewers. A main interviewer held the conversation with the interviewee and asked the questions according to the interview guide. A so-called shadow interviewer was present to listen and note what questions were asked, and to make sure none were forgotten. At the end of the interview, the shadow interviewer also had the chance to ask questions to follow up on interesting aspects that emerged. All interviews were conducted in English and remotely via a GDPR-compliant conference tool. We recorded each interview to create a transcript later on, after which we destroyed the recordings. All transcripts have been manually checked and corrected by us for possible errors.

Pre-Questionnaire. Before the actual interview, each participant had to fill a short pre-questionnaire. This had multiple purposes. (1) We screened participants by only accepting those who experienced secret leakage. (2) We explained the purpose of the study and obtained consent for participation, our data handling, and recording. (3) Finally, we asked several demographics and background questions, which also helped the interviewer to prepare for the interview.

Piloting. We iteratively tested and improved the initial version of the interview guide. This mainly included three cognitive walkthroughs with usable security researchers. We used this interview simulations to obtain feedback on question clarity, completeness, and to generally improve the interview guide with the interviewed researchers' experience. After each interview, we tweaked question clarity to ensure a good interview flow. This was followed by two pilot interviews with developers from Upwork.

3.2.2 Recruitment and Inclusion Criteria

As the goal of this study is to investigate secret leakage prevention as well as remediation, we decided to only interview developers who experienced secret leakage and therefore can report on remediation and past incidents. This was the only eligibility criteria to participate in an interview. Two developers from Upwork who also participated in our survey and stated that they experienced code secret leakage were invited and compensated with $60. Apart from the two piloting interviews with Upwork users, we recruited the remaining twelve participants from GitHub – following the same approach as for the online survey (cf. Section 3.1.2). Due to institutional restrictions, we could not compensate GitHub interview participants directly. Instead, we could offer these participants to sponsor a GitHub project of their choice with $60.

3.2.3 Interview Structure and Interview Guide

In the following, we describe the interview structure and questions. We outline all sections, each containing top-level questions and corresponding follow-up questions. The full interview guide can be found in Appendix B.

Introduction. Each interview started with greeting the participant, explaining the interview's purpose and procedure, and obtaining consent from the participants. We underlined that we are only interested in personal opinions and experiences and not judging their case of secret leakage. The participants could skip questions anytime.

Code Secrets. In the first section, we talked about code secrets and established a shared understanding of what a code secret is. Moreover, we asked participants about their broader experiences with code secrets. This included where they came into touch with code secrets and their experiences regarding sensitivity, and code secret access.

Secret Leakage and Remediation Approaches. We continued with the code secret leakage incident that the participant had experienced. First, we established a uniform understanding of what a secret leak is. Then, we queried participants about their most impactful or latest (depending on what they remember best) secret leak. To get detailed insights, we asked for reasons why the leak occurred, consequences, and changes to secret handling due to the incident. Related to the leak, we asked for its remediation, including experiences, and challenges, involved individuals/teams, and consulted resources.

Secret Leakage Prevention Approaches. The third section was about preventive measures against secret leakage. We asked questions on approaches that participants have used, including their experiences, understanding of the approach, and any challenges. We also asked for approaches they tried to use but failed with and the reason for this, as well as approaches they know of but do not use. If a participant had not taken any prevention approach, we instead asked for potential reasons.

The section concludes with an open question on wishes and improvements for future prevention approaches.

Outro & Debriefing. After we asked all the questions, we held a debriefing with the participants to clarify any remaining questions, to give participants an opportunity to add something we might not have specifically asked for, and to gather feedback for the interview.

3.2.4 Analysis and Coding

We used an iterative open-coding approach to analyze all interview transcripts [36–38]. First, two researchers developed an initial codebook based on their interview impressions and the interview guide. Afterward, the same two researchers coded the interviews in multiple rounds. After each iteration, they resolved conflicts by consensus discussion or by introducing new sub-codes. We continued iterative coding until no new codes and themes emerged [40, 41]. We do not report inter-coder agreement. We resolved each conflict immediately when it emerged (resulting in a hypothetical final agreement of 100%) [27, 39, 42–44]. The final codebook is part of our replication package (cf. *Availability* section).

3.3 Participant Demographics

We hired 109 respondents for the survey, and recruited 14 participants for the interviews. Overall, our survey respondents and interview participants were predominantly male, with roughly 10% female, and were about 30 years old on average. Country-wise, we have a highly diverse sample of survey respondents from more than 33 distinct countries, including the U.S., India, and Germany as the top three, as well as Canada, the UK, Russia, Pakistan, Portugal, the Netherlands, Mexico, Australia, Egypt, Brazil, and Indonesia. Interview participants were from nine distinct countries, including the U.S., India, and Pakistan as the top three, as well as Canada, Belarus, Italy, Kenya, and Brazil. While most survey respondents were full-time (71, 65.1%) or part-time employees (11, 10.1%), some participants reported to be self-employed/freelancers (33, 30.3%), or students (12, 11.0%). Overall, the respondents and participants were highly experienced, with the majority of survey respondents (59, 54.1%) having developed software for more than five years. About 85% of the survey respondents and 95% of the interview participants said they taught themselves how to program, often in addition to other ways of learning, e. g., at college or university, on the job, or in online classes. In total, the demographics in terms of gender, age, top three countries, and education are comparable to those of the 2022 Stack Overflow developer survey [2]. Table 1 provides the detailed overview of survey respondents' and interview participants' demographics.

Table 1: Selected participant demographics from both the survey and interviews. We omit "Other" and "Prefer not to disclose" answers for space reasons.

	Survey			**Interviews**
	Upwork	**GitHub**	**Combined**	
Participants:				
Started	52	101	153	n/a
Finished	51	59	110	n/a
Valid/Total ($n =$)	50	59	109	14
Gender:				
Male	86.0%	88.1%	87.2%	92.9%
Female	10.0%	1.7%	5.5%	0.0%
Non-Binary	0.0%	6.8%	3.7%	7.1%
Age [years]:				
Median	29.0	33.0	30.0	28.0
Mean	31.3	34.9	33.2	32.1
Country of Residence:				
U.S.	2.0%	32.2%	18.3%	21.4%
India	20.0%	3.4%	11.0%	21.4%
Germany	0.0%	18.6%	10.1%	0.0%
Pakistan	14.0%	3.4%	8.3%	14.3%
Other	60.0%	40.7%	49.5%	42.9%
Development/Programming Education:[1]				
Self-taught	92.0%	94.9%	93.6%	85.7%
College/University	54.0%	62.7%	58.7%	71.4%
On-the-job training	72.0%	42.4%	56.0%	57.1%
Online class	60.0%	28.8%	43.1%	35.7%
Coding camp	18.0%	8.5%	12.8%	0.0%

[1] Multiple answers allowed; may not sum to 100%.

3.4 Ethics & Data Protection

The research in this paper was approved by one of our institutions' ethical review board (ERB) (IRB equivalent). Overall, we adhere to the principles for ethical research outlined in the *Menlo Report* [45]. In addition, we handle all data according to the strict General Data Protection Regulation (GDPR) laws of the European Union (EU). Furthermore, all data is stored in de-identified form, so there is no link between the participants' survey or interview answers and their identity. Before participating in our studies, we encouraged potential participants to familiarize themselves with consent and data handling information on the study website. We obtained informed consent from all participants for participation in the study and having their interview's audio recorded and transcribed. Before, during, and after the interview, (potential) participants were able to contact us at listed contact addresses for any questions or additional information. Reporting security incidents such as leaked secrets might be very sensitive for participants. Therefore, we always offered the option to skip a question or select "Prefer not to answer".

Recruiting developers from GitHub has been common in the past [27, 46, 47]. However, a change in GitHub's terms of service prohibits contacting their users for research purposes [48, § 7]. Therefore, we suggest researchers avoid this recruitment procedure in the future.

We compensated all Upworkers who completed the survey with $25. Due to institutional restrictions, we could not compensate survey participants recruited from GitHub. When conducting the interview study later, we solved this issue and could offer compensation to interview participants recruited from GitHub.

3.5 Limitations

Our work includes some limitations typical for these survey and interview studies and should be interpreted in context. In general, self-report studies may suffer from several biases, including over- and under-reporting, sample bias, self-selection bias, and social desirability bias [46, 49–52]. Developers who agreed to speak with us could be more (or less) security-conscious than those who declined.

Furthermore, focusing our recruitment on Upwork and GitHub participants might introduce a sampling bias by excluding developers active on other platforms. We chose GitHub due to its high popularity compared to other platforms like GitLab. Kaur et al. compared different freelancer recruitment channels for studies with developers [53], and Upwork appears to be the best choice for our study. It offers recruitment for freelancers worldwide, allowing us to gather a more complete picture than country-specific platforms. Our demographics also reflect this diversity (Section 3.3).

4 Survey on Secret Management

In this section, we report on the findings based on all 109 valid survey responses (153 respondents started, one respondent tried to scam us by copying answers from websites). This includes an overview of 18 prevention and remediation approaches that survey respondents used. We include survey respondents' quotes as transcribed, with minor grammatical corrections and omissions marked by brackets ("[...]"). Survey respondents are numbered with a leading *S* (e. g., S4). Below, we report on the most relevant and interesting survey questions and responses. A full list of counts and codes for all questions is provided as part of our replication package (cf. *Availability* section). For the interview study that we conducted with developers who experienced secret leakage, we report on their experiences in Section 5.

4.1 Version Control Systems and Platforms

In the survey, we asked respondents for the different VCSs and corresponding platforms they used within the last 12 months. All 109 respondents reported using Git, which was by far the most popular VCS. In addition to Git, 12 respondents (11.0%) used Subversion, being the second most popular VCS. Others mentioned Concurrent Versions System (CVS) (5, 4.6%), Microsoft Team Foundation Version Control (TFVC) (4, 3.7%), Perforce (2, 1.8%), and Mercurial (2, 1.8%).

Closely related to the high prevalence of Git, both GitHub and GitLab were the platforms most often reported by 99 survey respondents (90.8%) and 48 respondents (44.0%), respectively. Besides these public services mainly known for

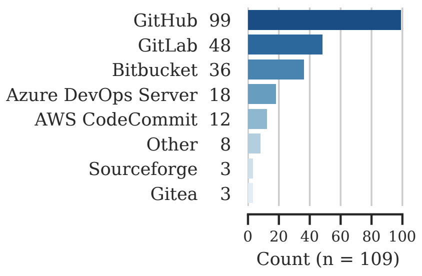

Figure 2: Platform usage reported by the survey respondents. We allowed multiple answers.

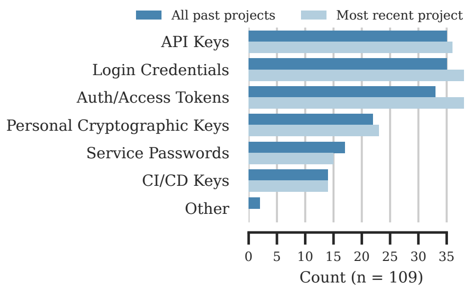

Figure 3: Secret usage reported by the survey respondents. We allowed multiple answers.

open-source software development, respondents reported services targeted at companies that offer commercial features for private source code repositories. This included cloud solutions like Microsoft's Azure DevOps Server (18, 16.5%), Amazon Web Services (AWS) CodeCommit (12, 11.0%), and Google Cloud (GC) Source Repositories (2, 1.8%), but also self-hosted solutions like Gitea (3, 2.8%). One respondent reported to use general file synchronization solutions instead of specialized services or hosting platforms. Figure 2 depicts system and service usage in detail.

Overall, hosting source code repositories using a third-party service or provider is the most prevalent for both public (95, 87.2%) or private (77, 70.6%) repositories. Contrary, self-hosting repositories were reported less than half as often and are more prevalent for private repositories (39, 35.8%) than for public ones (26, 23.9%).

4.2 Secrets, Access, and Threat Model

Regarding the types of secret information, our respondents reported that they predominantly handled login credentials (35, 32.1%), application programming interface (API) keys (35, 32.1%), and authentication and access tokens (33, 30.3%) within their past projects. This is similar for their most recent projects, as shown in Figure 3. Respondents also said that they encountered cryptographic keys, either for personal or server use, as well as special service passwords or keys used in CI/CD pipelines.

Regarding secret access and sharing, the respondents re-

Table 2: Summary of which groups of persons had access to secrets (besides the participants themselves); percentages normalized to 100% for all groups of each access level.

	Other team members	Management	Members of other teams	Public
Had access	156 (70,5%)	79 (35.7%)	34 (15.8%)	8 (3.6%)
Had no access	49 (22.2%)	116 (52.5%)	170 (76.9%)	203 (91.9%)
I don't know	2 (0.9%)	10 (45.2%)	4 (1.8%)	3 (1.4%)
Prefer not to disclose	14 (6.3%)	16 (7.2)	12 (5.4%)	7 (3.2%)

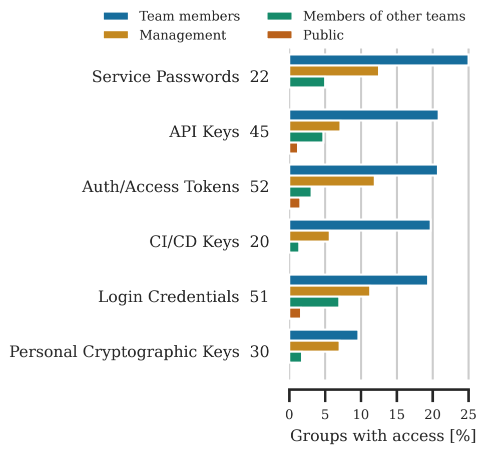

Figure 4: Respondents' reports on who had access to which secret information in their most recent project. Not considering "I don't know" and "Prefer not to disclose" answers; normalized to 100% for all groups of each secret type.

ported that other members within the same team had access 171 times. Most commonly, these were the management (90) and members of other teams (42). The general public had access in only eight cases. Conversely, the public had no access in the majority of cases, which can be inferred from Table 2 among other details. Most respondents shared secrets with their team members, some with their management, only a few with other teams, or with the public (cf. Figure 4). Developers shared passwords for services, API keys, and authentication/access tokens more often than personal keys.

Access control for secrets was most often configured by the respondents themselves, as 44 respondents (40.4%) reported. Sporadically, these decisions were jointly made (15, 13.8%), or at least with some involvement of the respondent (18, 16.5%). In those cases, decisions were made with architects, team leaders, management, the whole development team, or a security team.

4.3 Code Secret Leakage Incidents

While a third of all survey respondents (33, 30.3%) reported a code secret leakage in the past, only a few participants provided further information in the corresponding free text field. Survey respondents mainly reported on two types of incidents: secrets that were included in source code, and secrets that were placed in dedicated files, e. g., secret files or *.env* files, and had been pushed to the repo accidentally, e. g., "*I mistakenly pushed my .env file which includes all of my API keys to the Github repository.*" (S150).

Impact of Code Secret Leakage. Respondents faced various consequences of code secret leakage. They reported on, e. g., delayed project schedules and service downtimes due to the code secret leak. Furthermore, worse consequences occurred, like the leakage of confidential data, or team members that got fired because of the code secret leak, e. g.,

> "*There was a code secret leakage in one of the projects that I was part of. It led [...] to the loss of confidential information to the public. The project team leader had to cut every team member of the project because, [in his opinion, one team member was involved in the leakage.]*"— S49

4.4 Prevention and Remediation Approaches

Based on the survey answers, we found 18 approaches to prevent and remediate code secret leakage that participants had used before, or at least knew about. Table 3 provides an overview of both the nine prevention and nine remediation approaches, and their prevalence rates. Each approach is a theme that we extracted from all survey responses.

While most respondents reported the combined usage of multiple approaches (e. g., externalizing secrets and keeping them out of repositories, or encrypting them within the repositories), some approaches are designed for specific use cases and occurred less frequently (e. g., code or secret reviews). This applies to both prevention and remediation approaches. Approaches were either technical or organizational.

Prevention Approaches. The most reported measures were a combination of externalizing secrets (60, 55.0%), e. g., using dedicated config files or using environment variables, and blocking secrets from the repository itself (32, 29.4%), e. g., by using a *.gitignore* file. About a quarter of the participants (30, 27.5%) indicated to have stored secrets encrypted to prevent misuse if the secrets had to stay in the repository.

Remediation Approaches. The majority of participants (59, 54.1%) reported on renewing or at least revoking a leaked secret to prevent further misuse. In addition, they (19, 17.4%) reported cleaning the VCS history, or removing the secret (12, 11.0%) from source code without cleaning the VCS history.

5 Interviews: Experiences with Secret Leakage

Based on the 14 developer interviews, we report in-depth qualitative findings below. We use quantifiers to determine which qualitative findings were more relevant or minor. They do not serve as quantitative statistical. We include interview participants' quotes with minor grammatical corrections and omissions marked by brackets ("[...]"). Interview participants are numbered with a leading *I* (e. g., I4).

Table 3: Approaches for preventing and remediating code secret leakage as reported in the survey ($n = 109$).

Approach	Description	#	%
Prevention			
Externalize Secrets	Separation of code secrets and committed code so that secrets are loaded at runtime, e.g., storing secrets on a central server or secret management system, using environment variables or files [Hashicorp Vault, Azure Key Vault, AWS Vault, *Password, KeePass, Doppler, python-decouple, GitLab CI, GitHub CI, Travis CI]*	60	55.0%
Block Secrets	Prevent code secrets to be contained in code, config, or any other files or prevent including them in publicly available source code repositories, for example, usage of *.gitignore* files, minimizing secret usage in general or use none, remove secrets from version control before publishing to repository	32	29.4%
Encrypted Secrets	Use encryption to store secrets securely within source code repositories [git-secret, git-crypt, SOPS, GPG, kube-seal]*	30	27.5%
Restrict access	Limit the scope of entities including systems and users with access to code secrets, e.g., by user management, policies, role-based access control	19	17.4%
Monitoring	Regular scanning for code secrets and leaks both locally and remote e.g., using secrets scanners in CI/CD pipelines or pre-commit hooks, or review which entities have/had access [SonarQube, Checkmarx, GitGuardian, AWS Cloud Trail]*	16	14.7%
Education & Awareness	Raise awareness for code secret leakage and educate developers how to handle code secrets, e.g., coding guide, best practices wiki	9	8.3%
Other	Miscellaneous other approaches named by participants, not limited to secret handling	8	7.3%
Rotation	Use short-lived secrets, rotate them periodically [Doppler]*	6	5.5%
Code & Secret Reviews	Manual code reviewing which also focus on code secrets; four or more eyes principle to approve code changes	4	3.7%
Remediation			
Renew or Revoke Secret	Invalidate leaked code secrets to prevent any future misuse [Doppler]*	59	54.1%
Cleanup VCS History	Remove leaked secrets from VCSs whole history, e.g., by rewriting the history, clean caches, or reinitialize the whole repository [BFG Repo Cleaner]*	19	17.4%
Analyze Leak	Analysis and forensics on the code secret leak to identify root causes or how the leak was exploited, e.g., by auditing logs or consulting security experts,	17	15.6%
Removal from Source Code	Remove leaked secrets from the current code base. This doesn't include version history, caches or similar	12	11.0%
Notify Concerned Roles	Inform stakeholders affected or involved in the leak, e.g., security team, management, customers, providers, authorities	8	7.3%
Access Management	Re-evaluation of access control concepts and applying more restrictive access management if needed	6	5.5%
Retract Repository	Delete public repositories affected by the leak or make them private, possibly temporarily until remediation is completed	5	4.6%
Systemic Consequences	Applying consequences due to the secret leak, e.g., new processes, specific education, removal of team members or clients	3	2.8%
Server Operations	Actions taken to remediate secret leakage in running software, e.g., by backuping systems, or pruning and re-initializing servers	2	1.8%

* Tools our respondents used.

5.1 Secret Leakage Prevention Approaches

Interview participants used different approaches to prevent code secret leakage. They also elaborated on several factors on applying these measures, challenges, and lessons learned. Most participants used approaches to block secrets from getting in their repositories: half of the participants used environment variables to avoid statements of secrets in the source code, many used secret managers like *HashiCorp Vault* [54] or *AWS Parameter Store* [55], and a few used *.gitignore* files. A few participants reported that they educated and trained developers to raise their awareness. Moreover, a small number of participants reported that they use secret scanners and also that they use separate environments for development and production. Notably, some participants did not use prevention approaches before encountering code secret leakage.

Factors. Participants reported on several factors that influenced the choice and usage of different prevention approaches. Most important for all participants' satisfaction with an approach was that the approach worked well (in terms of usability and actual code secret leakage prevention). Other than that, approaches have to be effective, efficient, secure, and compliant with company requirements. Participants reported several factors that lead to a negative influence on the usage of prevention approaches. One participant reported that they do not rely on third parties. In consequence, they did not integrate a third-party code secret scanner. Another participant reported, that there were too many constraints for the approach to work: "*We are not going to use vault because it would be too many constraints for us.*" (I3). At least, one participant said, that the integration of the approach to their infrastructure would be too complicated.

5.1.1 Challenges

While a third of all survey respondents (33, 30.3%) reported to have had code secret leakage in the past, only a few respondents (7, 6.4%) reported that they had failed to apply a code secret handling approach when using environment variables, the `git filter-branch` command, or external tools, e. g., vaults. We could identify further problems with prevention measures from the participants we interviewed.

Cost and Time Constraints. Almost all interview participants complained about time and cost requirements of adopting a code secret management tool. This included the time it took to set up a method or educate all involved developers about the method. It also involved adopting the method into existing projects, often requiring refactoring work.

Documentation. Finally, documentation was one of the more commonly cited challenges, especially "*the lack of actual documentation available [for] open source software or open source approaches.*" (I14). This required special care and more time from developers during the setup of an approach, while securely deploying an approach in their infrastructure, and while using it.

Awareness and Education. The interviewees also indicated that getting developers to work carefully and use a secret management approach was a challenge: if a tool required too many workflow changes, it would slow down development, so developers try to bypass prevention approaches, e. g., "*Someone was doing something off the books [...]: They were just creating another repository [...] not within the organisation, but maybe just under a personal account or something. Those you can't really fix with tooling, at the end of the day those are just people problems [...] and we can fix that through training [...][or] policy.*"— I6.

Maintenance. Maintaining an approach can be challenging, for example, having to update secret definitions in secret scanners. About half of our interview participants reported that maintenance proved to be challenging when adopting prevention approaches. The biggest complaint was that maintenance was too time-consuming and not user-friendly.

5.1.2 Lessons Learned

All in all, participants learned from their previous experiences regarding code secret leakage prevention. Many participants suggested using more (automated) secret scanning, e. g.,

"*We need an approach to statically analyze [...][code for] secret leakage. For example, we are totally missing this kind of automation in our pipeline.*"— I8.

Furthermore, a few participants used a prevention approach because it is cheaper compared to facing a code secret leakage incident. Moreover, a few participants stated that they would like to know if the company's prevention approach would work in case of a potential leak. They felt uncomfortable because the approach was never tested. Besides, the human error seemed to be the most relevant factor when it comes to code secret leakage. Therefore, some participants suggested to focus on a better onboarding of new developers.

5.2 Secret Leakage Incidents

There are many ways to leak code secrets in VCSs. They depend on the place of a leak, the secret type, if and how a leak was recognized, and the potential consequences. This section sheds light on incidents that our participants reported. We learned that some participants experienced code secret leakage multiple times, e. g., in the same or across different software projects. A few participants stated that it happened repeatedly at a high frequency. "*[Code secret leakage] happens four or five times a year, I would say.*" (I5).

Places of Leak. Our interview participants experienced code secret leaks in various places. Most participants reported a secret leak through public source code repositories, including GitHub and GitLab. Few participants reported leaks through private repositories, to which only a specific group of potentially unauthorized users had access. One participant experienced code secret leakage through the log files of the *GitHub Workflow* feature [56]. Another participant reported secret leaks on platforms such as *pastebin* [57] or *gist* [58], e. g.,

"*It wasn't like 'I'm going to publish this on the Internet.' They had actually copied and pasted some code to get assistance from somebody in a gist, and the gist happened to be public.*"— I11.

Type of Leak. The type and frequency of secret leaks vastly varied between participants. All participants reported secret leaks through hard-coded information in source code or configuration files of a project. Some interviewees experienced leaks through accidental commits of configuration files that included API keys, tokens, or login credentials, for example. Other participants leaked access keys and API tokens they used for authentication purposes in deployment infrastructures. Few participants leaked AWS tokens [59]. Finally, few interviewees leaked passwords for databases containing sensitive internal information or customer data. One participant accidentally pushed all their project secrets to a publicly available GitHub repository, e. g.,

"*[I was] pushing the commits to GitHub and when I pushed the remote repository, I found that my [password manager database] has gone into GitHub without me wanting it to go to there.*"— I10.

Leak Detection. The impact of code secret leaks highly depends on their detection. In particular, the time span until leaks are detected is important. Most of our participants reported that GitHub notified them in case of a secret leak so that they could respond quickly. In two cases, GitHub even triggered the revocation of the leaked secrets.

Although GitHub notifications provide immediate feedback in case of secret leaks, some participants reported, that they discovered the secret leak only some weeks after the notification. "*It was probably out there for a couple of weeks. So, yes, that was not amazing.*" (I11). Furthermore, some participants noticed their leak randomly while they did another task, e. g., reading the logs to debug a program error . In addition, a few participants were informed by random people that saw their secret in the repository, e. g., "*Actually, it was someone who saw it for me because it was really recently.*" (I13). Moreover, a participant got informed by a third-party secret scanner.

Impact. We observed two different types of consequences caused by code secret leakage. Consequences may directly affect the company or software team that is responsible for the leak. However, we also saw consequences for external stakeholders, such as clients of the developed software or the customers of that client. Most of our participants reported, that the secret leak caused additional workload for the team. This included the investigation of the leak, as well as the remediation process. Furthermore, some participants told us, that their company or their team suffered from financial damage. A few participants reported that they experienced reputational damage to their company, e. g.,

"*All these issues were there, and we had to tell our clients that this happened, and we had to release it in that there was a security breach [...].*"— I11

Besides these consequences, one participant told us that their leaked secrets got used to crypto mine on their systems. In a few cases, external stakeholders got affected since they had to renew all customer and client secrets to prevent misuse.

Root Causes, Access, and Threat Model. There were various reasons for code secret leakage. Most participants reported that their leaks happened because developers, especially new developers that recently joined the team, were not aware of the consequences of a leak. Furthermore, most participants stated that they did not use any prevention approaches before an incident happened. In summary, developers leaked code secrets because they hard-code secrets in source code, or they did not add secret-containing files to a *.gitignore* file.

Access control configurations for code secrets and the threat model of the developers can affect the likelihood of code secret leakage. Most participants could describe a threat model for their use cases and included sharing secrets as a risk. Some of them have placed special emphasis on confidentiality. One participant stated that code secrets were not critical to them at all.

Some developers granted access to secrets on request. Few participants reported that their companies or team considered all employees as trusted and granted them full access to all secrets, e. g., "*Really just any time you ask, you'll just get access to whatever you want.*" (I6).

5.3 Secret Leakage Remediation

All interviewees experienced code secret leakage. Below, we report on the developers' experiences during the remediation approaches, their challenges, and lessons learned. Most participants reported to have revoked or updated the affected code secrets after they got aware of the leakage. Some immediately implemented remediation. They also analyzed root causes to prevent further future leaks. Few participants remediated the leak by taking down the repository or making the repository private. Moreover, few reported developer education measures on handling secrets securely. In addition, some participants removed the leaked code secrets by rewriting git history or deleting the secret with a new commit. Notably, a small number did neither revoke nor update leaked secrets. Participants repeatedly reported misconceptions of how *git* works.

While they used several approaches, the kinds of incidents can be various, including the consequences caused by the leak. Depending on the leakage and the consequences, participants chose different remediation approaches.

Challenges. Overall, most participants described the process of remediation as cumbersome. Several complained that they were facing an incident response process which they had never used before and was complicated to use. Some participants reported, that it was hard to estimate the consequences of their incident. Without being aware of all consequences, it was difficult to implement remediation, e. g.,

> "*It was a challenge not being able to say with 100% certainty that these secrets had never been misused [. . .]. There are scenarios where somebody could've [. . .] used it in such a way that it would be very difficult to detect, and we would have missed it. I didn't like that feeling that I couldn't say with certainty there was no impact.*"— I4.

Moreover, few participants complained, that no all-in-one solution exists for any kind of incident. Selecting, learning, and applying different or multiple remediation approaches would be too complex and time-consuming.

Lessons Learned. Most participants reported that their remediation process worked well in general. Of those, a few would apply the same process for future incidents. Besides that, participants requested changes for future incidents. While many participants expressed the need for better tooling to faster and easier remediate an incident, a few generally requested secret scanning to prevent or at least faster remediate leakage.

6 Discussion

This section discusses our findings, makes recommendations for developers and service providers, and provides ideas for future research. We base the discussion on both studies, using their individual findings to complement each other.

Encountering Code Secret Leakage. Our first point of discussion returns to the 30.3% of survey respondents that experienced code secret leakage, which turns out to be highly prevailing (cf. Section 4.3) compared to related work. One reason for this is that previous work [5, 13–16] focused on detectable secrets, for example, API keys, which were included in our survey. However, we additionally asked for credentials that need to be shared, and encryption keys a program needs to access. Our participants reported relying on the externalization, blocking, and encryption of secrets. Monitoring secrets through code scanners was less commonly mentioned, which we relate to the high false positive rates [5, 13] that make manual developer review necessary, e. g.,

> "*Most of the time, it just raises warnings about some secrets that are really supposed to be in the code and you have to manually exclude it from being scanned.*"— I13.

Usability and Adoption Aspects. Few participants reported significant challenges with secret handling approaches. Many participants deployed specific approaches due to good usability. At the same time, using approaches that participants used before or already knew about was a major theme. This might indicate an adoption burden that developers could not be willing to overcome, since light-weight approaches like blocking secrets via VCS (e. g., `.gitignore`) were adopted often while more resource-intensive approaches were not. For example, using short-living secrets that are rotated regularly, is a good fit to reduce the temporal attack surface and therefore potential secret leakage's impact – but was reported rarely. However, this would require more automation. While *Secret-as-a-Service (SaaS)* solutions and secret management tools like *HashiCorp Vault* can provide this out-of-the-box, developers, especially in small development teams, might not use them because they require additional setup and learning (cf. Section 5.1). Our findings regarding tool usage are consistent with previous research that focused on security tool adoption [60, 61]. Overall, we believe that approaches need to be light-weight to be adopted, or ideally require no developer effort at all. An excellent example for the latter is GitHub's secret scanning program, which is enabled by default for all public repositories [62] – therefore driving adoption at scale.

Secret Leakage Reporting. Interestingly, we found a mismatch in our survey between the low number of reports with problems using a code secret handling approach (7, 6.4%) in relation to the number of people that experienced leakage (33, 30.3%). Developers could experience leaks in their teams, for example, through a team member accidentally leaking hard-coded data not sufficiently secured through secret management approaches. Possible explanations include that developers do not relate the experienced code secret leakage with failed secret handling approaches. As we found in our interviews, they may also have not tried to use any approach, so they could not fail, e. g., "*We were a startup, [we didn't had any prevention approaches in place], we took all the measures after the secret leakage.*" (I2). Moreover, as a final factor, developers may have used an insufficient approach, so they have not failed this approach directly, but leaked a code secret nevertheless. If developers are unaware of problems or do not report them, it does not mean there are no issues.

Access Control Models. Some developers reported not feeling responsible for secret leakage. Instead, other team members, other teams, management, and clients had access to secrets and caused leaks. This is likely caused by secrets being shared as part of the development or deployment process (cf. Figure 4). To better support developers with this, future research should work on secure and easy-to-use secret-sharing and management platforms providing secret transfer and revocation between involved stakeholders.

Lack of Helpful Resources. In both the survey and the interviews, we found a lack of comprehensive resources that guide developers in the case of secret incidents. This is a burden for developers when they experience secret leakage. Although secret leakage is widespread, developers do not encounter it on a daily basis; they cannot be expected to instantly know what measures to take. Nonetheless, an incident ideally requires instant action. Therefore, we argue that easily accessible online resources are needed, and should contain actionable steps for easy, fast, and secure remediation. We think that code hosting platforms are a place to provide such information. This also holds for prevention approaches: developers complained about insufficient or missing documentation for deployment or usage (cf. Section 5.1 and 5.3).

Constructive Incident Handling. One interesting consequence of secret leakage was the firing of a full team or the complete termination of client contracts as a reaction to a secret leakage, potentially caused only by a single person. Considering our findings on the resources for secret leakage in general, we assume that this is mainly a problem with education and awareness, so *systemic consequences* like these are unlikely to prevent further leakage, especially considering that team members or clients are usually replaced with new, potentially less experienced stakeholders. Furthermore, it may discourage developers from reporting secret leakage to other team members or leaders, e. g., "*I didn't ask anyone, I knew what to do, I just responded directly.*" (I5). This can be highly problematic considering that the same participant applied insufficient remediation approaches and others could still misuse the leaked secrets. We propose to instead use restrictive *access management*, support through security teams, and education of team members and clients to prevent issues in companies [63].

Developer Awareness. Developers must be more aware of the risk and consequences of code secret leakage. After studying the experiences, challenges, and needs that developers had with prevention approaches, the human factor seems to be the most relevant regarding code secret leakage prevention, e. g.,

> "*Even with all the technology [...] to prevent secret leakage, the biggest contributor to secret leakage is the human factor, or negligence.*"— I2.

Companies need to properly onboard new developers to reduce the risk of secret leakage by training them or by policies (cf. Section 5.1.1). They also need to be trained and should be supported in developing and understanding threat models [64]. Typically, companies can also strengthen their cybersecurity advocates, as recommended by Haney et al. [65].

Manual vs. Automation. We found that developers desire increased automation for code secret leakage prevention. However, usability and maintenance challenges make the use of automated approaches, including code secret scanners, vaults, and automated rotation services, complicated. Manual approaches, such as restrictive access models and manual blocking of secrets in the repository, are subject to human error. Our findings illustrate the challenges of making the right trade-off decisions between security, ease of use, and maintainability of manual and automated approaches. Ultimately, the approach should be based on the developer or team's specific use case, considering their unique requirements and constraints.

6.1 Recommendations for Developers

We derive recommendations based on the approaches that we identified (cf. Table 3). We discuss these recommendations for more secure and usable code secret management for developers and focus on approaches that our survey and interview participants found to be secure and usable.

Prevention. We suggest combining different approaches to decrease the likelihood of code secret leakage. First, developers should *externalize secrets*, e. g., using environment variables or tools like vaults and secret managers, and *block secrets* in the repositories using, e. g., .gitignore files. These approaches can both prevent accidental commits. *Monitoring* , e. g., using secret scanners, especially as a pre-commit approach, can provide additional security. Some scenarios require developers to share code secrets with others. In such cases, we recommend to *encrypt secrets*, so unauthorized third parties cannot access them. Tools like e. g., git-secret, SOPS and GPP support comfortable secret encryption.

Remediation. Typical steps that should always be taken to remediate code secret leakage effectively are to *renew or revoke secret*s that leaked to prevent further misuse of affected services, *analyze leak*s to identify the root causes of the leak, and *revise the access management* using those results, e. g., apply more restrictive access management if needed. We also consider it essential to *notify the concerned roles* (e. g., management, security team, customers) for legal and ethical reasons, if not to get the appropriate help from security and privacy experts. Overall, we consider the above steps necessary because these steps will handle all consequences of a secret leak. The other approaches (cf. Table 3) can be used to complement the essential ones and should be considered as a second step in the circumstantial situation. *Removal from source code* and *cleaning up VCS history* are important steps. However, they cannot save a leaked secret on GitHub or similar platforms since those services are frequently crawled for archival purposes [5]. This, and the risk of archiving of public websites emphasize the need to *renew or revoke secret*s that have leaked in public spaces. *Server Operations* and *systemic consequences* (e. g., introducing new processes) depend heavily on company policies, the type of leak, and how well leakage damage can be prevented when developers can just *renew or revoke secret*s that have leaked. In case developers have remediation approaches prepared, we suggest testing them to make sure they work as expected in case of a leak.

6.2 Recommendations for Service Providers

Based on our findings, we discuss recommendations for improved code secret management for service providers.

Improving Online Documentation. We identified a lack of available documentation for secret leakage countermeasures. As platforms are the central instance, those can potentially reach many developers and should therefore provide easy-to-understand, accessible, and actionable guidance on secret leakage prevention and (especially) remediation. We believe that the approaches and recommendations for developers identified in this paper could be a good starting point for providing comprehensive guidance by service providers.

Provide and Extend Secret Scanning. We highly appreciate the platforms' effort in deploying large-scale secret *monitoring*, e. g., GitHub [62] and GitLab [66], which automatically scan public repositories for secrets and notify developers in case of a leak. In the example of GitHub, secrets with a known format are scanned using regular expressions. Those formats are supplied by several partnering service providers and limited to their API keys and access tokens. Additionally, the tokens are checked by the partner services for validity and automatically revoked if valid [67]. We believe this to be a suitable approach as it allows fully automated and, therefore, instant remediation. This is currently limited to a set of secrets and would be better if it included more types of secrets, like SSH keys. Although those cannot be revoked automatically (there is no central service provider), at least notifying the developers would be possible.

6.3 Outlook

Usability of Prevention and Remediation Approaches. To investigate and improve the Those would have to solve a programming task applying different code secret leakage prevention approaches to measure and compare their usability, or have to remediate a given secret leak.

Improving Secret Detection and Leakage Prevention. As discussed in related work (Section 2), general secret detection has high false-positive rates [13–16]. Future work should aim to improve detection accuracy so that platforms and developers can get useful secret scanners at hand. Another aspect that needs to be researched is secret leakage prevention. One approach is to develop and evaluate API designs that aim to prevent secret leakage by forcing a strict separation of code and data, e. g., the secrets. This may ensure security by default. The appropriate time for both secret scanning and prevention remains uncertain.

Comparison of Supporting Tools. Several tools support our identified approaches, e. g., secret scanners or vaults to externalize secrets and enable automatic rotation. A user study could investigate the challenges of using these tools. Additionally, how supportive they are in preventing and remediating code secret leakage could be measured.

7 Conclusion

In our online survey with 109 experienced software developers, we learned about their experiences with code secret leakage. We identified nine approaches developers use to prevent code secret leakage and nine approaches for code secret leakage remediation. 30.3% of the survey respondents experienced code secret leakage in the past. In 14 in-depth, semi-structured interviews with developers who experienced code secret leakage, we identified several problems and challenges that developers face when preventing and remediating code secret leakage. We make recommendations for both developers and service providers and outline ideas for future research based on our analysis. Overall, we strongly recommend that developers take preventative measures before any secret is accidentally leaked and be aware of the risks associated with leaking secret information.

Acknowledgments

We want to thank all survey participants and interviewees for supporting our research. Furthermore, we thank the anonymous reviewers and our shepherd for their constructive feedback. This research was funded in part by the VolkswagenStiftung Niedersächsisches Vorab – ZN3695, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972, and NSF grants CNS-2206865 and CNS-22 07008. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

Availability

To allow full replication of our research as well as meta-research, we provide a replication package at `https://doi.org/10.25835/xfc2h3pg`.

The replication package includes:

1. The full survey and interview recruitment materials (including Upwork post and invitation, as well as GitHub invite messages).
2. The survey screening questions and interview pre-survey questionnaire.
3. The survey and interview consent form.
4. The survey questionnaire and interview guide.
5. The survey and interview codebook.
6. The background section on version control, source code platforms, and secret information.

References

[1] Git Project. *Git*. `https://git-scm.com/` (visited on 02/02/2023).

[2] Stack Overflow. *Stack Overflow Developer Survey 2022*. `https://survey.stackoverflow.co/2022/` (visited on 08/08/2022). 2022.

[3] GitHub Inc. *GitHub*. `https://github.com/`.

[4] GitLab Inc. *GitLab*. `https://gitlab.com/`.

[5] Michael Meli, Matthew R McNiece, and Bradley Reaves. "How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories." In: *NDSS*. 2019.

[6] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation". In: *Proc. 26th Network and Distributed System Security Symposium (NDSS'19)*. 2019.

[7] Dwayne McDaniel. *Toyota Suffered a Data Breach by Accidentally Exposing A Secret Key Publicly On GitHub*. `https://blog.gitguardian.com/toyota-accidently-exposed-a-secret-key-publicly-on-github-for-five-years/` (visited on 02/02/2023).

[8] Mike Hanley. *We updated our RSA SSH host key*. `https://github.blog/2023-03-23-we-updated-our-rsa-ssh-host-key/` (visited on 05/16/2023).

[9] Eyal Katz. *8 Proven Strategies To Protect Your Code From Data Leaks*. `https://spectralops.io/blog/8-proven-strategies-to-protect-your-code-from-data-leaks/` (visited on 02/01/2023).

[10] Beata Berecki. *Best Practices for Source Code Security*. `https://www.endpointprotector.com/blog/your-ultimate-guide-to-source-code-protection/` (visited on 02/06/2023).

[11] CheatSheets Series Team. *Secrets Management Cheat Sheet*. `https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html` (visited on 02/06/2023).

[12] Mackenzie Jackson. *Best practices for managing and storing secrets including API keys and other credentials*. `https://blog.gitguardian.com/secrets-api-management/` (visited on 02/06/2023).

[13] 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*. 2015, pp. 396–400.

[14] Aakanksha Saha, Tamara Denning, Vivek Srikumar, and Sneha Kumar Kasera. "Secrets in Source Code: Reducing False Positives using Machine Learning". In: *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*. 2020, pp. 168–175.

[15] Sofiane Lounici, Marco Rosa, Carlo Negri, Slim Trabelsi, and Melek Önen. "Optimizing Leak Detection in Open-source Platforms with Machine Learning Techniques". In: *Proc. 7th International Conference on Information Systems Security and Privacy (ICISSP)*. SciTePress, 2021.

[16] Sabrina Kall and Slim Trabelsi. "An Asynchronous Federated Learning Approach for a Security Source Code Scanner". In: *Proc. 7th International Conference on Information Systems Security and Privacy (ICISSP)*. SciTePress, 2021.

[17] Runhan Feng, Ziyang Yan, Shiyan Peng, and Yuanyuan Zhang. “Automated Detection of Password Leakage from Public GitHub Repositories”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 175–186.

[18] Md Rayhanur Rahman, Nasif Imtiaz, Margaret-Anne Storey, and Laurie Williams. “Why secret detection tools are not enough: It’s not just about false positives-An industrial case study”. In: *Empirical Software Engineering* 27.3 (2022), p. 59.

[19] S. Basak, L. Neil, B. Reaves, and L. Williams. “What are the Practices for Secret Management in Software Artifacts?” In: *2022 IEEE Secure Development Conference (SecDev)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2022, pp. 69–76.

[20] Hala Assal and Sonia Chiasson. “Security in the Software Development Lifecycle”. In: *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security*. SOUPS ’18. Baltimore, MD, USA: USENIX Association, 2018, pp. 281–296.

[21] 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*. CHI ’19. Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–13.

[22] Julie M. Haney, Simson L. Garfinkel, and Mary F. Theofanos. “Organizational practices in cryptographic development and testing”. In: *2017 IEEE Conference on Communications and Network Security (CNS)*. 2017, pp. 1–9.

[23] Julie M. Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. “"We make it a big deal in the company": Security Mindsets in Organizations that Develop Cryptographic Products”. In: *Fourteenth Symposium on Usable Privacy and Security*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 357–373.

[24] Andrew Ruef, Michael Hicks, James Parker, Dave Levin, Michelle L Mazurek, and Piotr Mardziel. “Build it, break it, fix it: Contesting secure development”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 690–703.

[25] Daniel Votipka, Kelsey R. Fulton, James Parker, Matthew Hou, Michelle L. Mazurek, and Michael Hicks. “Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It”. In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 109–126.

[26] Hernan Palombo, Armin Ziaie Tabari, Daniel Lende, Jay Ligatti, and Xinming Ou. “An Ethnographic Understanding of Software (in) Security and a Co-Creation Model to Improve Secure Software Development”. In: *Proceedings of the Sixteenth USENIX Conference on Usable Privacy and Security*. SOUPS’20. USA: USENIX Association, 2020.

[27] D. Wermke, N. Wohler, J. H. Klemmer, M. Fourne, Y. Acar, and S. Fahl. “Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects”. In: *2022 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2022, pp. 1880–1896.

[28] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. “Why Do Developers Get Password Storage Wrong?: A Qualitative Usability Study”. In: *Proc. 24th ACM Conference on Computer and Communication Security (CCS’17)*. ACM, 2017.

[29] Tamara Lopez, Thein Tun, Arosha Bandara, Levine Mark, Bashar Nuseibeh, and Helen Sharp. “An Anatomy of Security Conversations in Stack Overflow”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. 2019.

[30] Qualtrics LLC. *Qualtrics*. `https://www.qualtrics.com/`.

[31] Stanley Presser, Mick P Couper, Judith T Lessler, Elizabeth Martin, Jean Martin, Jennifer M Rothgeb, and Eleanor Singer. “Methods for testing and evaluating survey questions”. In: *Public opinion quarterly* 68.1 (2004), pp. 109–130.

[32] Upwork Global Inc. *Upwork*. `https://www.upwork.com/` (visited on 02/01/2023).

[33] Upwork Global Inc. *Software Developer Hourly Rates*. `https://www.upwork.com/hire/software-developers/cost/` (visited on 02/01/2022).

[34] Dominik Wermke, Noah Wöhler, Jan H. Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. “Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects”. In: *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*. IEEE Computer Society, May 2022.

[35] Xunhui Zhang, Tao Wang, Yue Yu, Qiubing Zeng, Zhixing Li, and Huaimin Wang. “Who, What, Why and How? Towards the Monetary Incentive in Crowd Collaboration: A Case Study of Github’s Sponsor Mechanism”. In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI ’22. New Orleans, LA, USA: Association for Computing Machinery, 2022.

[36] Kathy Charmaz. *Constructing Grounded Theory*. SAGE Publications, 2014.

[37] Anselm Strauss and Juliet M Corbin. *Grounded theory in practice*. SAGE Publications, 1997.

[38] Juliet Corbin and Anselm Strauss. "Grounded theory research: Procedures, canons and evaluative criteria". In: *Zeitschrift für Soziologie* 19.6 (1990), pp. 418–427.

[39] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. "Reliability and Inter-Rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice". In: *Proc. ACM Hum.-Comput. Interact.* 3.CSCW (Nov. 2019).

[40] Melanie Birks and Jane Mills. *Grounded Theory: A Practical Guide*. Jan. 2015.

[41] Cathy Urquhart. *Grounded Theory for Qualitative Research: A Practical Guide*. Jan. 2013.

[42] Dominik Wermke, Jan H. Klemmer, Noah Wöhler, Juliane Schmüser, Harshini Sri Ramulu, Yasemin Acar, and Sascha Fahl. ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain". In: *In Proceedings of the 44th IEEE Symposium on Security and Privacy (IEEE S&P '23)*. IEEE Computer Society, May 2023.

[43] Nicolas Huaman, Alexander Krause, Dominik Wermke, Jan H. Klemmer, Christian Stransky, Yasemin Acar, and Sascha Fahl. "If You Can't Get Them to the Lab: Evaluating a Virtual Study Environment with Security Information Workers". In: *Eighteenth Symposium on Usable Privacy and Security, SOUPS 2022, Boston MA, USA, August 8-9, 2022*. Boston, MA: USENIX Association, Aug. 2022.

[44] Marco Gutfleisch, Jan H. Klemmer, Niklas Busch, Yasemin Acar, M. Angela Sasse, and Sascha Fahl. "How Does Usable Security (Not) End Up in Software Products? Results From a Qualitative Interview Study". In: *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*. IEEE Computer Society, May 2022.

[45] Erin Kenneally and David Dittrich. "The Menlo Report: Ethical principles guiding information and communication technology research". In: *SSRN Electronic Journal* (Aug. 2012).

[46] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. "Comparing the Usability of Cryptographic APIs". In: *Proc. 38th IEEE Symposium on Security and Privacy (SP'17)*. IEEE, 2017.

[47] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. "Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse". In: *Proc. 14th Symposium on Usable Privacy and Security (SOUPS'18)*. USENIX Association, 2018.

[48] GitHub. *GitHub Acceptable Use Policies*. `https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies` (visited on 05/05/2023.

[49] Elissa M. Redmiles, Ziyun Zhu, Sean Kross, Dhruv Kuchhal, Tudor Dumitras, and Michelle L. Mazurek. "Asking for a Friend: Evaluating Response Biases in Security User Studies". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. ACM, 2018.

[50] Mariana Peixoto, Dayse Ferreira, Mateus Cavalcanti, Carla Silva, Jéssyka Vilela, João Araújo, and Tony Gorschek. "On understanding how developers perceive and interpret privacy requirements research preview". In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. REFSQ 2020. Springer, 2020.

[51] Awanthika Senarath and Nalin A. G. Arachchilage. "Why developers cannot embed privacy into software systems? An empirical investigation". In: *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. EASE'18. ACM, 2018.

[52] Abraham H. Mhaidli, Yixin Zou, and Florian Schaub. ""We Can't Live Without Them!" App Developers' Adoption of Ad Networks and Their Considerations of Consumer Risks". In: *Proc. 15th Symposium on Usable Privacy and Security (SOUPS'19)*. USENIX, 2019.

[53] Harjot Kaur, Sabrina Amft, Daniel Votipka, Yasemin Acar, and Sascha Fahl. "Where to Recruit for Security Development Studies: Comparing Six Software Developer Samples". In: *31st USENIX Security Symposium, USENIX Security '22, Boston MA, USA, August 10-12, 2022*. USENIX Association, Aug. 2022.

[54] HashiCorp. *HashiCorp Vault*. `https://www.vaultproject.io/` (visited on 02/01/2023).

[55] Amazon Web Services. *AWS Systems Manager*. `https://docs.aws.amazon.com/systems-manager` (visited on 02/01/2023).

[56] GitHub Inc. *GitHub Workflow*. `https://docs.github.com/en/actions/using-workflows` (visited on 02/01/2023).

[57] Pastebin. *Pastebin.* https://pastebin.com/ (visited on 02/01/2023).

[58] GitHub Inc. *GitHub gists.* https://gist.github.com/ (visited on 02/01/2023).

[59] Amazon Web Services. *Amazon Web Services.* https://aws.amazon.com (visited on 02/01/2023).

[60] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. "Quantifying Developers' Adoption of Security Tools". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: Association for Computing Machinery, 2015, pp. 260–271.

[61] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. "Social Influences on Secure Development Tool Adoption: Why Security Tools Spread". In: *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*. CSCW '14. Baltimore, Maryland, USA: Association for Computing Machinery, 2014, pp. 1095–1106.

[62] GitHub Inc. *Secret Scanning.* https://docs.github.com/en/code-security/secret-scanning/about-secret-scanning (visited on 02/01/2023).

[63] Nicolas Huaman, Bennet von Skarczinski, Christian Stransky, Dominik Wermke, Yasemin Acar, Arne Dreißigacker, and Sascha Fahl. "A Large-Scale Interview Study on Information Security in and Attacks against Small and Medium-sized Enterprises". In: *Proc. 30th Usenix Security Symposium (SEC'21).* USENIX Association, 2021.

[64] Adam Shostack. "Elevation of Privilege: Drawing Developers into Threat Modeling". In: *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14).* San Diego, CA: USENIX Association, Aug. 2014.

[65] Julie M. Haney and Wayne G. Lutters. ""It's Scary…It's Confusing…It's Dull": How Cybersecurity Advocates Overcome Negative Perceptions of Security". In: *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018).* Baltimore, MD: USENIX Association, Aug. 2018, pp. 411–425.

[66] GitLab Inc. *Secret Detection.* https://docs.gitlab.com/ee/user/application_security/secret_detection/ (visited on 02/01/2023).

[67] GitHub Inc. *Secret scanning partner program.* https://docs.github.com/en/developers/overview/secret-scanning-partner-program (visited on 02/01/2023).

A Analysis: Online Resources on Secret Handling

As part of the exploration phase for this study, we researched online guides and documents in summer of 2021 that cover secret leakage prevention and remediation, e. g., discussing best practices. While we mainly used the insights for the construction of survey questions and the interview guide, the guide analysis itself yielded some minor results that we therefore want to report here for completeness.

Interestingly, the online resources covered only a subset of approaches compared to the survey (cf. Table 3). Regarding leakage prevention, all but *Code & Secret Reviews* were mentioned. Contrary to the survey, *Block Secrets* (12 documents) was the most popular approach, followed by *Monitoring* (11). *Externalizing* and *encrypting* secrets both occurred in 9 documents, and *restricting access* in 4. The remaining approaches occurred only once. Regarding leakage remediation, only four of the overall nine approaches from the survey could be found. As in the survey, *Renew or Revoke Secret* is the most common approach (7 documents). This is followed by *cleaning VCS history* (2), *analyzing leaks* in detail (1), and *access management* considerations (1). That said, online resources are seemingly more incomplete for remediation of an incident than for prevention. However, the online guides covered the important approaches and lacked only minor ones.

Overall, we found only a few resources on handling code secrets. Many of them were incomplete (only covering one or a few approaches) or inconsistent with each other. All this indicates a lack of helpful information sources in terms of secret handling for developers.

B Interview Questions

In the following, we present the interview questions used to conduct the semi-structured interviews. The initial demographic and screening pre-survey including the consent form, and the complete interview guide can be found in the replication package (cf. *Availability* section). We used a numbering format where *S* stands for section and *Q* for question.

S1 Code Secrets

S1Q1. Secret Types: Please tell us about the kind of secret information you come into contact with when writing and maintaining your source code.

Definition Code Secret: A code secret is any secret information that your program needs to access without user-input. For example, this could be an API key or a private key.

- ☐ **S1Q1.1 Use Cases:** Please provide some typical examples of where you used code secrets before?
- ☐ **S1Q1.2 Sensibility:** What do you think about the "sensibility" of these secrets? How confidential/critical are these secrets?
- ☐ **S1Q1.3 Access:** How do you decide on components or users that can access these code secrets?

- □ **S1Q1.4 Participants:** Who (components and people) typically had access to these secrets?

S2 Code Secret Leakage & Recommendation

You are here because you experienced code secret leakage in the past.
Definition: Code Secret Leakage refers to leaking code secrets, for example through source code repositories, build scripts or CI and similar approaches to source code sharing.
Examples: Also, a push of an API-Key to a publicly available source code platform without consequences is considered as a code secret leak. Or if the leak is inside a company platform (e.g., pushing to an internal repo where only other members of the company have access).

To better understand the prevalence of code secret leakage, we would like to know how often you have experienced it.

S2Q1. Prevalence: How often did you experience secret leakage?

Please elaborate on your most impactful or latest code secret leakage and the experiences you had with it.

S2Q2. Becoming aware: How did you recognize that a secret leakage happened?

S2Q3. Experience

- □ **S2Q3.1 Reason:** How did the code secret leak happen?
- □ **S2Q3.2 Consequences:** Can you describe the immediate consequences?
  - **S2Q3.2.1 For the Company:** What consequences were there for the company? Cyberattacks? Monetary damage?
  - **S2Q3.2.2 Authorities (If they got attacked):** Did you attempt to contact authorities/ prosecute the attackers?
  - **S2Q3.2.3 For external Stakeholders:** Data Leakage/Monetary Damage/ Other inconvenience for the client or other parties?
- □ **S2Q3.3 Changes:** Were there any new measures/approaches introduced to prevent secret leakage in the future?

S2Q4. Remediation: How did you react to the incident? How did you remediate the code secret leak?

- □ **S2Q4.1 Experiences:** What were your experiences when applying the approach(es)? What went well?
- □ **S2Q4.2 Involved Roles:** Who was actually involved in remediating the code secret leak?
- □ **S2Q4.3 Challenges:** Did you encounter any challenges? Why?
- □ **S2Q4.4 Resources:** Please tell us about the information sources you used to remediate the code secret leak. In example, online blogs, official documentation, other developers, or the it-security team / incident response team.
- □ **S2Q4.5 Satisfaction:** If a code secret leak happen again, would you like to apply the same process for future code secret leakage remediation, or is there something you would like to change or improve?

S3 Prevention Approaches

Let's talk about your experiences regarding code secret leakage prevention approaches.

S3Q1. Used approaches: If you used prevention approaches before, please provide an example of approaches or tools you used to prevent code secret leakage. *(If missing, provide examples: Externalize Secrets, Block Secrets, Monitoring, Restrict Access)*

- □ **S3Q1.1 Understanding:** Please describe how you used the approach/tool in your project and why?
- □ **S3Q1.2 Reason:** What was the rationale, or reason, why this approach was introduced in your processes?
- □ **S3Q1.3 Experiences:** Please tell us about the experiences with these approaches and tools. Would you consider it easy to use, effective, or secure?
- □ **S3Q1.4 Challenges:** Please tell us about any challenges or problems you faced.
- □ **S3Q1.5 Satisfaction:** Does the approach fulfill your needs, or do you desire changes to improve the approach?

S3Q2. Used approach and failed with: Have you had problems trying to apply an approach or failed with an approach? Please tell us about them.

- □ **S3Q2.1 Reason:** What was the reason you decided for the approach you failed with?
- □ **S3Q2.2 Causes of Failure:** What caused the approach to fail (in your specific accident)?
- □ **S3Q2.3 Improvements:** What changes do you suggest could improve the concept?

S3Q3. Known, but unused approaches: Can you name any further approaches that you did not use so far and why?

- □ **S3Q3.1 Reason**: Why did you decide not to use those?
  - **S3Q3.1.1 Future Use:** Do you want to try them for future projects?
  - **S3Q3.1.2 Experiences:** Do you think the approach would be easy-to-use?
  - **S3Q3.1.3 Challenges:** Do you think you might encounter any challenges or problems along the way? What are these challenges?
- □ **S3Q3.2 Satisfaction:** Do you think the approach would fulfill your needs, or are you requesting changes to improve the approach?

S3Q3 [IF in S3Q1 stated]. Secret Scanner: A popular approach to detect secret leaks, are so-called secret scanners, that scan code, repositories, or commits for any contained secrets. What are your experiences if you have any?

- □ **S3Q3.1 Utilization:** When is this scanner executed in your project? (CI/CD, pre-commit, regular, etc.?)
- □ **S3Q3.2 False Positives:** Do you think the scan results are reliable? Why?
- □ **S3Q3.3 Challenges:** Did you encounter any challenges or problems along the way? What are these challenges? Have you solved them?

S3Q4. Not used: [If not used any approaches/tools]: Have you ever considered using an approach to prevent code secret leakage? Can you elaborate on your decision?

S3Q5. Needs: Looking back at all your experiences with code secret leakage and its prevention, what do you currently miss or think might be helpful to successfully prevent code secret leakage? *(Then if nothing comes to their mind, we could specifically ask for resources, tools, approaches.)*