It Should Be Easy but... New Users' Experiences and Challenges with Secret Management Tools

Lorenzo Neil
North Carolina State University
Raleigh, United States of America
lcneil@ncsu.edu

Deepthi Mungara
Paderborn University
Paderborn, Germany
deepthi.mungara@uni-paderborn.de

Laurie Williams
North Carolina State University
Raleigh, United States of America
lawilli3@ncsu.edu

Yasemin Acar
Paderborn University & The George Washington University
Paderborn, Germany
yasemin.acar@uni-paderborn.de

Bradley Reaves
North Carolina State University
Raleigh, United States of America
bgreaves@ncsu.edu

Abstract

Software developers face risks of leaking their software secrets, such as API keys or passwords, which can result in significant harm. Secret management tools (SMTs), such as HashiCorp Vault Secrets or Infisical, are highly recommended by industry, academia, and security guidelines to manage secrets securely. SMTs are designed to help developers secure their secrets in a central location, yet secrets leaks are still commonplace, and developers report difficulty in learning how to setup and use SMTs. While SMTs typically come with publicly available help resources (e.g., tool documentation and interfaces), it is unclear if these actually help developers learn to effectively use SMTs. Without usable help resources that onboards developers, quick adoption and effective use of SMTs may be unrealistic.

In a qualitative two-step study, we observed 21 new users in person while they used SMTs to perform two secret management tasks: secret storage and access, then secret injection. We interviewed participants after each task to identify their challenges and experiences using SMTs, with the assistance of help resources.

While our study sample is narrow, it serves as a reasonable proxy for new developers who are likely to adopt SMTs early in their careers. We found that even in a laboratory setting where new users found tool functionality and interface flexibility helpful, they still experienced increased difficulty to effectively use SMTs to securely remediate a hard-coded secret when they felt tool documentation was insufficient. Insufficient tool documentation motivated participants to deviate from official tool documentation to access secondary sources or attempt workaround methods. Specific challenges reported by participants were tool documentation content quality, navigation difficulties with both tool documentation and web interfaces for finding helpful content, and supportive tool features. We explain how these challenges negatively affect participant experiences adopting SMTs, and suggest recommendations on tool documentation and interfaces for SMT developers. If developers cannot simply and quickly manage secrets securely, secret leakage will continue to be commonplace.

CCS '25, Taipei
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765065

CCS Concepts

• **Security and privacy → Usability in security and privacy**.

Keywords

secret management tools; usable security; user interfaces; documentation

ACM Reference Format:
Lorenzo Neil, Deepthi Mungara, Laurie Williams, Yasemin Acar, and Bradley Reaves. 2025. It Should Be Easy but... New Users' Experiences and Challenges with Secret Management Tools. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3765065

1 Introduction

Software developers need to store software secrets (e.g., passwords, API keys), which grant access to sensitive or private information (e.g., account access, company bank information). Without a proper secret management approach in place, developers may hard-code secrets into source code, public repositories, or configuration files. A highly recommended option by industry, latest guidelines, and academic research [12, 14, 53] are secret management tools (SMTs). These can securely store sensitive developer secrets in a centralized location. However, even with the existence of SMTs, developers still leak secrets [14, 25, 47, 52, 53], which can lead to a wide exposure of sensitive information [1, 19, 48, 61, 64].

Like most software, SMTs come with online public documentation that onboards developers on how to store, manage, and distribute secrets. Prior research efforts have highlighted issues within software tool documentation that affect developers' abilities to use the tools to perform software development related tasks [2, 3, 28, 30, 72, 79, 84]. Limitations for software development tool documentation may extend to SMT tool documentation, yet the extent of the impact of the quality of SMT tool documentation on developers' experiences in using SMTs is unclear. Prior research shows that SMT adoption is widely a burden for developers, largely because of setup and learning constraints during initial adoption of the tools [14, 53]. If SMT tool documentation cannot minimize onboarding constraints, then new users will struggle to effectively use SMTs to centrally secure their secrets.

To understand the experiences and challenges new users face when using SMTs, we ran a qualitative two-step laboratory study to investigate the following research questions:

RQ1: *Can new users effectively use a SMT to securely remediate a hard-coded secret?*

RQ2: *What factors encourage or inhibit SMT usage success?*

We observed participants with prior secret management experience, using one of three cross-cloud SMTs: Doppler [26], HashiCorp Vault Secrets [41], and Infisical [46]. Participants performed two tasks that involved securing a hard-coded secret: Access a stored secret from the command line, then inject the secret into a local Python application. We chose these tasks to reflect a realistic onboarding scenario where a developer addresses a common issue involving a hard-coded database password in the codebase. This creates security risks through version control or logs [58, 69]. The study tasks involve securing the secret using an SMT and injecting it at runtime, covering essential skills such as secret storage, access, and cloud integration. These tasks are based on real workflows and demonstrate how developers use secret management tools to reduce vulnerabilities [78]. Following each task, we interviewed participants about their experiences and challenges using SMTs and SMT help resources.

Although documentation had always been a consideration in our study design, our focus on it sharpened during the study based on participant responses (Section 4.3). We observed that participants were generally satisfied with the different functionality and user interfaces provided by the SMTs. While we noticed observable differences in task completion times between SMTs, all were functionally indistinguishable towards task success. However, even while appreciating SMT functionalities and interfaces, new users in our study struggled to effectively use SMTs to securely remediate a hard-coded secret. Participant interviews revealed that their ability to fully remediate the hard-coded secret was mostly affected by the quality of the available tool documentation.

Major limitations in SMT tool documentation that negatively impacted SMT usage were the following: Insufficient information for relevant command line arguments, lack of a relevant set of use cases in examples provided for command line interface (CLI) commands, and inconsistent versions of descriptions for CLI commands maintained across documentation. Participants also reported a lack of clarity when using the tool documentation and web interfaces to find technical guidance such as specific commands or debug information. When encountering limiting factors in SMT tool documentation, participants often deviated from tool documentation to access secondary sources and attempt workaround methods. However, participants largely reported that secondary sources were less helpful for their specific needs and therefore experienced more time and effort to troubleshoot and explore solutions. Our study conclusions contribute towards addressing research areas both specific towards SMTs [12, 14, 53] and more broadly such as tool usability [2, 51, 100] and documentation [33, 39, 80]. Our work can serve as a reference for SMT vendors who need to create effective tool documentation and more user-friendly tool interfaces for their intended audience, regardless of their technical expertise, to enable broad, secure adoption of effective secret management tools. Thus, contributing to ensure secret leakage becomes less widespread.

2 Background and Related Work

We present a brief background on SMTs and discuss related work for secret management, software tool documentation, and security tool usability and adoption.

2.1 Secret Management

In response to secret leakage, recommended guidelines for software secret management from organizations [21, 83], coding platforms [35, 36, 60], and security blogs [52] were created to guide developers on how to secure secrets. Academic research has also addressed secret management leakage and practices. Meli et al. [64] conducted studies to characterize widespread secret leakage in public repositories. Similar work has outlined solutions [12, 88, 96] and proposed datasets [13] to help developers detect and prevent exposed secrets in their repositories. Basak et al. [14] characterized challenges and solutions reported in questions related to checked-in secrets on Stack Exchange, while Krause et al. [53] surveyed 109 developers and interviewed 14 developers to learn common challenges with managing and leaking secrets in code repositories.

Both of their works motivate this study, as they highlight that developers want to use SMTs but struggle with adoption, often citing a lack of available documentation as a key challenge [14, 53]. We further this effort by observing how participants use SMTs and the challenges they face in effectively managing secrets in direct relation to the available documentation.

2.2 Secret Management Tools

Secret management tools are highly recommended for securely storing sensitive software secrets in a centralized location [34, 59]. SMTs can offer different functionalities or use cases. Cloud-specific SMTs are built on major cloud ecosystems such as AWS [9], Google Cloud Platform [37], or Microsoft Azure [65], and are typically used by companies already integrated into those ecosystems. On the other hand, Cross-cloud SMTs are usually offered either as Software-as-a-Service (SaaS) or as a managed service that can be self-hosted or managed [59]. Cross-cloud SMTs are not tied to a specific cloud and offer easier onboarding for developers not working within large cloud ecosystems. Open source SMTs are self-hosted by developers, while in-house SMTs are custom-built by companies for their own use. While both give developers much creative freedom, these services must be regularly maintained by the company or developers themselves, therefore requiring significant overhead. Interview findings from Krause et al. [53] show that developers report setup and learning constraints when first adopting SMTs. For our work, we investigate new user experiences with three cross-cloud SMTs and the respective tool documentation.

2.3 Software Tool Documentation

Much of the prior work towards software documentation specifically investigated challenges with API focused software documentation that prevents developers from effectively and securely using

APIs [2–4, 42, 85, 90, 95, 97, 98]. Challenges highlighted from prior work that affect the quality of software documentation include, but are not limited to, insufficient coding examples [2, 4–6, 71, 95], incorrect or ambiguous explanations [22, 67, 98, 101], as well as non intuitive presentation of material [72, 74]. Such challenges within software documentation make it more difficult for developers to learn new technology [7, 22, 73] and in some cases affect developer productivity [42, 68, 76, 87, 99]. In 2003, Lethbridge et al. [56] interviewed software engineers to learn how they used software documentation. 61% of the participants in their study felt software documentation was most effective when learning new software and that they prefer simplified documentation, while tending to ignore complex or time consuming documentation [56]. More recent work highlights how developers seek both, official documentation and also other resources such as Stack Overflow [3, 7, 11, 86, 93] to learn new technology. Stack Overflow in particular has been highlighted in prior work as a resource developers access to find answers to challenges addressed by their community [79, 93]. Parnin et al. [79] examined crowd contributed API documentation from Stack Overflow and found that while the crowd documentation provided many examples and explanations for API elements, the rate of information produced from the crowd would not be fast enough to outright replace API documentation [79]. A lab study conducted by Acar et al. [3] observed Android developers writing security-and-privacy-relevant code under time constraints using different information sources. Their findings reported that developers who used Stack Overflow as a resource wrote less secure but more functional code as opposed to developers who used official Android documentation [3]. While we see the critical need to create effective software documentation, there is a lack of incentive or motivation to create such documentation [8, 79]. Arya et al. [8] interviewed 26 volunteer documentation contributors to learn their motivations for contributing to software documentation. Their work concluded that contributors to software documentation were mostly self-motivated, including personal experiences with inadequate software documentation or pursuits for content creation.

Challenges with software documentation content, presentation, and availability as highlighted from prior work serve as motivation for this work. We extend the body of literature by observing how challenges for SMT tool documentation affect our participants' abilities to use SMTs to perform tasks involving managing secrets. We also observe how tool documentation content challenges motivate participants to find secondary sources for further assistance or try workaround methods.

2.4 Security Tool Usability and Adoption

The usability of security tools impacts developers' ability to securely complete tasks [2, 23, 94, 102]. Storey et al. [94] surveyed software developers at a company to learn about challenges they faced in regards to their job satisfaction and perceived productivity. Challenges with software architecture and finding relevant information were two of the most common challenges their survey participants reported in regards to their job satisfaction and perceived productivity. Acar et al. [2] compared the usage of different cryptographic API usage by Python developers to write secure code and found that simplified libraries within the APIs helped developers produce more secure code than the comprehensive libraries. Related prior work also echoes the theme that simplifying information or the design of tools improves tool usability [45, 89]. In response, prior work has suggested new approaches aimed to help developers write more secure code when using cryptographic APIs [38, 54]. Gorski et al. [38] observed improvements in code security when comparing developers who used Python's PyCrypto API to developers with a version of PyCrypto with integrated security advice. Krüger et al. [54, 55] introduced CogniCrypt, their proposed tool for helping developers write code and implement tasks in a secure manner.

Much of the prior efforts on examining security tool usability and adoption focused on APIs or other specific areas. We use prior work as an influence to shape how we investigate the usability of SMTs, which has only been briefly discussed in prior work [14, 53]. We also examine how provided tool documentation and the availability of secondary sources impact the ability of participants to use SMTs for managing secrets.

3 Methodology

We conducted a qualitative two-step study on SMT use with 21 Computer Science (CS) Master's students in July 2024, consisting of an observation study followed by an interview study [17, 49]. Participants were observed while using SMTs, with the assistance of publicly available tool documentation from the SMTs, to perform two tasks: Task 1 involves storing and accessing a secret and Task 2 involves injecting a secret into a local application. After each task, we interviewed participants to learn about their experiences and any challenges they faced while completing each task.

The extended version of our paper includes the interview guide, codebook, and demographics [75]. The lab study protocol, SMT tool help documents, and related materials used in the study are available at https://doi.org/10.5281/zenodo.17018637.

3.1 Recruitment and Eligibility Criteria

Our population of interest were developers with academic or professional experience with managing secrets for coding projects. For feasibility, we focus our recruitment towards a sub-population: CS Master's students at our university. Many local CS Master's students either have experience managing secrets from their coding projects, prior employment positions, research projects, or academic courses, and are therefore an appropriate sample for our study. We focused on participants with prior secret management experience, so that true new SMT users can be expected to fare either the same or worse.

We recruited CS Master's students through a public email forum offered by our CS graduate office. We described our research overview and goals, and provided a link to our Qualtrics [82] screening survey for participants to indicate interest. We required participants to participate in person at our research campus and have either prior experience or general understanding of the following: Linux or MacOS terminal commands, Python file execution, environment variable usage, and secret management, all essential for performing our research tasks. Participants were only eligible if they met all of our criteria. Once eligible participants filled out our screening survey, we sent them a link to our Qualtrics [82] informed consent form. Our informed consent form asked for consent to participate in person and agree to be recorded through both video and audio. Participants were compensated $30 per half hour for their participation in the study.

We ended participant recruitment when we observed common usability issues and themes were repeated, after observing no new insights, from participants' performances to address **RQ1**, and from participants' interview responses to address **RQ2**. Participant demographics shows the breakdown of their prior professional experience, academic experience, gender, and the SMT they were assigned to use in the study. All but one of our participants had prior experience with managing secrets in a company, either through a prior internship or full time employment. Nine participants held a combination of both professional experience and academic coursework experience with managing secrets. They named passwords, tokens, API keys, and user credentials as examples of secrets they were tasked with managing in prior experiences. Participants mentioned using the secret management tools Amazon Web Services (AWS) Key Management Service [9], Azure Key Vault [65], and Google Cloud Platform (GCP) [37].

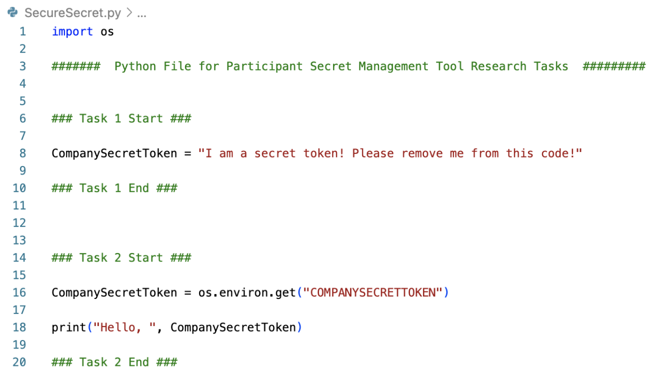
3.2 Study Design

3.2.1 SMT Choice. We chose the following three cross-cloud SMTs: Doppler [26], HashiCorp Vault (HCP) Vault-Secrets [41], and Infisical [46]. We chose cross-cloud SMTs because they do not require substantial experience or access with specific cloud providers. Hence, we avoided cloud-specific tools, such as AWS, which require substantial experience with that particular cloud deployment.

All three SMTs offer public online tool documentation and free versions of their tools. These three specific cross-cloud SMTs were also three of the more common cross-cloud SMTs [34, 59]. All three SMTs provide a web user-interface (UI) and installable command-line interface (CLI). The web UI allows users to create free accounts and store secrets. Secret storage dashboards allow users to create and manage projects that store their secrets. From the CLI perspective, users can access secrets from their web secret storage projects and distribute them across local applications without the need to store secrets locally.

3.2.2 Study Protocol. Our study uses a between-subjects research design [18], where each participant met with a researcher on campus and performed two research tasks with only one of the three SMTs. Therefore, seven participants performed two secret management related tasks with each SMT, totaling 21 participants overall. The Tables 1, 2 lists which SMTs participants used for our study.

We borrow inspiration from related work by providing participants with a scenario-based description for each task [3, 10, 77]. Each scenario imagines the participant working for a software company with the goal to use the provided SMT, with respective tool documentation, to remediate a hard-coded secret within a company Python file shown in Figure 1. The Python file, `SecureSecret.py`, consists of one or two lines of code for each task. One of the lines of code for Task 1 includes the hard-coded secret `CompanySecretToken`. Participants were provided with a MacBook Pro which held the Python file in a Visual Studio Code (VS) Text Editor work space [66]. Participants also were required to use the



```python
import os

#######  Python File for Participant Secret Management Tool Research Tasks  #########


### Task 1 Start ###

CompanySecretToken = "I am a secret token! Please remove me from this code!"

### Task 1 End ###



### Task 2 Start ###

CompanySecretToken = os.environ.get("COMPANYSECRETTOKEN")

print("Hello, ", CompanySecretToken)

### Task 2 End ###
```

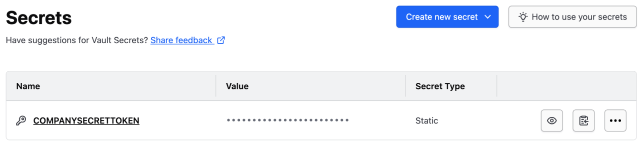
Figure 1: Screenshot image of the Python file, `SecureSecret.py`, used by participants for Task 1 and 2.

terminal prompt within VS to enter SMT-based CLI prompts to complete steps for each task. Lastly, participants were given account access to the web UI for their SMT which held the scenario-based project, "*Company Secret Storage*", they store `CompanySecretToken` in.

While we described the first task to participants, we provided them links to online tool documentation from the SMT they worked with. The links explicitly provide information for each task within their core CLI commands documentation, quick starter guides, and documentation that covers managing secrets or projects. We present these links to the participant in a Google Doc where we also provide labels for the titles of each link. The list of links we presented for Doppler, HCP Vault Secrets, and Infisical can be found in our replication package. We did this to help the participants reduce trivial lookup time and decrease the disadvantages of finding advice on different SMTs.

Participants were given 30 minutes to complete each task, a duration chosen based on pilot testing and complexity of tasks. We allowed stoppages of time when participants asked questions or wanted to take a break. We also allowed participants to skip a task or not answer a question. If participants reached a point in which they felt the SMT tool documentation was unclear or not helpful, we allowed participants to search online for secondary sources for more information. We allowed participants to use any website (e.g., YouTube or Stack Overflow) or medium (e.g., text or video). However, we did not allow participants to use generative AI services (e.g., ChatGPT). We set this restriction since not all participants may be familiar with such services and we wanted to ensure participants have the same resource availability and all know how to use their resources.

After participants completed a task, we asked them several questions about their experiences and challenges using the SMT and tool documentation. Participants were also asked a couple of warm-up questions before the tasks, which collected their prior experience with secret management. After participants answered questions about both tasks, they were then asked a set of wrap-up questions. We explain in detail our post task interview development and structure in Section 3.3.1 and 3.3.2.



Name	Value	Secret Type
COMPANYSECRETTOKEN	••••••••••••••••••••••••	Static

Figure 2: Screenshot image showing the HCP Vault Secrets dashboard for the *Company Secret Storage* project with `CompanySecretToken` stored.

```
Command:
    $ hcp auth login
Output:
    ....
    Successfully logged in!
Command:
    $ hcp profile init --vault-secrets
Output:
    ....
    ✓ App with name "CompanySecretStorage" selected
Command:
    $ hcp vault-secrets secrets open "COMPANYSECRETTOKEN"
Output:
    Secret Name:    COMPANYSECRETTOKEN
    ....
    Value:          I am a secret token! Please remove me from this code!
```

Figure 3: Sample presentation of the list of CLI commands required to complete Task 1 for HCP Vault Secrets.

3.2.3 Task Selection. We designed each task to be general and be equally applicable to all SMTs we study and to reflect real-world security practices, focusing on secure secret management and minimizing the risk of hardcoded secrets, a common security vulnerability [15, 69]. These tasks also reflect common, security-critical workflows that developers regularly perform when managing secrets in the development and deployment process [78]. Using both the UI and CLI as part of our task design enables us to uncover usability issues that could affect whether developers adopt security tools and how they use them over time. Both tasks were designed to be completed within 30 minutes. By evaluating how participants handle secret storage, retrieval, and injection, our study addresses the security and usability perceptions of developers when using SMTs. Participants also do not need extensive knowledge of the SMT to perform these tasks. With the tool documentation provided to participants, as well as secondary sources, participants were instructed to complete the following two tasks:

Task 1: Remove `CompanySecretToken` from `SecureSecret.py`, store `CompanySecretToken` within *Company Secret Storage*, and then access `CompanySecretToken` from the command line using the SMT CLI.

Task 2: Inject `CompanySecretToken` into `SecureSecret.py`.

Task 1 starts with participants removing `CompanySecretToken` from `SecureSecret.py`, as shown in Figure 1. For this, we simply have them delete it from the file. Participants then store `CompanySecretToken` using web UI of a SMT in the project named *Company Secret Storage* as shown in Figure 2. The last step for Task 1 is to then use the SMT CLI to print the plaintext value of `CompanySecretToken`. A presentation for the number of steps required for this task from the CLI perspective is shown in Figure 3.

Task 2 involves a process called "secret injection" [40]. For secret injection, a secret is retrieved from the SMT's central storage, and

```
Command:
    $ hcp vault-secrets run -- python3 SecureSecret.py
Output:
    Hello,  I am a secret token! Please remove me from this code!
```

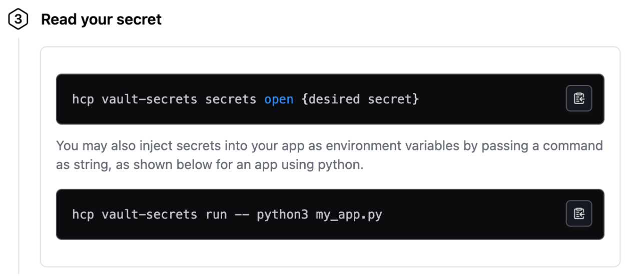
Figure 4: Sample presentation of the injection run CLI command required to complete Task 2 for HCP Vault Secrets.

then passed to an application at run time as an environment variable. Secret injection ensures that the secret is not stored locally or within the application. As shown in Figure 1, there are two lines under Task 2 within `SecureSecret.py`. The first line retrieves an environment variable named `CompanySecretToken`. Once the participant completes Task 1, they would have successfully logged in and initialized the SMT with the CLI. Participants can then reference secrets stored in the SMT from the CLI through the usage of environment variables. We take care of this step with the first line of code which creates the environment variable. The second line simply prints "`Hello, `" in front of the value of the environment variable we created for `CompanySecretToken`.

After participants complete Task 1, injecting `CompanySecretToken` only requires one command as shown in Figure 4. This is common for all of the SMT's for each Task 2 as each SMT's injection command consists of a similar syntax which all commonly use the term "run". The "run" term is followed by two dashes and then the normal start command one would input to run the application in which they are injecting a secret into (e.g., `python3 SecureSecret.py` in our case). The expected output for Task 2 is: `Hello, I am a secret token! Please remove me from this code!`.

Study considerations: We initially chose a Python local file for Task 2 since before our pilot study, none of the SMTs held a specific Python example for their secret injection run advice. We also chose Python since it is the most popular coding language as well [20, 50, 92], and would provide a common language that developers code in already. We carefully designed these tasks to minimize any advantages between the SMTs and their tool documentation. However, after our pilot study, HCP Vault Secrets updated their documentation in their web dashboard to add an example of how to complete Task 2 in Python, as shown in Figure 5. We also noticed that Doppler references an external PyPI webpage [81] in their "Development/Editors" documentation which also shows an example of how to complete Task 2 in Python. Infisical throughout the time of our full study did not include such content in neither their public help documentation nor web UI. We decided to continue our experiments as planned and take note if these additions to the Doppler help documentation and HCP Vault Secrets web dashboard had any affect on the participants completing the tasks. We viewed this phenomenon as a natural experiment. Meaning, we observed differences in how participants performed on tasks using tool documentation when naturally presented with different content specificity.

3.2.4 Data Collection. We recorded whether participants completed each task and their completion times for each task. We also recorded two specific actions from participants that indicate deviations from the expected or "recommended" path a user would



```
hcp vault-secrets secrets open {desired secret}
```

You may also inject secrets into your app as environment variables by passing a command as string, as shown below for an app using python.

```
hcp vault-secrets run -- python3 my_app.py
```

Figure 5: Additional part of HashiCorp vault secrets documentation in their web dashboard that reflects the required CLI command for Task 2.

take to complete these tasks in a real setting. The expected path we refer to is that participants complete a task using the SMT and steps recommended by the SMT tool documentation to complete each task within the time constraint. The two actions we recorded which represent a deviation from the expected path are usage of secondary sources and attempts at workaround methods.

The usage of secondary sources means that for a given task, a participant searched for and accessed a source outside of the tool documentation provided by the SMT in order to complete a task. By "workaround methods", we refer to trying commands or actions that were not the recommended method suggested by the SMT tool documentation to complete a task (e.g., exporting tokens to access secrets or using the injection command to print the plaintext version of a secret in the CLI in Task 1). These actions allowed us to ask follow-up questions in our post-task interviews to see why participants deviated from the expected path in order to complete a task. We present these findings in detail in Section 4.1.

3.3 Post-Task Interviews

3.3.1 Interview Guide Development. We interviewed participants after each task to get in-depth responses about their experiences during each task. We focused our questions on the usage of the SMT, tool documentation, and usage of secondary sources if they accessed them. To develop questions addressing this focus, we first referred to prior work on software documentation [4, 79] and tool observation studies [2, 84]. Prior work has highlighted content challenges such as insufficient code snippets, explanatory text, or lack of references for a wide range of use cases. Therefore, we used known related challenges from prior work as well as our research questions to draft several high-level categories of questions. We then piloted our interview guide with our observational study design with three PhD research students in our research lab to refine our interview guide and study design. Two of the PhD students pursue research in software engineering and software supply chain security, while the third PhD student pursues research in usable security and privacy. We updated our interview guide based on feedback from pilots.

3.3.2 Interview Structure. Warm-up and wrap-up questions are asked before and after both tasks, respectively. The following high-level categories here represent questions we asked participants:

Warm-up Questions: Before Task 1, we asked a set of warm-up questions to learn about their academic and/or professional experience with secret management, as well as the types of secrets and tools they have used.

Overall Experience: The first question we asked after participants finished a task is to describe their overall experience with the task.

Negative Emotions: We asked participants if they felt any negative emotions when performing each task. Negative emotions can include annoyance, confusion, fatigue, frustration, or any form of disapproval/dissatisfaction. This focus helps identify usability issues that may not emerge from general feedback, while positive experiences were captured in the overall experience question.

Documentation Content: We specifically asked participants about challenges they experienced using the content of the tool documentation itself to complete each task. Content includes (but is not strictly limited to) code snippets, code blocks, CLI commands, coding or web UI examples, explanatory text, images, or videos.

Documentation Structure and Formatting: We specifically asked participants about challenges they experienced with the structure and formatting of the tool documentation. Structure includes (but is not strictly limited to) section headers, bullet points, tabs, or toolbars. Formatting includes (but is not strictly limited to) text, pictures, or videos used in the advice.

Secondary Source Usage: We then asked participants about their usage of secondary sources and if those sources were more challenging to use or more helpful than the official tool documentation.

Wrap-up Questions: We wrap up the study by asking participants if they would consider using the SMT in the future, what their general preferences are for receiving information from sources, and for any last comments or questions about the study.

3.3.3 Analysis and Coding. All interviews were recorded in person via Zoom and then transcribed using a Whisper's multi-lingual medium model [29] into written transcripts. Interview transcripts were analyzed through qualitative deductive and inductive coding. We first developed an initial codebook based on our interview and research questions. Using this initial codebook, two of the researchers met to code a subset of the transcripts and compare their coding results. After the subset coding, the two researchers then split coding evenly (13 and 14 transcripts coded respectively) and met weekly to discuss coding disagreements and revise the codebook to accurately reflect our research interests. The two researchers also double coded six of the transcripts and checked inter-coder agreement weekly [62] to ensure we accomplished high agreement when independently coding transcripts (Krippendorff's Alpha > 0.75 for both researchers during independent coding [31, 32]). We continued to iteratively code the interview transcripts until no new codes emerged and all transcripts were coded.

3.4 Data Protection and Ethical Standards

We took multiple steps to protect participants' privacy and data, while also ensuring all participants were treated ethically and with respect. Before participants performed our study, we described our research goals to them and obtained consent from all participants. Participants were also informed on how their information would be protected on our consent form. During the study, we briefed participants on their rights during the study, including the right to skip any tasks and questions, and the right to withdraw without loss of benefits. We obtained video recordings and journal notes from each session, as well as written transcripts from audio recordings and performed both quantitative and qualitative analyses on our data. Participants consented to transcription, and all transcription was done with local models [29]. Written transcripts and journal notes were de-identified by replacing personally identifiable information (participant names, organizational names) with pseudonyms such as P01. We also did not request identifying, confidential, or private information about our participants or their prior experience in our study. We used end-to-end encrypted tools in all of our study communications and data storage components. Our study protocol was approved by our University's Institutional Review Board (IRB).

Further, we ensured that none of the CS Master's students held any prior relationships (academic or professional) or interactions with the researcher leading the in-person observational tasks and interviews. We also informed participants they would only receive monetary compensation, and their involvement would not result in any academic credit compensation or hold any bearing on their academic status.

3.5 Limitations

Participants were placed under three constraints during this study. The first being a 30 minute time constraint to complete each task. Such a time constraint may have influenced participants to overlook or skip steps that they normally would take in order to finish the task. Also, emotions of stress or fear of incompletion may also have been a factor for participants during this study. We informed participants that their compensation is not dependent on performance, but rather based on the amount of time they take to complete tasks. We also allowed participants breaks at any time, allowed questions in which we pause the time, or allowed them to skip a task if they chose to do so. The second constraint is that participants only used one of the three SMTs during their tasks. We designed our study this way because we did not want to overburden or confuse participants with too many tools or tasks, and also to avoid learning effects. Our third constraint prohibited using generative AI services when searching for online SMT tool documentation. Allowing AI tool use would introduce many confounding variables. One issue is assessing and controlling for user experience and competence with AI. Another is accounting for tool and model choice, model performance, and if AIs are biased to perform better against older or more popular tools. Studying the effectiveness of GenAI for this setting will require novel methodologies worthy of their own future research efforts.

Our study sample consists of CS Master's students with mixed levels of secret management experience from professional or academic backgrounds, thus potentially contributing to confounding effects to our study by analyzing participants with varying levels of software development maturity. However, focusing on CS Master's students serves as a proxy for junior developers, which is reflective of the new user study population we focus on.

4 Results

This section presents the findings of the challenges and experiences participants faced when using an SMT to perform two secret management related tasks. In Section 4.1, we report findings for participant task completion, completion times, usage of secondary sources, and attempts at workaround methods for each task. In Section 4.1.1, we present noticeable differences for participant performance for tasks across the three SMTs. The remaining sub-sections report in-depth qualitative findings from post-task interviews with participants who reported their challenges and experiences using SMTs to complete our secret management tasks. Our qualitative findings aim to explore the context for why developers widely report setup and learning constraints for SMTs when adopting them [14, 53]. In Section 4.2, we report participants' insights and responses regarding the functional impact of the SMTs they used. In Section 4.3, we report specific challenges with the tool documentation that contributed to reported task challenges and deviations from the expected methods as provided from the tool documentation. Lastly, in Section 4.4, we explain responses participants gave for why they accessed secondary sources, and also explain how helpful participants reported that access to secondary sources was for their tasks.

4.1 Participant Performances

Tables 1 and 2 show the performance statistics for participants. Overall, we observe noticeable differences in participant performance between tasks. 21 participants completed Task 1 within 30 minutes, whereas only 18 participants completed Task 2 within 30 minutes. We also notice that participants on average completed Task 1 in 11 minutes and 8 seconds, whereas participants on average completed Task 2 in 16 minutes and 18 seconds. More importantly, we noticed that only eight participants accessed a secondary source or tried a workaround method in Task 1, while 16 participants accessed a secondary source or tried a workaround method in Task 2. In Task 2, 12 participants accessed a secondary source, and only two participants accessed a secondary sources in Task 1. Twice as many participants tried a workaround method in Task 2 (14) as they did in Task 1 (7).

Our findings indicate that the number of participants that deviated from the expected path, as described in Section 3.2.4, doubled in count from Task 1 to Task 2. More than two-thirds of participants struggled to solely stick with the provided tool documentation and the expected methods for each task. We emphasize this as a focus in our findings since Task 1 requires several more individual steps than Task 2. Task 1 requires participants to remove `CompanySecretToken` from `SecureSecret.py`, store `CompanySecretToken` in the SMT web UI *CompanySecretStorage*, login and initialize into the SMT CLI, and finally print the value of `CompanySecretToken` from the command line. Task 2 on the other hand only requires the participant to input one command in the SMT CLI which performs the injection run process as shown in Figure 4. It is unclear what specifically contributed to different paths participants took for each task by only looking at our data from Tables 1 and 2. Therefore, we use Sections 4.2, 4.3, and 4.4 to find explanations for why participants experienced increases in measured difficulty and deviations from solely relying on tool documentation from the SMTs.

Table 1: Participants' Performance By SMT in Task 1.

	Completed	SMT Used	Secondary Source	W.A Method	Time
P16	✓	HCP VS			1:52
P21	✓	HCP VS			2:41
P12	✓	HCP VS			3:34
P09	✓	HCP VS			5:35
P03	✓	HCP VS			7:10
P01	✓	HCP VS			8:57
P11	✓	HCP VS			14:03
P07	✓	Infisical			5:40
P05	✓	Infisical		✓	7:56
P20	✓	Infisical			8:31
P10	✓	Infisical		✓	10:39
P02	✓	Infisical			17:03
P04	✓	Infisical	✓		19:45
P14	✓	Infisical	✓	✓	23:42
P15	✓	Doppler		✓	6:44
P08	✓	Doppler		✓	9:43
P19	✓	Doppler			10:33
P06	✓	Doppler		✓	11:52
P13	✓	Doppler			12:17
P18	✓	Doppler			22:07
P17	✓	Doppler		✓	23:28
Overall	21		2	7	11:08†

† Average

Table 2: Participants' Performance By SMT in Task 2.

	Completed	SMT Used	Secondary Source	W.A Method	Time
P12	✓	HCP VS			4:37
P09	✓	HCP VS	✓	✓	7:15
P03	✓	HCP VS	✓		12:13
P16	✓	HCP VS		✓	16:44
P11	✓	HCP VS	✓	✓	21:03
P21	✓	HCP VS	✓	✓	28:44
P01		HCP VS	✓	✓	>30:00
P10	✓	Infisical			1:04
P20	✓	Infisical			2:00
P05	✓	Infisical			4:04
P04	✓	Infisical		✓	5:18
P07	✓	Infisical	✓		16:23
P02		Infisical	✓	✓	>30:00
P14		Infisical		✓	>30:00
P08	✓	Doppler		✓	7:50
P13	✓	Doppler	✓	✓	13:20
P06	✓	Doppler	✓	✓	13:28
P19	✓	Doppler			22:51
P17	✓	Doppler	✓	✓	23:10
P15	✓	Doppler	✓	✓	23:23
P18	✓	Doppler	✓	✓	29:05
Overall	18		12	14	16:18†

† Average

4.1.1 Task Performance Comparing SMTs. Here, we briefly present differences in participant performance statistics we noticed between the three SMTs for each task.

Task 1: First, we examine how SMTs vary in their time-to-completion. Doppler (med. 11.9 minutes, *sigma* = 6.41) and Infisical (med. 10.7 minutes, *sigma* = 6.84) have task distributions with a center approximately double the median completion time of HCP Vault Secrets (med. 5.58 minutes, *sigma* = 4.25). None of the seven HCP Vault Secrets participants accessed secondary sources or tried workaround methods. As P16 noted, the tool was straightforward to use: "*I liked how easy it was to create a token, to see the token …new developers are going to join and may use it more easily. There's no complexity.*" (P16) Whereas Infisical had the most people trying secondary sources and workaround methods. Because we only have seven participants in each group, statistical power is quite limited, and hypothesis tests should not be considered definitive. Nevertheless, we performed a Kruskal-Wallis test out of curiosity. We found a statistically significant difference in distributions (H = 6.50, P = 0.39) with a "large" effect size (^{2} = 0.250).

Task 2: We repeat the same analysis for Task 2. Unlike Task 1, we had three participants who failed to complete the task. Two of those participants used Infisical, which was the same SMT with the most participants not completing Task 2. For example, P14 struggled to understand how the tool linked the web portal to the local Python file, stating: "*I wanted to link the Python file to the Infisical path where we store the key …I did not know just running the file would connect those two [Infisical web portal to the Python file].*" (P14) Infisical also had the lowest number of participants who accessed secondary sources. We arbitrarily assigned a value of 30 minutes time for these participants, though we find using a higher value did not change the statistical conclusions. Doppler (med. 22.9 minutes, *sigma* = 7.53), Infisical (med. 5.3 minutes, *sigma* = 12.85) and HCP Vault (med. 16.74 minutes, *sigma* = 9.95) vary widely in median but largely have overlapping distributions. We also performed a Kruskal-Wallis test on these times, finding no statistically significant difference in the distributions (H = 1.41, P = 0.495). Again, with low sample sizes these findings should be considered provisional.

4.2 Tool Usability Feedback

Participants highlighted both positive aspects and suggestions for the SMTs from the study, indicating how different functionalities influenced their success in using the tool and completing the tasks. The copy to clipboard functionality was appreciated by P19, who mentioned they can easily copy to the clipboard and paste it in the *"terminal directly"* along with the ease of creating tokens as mentioned by P16, "*I liked how easy it was to create a token, to see the token and it's like if there's a company that's using this application, new developers are going to join and may use it more easily. There's no complexity.*" (P16) P21 also mentioned that the web dashboard was more helpful while working on the task, with its *"UI is particularly helpful"* in managing secrets and allowing users to view all applications at once. Many saw the SMT as a *"good option for managing secrets"* with several others expressing interest in using it for future projects and exploring its integration capabilities. Participants also compared the tool to others, some felt it is a strong alternative to Git Lab secrets, while others mentioned preferring established cloud options like AWS for its documentation and familiarity like P15 stated, "*I'm more of an AWS guy and it already has something like KMS. Like if you use AWS, KWS, EC2, you really don't have to do anything else, it is all in one place. I don't think I'd want to use Doppler. Also, the documentation for KMS is really nice, people have been using it for a while.*" (P15)

In terms of personal versus professional usage, participants like P01 generally felt the tool was more suited for *"company or team-based projects rather than personal use"*.

P09 mentioned that the *"pricing structure of the tool"* is one of the deciding factors for adoption. Some participants like P04 also mentioned that the tool was easy to set up and had clear documentation and an intuitive user interface. P13 felt that the dashboard and command-line interface further improved the overall usability by stating, "*I think the tool was very helpful in managing the secrets. I like the way we can interact with the dashboard like they have created a dashboard that then we can interact via the visual interface and we can import the secrets as well if we don't want to use the command line. Whereas people who love the command line, have a dedicated command line as well. So I like that idea pretty much.*" (P13)

Participants also pointed to various features that supported their needs where P16 appreciated the *"availability of API, CLI and customization"*. Whereas, P08 also pointed out that for projects, relying on a tool like Doppler *"could create dependency and difficulties in migration"*. P01 faced login issues, mentioning they *"didn't know how the login works"*. P15 also highlighted problems with the save feature, stating they *"didn't really like it"* and P04 also pointed out issues with handling special characters, stating it *"could improve on string escape characters"*.

The study also revealed usability differences between SMTs in Section 4.1.1, where HCP Vault Secrets had faster completion times and fewer instances of participants seeking secondary sources compared to Infisical. This could indicate that its workflow is easier or more intuitive for users. In contrast, Infisical had a higher rate of task failure and more users relying on secondary sources, which might point to usability challenges like input handling limitations, and flexibility. The flexibility offered by different user interfaces could play a role in adoption, though further research is needed to better understand how these factors impact long-term usage and success.

4.3 Tool Documentation Challenges

In this section, we specifically focus on challenges and experiences directly reported from using SMT tool documentation and web UI to complete each task.

4.3.1 Documentation Content. Tool documentation was widely reported as providing enough information and being generally helpful by participants during Task 1. 13 participants even cited specific documentation they found helpful. For example, both P04 and P10 mentioned that the Infisical CLI quickstart guide was helpful for managing secrets and gave basic steps to do so. P03 reported that HCP Vault Secrets was *"easy to navigate"* and appreciated the usage of images in documentation. P18 and P19 appreciated CLI command descriptions in Doppler's CLI Guide.

There were a few instances when participants mentioned a lack of clarity over CLI descriptions and usage during Task 1. P09, P11, and P19 expressed confusion on applying CLI secrets commands when command line arguments were not explicitly defined on the same page in the documentation. Participants reported confusion over CLI usage when they felt the content did not sufficiently cover the relationship between CLI commands and secret management processes. Both P04 and P19 felt the content they used did not fully address immediate actions required to complete the initialization and login process for using the CLI.

Participants who attempted workaround methods for Task 1 mostly tried CLI commands to export secrets and also the injection run command which was needed for Task 2. None of the participants were aware they were going to inject secrets before Task 2, so some of them used the injection run command to perform injection to complete Task 1. P15 mentioned the following as to why they used the injection run command for Task 1: "*So my task was basically to output the secret, right? So the first thing that I did was like doppler run...*" (P15)

For Task 2, participants largely reported tool documentation as being much less helpful and consequently reporting Task 2 as more challenging than Task 1. 17 participants reported that content for the injection run command contributed to challenges they experienced when completing Task 2. Participants mentioned the following problems: lack of sufficient information for relevant command line arguments, lack of a relevant set of use cases in examples provided, and inconsistent documentation that covers secret injection.

Command Line Arguments: Several participants mentioned that command line arguments were introduced in the tool documentation for the injection run command. However, they were confused reading because they felt there was a lack of explicit information on how to apply them and how they related to the injection run command. P09 and P16 both referred to injection run command documentation from HCP Vault Secrets CLI page. They did not understand the `duration` flag shown at the end of the command in the documentation. P16 could not figure out how to make the `duration` flag work with the injection run command. P16 also mentioned a lack of understanding towards another flag used in the documentation: "*And the first thing, if you see, they have mentioned app, my-app, the application you want to pull all secrets from. But you don't understand its purpose or it doesn't say what happens if you don't do it. I had to try it myself.*" (P16)

Example Use Cases: Six participants explicitly stated examples provided from the tool documentation were a challenge when figuring out how to apply the injection run command to the study Python file, *SecureSecret.py*. Before our full study, injection run command examples shown in all of the SMTs tool documentation did not explicitly present a Python example, unlike other languages and frameworks such as Flask, Golang, NPM, and command line functions like echo. Participants wanted more clarification from tool documentation on how languages or frameworks used in the examples are simply just examples and not required to run the command itself. P17 explained how they were initially confused with the explicit injection content in Doppler's Secret Access CLI Guide since it primarily used an example with Node Package Manager (NPM) without any other examples present: "*This npm start is just a command example... They should have mentioned that npm start is just an example you can type in your command instead of start. That got me a little confused. Otherwise, the task was easier but it took me some time to figure out the exact solution.*" (P17)

Inconsistent Command Presentations: Several participants mentioned they observed discrepancies in relevant command presentations used in content describing secret injection across different documentation pages. Multiple participants that used either Inifisical or HCP Vault secrets reported that both tool's quick start guides and their CLI documentation pages differed in both the examples and arguments present for their injection run command descriptions. P05 mentioned that while Infisical's injection run command page provides a bare example without flags or arguments, the Infisical quickstart guide only provides examples with sample flags and arguments. P05 later explained that led them to try the injection command with incorrect flags and arguments because they did not know beforehand those were not necessary, thus received errors at first. Participants that used HCP Vault Secrets faced this challenge even with the existence of added dashboard documentation that provides a specific Python3 example for the injection run command, as shown in Figure 5. For participants who noticed the added dashboard content, they mentioned that the inclusion of the Python injection run command example helped them complete the task. Specifically, participants P03 and P09 found the HCP Vault Secrets Python3 example helpful for understanding how to use secret injection feature. Participant P03 noted, "*One is like not having to type out the credentials again and again, and not having to hard code the credentials which is not ideal. Scoring it in the environment variables with secret injection is a lot safer.*" (P03) Similarly, P09 explained, "*They have this feature and I can inject my secret. So it is like completely decoupled from my main application.*" (P09)

However, they clarified it was not intuitive to expect secret injection documentation within the web dashboard or outside of the public tool documentation. P11 noticed the Python example in the HCP Vault Secrets dashboard and stated that *"maintaining two different pages and they have two different contents, obviously confuses people"*. P03 echoed a similar response: "*I would say the first one was easier because the links to the documentation were available... In my mind, I was assuming that there will be similar documentation available somewhere. And it did not strike me that I can just take a look at the portal.*" (P03)

We believe insufficient information, a lack of relevant use cases in examples, and inconsistent documentation contributed to participants deviating from strictly following tool documentation. From our interview responses, participants reported struggling with understanding how to perform secret injection in Python when reading tool documentation command line arguments and examples. Inconsistent presentations of injection run commands also confused participants as to how to perform secret injection in Python.

4.3.2 Documentation Structure and Formatting. Here we report responses from participants towards the structure and formatting of the tool documentation. We also briefly report specific navigation difficulties participants experienced with the search functionality in the tool documentation.

Structure: The structure of the tool documentation received generally positive feedback. Participants appreciated the overall clarity in structure, including well-defined headers and organized sections like P18 mentioned, *"everything is labeled nicely"*. Both P04 and P20 felt the web UI for the tool documentation was clear and minimalist in design. P04 specifically appreciated when titles, headers, and tabs were used to separate topics.

Content Format Preferences: Regarding content format, participants had mixed preferences. Some participants preferred text for its searchability and ease of reference. When asked the reason for preferring text, P09 explained they *"prefer reading text over videos"* as it allows them to search for keywords and scan information faster. Other participants found code examples and videos beneficial for visual learning. For coding issues, P21 mentioned, *"text with code blocks is very helpful"*. P17 explained their preference for videos: "*First reference would be video because if I want to get help for something then it would be better for me to see that the other person is doing the same thing and explaining it in a video.*" (P17)

Few participants mentioned their content preference depended on the complexity of the task at hand. P01 stated, *"minor problem may be a text, a bigger problem somewhere in between text and video"*. Overall, participants held mixed preferences regarding format, suggesting that offering mixed formats could better support varied user needs.

Navigation Difficulties: During both tasks, participants reported specific navigation difficulties for the tool documentation. P04 felt that the *"search functionality could be improved"* in the documentation, as they struggled to find specific terms or commands, and mentioned that the *"ease of search was missing."* Participants found it most challenging to use the search functionality from the tool documentation to find sufficient information for secret injection, as P04 mentioned: "*... when I typed injection, it did not give me the desired response. So I think it is good that they are returning an LLM response, but I would also appreciate if they would have returned a link to where I could find what I actually wanted. They're basically giving me a ChatGPT kind of response, but I would also appreciate the source of that thing.*" (P04)

As navigation difficulties increased for participants during Task 2, their time and effort also increased as they needed to perform more troubleshooting and exploration.

4.4 Secondary Sources

Here, we focus on participants responses towards why they accessed secondary sources and how helpful they reportedly found secondary sources. We do not analyze the secondary sources in-depth themselves (e.g., content, structure, formatting, or source of origin). We only provide some examples and briefly discuss relevant content from examples participants mentioned where relevant.

Only two participants during Task 1 accessed secondary sources online for further assistance, as shown in Table 1. One of those participants, P14, mentioned they accessed a secondary source to reduce confusion for the injection run command by finding more examples. As we previously mentioned in Section 4.3.1, a few participants from Task 1 attempted the injection run command intended for Task 2 because they thought that command would produce the intended output required for Task 1. This lead P14 to seek secondary sources on how to use the injection run command during Task 1. We explain implications behind how this can lead to complications broadly for users adopting new software technology through tool documentation in Section 5.2.3.

Seeking further assistance on how to perform secret injection in Python became the common reason for participants accessing secondary sources during Task 2. As shown in Table 2, twelve participants searched for and accessed secondary sources outside the tool documentation during Task 2. Ten of those participants explained they needed information which was not addressed in the tool documentation. P06 stated the tool documentation was not necessarily confusing, but they needed *"maybe a little more description"*. P21 felt the examples provided were not elaborate for completing Task 2. This reasoning led participants to search the web for secondary perspectives on how to inject secrets with their SMT. P18 described this process: "*I scrolled quickly through the main documentation and I was stuck. So it's like whenever you are stuck at something, you need some other perspective.*" (P18)

Finding solutions from others online motivated participants in our study to use sites like StackOverflow. P15 felt that while the Doppler CLI Guide was *"really good to get started"*, StackOverflow is better for finding the *"specific piece of code you are actually looking for"*. P21 mentioned they appreciate added background and debugging advice from users on StackOverflow. Participants who searched for secondary sources often indicated those sources helped them more than the tool documentation when they received that added bit of information or example usage which they felt was missing from the tool documentation, specifically for Task 2. P06 googled broadly for information for running Doppler with Python files and mentioned that a Dev Community blog post [24] gave better descriptions for running the injection run command with Doppler for Python files.

However, more often than not, participants would either try multiple search queries and multiple distinct secondary sources before either finding one that helped or eventually going back to the tool documentation for Task 2. P03 even mentioned that when they did try to Google for secondary sources, the results returned *"were totally irrelevant."* P15, while searching for sources, clicked on a blog post by Medium Corp. [63] which they believed would give relevant advice for secret injection. However, P15 later clarified that the blog *"was very complex and covered kubernetes. It didn't actually tell me what I wanted."*

As mentioned in Section 3.2.3, Doppler provides a hyperlink within the Development/Editors Section [27] for Python which directs users to a external doppler-env-package page [81] that shows an injection run command example in Python. While three of the Doppler participants specifically mentioned that the Python example for secret injection from the doppler-env-package page helped, finding the page was a challenge. P15 mentioned that doppler-env-package page [81] *"wasn't really anywhere in the documentation."* P18 on the other hand noticed the hyperlink for the doppler-env-package page, however the surrounding headers and descriptions mentioned configuration and installation steps. Therefore, P18 early only thought any hyperlinks within that section were *"only about installation."* They did not reach the doppler-env-package page until Googling for more documentation and being directed to it.

The motivation from participants to acquire secondary sources increased noticeably when the tool documentation provided by the SMTs were perceived to not be sufficient enough in their explanations or examples, or provided inconsistent information. Even with the direct Python examples of injection run commands from HCP Vault Secrets web dashboard and Doppler's external doppler-env-package page, participants experienced difficulty in finding that information. HCP Vault Secrets participants did not expect relevant content for their tasks to be in their web dashboard, and Doppler participants could not find the doppler-env-package hyperlink which is mentioned in a section not specifically related to secret management. Other secondary sources were only helpful if they gave descriptions or code snippets participants felt that the tool documentation did not provide. However, most of the secondary sources accessed by participants were not much more helpful than the tool documentation, leading participants back to the tool documentation to finish Task 2.

5 Discussion

We discuss how our findings provide context for why new users may report challenging experiences when first setting up and using SMTs to manage secrets. We discuss SMT-specific conclusions towards new user SMT experiences in Section 5.1. Then, we discuss broadly applicable conclusions towards tool usability and documentation in Section 5.2. We use both our participant performance and interview findings to complement each other. We conclude with providing recommendations for SMT providers when designing SMT tool documentation and interfaces in Section 5.3.

5.1 RQ1: New Users and SMT Usage

Participants joined our study with prior experience in secret management and related programming concepts from our eligibility criteria. Some held experience using SMTs not from this study while others used a SMT for the first time during this study. We relieved participants of typical onboarding duties and provided a minimal example use case to remediate a hard-coded secret as described in Section 3.2. However, we did not not observe noticeable differences in participant task completion time and their ability to complete tasks with using workaround methods or secondary sources. Therefore, our work serves as a baseline for the best-case scenario for how new users would realistically perform with the set up and usage of SMTs to centrally secure their secrets.

Our findings support prior work towards identifying how developer experiences using new tools can be drastically impacted by the quality of available resources present [3, 12, 38, 53]. Even in a laboratory study of the best-case scenario, the ability of participants to effectively and securely remediate a hard-coded secret changed drastically between each task. The number of new users that accessed secondary sources or attempted workaround methods doubled from Taks 1 to Task 2. Even though participants completed Task 1 with relative ease, Task 2 is the process which helps ensure users can securely run local applications without hardcoding secrets. Both tasks were designed to represent fundamental processes for SMTs which aim to reduce security vulnerabilities for managing secrets, while being short enough to complete consecutively with four CLI command prompts as shown in Figures 3 and 4. Yet, even in our laboratory study with prior secret management experience, participants spent approximately half an hour and deviated from official tool documentation guidance just to complete both tasks. Even if participants gained incidental knowledge to complete Task 2 during Task 1, Task 2 is designed so that participants still needed to perform secret removal and secret storage steps with the SMT in Task 1. Therefore, our conclusions do not change when observing the total experience for both tasks. If participants in our laboratory setting struggled to effectively perform both tasks, then it is likely new users in a real world setting experience similar challenges with even less guidance on how to effectively use SMTs to securely manage secrets, which demonstrates a contribution to adoption burdens reported by developers in prior work [14, 53].

5.2 RQ2: Functionality and Documentation Factors that Impact SMT Usage

We examine the functional, positive, and negative aspects of participants' experiences using SMTs and tool documentation to manage secrets. Participants reported that well-structured documentation supported task completion, while unclear CLI instructions and limited examples created challenges.

5.2.1 Functionality Impact. We performed statistical analyses to see if we could identify any differences in participant performances across the three SMTs. While we do not claim definitive statistical significance, we found median completion times across SMTs varied widely, but largely overlapped in range. For example, in Tables 1 and 2, three and two participants completed Task 1 and 2 in under four minutes, respectively. On the flip side, three and nine participants took at least 20 minutes to complete Task 1 and 2, respectively. Thus, we see the possibility for participants to effectively use SMTs to secure secrets quickly, but we also see many other participants look at the same documentation and take considerably longer. Therefore, we reason the SMTs in our study were functionally indistinguishable compared to each other when participants used them to manage secrets. However, by functionally indistinguishable, we refer specifically to their usability features, such as those highlighted in Section 4.2, and not functional equivalence between the SMTs. Differences in participant task completion times likely reflect SMT usability variations between each task. While we acknowledge that our sample size limits statistical significance, our work serves as an appropriate exploratory study addressing first-time user experiences with SMTs [14, 53].

Participants pointed out several UI and interface features that influenced their experience with SMTs. They appreciated intuitive web interfaces, copy-to-clipboard options for command snippets, and well-structured layouts with clear headers, all of which made their tasks easier. These usability improvements align with findings from recent studies, which highlight the importance of UI design and guided workflows in secret management tools [91]. We also noticed that those who found the UI intuitive had fewer difficulties managing secrets, while those who struggled with the interface took longer to complete tasks and felt more frustrated. Previous work has similarly shown that poorly designed interfaces and unclear documentation can cause secrets to be mismanaged or accidentally leaked [44]. This supports our findings that while SMTs may offer similar functionality, the quality of SMT tool UI design has a significant impact on overall usability and adoption by helping users navigate and complete tasks more smoothly [13].

5.2.2 Documentation Positive Factors and Experiences. Participants provided multiple positive statements about the tool documentation they used with the SMT, mostly while performing Task 1. Similar to prior efforts [2, 45, 56, 70, 89], participants in our study largely appreciated tool documentation that simplified their process of finding relevant information and provided just enough information to complete each task. Additionally, several participants also expressed a preference for video-based tutorials, indicating that these visual formats are effective and make it easier to understand and follow the instructions, including for non-expert users of a new tool [57]. Participants felt that the tool documentation provided enough information to complete Task 1. The inclusion of quick starter guides, and structural features like well-defined headers and organized sections were common aspects participants felt made their experience easier. We provide further insight from prior work in directly observing that as participants felt more positive about tool documentation, they completed their task in fewer time with less challenges along the way. Most importantly, we saw that participants were less likely to deviate the expected path while using SMTs to manage secrets when they reported positively about tool documentation. Thus, decreasing the risks of traversing through irrelevant information, implementing insecure methods, or an inability entirely to learn how to use new tools.

5.2.3 Documentation Negative Factors and Experiences. Participant feedback revealed that documentation challenges were significant barriers, leading us to shift our focus toward examining the usability of SMT tool documentation. Our findings support related efforts by continuing to highlight tool documentation challenges such as insufficient coding examples [2, 4–6, 71, 95], incorrect or ambiguous explanations [4, 22, 67, 90, 97, 98, 101] and the growing use of LLMs to generate documentation, which may introduce inaccuracies and further mislead users [16, 43]. However, our work extends prior efforts by exploring challenges participants report with using SMT-specific tool documentation covering CLI commands. New users already face a learning curve when first adopting SMTs [14, 53]. If new users encounter tool documentation with insufficient descriptions for relevant CLI command line arguments, or without a broad range of example use cases for CLI commands, they are less likely to know how to effectively use the SMT CLI functionality. Therefore, new users are also less likely to understand how to securely manage their secrets with SMTs when reading provided tool documentation. We discuss recommendations that SMT providers should consider when writing tool documentation to reduce participant confusion when learning SMT CLI commands in Section 5.3.

Our work also provides more context towards the experiences new users have when searching secondary sources for technical assistance as highlighted in prior work [3, 7, 11, 79, 86, 93]. Most relatedly, Acar et al. [3] observed that Android code written by developers produced different levels of security and functionality depending on the information source the developer used for assistance. In our study, participants complained that information from secondary sources was either no different than what the tool documentation provided, overly complex for their task, or completely irrelevant. In a real world setting, new users learning a tool search for information sources and need to determine if the source they are looking at is proposing methods that will help them achieve the expected outcome they require. The extent of harm for applying incorrect, or worse insecure, methods increases if new users feel inclined to spend more time exploring secondary sources after unsuccessfully trying official tool documentation. Therefore, making it more difficult for new users to effectively use or adopt new tools. We believe our findings for participants using SMT tool documentation add context to adoption burdens reported by developers [14, 53]. Specifically, if SMT tool documentation is challenging to use, and secondary sources discussing SMTs only add more confusion, then developers may be less willing to further adopt SMTs. Developers interested in adopting SMTs may have challenges using SMT tool documentation to manage their secrets or navigating different information sources to find relevant solutions for their use cases.

5.3 Recommendations

Here, we offer recommendations to SMT providers regarding tool functionality and documentation.

Dedicated Sections for Technical Terms: Many participants recommended including dedicated sections in the documentation to explain technical jargon, such as command flags, subcommands, and arguments. These sections could provide clear definitions, examples, and a glossary to help users, especially beginners, navigate the documentation more effectively.

Visual Aspects for Demonstration: Additionally, participants also suggested including visual aspects in the documentation like flow diagrams to demonstrate how commands interact with values, arguments, and output. These diagrams could provide a clear logical flow for the key tasks in the SMTs like logging in, initializing processes, and accessing secret projects. While resolving issues, users could also benefit from having debug examples, code snippets, and guidance.

Structural Improvements: Another recommendation is regarding the structuring of CLI documentation with a base example at the top, followed by examples with subcommands, flags, and arguments. This approach keeps all relevant information on one page, reducing the need for users to search elsewhere, and improving the overall user experience by minimizing lookup time. An effective structure and formatting with clear headings, a logical flow, and tables to represent a list of commands, subcommands, and argument explanations would help users quickly access and understand the information. Also, adding hyperlinks or resources in the documentation to guide users on installing necessary programming environments or dependencies would help beginners to quickly set up and follow the rest of the instructions to complete their tasks.

Clear Placeholders in Commands: Participants reported confusion over which CLI argument parameters were necessary to replace or use verbatim. Documentation authors should use obvious placeholders (e.g., `your_project_name`) to help users avoid confusion.

User-friendly Features: Providers are also recommended to continue to include user friendly features like copy to clipboard for ease of use and also consider adding search functionality and quick navigation links to improve the tool usability and help users complete their tasks more efficiently.

Incorporating Visual Media: Many participants expressed a preference for video-based tutorials as part of the documentation, as this format can help them better understand how to use tool features. Therefore, providers should aim to incorporate visual elements such as interactive demos and walkthroughs that demonstrate command usage, argument flows, common troubleshooting steps, and mirror real usage scenarios.

LLM Integration: Participants struggled when documentation lacked concrete examples or when search features failed to return relevant information. While LLMs can help generate diverse, contextual examples, participants also noted confusion when responses were vague or lacked sources. We recommend that SMT providers use LLMs to augment, not replace, documentation by generating accurate, targeted examples that are clearly linked to official sources.

6 Conclusion

In our exploratory qualitative two-step laboratory study into the usage of SMTs, we observed how even new users with prior secret management experience and an easier onboarding experience struggled to perform secret management tasks using a SMT. We identified specific challenges relating to tool documentation content, structure, and formatting that participants reported impacted their ability to effectively and securely remediate a hard-coded secret. Participants responded to tool documentation and interface challenges by deviating towards accessing secondary sources and attempting workaround methods, while cycling back to tool documentation and experiencing added difficulty. We make recommendations for SMT vendors to improve the ecosystem of guidance for new SMT users.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Award Nos. CNS-2055554 and CNS-2206865. We want to thank Anna Lena Rotthaler for her valuable feedback and interest in this research. We are thankful to all study participants for generously sharing their time and insights. We also appreciate the anonymous reviewers for their constructive suggestions and guidance.

References

[1] Lawrence Abrams. 2024. New York Times source code stolen using exposed GitHub token. https://www.bleepingcomputer.com/news/security/new-york-times-source-code-stolen-using-exposed-github-token/. Accessed: 2024-00-00.

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

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

[4] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L Mazurek, and Sascha Fahl. 2017. Developers need support, too: A survey of security advice for software developers. In *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, 22–26.

[5] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd. 2020. Software documentation: the practitioners' perspective. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 590–601.

[6] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1199–1210.

[7] Deeksha M Arya, Jin LC Guo, and Martin P Robillard. 2023. How programmers find online learning resources. *Empirical Software Engineering* 28, 2 (2023), 23.

[8] Deeksha M Arya, Jin LC Guo, and Martin P Robillard. 2024. Why People Contribute Software Documentation. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering*. 91–96.

[9] AWS. 2024. AWS Key Management Service. https://aws.amazon.com/kms. Accessed: 2024-00-00.

[10] Wei Bai, Moses Namara, Yichen Qian, Patrick Gage Kelley, Michelle L Mazurek, and Doowon Kim. 2016. An Inconvenient Trust: User Attitudes toward Security and Usability Tradeoffs for {Key-Directory} Encryption Systems. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. 113–130.

[11] Sebastian Baltes, Christoph Treude, and Martin P Robillard. 2020. Contextual documentation referencing on stack overflow. *IEEE Transactions on Software Engineering* 48, 1 (2020), 135–149.

[12] Setu Kumar Basak, Lorenzo Neil, Bradley Reaves, and Laurie Williams. 2022. What are the practices for secret management in software artifacts?. In *2022 IEEE Secure Development Conference (SecDev)*. IEEE, 69–76.

[13] Setu Kumar Basak, Lorenzo Neil, Bradley Reaves, and Laurie Williams. 2023. SecretBench: A Dataset of Software Secrets. *arXiv preprint arXiv:2303.06729* (2023).

[14] Setu Kumar Basak, Lorenzo Neil, Bradley Reaves, and Laurie Williams. 2023. What Challenges Do Developers Face About Checked-in Secrets in Software Artifacts? *arXiv preprint arXiv:2301.12377* (2023).

[15] Supraja Baskaran, Lianying Zhao, Mohammad Mannan, and Amr Youssef. 2023. Measuring the leakage and exploitability of authentication secrets in super-apps: The wechat case. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. 727–743.

[16] Avinash Bhat, Disha Shrivastava, and Jin LC Guo. 2024. Do LLMs meet the needs of software tutorial writers? Opportunities and design implications. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*. 1760–1773.

[17] Bjørnar Blaalid. 2018. "Tricks of the trade" – The art and method of combining interviews and participating observations to generate data on drug users participating in rehabilitation programs. *ResearchGate* (04 2018), 13.

[18] Raluca Budiu. 2024. Between-Subjects vs. Within-Subjects Study Design. https://www.nngroup.com/articles/between-within-subjects/. Accessed: 2024-00-00.

[19] Matt Burgees. 2024. Thousands of Corporate Secrets Were Left Exposed. This Guy Found Them All. https://www.wired.com/story/secret-hunting-bill-demirkapi/. Accessed: 2024-00-00.

[20] Stephen Cass. 2024. The Top Programming Languages 2024. https://spectrum.ieee.org/top-programming-languages-2024. Accessed: 2024-00-00.

[21] Ramaswamy Chandramouli. 2024. Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD pipelines. https://csrc.nist.gov/pubs/sp/800/204/d/ipd?ref=blog.gitguardian.com. Accessed: 2024-00-00.

[22] Jie-Cherng Chen and Sun-Jen Huang. 2009. An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software* 82, 6 (2009), 981–992.

[23] Lan Cheng, Emerson Murphy-Hill, Mark Canning, Ciera Jaspan, Collin Green, Andrea Knight, Nan Zhang, and Elizabeth Kammer. 2022. What improves developer productivity at google? code quality. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1302–1313.

[24] Dev Community. 2024. Build and deploy a web app with Python, Flask, and Doppler. https://dev.to/lordghostx/build-and-deploy-a-web-app-with-python-flask-and-doppler-9jm. Accessed: 2024-00-00.

[25] Markus Dahlmanns, Constantin Sander, Robin Decker, Klaus Wehrle, Jan Pennekamp, Anastasiia Belova, Thomas Bergs, Matthias Bodenbenner, Andreas Bührig-Polaczek, Ike Kunze, et al. 2023. Secrets Revealed in Container Images: An Internet-wide Study on Occurrence and Impact. In *ACM Transactions on Internet Technology*. ACM, 252–266.

[26] Doppler. 2024. The New Era of Secrets Management. https://www.doppler.com/. Accessed: 2024-00-00.

[27] Doppler. 2024. Python. https://docs.doppler.com/docs/vscode-python. Accessed: 2024-00-00.

[28] Neil A Ernst and Martin P Robillard. 2023. A study of documentation for software architecture. *Empirical Software Engineering* 28, 5 (2023), 122.

[29] Hugging Face. 2024. openai/whisper-medium. https://huggingface.co/openai/whisper-medium. Accessed: 2024-00-00.

[30] Marcel Fourné, Daniel De Almeida Braga, Jan Jancar, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. 2024. "These results must be false": A usability evaluation of constant-time analysis tools. In *33th USENIX Security Symposium (USENIX Security 2024)*.

[31] Deen Freelon. 2024. *ReCal2: Reliability for 2 Coders*. Accessed: 2024-00-00.

[32] Deen Freelon. 2024. ReCal3: Reliability for 3+ Coders. http://dfreelon.org/utils/recalfront/recal3/. Accessed: 2024-00-00.

[33] Peggy Fung, Lam-for Kwok, and Dennis Longley. 2003. Electronic information security documentation. In *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21* (Adelaide, Australia) *(ACSW Frontiers '03)*. Australian Computer Society, Inc., AUS, 25–31.

[34] g2. 2024. Best Secrets Management Tools". https://www.g2.com/categories/secrets-management-tools. Accessed: 2024-00-00.

[35] GitHub. 2024. Managing your account-specific secrets for GitHub Codespaces. https://docs.github.com/en/codespaces/managing-your-codespaces/managing-your-account-specific-secrets-for-github-codespaces. Accessed: 2024-00-00.

[36] GitHub. 2024. Using secrets in GitHub Actions. https://docs.github.com/en/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions. Accessed: 2024-00-00.

[37] Google. 2024. Google Secret Manager. https://cloud.google.com/secret-manager. Accessed: 2024-00-00.

[38] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 2018. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic {API} misuse. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. 265–281.

[39] Peter Leo Gorski, Sebastian Möller, Stephan Wiefling, and Luigi Lo Iacono. 2022. "I just looked for the solution!"On Integrating Security-Relevant Information in Non-Security API Documentation to Support Secure Coding Practices. *IEEE Transactions on Software Engineering* 48, 9 (2022), 3467–3484. doi:10.1109/TSE.2021.3094171

[40] HashiCorp. 2024. hcp vault-secrets run. https://developer.hashicorp.com/hcp/docs/cli/commands/vault-secrets/run. Accessed: 2024-00-00.

[41] HashiCorp. 2024. What is HCP Vault Secrets? https://developer.hashicorp.com/hcp/docs/vault-secrets/. Accessed: 2024-00-00.

[42] Andrew Head, Caitlin Sadowski, Emerson Murphy-Hill, and Andrea Knight. 2018. When not to comment: Questions and tradeoffs with API documentation for C++ projects. In *Proceedings of the 40th International Conference on Software Engineering*. 643–653.

[43] Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675* (2023).

[44] Yizhan Huang, Yichen Li, Weibin Wu, Jianping Zhang, and Michael R Lyu. 2024. Your code secret belongs to me: neural code completion tools can memorize hard-coded credentials. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2515–2537.

[45] Soumya Indela, Mukul Kulkarni, Kartik Nayak, and Tudor Dumitraş. 2016. Helping Johnny encrypt: Toward semantic interfaces for cryptographic frameworks. In *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 180–196.

[46] Infisical. 2024. Open Source Secret Management. https://infisical.com/. Accessed: 2024-00-00.

[47] itnews. 2024. AWS urges developers to scrub GitHub of secret keys. https://www.itnews.com.au/news/aws-urges-developers-to-scrub-github-of-secret-keys-375785. Accessed: 2024-00-00.

[48] Mackenzie Jackson. 2024. 8.5% of Docker Images Expose API and Private Keys. https://blog.gitguardian.com/8docker-images-api-and-private-keys/. Accessed: 2024-00-00.

[49] Shazia Jamshed. 2014. Qualitative research method-interviewing and observation. *Journal of Basic and Clinical Pharmacy* 05, 04 (2014), 87–88.

[50] Paul Jansen. 2024. TIOBE Index for November 2024. https://www.tiobe.com/tiobe-index/. Accessed: 2024-00-00.

[51] Johannes Kaiser and Martin Reichenbach. 2002. Evaluating security tools towards usable security: A usability taxonomy for the evaluation of security tools based on a categorization of user errors. In *IFIP World Computer Congress, TC 13*. Springer, 247–256.

[52] Eyal Katz. 2024. 5 Ways to Prevent Secrets Sprawl. https://spectralops.io/blog/5-ways-to-prevent-secrets-sprawl/. Accessed: 2024-00-00.

[53] Alexander Krause, Jan H Klemmer, Nicolas Huaman, Dominik Wermke, Yasemin Acar, and Sascha Fahl. 2023. Pushed by Accident: A {Mixed-Methods} Study on Strategies of Handling Secret Information in Source Code Repositories. In *32nd USENIX Security Symposium (USENIX Security 23)*. 2527–2544.

[54] Stefan Krüger, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler, et al. 2017. Cognicrypt: Supporting developers in using cryptography. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 931–936.

[55] Stefan Krüger, Michael Reif, Anna-Katharina Wickert, Sarah Nadi, Karim Ali, Eric Bodden, Yasemin Acar, Mira Mezini, and Sascha Fahl. 2023. Securing your crypto-api usage through tool support-A usability study. In *2023 IEEE Secure Development Conference (SecDev)*. IEEE, 14–25.

[56] Timothy C Lethbridge, Janice Singer, and Andrew Forward. 2003. How software engineers use documentation: The state of the practice. *IEEE software* 20, 6 (2003), 35–39.

[57] Songsong Liu, Shu Wang, and Kun Sun. 2024. Having Difficulty Understanding Manuals? Automatically Converting User Manuals into Instructional Videos. *Proceedings of the ACM on Human-Computer Interaction* 8, EICS (2024), 1–19.

[58] Nikolaos Lykousas and Constantinos Patsakis. 2024. Decoding developer password patterns: A comparative analysis of password extraction and selection practices. *Computers & Security* 145 (2024), 103974.

[59] Vlad Matsiiako. 2024. Top-10 Secret Management Tools in 2024. https://infisical.com/blog/best-secret-management-tools. Accessed: 2024-00-00.

[60] Dwayne Mcdaniel. 2024. A look at the future of supply chain and national security: Updates from CISA and NIST. https://blog.gitguardian.com/software-supply-chain-security-updates-from-cisa-and-nist/. Accessed: 2024-00-00.

[61] Dwayne Mcdaniel. 2024. 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/. Accessed: 2024-00-00.

[62] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice. *Proceedings of the ACM on human-computer interaction* 3, CSCW (2019), 1–23.

[63] Medium. 2024. Injecting secrets to Kubernetes containers from the Doppler secrets manager. https://medium.com/@peterkracik/injecting-secrets-to-kubernetes-containers-from-the-doppler-secrets-manager-ef491a20f45b. Accessed: 2024-00-00.

[64] Michael Meli, Matthew R McNiece, and Bradley Reaves. 2019. How bad can it git? characterizing secret leakage in public github repositories.. In *NDSS*.

[65] Microsoft. 2024. Azure Key Vault. https://learn.microsoft.com/en-us/azure/key-vault/. Accessed: 2024-00-00.

[66] Microsoft. 2024. Visual Studio Code. https://code.visualstudio.com/. Accessed: 2024-00-00.

[67] Justin Middleton, Emerson Murphy-Hill, and Kathryn T Stolee. 2020. Data analysts and their software practices: A profile of the sabermetrics community and beyond. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–27.

[68] Emerson Murphy-Hill, Ciera Jaspan, Caitlin Sadowski, David Shepherd, Michael Phillips, Collin Winter, Andrea Knight, Edward Smith, and Matthew Jorde. 2019. What predicts software developers' productivity? *IEEE Transactions on Software Engineering* 47, 3 (2019), 582–594.

[69] Olha Mykhaylova, Taras Fedynyshyn, and Artem Platonenko. 2024. Hardcoded credentials in Android apps: Service exposure and category-based vulnerability analysis. *Cybersecurity Providing in Information and Telecommunication Systems II 2024* 3826 (2024), 206–211.

[70] Mathieu Nassif, Alexa Hernandez, Ashvitha Sridharan, and Martin P Robillard. 2021. Generating unit tests for documentation. *IEEE Transactions on Software Engineering* 48, 9 (2021), 3268–3279.

[71] Mathieu Nassif, Zara Horlacher, and Martin P Robillard. 2022. Casdoc: unobtrusive explanations in code examples. In *Proceedings of the 30th IEEE/ACM international conference on program comprehension.* 631–635.

[72] Mathieu Nassif and Martin P Robillard. 2023. A Field Study of Developer Documentation Format. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems.* 1–7.

[73] Mathieu Nassif and Martin P Robillard. 2023. Identifying Concepts in Software Projects. *IEEE Transactions on Software Engineering* 49, 7 (2023), 3660–3674.

[74] Mathieu Nassif and Martin P Robillard. 2023. Non Linear Software Documentation with Interactive Code Examples. *arXiv preprint arXiv:2311.18057* (2023).

[75] Lorenzo Neil, Deepthi Mungara, Laurie Williams, Yasemin Acar, and Bradley Reaves. 2025. Extended Version: It Should Be Easy but... New Users Experiences and Challenges with Secret Management Tools. arXiv:2509.09036 [cs.HC] https://arxiv.org/abs/2509.09036

[76] Abi Noda, Margaret-Anne Storey, Nicole Forsgren, and Michaela Greiler. 2023. DevEx: What Actually Drives Productivity: The developer-centric approach to measuring and improving productivity. *Queue* 21, 2 (2023), 35–53.

[77] Sanghak Oh, Kiho Lee, Seonhye Park, Doowon Kim, and Hyoungshick Kim. 2024. Poisoned chatgpt finds work for idle hands: Exploring developers' coding practices with insecure suggestions from poisoned ai models. In *2024 IEEE Symposium on Security and Privacy (SP).* IEEE, 1141–1159.

[78] OWASP Cheat Sheet Series. 2025. Secrets Management Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html. https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html Accessed: 2025-07-23.

[79] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. 2012. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. *Georgia Institute of Technology, Tech. Rep* 11 (2012).

[80] Reinhold Plösch, Andreas Dautovic, and Matthias Saft. 2014. The Value of Software Documentation Quality. In *2014 14th International Conference on Quality Software.* 333–342. doi:10.1109/QSIC.2014.22

[81] PyPI. 2024. doppler-env 0.3.1. https://pypi.org/project/doppler-env/. Accessed: 2024-00-00.

[82] Qualtrics. 2024. qualtrics. https://www.qualtrics.com/. Accessed: 2024-00-00.

[83] ReversingLabs. 2024. Secrets Exposed: How to mitigate risk from secrets leaks — and prevent future breaches. https://www.reversinglabs.com/blog/secure-your-development-secrets-3-essential-steps. Accessed: 2024-00-00.

[84] Martin P Robillard. 2009. What makes APIs hard to learn? Answers from developers. *IEEE software* 26, 6 (2009), 27–34.

[85] Martin P Robillard and Yam B Chhetri. 2015. Recommending reference API documentation. *Empirical Software Engineering* 20, 6 (2015), 1558–1586.

[86] Derek Robinson, Neil A Ernst, Enrique Larios Vargas, and Margaret-Anne D Storey. 2022. Error identification strategies for Python Jupyter notebooks. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension.* 253–263.

[87] Judith Segal. 2007. Some problems of professional end user developers. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007).* IEEE, 111–118.

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

[89] Justin Smith, Lisa Nguyen Quang Do, and Emerson Murphy-Hill. 2020. Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020).* 221–238.

[90] SM Sohan, Frank Maurer, Craig Anslow, and Martin P Robillard. 2017. A study of the effectiveness of usage examples in REST API documentation. In *2017 IEEE symposium on visual languages and human-centric computing (VL/HCC).* IEEE, 53–61.

[91] Prakash Somasundaram. 2024. Unified Secret Management Across Cloud Platforms: A Strategy for Secure Credential Storage and Access. *Int. J. Comput. Eng. Technol* 15 (2024), 5–12.

[92] Stackoverflow. 2024. 2024 Developer Survey. https://survey.stackoverflow.co/2024/technology#2-programming-scripting-and-markup-languages. Accessed: 2024-00-00.

[93] Margaret-Anne Storey, Daniel Russo, Nicole Novielli, Takashi Kobayashi, and Dong Wang. 2024. A disruptive research playbook for studying disruptive innovations. *ACM Transactions on Software Engineering and Methodology* (2024).

[94] Margaret-Anne Storey, Thomas Zimmermann, Christian Bird, Jacek Czerwonka, Brendan Murphy, and Eirini Kalliamvakou. 2019. Towards a theory of software developer job satisfaction and perceived productivity. *IEEE Transactions on Software Engineering* 47, 10 (2019), 2125–2142.

[95] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API documentation. In *Proceedings of the 36th international conference on software engineering.* 643–652.

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

[97] Christoph Treude and Martin P Robillard. 2016. Augmenting API documentation with insights from stack overflow. In *Proceedings of the 38th International Conference on Software Engineering.* 392–403.

[98] Gias Uddin and Martin P Robillard. 2015. How API documentation fails. *Ieee software* 32, 4 (2015), 68–75.

[99] Stefan Wagner and Emerson Murphy-Hill. 2019. Factors that influence productivity: A checklist. *Rethinking productivity in software engineering* (2019), 69–84.

[100] Thomas Weber, Alois Zoitl, and Heinrich Hußmann. 2019. Usability of Development Tools: A CASE-Study. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C).* 228–235. doi:10.1109/MODELS-C.2019.00037

[101] Fengcai Wen, Csaba Nagy, Gabriele Bavota, and Michele Lanza. 2019. A large-scale empirical study on code-comment inconsistencies. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC).* IEEE, 53–64.

[102] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. 2015. Quantifying developers' adoption of security tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* 260–271.