What Challenges Do Developers Face About Checked-in Secrets in Software Artifacts?

Setu Kumar Basak*, Lorenzo Neil†, Bradley Reaves‡ and Laurie Williams§
North Carolina State University, USA
Email: *sbasak4@ncsu.edu, †lcneil@ncsu.edu, ‡bgreaves@ncsu.edu, §lawilli3@ncsu.edu

2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) | 978-1-6654-5701-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE48619.2023.00141

***Abstract*—Throughout 2021, GitGuardian's monitoring of public GitHub repositories revealed a two-fold increase in the number of secrets (database credentials, API keys, and other credentials) exposed compared to 2020, accumulating more than six million secrets. To our knowledge, the challenges developers face to avoid checked-in secrets are not yet characterized. *The goal of our paper is to aid researchers and tool developers in understanding and prioritizing opportunities for future research and tool automation for mitigating checked-in secrets through an empirical investigation of challenges and solutions related to checked-in secrets*. We extract 779 questions related to checked-in secrets on Stack Exchange and apply qualitative analysis to determine the challenges and the solutions posed by others for each of the challenges. We identify 27 challenges and 13 solutions. The four most common challenges, in ranked order, are: (i) store/version of secrets during deployment; (ii) store/version of secrets in source code; (iii) ignore/hide of secrets in source code; and (iv) sanitize VCS history. The three most common solutions, in ranked order, are: (i) move secrets out of source code/version control and use template config file; (ii) secret management in deployment; and (iii) use local environment variables. Our findings indicate that the same solution has been mentioned to mitigate multiple challenges. However, our findings also identify an increasing trend in questions lacking accepted solutions substantiating the need for future research and tool automation on managing secrets.**

I. Introduction

In March 2022, GitGuardian stated that the number of secrets exposed on public GitHub repositories doubled in 2021 compared to 2020, reaching a total of over six million secrets [1]. To perform authentication across software artifacts as part of system integration, software developers need secrets (database credentials, API keys, and other credentials). During software development, these secrets may need to be shared by developers working on a team, and after deployment may need to be distributed to applications.

Version control system (VCS) repositories, such as GitHub [2] and GitLab [3], are widely used by developers for managing source code. However, the VCS repository's nature makes securing secrets in developer projects challenging. In 2019, Meli et al. [4] studied a 13% snapshot of public GitHub repositories and found over 200K API keys checked into the repositories. Secrets are not only pushed into VCS repositories by developers but also kept in Android and iOS application packages [5]. Secrets in software artifacts (CWE-798: Use of Hard-coded Credentials [6]) have also been identified as a CWE Top 25 Most Dangerous Software Weaknesses [7].

While the checked-in secrets issue is well-known through prior works [4], [8], [9], [10], little is known about developers' technical challenges in preventing secrets from being stored in software artifacts. Developers query online forums, such as a developer who posted a question on how to keep secrets out of VCS repositories [11]. Systematically analyzing questions asked by developers and solutions posed by others can reveal the technical challenges and practices adopted by the developers to protect the secrets.

The goal of our paper is to aid researchers and tool developers in understanding and prioritizing opportunities for future research and tool automation for mitigating checked-in secrets through an empirical investigation of challenges and solutions related to checked-in secrets.

In this study, we analyze developers' questions and related solutions about checked-in secrets and provide answers to the following research questions:

- **RQ1:** What are the technical challenges faced by developers related to checked-in secrets?
- **RQ2:** What solutions do developers get for mitigating checked-in secrets?

Users can post questions describing a particular technical challenge for which they need support on Stack Exchange [12], a major question and answer (Q&A) site. An answer is a suggestion or solution to a technical challenge. Users can pose multiple answers to a question, but either zero or one answer is accepted. The answer approved by the user who posted the question is termed as the *accepted answer*. We refer to a question lacking an accepted answer or having no answers as a *question with unsatisfactory answer*.

We extracted 779 questions related to checked-in secrets from Stack Exchange spanning from September 2008 to December 2021. From these questions, we conducted a qualitative analytical approach called card sorting [13] to determine the question categories and related answer categories. We also perform quantitative analysis of question categories, which will help researchers and tool developers prioritize further study and tool development. In addition, the answer categories we presented give insights into which practices developers may have adopted. Following is a summary of the paper's contributions:

- A set of challenges faced by the developers about checked-in secrets; and

1558-1225/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE48619.2023.00141

- A set of solutions or suggestions posed by other developers to mitigate the checked-in secret challenges

The rest of our paper is structured as follows: The methodology used in our work is described in Section II. We discuss our findings and recommendations in Section III and IV, respectively. The ethics and limitations of our paper is discussed in Section V and VI, respectively. Section VII summarizes previous research findings pertinent to our paper. Finally, Section VIII draws the paper's conclusion.

II. Methodology

We provide our four-step process for data collection and question and answer analysis as follows:

A. Step 1: Q&A Site Selection

For collecting questions related to checked-in secrets, we selected Stack Exchange [12] which has been extensively used to gain insights from developers' questions to align future research and guide tools providers [14], [15]. Stack Exchange consists of 179 Q&A sites [12]. We extract the name and description of all the sites and manually read them. Then, we select sites that allow questions related to software development, software engineering, and software security. For example, the site "Software Engineering" can feature queries from developers, according to the site description "Q&A for professionals, academics, and students working within the systems development life cycle". The first author selected three Q&A sites: "Stack Overflow" [16], "Information Security" [17] and "Software Engineering" [18]. The basic statistics of the three sites are shown in Table I. In Step 2, we use these sites for question collection.

TABLE I: Basic statistics of Stack Overflow (SO), Information Security (IS) and Software Engineering (SE) sites[1]

Site	#Questions	#Answers	#Users	#Questions/Day
SO	23m	34m	18m	5.5k
IS	66k	114k	228k	9.6
SE	61k	173k	352k	5.5

B. Step 2: Content Collection

Start with initial tags and keywords for title and body: To increase the likelihood of speedy response and aid in automated search, each question can be given one or more tags [19]. Tags allow the extraction of questions that are specific to a given technology. For example, the tag "secret-key" can be used for identifying questions related to checked-in secrets according to the tag description "Use this tag for questions related to the creation, storage and usage of secret keys". Initially, we select "secret-key" and "access-keys" tags. Users can also post questions without giving tags. To avoid missing candidate questions, we use secrets-related keywords, such as "expose", "protect", and "sensitive", to search in the body and title of the questions to extract relevant questions.

[1]Based on data retrieved from the Stack Exchange Data Explorer [12] on June 2022

Extract questions from Stack Exchange data explorer: The Stack Exchange dataset is accessible publicly via data dumps [20] and the Stack Exchange data explorer [19]. The data dumps are released quarterly, whereas the online Stack Exchange data explorer provides the most recent data. We use the tags and keywords in a SQL query and extract data from the Stack Exchange data explorer instead of data dumps. We collect the ID, title, body, accepted answer, view count, score, creation date, closed date, and tags of each extracted question from the three sites identified in Step 1. We collected 6022, 2591, and 1415 questions from Stack Overflow, Information Security, and Software Engineering sites, respectively.

Identify relevant questions: We manually inspected each question's title and body and accepted questions with a discussion related to checked-in secrets while rejecting all others.

Find new relevant tags and keywords: We use snowball sampling [21] which is a non-probability sample selection technique to locate hidden populations by relying on the characteristics of initial sample. Since a question can have multiple tags, we find new relevant tags by looking at all the tags present in the questions. For example, the question "Where to keep static information securely in Android app?" [22] can be found by "secret-key" tag. The question also has tags "access-token" and "security" which we can add to our list of tags for finding more questions. Similarly, add new keywords by reading the title and body of the question. Altogether, we used 59 tags and 42 keywords which can be found in Table II.

TABLE II: List of Tags and Keywords used to extract questions from Stack Exchange sites

Tags	Keywords
secret-key, access-keys, access-token, security, credentials, passwords, api-key, private-key, app-secret, connection-string, sensitive-data, environment-variables, config-files, certificate, configuration, google-api, amazon-s3, oauth, youtube-api, stripe-api, square, paypal, braintree, amazon-mws, gmail-api, twilio-api, mailgun, mailchimp, google-drive-api, key-management, development-process, coding-style, password-protection, source-code-protection, code-security, source-code, secure-coding, open-source, azure-key-vault, password-storage, password-management, key-exchange, confidentiality, sensitive-data-exposure, web-development, git, gitignore, version-control, github, svn, tfs, gitlab, repository, bitbucket, launchpad, mercurial, git-rewrite-history, git-history, git-filter-branch	expose, exposing, protect, protecting, sensitive, remove, removing, commit, committing, share, sharing, keep, keeping, manage, managing, delete, deleting, clear, clearing, ignore, ignoring, secure, securing, store, storing, hide, hiding, avoid, avoiding, push, pushing, host, hosting, security, connection string, secret, password, credential, private key, token, api key, access key

Repeat and stop criteria: We repeat the previous step until we no longer found new tags and keywords in each set of extracted questions.

Finally, we identified 694 questions in Stack Overflow, 40 questions in Information Security, and 45 questions in Software Engineering. In total, we identified 779 questions from the three sites spanning from September 2008 to December 2021 which are available online [23]. The count of questions from each year before and after filtering is shown in Table III.

TABLE III: Question Count Per Year for Stack Overflow (SO), Information Security (IS) and Software Engineering (SE) sites

Year	SO[a]	SO[b]	IS[a]	IS[b]	SE[a]	SE[b]	Total[a]	Total[b]
2008	23	4	0	0	0	0	23	4
2009	136	22	0	0	0	0	136	22
2010	212	30	5	0	28	1	245	31
2011	284	43	73	0	163	5	520	48
2012	370	44	129	2	170	8	669	54
2013	447	48	160	6	146	7	753	61
2014	485	41	257	5	136	3	878	49
2015	481	47	361	5	152	4	994	56
2016	581	51	340	6	126	2	1047	59
2017	581	76	323	5	110	4	1014	85
2018	538	63	268	5	107	4	913	72
2019	518	54	235	1	102	3	855	58
2020	722	88	226	1	90	2	1038	51
2021	644	83	214	4	85	2	943	89

[a] Total number of questions before filtering
[b] Total number of questions after filtering

C. Step 3: Identifying Question and Answer Categories

From the 779 checked-in secrets-related questions, two authors independently apply card sorting [13], a qualitative analysis technique, to identify the question and answer categories. Card sorting is a qualitative technique for classifying textual items into categories [13]. Card sorting aids in creating informative categories and is commonly used in research [15]. The following three phases of card sorting are implemented in accordance with Zimmerman et al. [13]'s recommendations.

Preparation: Each question's ID, title, body, and accepted answer are collected.

Execution: The first and second authors perform card sorting by giving labels to each question and the corresponding answer and sort into categories. The body and title of the questions are used to derive question categories, whereas the accepted answers are used to derive answer categories.

Analysis: The obtained question and answer categories are cross-checked by both authors after the first and second authors finish their card sorting analysis individually. We use a negotiated agreement [24] to resolve the disagreed-upon categories. A negotiated agreement is an approach to discuss the disagreements among the raters to resolve disagreements when two or more raters code the same artifacts [24]. We resolve disagreements by discarding categories inappropriate for checked-in secrets or combining similar categories into one category. The first author determines 32 unique question categories and 16 unique answer categories. The second author determines 30 unique question categories and 14 unique answer categories. The first and second authors finalize 27 question and 13 answer categories by resolving the disagreements presented in Table IV and Section III, respectively.

D. Step 4: Analysis

We use the identified question and answer categories from Step 3 to answer our research questions.

1) RQ1: What are the technical challenges faced by developers related to checked-in secrets? We break down RQ1 into four sub-research questions as below:

- **RQ1.1** What are the questions developers ask about checked-in secrets?
- **RQ1.2** Which questions related to checked-in secrets exhibit more unsatisfactory answers?
- **RQ1.3** Which questions are the most popular among developers related to checked-in secrets?
- **RQ1.4** How do question categories related to checked-in secrets trend over time?

We investigate the four sub-research questions as following:

RQ1.1: What are the questions developers ask about checked-in secrets? We first provide the set of question categories to answer RQ1.1 along with a description and an example of each category which we determine from Step 3. Next, we compute the proportion of questions for each category x, QC(x).

RQ1.2: Which questions related to checked-in secrets exhibit more unsatisfactory answers? A question with no accepted answer could indicate that the developer who asked the question was dissatisfied with the responses. Lacking accepted answers or having no answers may suggest an important category that needs assistance. We answer RQ1.2 by quantifying which of the checked-in secrets-related question categories has more questions with unsatisfactory answers. We compute the proportion of questions with unsatisfactory answers for question category x, UNC(x).

Furthermore, we compute the proportion of questions with unsatisfactory answers for each year y, TUN(y) to see how the proportion of unsatisfactory answers related to checked-in secrets has changed over time.

RQ1.3: Which questions are the most popular among developers related to checked-in secrets? Developers can view a question and corresponding answers without becoming registered users on Stack Exchange. The number of total visits for a question by registered and non-registered users of the website is used to calculate the View Count of a question [19]. The View Count can help us observe which questions are most popular among the developers. Registered users can also vote up or down on questions. Upvotes indicate that users find the question helpful, well-researched, or thought-provoking. Downvotes indicate that users believe the question lacks real explanation, contains misleading information, or is poorly researched. A question's Score on Stack Exchange is calculated by subtracting the number of downvotes from the number of upvotes [25]. Rather than selecting a single metric, we use both View Count and the Score of the question as a better approximation for question popularity. Previous studies use a similar a popularity metric [14].

We use Spearman's rho ρ [26] to verify the rank correlation between View Count and Score. View Count is found to have a significant correlation with Score (ρ = 0.72, $\alpha < 0.001$). We use Feature Scaling [27] to normalize the View Count and Score values of each question by Equation 1 since the range of both the metrics are different.

$$X_{nor} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$

where X denotes the original value, X_{min} denotes the range's minimum value, X_{max} denotes the range's maximum value and X_{nor} denotes the normalized value.

To determine how popular a question is, we use the average of normalized View Count and Score values. Next, we calculate the popularity of each category x, PQ(x) using Equation 2. A question category x with a high popularity score means developers need support to mitigate the specific challenge.

$$\text{PQ}(x) = \frac{\text{sum of popularity score of questions in category } x}{\text{total questions in category } x} \quad (2)$$

RQ1.4: How do question categories related to checked-in secrets trend over time? We examine temporal trends, similar to previous studies [15], [28], to see how the number of questions relevant to the identified question categories changes over time. We first use Equation 3 to compute the temporal trend of category x for each month m.

$$\text{TT}(x, m) = \frac{\text{number of questions of category } x \text{ in month } m}{\text{number of questions in month } m} \quad (3)$$

Then, to see whether the observed trend is significantly increasing or decreasing, we use the Cox-Stuart test [29], a statistical method that compares earlier data points in a time series to later data points to evaluate the trend. To assess which question categories have increasing or decreasing trends, we apply a 95% statistical confidence level ($p < 0.05$). We term the temporal trend to be "Consistent" if we can not determine whether the trend is increasing or decreasing.

2) RQ2: What solutions do developers get for mitigating checked-in secrets? To answer RQ2, we first provide the answer categories to mitigate the challenges related to checked-in secrets, which we determine from Step 3. Then, we provide a mapping of answer categories to each of the question categories. From the question-answer category mapping, we can understand the solutions posed by developers to mitigate a specific technical challenge.

III. Results

In this section, we discuss our findings and answer our research questions.

A. Answer to RQ1: What are the technical challenges faced by developers related to checked-in secrets?

We answer the four sub-research questions of RQ1 in the following sub sections.

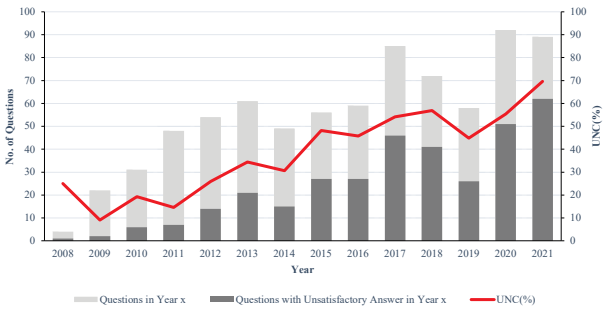

Fig. 1: Trend of Unsatisfactory Answer Per Year

1) Answer to RQ1.1: What are the questions developers ask about checked-in secrets? We identify 27 unique question categories of 9 domains, which we present in Table IV sorted based on the number of questions in a domain. The domain name, question category name, a description of the question category, and a representative example are provided for all the question categories. The number of questions in each category is indicated in parenthesis in the "Category" column.

The proportion of questions in each identified question category and the other four metrics mentioned in Section II are presented in Table V. The proportion of questions, percentage of unsatisfactory answers, popularity score, Cox-Stuart test value of temporal trend of questions, and the identified trend of questions in each question category are represented in the columns "QC(%)", "UNC(%) (Count)", "PQ", "Cox Stuart, p-value" and "Trend" respectively. According to Table V, the top four question categories based on QC metric are "(Deployment) Store/Version", "(Secrets) Store/Version", "(Secrets) Ignore/Hide", and "(VCS Feature) History Sanitize". These four categories constitute 56.1% of all questions.

2) Answer to RQ1.2: Which questions related to checked-in secrets exhibit more unsatisfactory answers? Table V shows that UNC scores of more than 40% are found in 16 of the 27 identified question categories. Our finding indicates that 44.3% of questions within our dataset have unsatisfactory answers. The top four question categories, "(Deployment) Store/Version", "(Secrets) Store/Version", "(Secrets) Ignore/Hide" and "(VCS Feature) History Sanitize" have UNC scores of 43.0%, 47.1%, 34.1% and 45.7% respectively.

Figure 1 presents the trend of unsatisfactory answers for each year between 2008 and 2021. We observe that the percentage of unsatisfactory answers shows an increasing trend. More than 50% of questions have unsatisfactory answers since 2017, thus indicating that the developers are not getting desired answers to mitigate the challenges of checked-in secrets.

3) Answer to RQ1.3: Which questions are the most popular among developers related to checked-in secrets? The popularity of each question category is presented in the "PQ" column of Table V. In our study, the popularity score varies between 0.005 and 0.030. For example, a question with Score 0 and View Count 17 has a PQ score of 0.005, whereas a question

TABLE IV: 27 question categories. References to all the examples and developer quotes are available online [30]

Domain	Category (Count)	Description	Example
Secrets	Q1: Store/Version (121)	We observe that the same questions of knowing the best way to store secrets have been asked for different technologies, such as ASP.NET and Python. We also observe developers asking about versioning the secrets for environments, such as development and production environments, where they do not know the consequences of storing secrets in VCS repositories.	*How should I store a password used by a service written in .NET?*
	Q2: Ignore/Hide (85)	We observe that developers are aware of the consequences of secrets presence in the source code and want to hide the secrets. As one developer stated: *"The credentials are hard-coded at the moment, but they should not be. What is the proper way of hiding them?"*. Developers also question about challenges faced in avoiding secrets from being committed to the VCS repository.	*Hide API keys from github public repo?*
	Q3: Exploitability (30)	Developers do not know whether storing a secret such as a Google API key or testing credentials in source code or a VCS repository can be exploited. For example, one developer stated: *"I'm making use of google API for location. Can the key be hardcoded? ... If it's sensitive, why is it sensitive and how can attackers exploit this?"*.	*Is having sensitive data in a PHP script secure?*
	Q4: Distribute (11)	Developers ask questions about sharing secrets with other developers so that they can run the project successfully in their environment. As one developer stated: *"How can I keep my API key secret, but have my project still be functional if someone clones the repo?"*. We observe that developers are unsure how to share secrets with specific developers without exposing them.	*Push to GitHub that project is still functional when the repo cloned?*
	Q5: Restriction (2)	We observe questions posted for restricting a specific group of developers from having access to secrets. For example, *"What happens if a malicious developer decides to steal the secret (say, an API key) and use it for malicious purposes? Is there a way to store secrets such that a backend developer doesn't have direct access to the API Key?"*.	*What are ways to manage secrets in a big organisation?*
Deployment	Q6: Store/Version (149)	Platform as a service (PaaS), such as Heroku [31] and Google App Engine [32], are commonly used to manage applications. During deployment, the code is fetched from the repositories. We observe developers asking questions about where to store the secrets needed for deployment since secrets are not pushed in the repository. Developers want to know the secure way of versioning secrets for deployment environments. This question category is the most frequently asked.	*Where to store sensitive files for heroku platform?*
	Q7: Improper Configuration (34)	As the configuration (config) files are ignored in the repository and source code is fetched from the repository for deployment, developers are getting exceptions due to improper configuration in the deployment server. We observe developers asking for help resolving the build and deploy-related exceptions. We observe that most of the exceptions are during Django application deployment.	*Azure Django App has SECRET_KEY Exception*
	Q8: Ignore/Hide (15)	During the build and deployment of an application, developers use the secrets present in the continuous integration and continuous deployment (CI/CD) scripts or the VCS repository. We observe developers asking to know the best practice of hiding the secrets from CI/CD scripts or repositories and perform successful build and deploy.	*Docker-Compose with Gitlab CI managing sensitive data*
	Q9: Dot File (3)	Developers deploy directly from VCS repositories using Git tools. They push sensitive dot files such as .git and .gitignore files that can be accessed at the website's root location. Previous research [4] has found secrets in the .gitignore file, even though the .gitignore file is designed to restrict unintended source files committing into VCS. We observe developers facing challenges restricting the dot files' access from the website's root.	*How to make .gitignore safe?*
VCS Feature	Q10: History Sanitize (81)	Developers accidentally or knowingly push sensitive information into the VCS repository. One developer stated: *"I am using a shared github repository to collaborate on a project ... I committed and pushed a script file containing a password which I don't want to share"*. The sensitive information remains in the VCS history even when removed in another commit. Developers ask questions about sanitizing the VCS history using different tools but could not use the tools properly. Rahman et al. [33] also observed developers bypassing secret scanning tools warning because of facing technical challenges of eliminating secrets completely from the VCS history.	*How to remove sensitive data from a file in github history?*
	Q11: Ignore Already Committed (14)	Knowing the exploitability of secrets present in source code, developers want to commit a default file without secrets. However, they want to untrack further local changes of the file from VCS repositories to avoid accidentally committing the local changes, and VCS does not support the functionality [34]. As a result, we observe developers ask questions about ignoring an already-committed file from VCS tracking.	*Stop tracking file in Git after a first commit?*
	Q12: Line Level Security (11)	*"Do any version control systems allow you to specify line level security restrictions rather than file level?"* stated by one developer. VCS, such as Git, only supports file-level restrictions. We observe developers wanting to mark specific lines in a file that contains secrets and tell the VCS to secure the lines to avoid exposing the secrets.	*hide or change value a line at git commit but not locally*
	Q13: Encrypt File (1)	We observe developers asking questions about if there is a way to encrypt a secrets-containing file before committing to VCS repositories.	*Encrypting files added to repos*
Configuration File	Q14: Store/Version (56)	Config files contain secrets. We observe developers face challenges storing the config files in the VCS repository since it would expose the secrets. For example, one developer stated: *"I'd like to version control the whole project, including config file, but I don't want to share my passwords"*.	*Preferred way to store application configurations?*
	Q15: Ignore/Hide (32)	We observe developers asking questions about ignoring or hiding sensitive secrets-containing config files such as the web.config and database.yml files from the VCS repository. Developers also complain about the lack of documentation or suggestions the specific technology provides on ignoring config files.	*Protecting the sensitive files from pushing to version control?*
	Q16: Distribute (9)	Developers face challenges sharing secrets-containing config files with other team members without exposing them publicly. For example, one developer stated: *"Should I add these 2 files to versioning or do I have to distribute these files manually to other team members?"*.	*Managing project config files in repository?*
	Q17: Exploitability (3)	Developers place environment variables replacing secrets in the config files and want to confirm the exploitability from outside. We also observe developers placing secrets in PHP .ini files and asking about the exploitability of the secrets. For example, one developer stated: *"Is better to hide somewhere .ini file and deny access via .htaccess?"*.	*Storing sensitive info. inside .ini file is good or bad approach?*
	Q18: Accessibility (3)	To avoid exposing secrets, developers load secrets dynamically by referencing external files in config files but get an undefined error. An example includes loading an external database settings file into a web.config file. We observe developers facing challenges in avoiding the undefined error and could not find the proper documentation.	*How to securely use credentials outside web.config?*
Pre-open Source	Q19: Cross-check (52)	We observe developers asking questions before open-sourcing their projects. The questions include should developers clean VCS history and what checklists should they run to avoid exposing secrets.	*OpenAuth & Open Source Projects?*

TABLE IV: 27 question categories (Continued). References to all the examples and developer quotes are available online [30]

Domain	Category (Count)	Description	Example
Client-Side Application	Q20: Store (28)	Developers work on client-side applications without a server-side implementation and store secrets on the client-side, such as in Javascript and Android applications. Developers face challenges in storing the secrets securely as secrets can easily be exposed from the developer console or by decompiling the binary packages.	*Securely storing secret data in a client-side web application?*
	Q21: Hide (14)	One developer stated: *"Using Javascript however, I don't feel comfortable that the client secret is exposed in my code ... because if someone looks at my source they have the client_id and client_secret which makes it possible to authenticate themselves with my code"*. We observe developers looking for ways to hide client-side application secrets.	*How do I hide API key in create-react-app?*
	Q22: Exploitability (5)	Developers ask questions to confirm whether the implementation of keeping secrets in the client-side application code is exploitable or not. *"Could I sleep at night knowing that I won't see "Super Cool Web App Hacked, change your passwords!" all over HN and Reddit ... as a result of this implementation."* stated by one developer.	*In iOS, is there leak risk if I write the secret key in the code?*
Secureness	Q23: Private Repository (13)	One developer stated: *"Is it safe for me to store my Amazon S3 keys/secrets in a private Github repo? I know that it is not safe for a public repo but I am wondering if a private repo is safe?"*. We observe developers asking about the safety of secrets present in a private repository.	*Storing Amazon S3 keys in private repo*
	Q24: Unpushed Branch (1)	We observe developers ask questions about the security of secrets if they do not push the secrets-containing branch to a public repository. For example, one developer stated: "Is there any chance my sensitive data could end up in the remote repository index somehow?".	*Commit password to branch that never pushed?*
External Secret Management	Q25: Setup (3)	We observe developers moving secrets to external secret management services, such as HashiCorp Vault [35] and Azure Key Vault [36]. However, developers face challenges in properly setting up these hardware security modules. Examples of such questions include where to store the vault key, the feasibility of using vaults, and how to store the database connection strings in the vault.	*Storing DB Connection Strings in Azure Key Vault*
Others	Q26: Importance (2)	We observe developers asking questions about why they should keep secrets out of the VCS repository. For example, one developer stated: *"It seems like common knowledge that it's a good practice to keep secrets files ... checked out of your git repository ... Why?"*.	*Why should you keep secrets out of your repository?*
	Q27: Decision (1)	One developer stated: *"Today I found what looked to be my supervisor's password in some code in version control ... How should I handle this situation?"*. We observe developers being hesitant about making decisions when they find secrets in the VCS repository.	*What should I do when I find sensitive info in VCS?*

TABLE V: Summary of identified question categories, sorted by decreasing question proportion (QC)

(Domain) Question Category	QC (%)	UNC (%) (Count)	PQ	Cox Stuart, p-value	Trend
(Deployment) Store/Version	19.2	43.0 (64)	0.020	$\Uparrow$, 0.11	Consistent
(Secrets) Store/Version	15.6	47.1 (57)	0.023	$\Uparrow$, 0.003	Increasing
(Secrets) Ignore/Hide	10.9	34.1 (29)	0.015	$\Uparrow$, 0.11	Consistent
(VCS Feature) History Sanitize	10.4	45.7 (37)	0.018	$\Uparrow$, < 0.001	Increasing
(Configuration File) Store/Version	7.2	39.3 (22)	0.022	$\Uparrow$, 0.5	Consistent
(Pre-open Source) Cross-check	6.7	40.4 (21)	0.010	$\Downarrow$, 0.3	Consistent
(Deployment) Improper Configuration	4.4	58.8 (20)	0.008	$\Uparrow$, < 0.001	Increasing
(Configuration File) Ignore/Hide	4.1	40.6 (13)	0.008	$\Downarrow$, 0.34	Consistent
(Secrets) Exploitability	3.9	56.7 (17)	0.014	$\Downarrow$, 0.59	Consistent
(Client-Side Application) Store	3.6	60.7 (17)	0.030	$\Uparrow$, 0.002	Increasing
(Deployment) Ignore/Hide	1.9	46.7 (7)	0.010	$\Uparrow$, 0.09	Consistent
(VCS Feature) Ignore Already Committed	1.8	28.6 (4)	0.007	$\Uparrow$, 0.29	Consistent
(Client-Side Application) Hide	1.8	35.7 (5)	0.022	$\Uparrow$, 0.13	Consistent
(Secureness) Private Repository	1.7	46.2 (6)	0.014	$\Uparrow$, 0.27	Consistent
(Secrets) Distribute	1.4	63.6 (7)	0.007	$\Uparrow$, 0.11	Consistent
(VCS Feature) Line Level Security	1.4	36.4 (4)	0.007	$\Downarrow$, 0.5	Consistent
(Configuration File) Distribute	1.2	66.7 (6)	0.007	$\Uparrow$, 0.14	Consistent
(Client-Side Application) Exploitability	0.6	40.0 (2)	0.008	$\Uparrow$, 0.5	Consistent
(Configuration File) Exploitability	0.4	0.0 (0)	0.012	$\Uparrow$, 0.5	Consistent
(Configuration File) Accessibility	0.4	33.3 (1)	0.008	$\Uparrow$, 0.13	Consistent
(Deployment) Dot File	0.4	33.3 (1)	0.007	$\Uparrow$, 0.5	Consistent
(External Secret Management) Setup	0.4	66.7 (2)	0.015	$\Uparrow$, 0.13	Consistent
(Others) Importance	0.3	100.0 (2)	0.005	$\Uparrow$, 0.25	Consistent
(Secrets) Restriction	0.3	50.0 (1)	0.007	$\Downarrow$, 0.75	Consistent
(VCS Feature) Encrypt File	0.1	0.0 (0)	0.008	$\Downarrow$, 0.5	Consistent
(Secureness) Unpushed Branch	0.1	0.0 (0)	0.005	$\Downarrow$, 0.5	Consistent
(Others) Decision	0.1	0.0 (0)	0.008	$\Downarrow$, 0.5	Consistent

with Score 12 and View Count 17847 has a PQ score of 0.030. The top three most popular question categories are "(Client-Side Application) Store", "(Secrets) Store/Version" and "(Client-Side Application) Hide". In Table VI, we also provide the question categories in descending order, sorted by PQ and UNC(%). Further observations are aided by the ranking of the 27 question categories:

- "(Client-Side Application) Store" and "(Client-Side Application) Hide" rank first and third based on the popularity score (PQ) and have a UNC score of 60.7% and 35.7%, respectively. The observation indicates that the questions related to storing and hiding secrets in client-

TABLE VI: Ranked Order of Question Categories Based on Popularity (PQ) and Unsatisfactory Answer Percentage (UNC)

Metric	(Domain) Question Category (Sorted in decreasing order of metric)
PQ	(Client-Side Application) Store, (Secrets) Store/Version, (Client-Side Application) Hide, (Configuration File) Store/Version, (Deployment) Store/Version, (VCS Feature) History Sanitize, (Secrets) Ignore/Hide, (External Secret Management) Setup, (Secureness) Private Repository, (Secrets) Exploitability, (Configuration File) Exploitability, (Pre-open Source) Cross-check, (Deployment) Ignore/Hide, (VCS Feature) Encrypt File, (Configuration File) Accessibility, (Others) Decision, (Configuration File) Ignore/Hide, (Client-Side Application) Exploitability, (Deployment) Improper Configuration, (Deployment) Dot File, (Secrets) Restriction, (Secrets) Distribute, (Configuration File) Distribute, (VCS Feature) Line Level Security, (VCS Feature) Ignore Already Committed, (Others) Importance, (Secureness) Unpushed Branch
UNC (%)	(Others) Importance, (Configuration File) Distribute, (External Secret Management) Setup, (Secrets) Distribute, (Client-Side Application) Store, (Deployment) Improper Configuration, (Secrets) Exploitability, (Secrets) Restriction, (Secrets) Store/Version, (Deployment) Ignore/Hide, (Secureness) Private Repository, (VCS Feature) History Sanitize, (Deployment) Store/Version, (Configuration File) Ignore/Hide, (Pre-open Source) Cross-check, (Client-Side Application) Exploitability, (Configuration File) Store/Version, (VCS Feature) Line Level Security, (Client-Side Application) Hide, (Secret) Ignore/Hide, (Configuration File) Accessibility, (Deployment) Dot File, (VCS Feature) Ignore Already Committed, (Configuration File) Exploitability, (Others) Decision, (Secureness) Unpushed Branch, (VCS Feature) Encrypt File

side applications are most popular among developers but do not receive satisfactory answers. Therefore, future research is needed on the client-side frameworks for securely managing secrets.

- "(Secrets) Store/Version" ranks second based on the popularity score (PQ) and has a UNC score of 47.1%. Our observation indicates that developers are showing more interest in the question of securely storing secrets for different technology frameworks such as ASP.NET, Ruby on Rails and Python. But, developers could not implement properly because of lacking proper documentation.
- "(Secrets) Distribute" and "(Deployment) Improper Configuration" question categories rank fourth and sixth for unsatisfactory answers, respectively. However, these question categories rank 22^{nd} and 19^{th} based on popularity score. Though the popularity score is low, developers are not receiving satisfactory answers for distributing secrets and fixing improper configuration errors during deployment. Therefore, future research can address secure secret distribution, and respective technology providers can provide proper documentation to fix improper configuration errors during deployment.
- We observe developers searching for VCS features to ignore the tracking of already-committed files to avoid local changes being accidentally committed in the VCS repository. An option exists to delete the file from remote repository and then ignore the file by placing the file name in the .gitignore file. However, developers do not want to delete and want a copy of the file in the remote repository, which VCS does not support [37]. Developers are also looking for line-level restrictions in VCS to hide secrets in particular lines of the source code. Though VCS has a feature called git smudge-clean [38] which can be used to replace a secret with a dummy value during commits, developers face difficulties in implementing the process. Despite "(VCS Feature) Ignore Already Committed" and "(VCS Feature) Line Level Security" ranking 25^{th} and 24^{th}, respectively, based on popularity score, the two question categories consist of 25 questions where developers are seeking the new VCS feature.

4) Answer to RQ1.4: How do question categories related to checked-in secrets trend over time? Figure 2 depicts the temporal trend of 15 question categories that have at least 10 questions. For each category, the figure provides a scatter plot with a smoothing plot with the trends highlighted. We can understand whether the trend of each question category is increasing, decreasing, or consistent from the "Cox Stuart", "p-value" and "Trend" columns of Table V. Table V highlights the question categories with a p-value less than 0.05 in grey.

From Table V, we observe an increasing trend in four question categories. While only four question categories showed increases, the trend is across 13 years of the data. We also observe that developers are posting more questions in "(Secrets) Store/Version", "(VCS Feature) History Sanitize", "(Deployment) Improper Configuration", and "(Client-Side Application) Store" categories, but their questions are not well-answered. The four question categories have a UNC score of more than 45%, and three out of four question categories are also in the top six categories based on the popularity score (PQ). The increasing trend of these four question categories substantiates the absence of proper documentation on managing secrets during the deployment and the need for future research on client-side frameworks. In addition, the increasing trend also substantiates the need to improve existing VCS history sanitizing tools to make integration easier for developers.

B. Answer to RQ2: What solutions do developers get for mitigating checked-in secrets?

We identify 13 answer categories from our analysis, which we present below based on the descending order of the number of questions in which StackExchange users suggest the specific answer category. For example, 179 answers to the 779 questions suggest the 'A1: Move Secrets out of Source Code/Version Control and Use Template Config File' category. We do not declare all the answer categories as best practices. Indeed, below we highlight the shortcomings of these answer categories as appropriate.

A1: Move Secrets out of Source Code/Version Control and Use Template Config File (179): Developers may put secrets, such as database credentials, in a file where the code for database functionalities are present. As a result, developers face challenges in hiding the credentials from VCS repositories. In such cases, developers are suggested to move the secrets to a config file. Then, the config file with original secrets

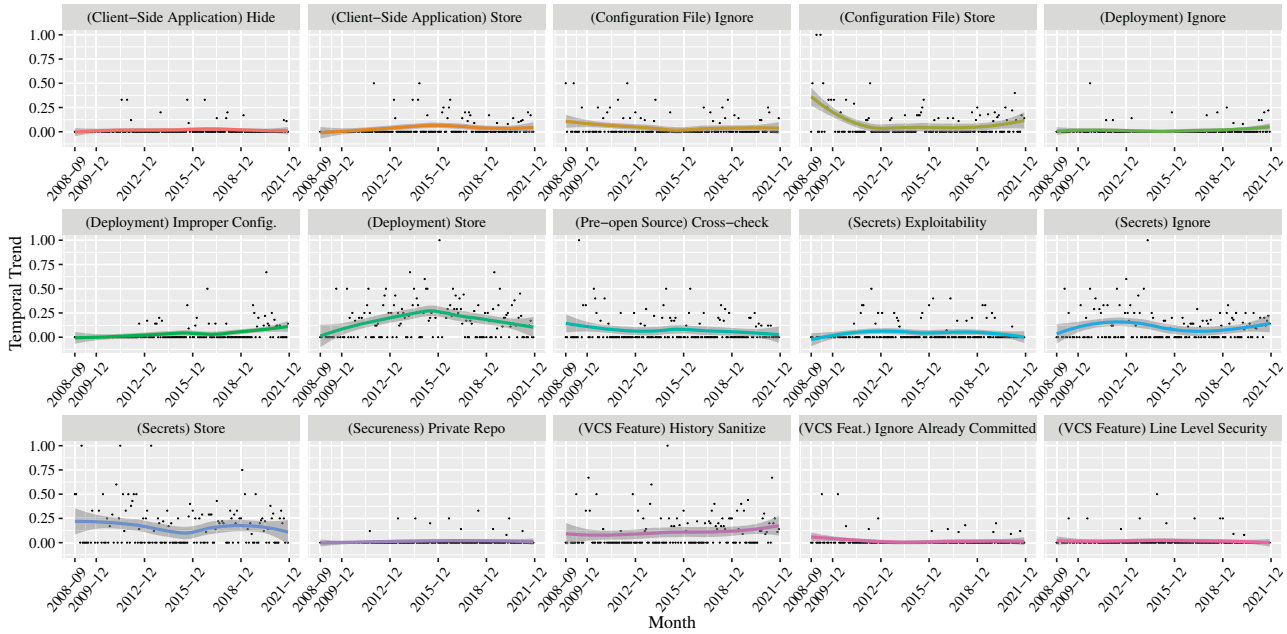

Fig. 2: Temporal Trend of each identified question category. The month of the x-axis is shown in three-year interval. The zero value of Temporal Trend indicates no question is posted on the specific month for a category.

should be ignored from the VCS repository, and a template config file should be committed to the repository. Template config files, such as database.sample.yml file of Ruby on Rails, contain the minimum configurations with dummy secrets to avoid build failure. Developers will replace the dummy secrets in their development environment. Furthermore, a .gitignore file should be included with all repositories to ignore the secrets-containing files. GitHub has published a collection of .gitignore templates [39] for different technologies.

A2: Secret Management in Deployment (78): We observe that developers mostly face challenges storing or versioning secrets for multiple environments during deployment. Configuration management systems, such as Ansible-Vault [40] and Chef-Vault [41], provide support for secret management. Developers are advised to use deployment variables, such as Heroku Config Vars [42], which create environment variables for respective environments. Developers are also suggested to keep the dot files such as .git and .hg files out of the root directory during deployment to avoid exposing secrets.

A3: Use Local Environment Variables (56): An environment variable is a dynamic object which is set outside of the application and used to avoid the storage of secrets in code or local config files. Developers are suggested to use environment variables to load the secrets at runtime. The benefits of using environment variables are switching secrets between deployed versions without modifying any code and making it less likely that secrets get checked into the repository. However, environment variables can leak secrets as they are passed down to child processes, which allows for unintended access [43].

A4: Rewrite VCS History (48): Secrets will not be removed entirely by removing in another commit as secrets will remain in the VCS history. Developers suggest removing secrets using git-filter-repo [44], git-filter-branch [45], and BFG repo cleaner [46]. Though official GitHub documentation [47] suggests using BFG repo cleaner instead of git-filter-repo and git-filter-branch, we have seen Stack Exchange users mostly suggest using the latter. GitHub has also suggested contacting them with the repository name to clear the secrets from their cache and advised to tell the project collaborators to do git rebase instead of git merge [47] though no Stack Exchange users' solutions suggested these actions.

A5: Store Encrypted/Obfuscated Secrets (39): Storing secrets as encrypted, encoded, or obfuscated is one of the solutions suggested by Stack Exchange users. Different encryption algorithms, such as AES and RSA, are suggested. In some cases, developers are suggested to encode secrets using Base64 encoding in Android applications. Another suggestion is to split the secrets into multiple parts and keep them in the source code. The number of parts should be high, so the attacker will have to check for more than a billion permutations. Tools such as git-secret [48] and git-crypt [49], are available for encrypting secrets-containing files. The disadvantage of encryption is to deal with the encryption keys securely.

A6: Use of External Secret Management Service (26): Developers are recommended to implement external secret management services, such as HashiCorp Vault [35] and AWS KMS [50]. These hardware security modules can safely store secrets with tightly-controlled access. However, because they are challenging to set up and maintain, these solutions may be unsuitable in some situations. In addition, they need a

significant investment of time and money.

A7: Load Externally and Use Secondary Private Repository (23): Since developers want to avoid committing secrets into VCS that are needed for the application's functioning, developers are advised to load secrets externally using AWS S3 or a secondary private repository. Since AWS S3 needs access keys to retrieve stored files, the same problem of storing the access keys may occur. A secondary private repository can be used to store secrets and loaded dynamically using git submodules [51]. However, private repositories are not free from exploitation by attackers [52].

A8: Revocation and Rotation (16): The first step to stop secrets sprawl is to revoke the secrets immediately. One developer suggested: *"The important bit: Consider your credentials compromised. Change them. No matter what you do at this point, they are no longer secure"* [30]. A good practice is to rotate the secrets periodically. Short-lived secrets prevent previously-undetected data breaches from posing a threat, as access will be cut off even if the breach is not identified.

A9: Server-Side Implementation (16): To avoid keeping secrets in client-side applications for fetching data from web services, developers are recommended to implement web service functionality on the server side. Then, the server will use the secrets and fetch data for the client side, thus removing the necessity to keep secrets in client-side applications.

A10: VCS Feature (Git Hooks and Flags) (10): To avoid secrets from pushing in VCS repositories, developers are suggested to implement git hooks [53] and git flags [54], [55]. The pre-commit and post-commit hooks can be used to filter and smudge before commit or after pull, respectively [38]. However, developers are warned as implementing git hooks properly is difficult. Developers are also suggested to use the git flags such as –skip-worktree [55] and –assume-unchanged [54] to prevent changes from being committed to existing files.

A11: Add Files to the Staging Area Explicitly (3): A simple strategy to avoid exposing secrets accidentally is to add files explicitly in the VCS staging area. Developers are suggested to avoid using wildcards (git add -A or git add *) for adding files, thus having complete control and visibility over what files are committed.

A12: Restrict API Access and Permissions (3): Since attackers frequently use secrets within their scope, detecting when they are doing so maliciously might be challenging. However, damage and lateral movement can be limited by restricting access and permissions of the secrets. For example, GitHub IP white-listing [56] can be employed to prevent any untrusted sources from accessing the GitHub repositories.

A13: VCS Scan Tools (1): Developers are advised to run VCS scan tools, such as TruffleHog [57] and Gitrob [58], before any commit or in an existing repository to find out the presence of secrets. The tools can find secrets buried in histories that manual searches and reviews will miss. However, tools may return a significant number of false positives [33].

The mapping of answers to each question category can be found online [59]. We observe that the same answer category has been mentioned to mitigate challenges of multiple question categories. For example, 'A1: Move Secrets out of Source Code/Version Control and Use Template Config File', 'A3: Use Local Environment Variables' and 'A2: Secret Management in Deployment' have been mentioned as part of a solution in 20, 12, and 10 out of 27 question categories, respectively.

IV. Discussion and Recommendations

Below we discuss our findings and make recommendations. In our discussion, we trace the questions and answers by their identifiers assigned in Table IV and Section III-B, respectively.

Tool enhancement. We find that developers face difficulty with properly sanitizing VCS history (Q10). Developers commonly use git-filter-branch [45] and git-filter-repo [44] to sanitize VCS history. However, both the tools have safety and usability issues which can easily corrupt the repository's history [60]. For example, these tools can easily mix up the old and new history of the repository. In addition, coming up with the correct shell script is difficult as developers find out if the sanitizing code script is right or wrong by trying the script out. Even worse, broken filters often result in silent incorrect rewrites without proper output. Even if the developers sanitize the VCS history properly using the tools, the tools can not clear the cache in the respective version control systems, such as GitHub, as the sensitive information can appear again from the cache, according to GitHub's official documentation[3]. As of now, clearing from the cache is a manual process that can be automated.

In addition, we observe that developers are suggested to use VCS scan tools (A13) to avoid accidentally committing secrets, but developers seem to bypass scan tool warnings due to high false positives [33]. There are currently many open-source and proprietary VCS scan tools [61], but developers find it challenging to choose one tool out of many. Researchers and tool developers can work on comparing the effectiveness and efficiency of the VCS scan tools and improving the tools by reducing false positive warnings.

We also found that developers want new VCS features, such as line-level security, where developers can quickly point to the specific lines to which they want to restrict visibility in the VCS (Q12). In addition, we found that developers want to ignore local changes of already-committed files from VCS tracking without removing the file from the repository (Q11). Though Stack Exchange users suggested using –assume-unchanged [54] and –skip-worktree [55] flags to ignore local changes of already-committed files from VCS tracking (A10), the official Git documentation suggests these flags not be used [34].

Recommendation 1: We recommend improving the existing tools, such as making the integration of VCS history sanitizing tools easy for the developers and reducing VCS scan tool false positives. We also recommend developing new tools for line-level security and ignoring local changes of already-committed files.

Documentation. We find that developers face challenges in securely managing secrets while developing with different technologies due to the absence of proper documentation (Q1, Q6-Q8). For example, Foursquare API documentation [62] suggests developers use a client secret in userless or server-side authentication. However, a developer did not understand the documentation and asked in Stack Exchange whether the secret could be used in the client-side authentication [63]. Developers also seem to query to understand the safest approach when multiple approaches are suggested in the same documentation [64]. For example, ASP.NET Core documentation suggests using local environment variables and secret manager tools to store secrets securely but does not specify which one will be the safest approach in specific use cases [65]. However, we agree that no solution will be perfectly secure, but the documentation should be clear and detailed so that developers understand which use cases are appropriate for each approach. Furthermore, we observe that developers want reference links on how to implement a specific approach suggested in the documentation. For example, Google API provides documentation of the best practices for securely using API keys [66]. However, developers could not figure out how to implement these suggestions as reference links to the specific suggestions are not given [67]. We also observe that documentation does not explicitly mention whether the particular suggestion, such as setting up continuous deployment in Azure Function, is for the development or production environment [68]. As a result, developers may implement a suggestion in the production environment that was intended for use in the development environment [69], thus exposing secrets to the attackers.

Recommendation 2: We recommend that each technology improve the technical documentation for managing secrets by i) clearly explaining the suggested approach's use cases and restrictions; ii) mentioning which approach will be safest for specific use cases when multiple approaches are suggested; iii) providing reference links to implement the suggested approaches; and iv) explicitly mentioning whether the particular approach is for development, production, or both environments.

Client-side applications. Often, developers architect applications with only a client-side implementation and only later realize they must securely embed a secret in the code they distribute. As a result, questions about client-side secret storage (Q20), were the most popular among all topics we studied, as seen in Table V. One solution is for the developer to operate an API for their app that wraps the third-party API and keeps the secret server-side. Instead, novice developers embed third-party API calls in the client because it seems easier, cheaper (no infrastructure costs), and functions as expected. Unfortunately, secrets in the client-side application can not be protected against even a basic adversary with access to a debugger or decompiler. Inspired by popular DRM schemes such as Apple's FairPlay Streaming [70], we posit that privileged system elements, such as virtual machines, runtimes, browsers, or kernels could provide an interface for secure secret management.

Recommendation 3: We recommend that kernels and privileged runtimes develop frameworks to provide secure secret management for client-only applications.

Guidelines. From the identified challenges in Table IV, we observe that developers have a knowledge gap about whether a secret is exploitable or not (Q3), why they should keep secrets out of VCS (Q26), and what to do if they find secrets in the source code (Q27). We also found that some solutions are insecure for managing secrets by analyzing the solutions posed by Stack Exchange users. For example, storing secrets as Base64 encoded in the source code can be exploitable as secrets can be decoded easily (A5). Furthermore, storing secrets in a private repository is not a safe approach (A7) as private repositories are not free from exploitation by attackers or insider threats [52], [71]. Therefore, a guideline to train developers on securely managing secrets can eliminate the knowledge gap, and developers can make correct decisions during development. The National Institute of Standards and Technology (NIST) [72] provides a framework SP 800-218 [73] to mitigate the risk of software vulnerabilities but does not have practices specific to securely managing secrets.

Recommendation 4: We recommend that NIST update the SP 800-218 framework by including practices specific to securely managing secrets to train developers.

V. Ethics

The contents of all the Stack Exchange sites are under Creative Commons (CC BY-SA 3.0) license [74] with the following requirements: "You are free to: *Share* - copy and redistribute the material in any medium or format, *Adapt* - remix, transform, and build upon the material for any purpose, even commercially" [74]. Stack Exchange inspires academics to utilize the data in research articles [75] and requires researchers to give attribution to posts using a direct link [76]. As a result, we include hyperlinks to connect our quotes to the original posts, which are available online [30].

VI. Threats to Validity

In this section, we discuss the limitations of our paper.

Q&A Site Selection: We did not collect questions from other Q&A sites, such as CodeProject [77] and Coderanch [78]. We accounted for this limitation by considering three Q&A sites of Stack Exchange instead of only using Stack Overflow.

Manual Analysis Bias: Caused by multiple interpretations and oversight, the manual analysis may induce bias. For example, the identified question and answer categories are susceptible to bias. We mitigated this bias by cross-checking the obtained question and answer categories and adding question and answer categories that both participants agreed on.

Closed Questions: The nature of inquiries about checked-in secrets in software artifacts may be broad, and Stack

Exchange moderators do not like such questions. As a result, the moderators may decide to close some of the important questions. However, only 52 questions were flagged as closed, accounting for less than 7% of the 779 questions in our dataset. We also observed that the closed questions had a high View Count (as high as 49471) and high Score (as high as 126) [79]. As a result, we claim that the closed questions of our dataset have remained active after being closed, proving the significance of the topics under discussion.

Popularity Metric: We measured the popularity metric of a question by taking the question's View Count and Score values into account. On the other hand, this metric may be biased because it ignores the time span of the views. Therefore, a new question with a low View Count and Score value may be regarded as unpopular. Also, Stack Exchange does not provide the temporal View Count of a question. As a result, a significant percentage of the View Count may accumulate when the question is initially posted or may have recently increased. Unfortunately, we have not yet arrived at a suitable treatment for this threat.

Counting Questions: We counted questions of a category posted by developers over time to find if a particular question category trends. There can be questions in that specific category that have been answered before, but developers are still posting new questions. It implies that the particular category continues to be a problem despite the ongoing effort. We agree that there can be a trend of decreasing questions of a category, but the problem may not be solved till today. However, we are not claiming those categories as of less importance. Instead, we are highlighting the recent ongoing problematic topics to the research community so that researchers can prioritize the challenges and work on resolving them.

Accepted Answer: We termed a question lacking an accepted answer as a *question with unsatisfactory answer*. However, a developer who posted the question may be satisfied with the suggested solution posted by Stack Exchange users. Nevertheless, the developer may forget or not know how to mark the suggested solution as accepted in Stack Exchange. Unfortunately, we have not yet arrived at a suitable treatment for this threat.

VII. Related Work

Prior work has found that root causes for widespread secret leakage were insecure practices, such as embedding hard-coded credentials [80], [81], organizational issues influencing software security vulnerabilities [82], [83], [84], and compromising security for functionality when managing software dependencies [85]. Researchers have looked into instances of such insecure developer practices within open-source projects [4], [8], [9], [86], [87]. Researchers have discovered hard-coded secrets as a prevalent practice, resulting in thousands of repositories on open-source coding platforms, such as GitHub and Openstack, leaking hard-coded secrets [4], [9], [88]. Within IaC scripts, Rahman et al. [10] looked for security smells, which are repeating coding patterns indicating a security flaw. They found 21,201 occurrences of seven security smells within 15,232 IaC scripts, and hard-coded credential is the most occurring smell with 1326 occurrences.

To understand more clearly the challenges that developers face, researchers have performed qualitative research into investigating what questions developers are asking on Stack Overflow (SO) [28], [89], [90], [15], [14] as developers constantly search in SO for guidance on solving a challenge during development. Tahir et al. [14] looked through 4000 posts from three Stack Exchange sites to see what developers were discussing about code smells and anti-patterns. They observed that developers frequently post questions on Stack Exchange to check the presence of smell in their code, effectively using Q&A sites as an informal code smell and anti-pattern detector.

We take motivation from the above studies and concentrate our research efforts on finding difficulties faced by developers for checked-in secrets in software artifacts. We also determine the solutions proposed by other developers to alleviate a specific challenge.

VIII. Conclusion

Software relies heavily on the use of secrets for authentication and authorization, and the exposure of secrets is increasing each day. By analyzing the questions developers ask, we can understand the challenges developers face regarding checked-in secrets. In our empirical study, we studied 779 questions posted on Stack Exchange to investigate the challenges faced by developers and the corresponding solutions posed by others to mitigate the challenges. We identified 27 challenges and 13 solutions. The four most common challenges, in ranked order, are: (i) store/version of secrets during deployment (Q6); (ii) store/version of secrets in source code (Q1); (iii) ignore/hide of secrets in source code (Q2); and (iv) sanitize VCS history (Q10). The three most common solutions, in ranked order, are: (i) move secrets out of source code/version control and use template config file (A1); (ii) secret management in deployment (A2); and (iii) use local environment variables (A3). In addition, we observe that the same solution has been mentioned to mitigate multiple challenges. We also observe an increasing trend in questions lacking accepted answers. Our findings will benefit researchers and tool developers who can investigate how the secret management process can be enhanced to facilitate secure development.

Acknowledgment

This work was supported by National Science Foundation 2055554 grant. The authors would also like to thank the North Carolina State University Realsearch research group for their valuable input on this paper.

References

[1] "The State of Secrets Sprawl 2022," https://blog.gitguardian.com/the-state-of-secrets-sprawl-2022, [Online; accessed March 16, 2022].
[2] "GitHub," https://github.com, [Online; accessed March 3, 2022].
[3] "GitLab," https://gitlab.com, [Online; accessed March 3, 2022].
[4] M. Meli, M. R. McNiece, and B. Reaves, "How bad can it git? characterizing secret leakage in public github repositories." in *NDSS*, 2019.

[5] S. Nichols, "Popular mobile apps leaking AWS keys, exposing user data," https://www.techtarget.com/searchsecurity/news/252500361/Popular-mobile-apps-leaking-AWS-keys-exposing-user-data, 2021, [Online; accessed December 25, 2021].

[6] "CWE: Common Weakness Enumeration," https://cwe.mitre.org/data/definitions/798.html, [Online; accessed December 25, 2021].

[7] "2021 CWE Top 25 Most Dangerous Software Weaknesses," https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html, [Online; accessed February 27, 2022].

[8] V. S. Sinha, D. Saha, P. Dhoolia, R. Padhye, and S. Mani, "Detecting and mitigating secret-key leaks in source code repositories," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 396–400.

[9] M. R. Rahman, A. Rahman, and L. Williams, "Share, but be aware: Security smells in python gists," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 536–540.

[10] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 164–175.

[11] "How to keep secret key information out of Git repository?" https://stackoverflow.com/questions/52293453, [Online; accessed February 27, 2022].

[12] "Stack exchange sites," https://stackexchange.com/sites, [Online; accessed December 23, 2021].

[13] T. Zimmermann, "Card-sorting: From text to themes," in *Perspectives on Data Science for Software Engineering*, T. Menzies, L. Williams, and T. Zimmermann, Eds. Boston: Morgan Kaufmann, 2016, pp. 137–141. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128042069000271

[14] A. Tahir, J. Dietrich, S. Counsell, S. Licorish, and A. Yamashita, "A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites," *Information and Software Technology*, vol. 125, p. 106333, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920300926

[15] A. Rahman, A. Partho, P. Morrison, and L. Williams, "What questions do programmers ask about configuration as code?" in *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 16–22. [Online]. Available: https://doi.org/10.1145/3194760.3194769

[16] "Stack Overflow," https://stackoverflow.com, [Online; accessed January 3, 2022].

[17] "Information Security," https://security.stackexchange.com, [Online; accessed January 3, 2022].

[18] "Software Engineering," https://softwareengineering.stackexchange.com, [Online; accessed January 3, 2022].

[19] "Stack exchange data explorer," https://data.stackexchange.com, [Online; accessed December 23, 2021].

[20] "Stack exchange data dump," https://archive.org/details/stackexchange, [Online; accessed December 23, 2021].

[21] T. P. Johnson, "Snowball sampling: introduction," 2014. [Online]. Available: https://doi.org/10.1002/9781118445112.stat05720

[22] "Where to keep static information securely in Android app?" https://stackoverflow.com/questions/61724202, [Online; accessed June 15, 2022].

[23] "GitHub Repository," https://github.com/setu1421/ICSE-2023-Artifacts, [Online; accessed January 28, 2023].

[24] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, "Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement," *Sociological Methods & Research*, vol. 42, no. 3, pp. 294–320, 2013.

[25] Y. Yao, H. Tong, T. Xie, L. Akoglu, F. Xu, and J. Lu, "Want a good answer? ask a good question first!" 2013.

[26] J. H. Zar, *Spearman Rank Correlation*. John Wiley & Sons, Ltd, 2005.

[27] "Feature Scaling," https://en.wikipedia.org/w/index.php?title=Feature_scaling&oldid=1075231919, [Online; accessed 11-March-2022].

[28] K. Bajaj, K. Pattabiraman, and A. Mesbah, "Mining questions asked by web developers," in *Proceedings of the 11th Working conference on mining software repositories*, 2014, pp. 112–121.

[29] D. R. Cox and A. Stuart, "Some quick sign tests for trend in location and dispersion," *Biometrika*, vol. 42, no. 1/2, pp. 80–95, 1955. [Online]. Available: http://www.jstor.org/stable/2333424

[30] "Example Links to Questions and Developer Quotes," https://figshare.com/s/edebdcb73def3bdb7cfb, [Online; accessed June 24, 2022].

[31] "Heroku," https://www.heroku.com, [Online; accessed March 7, 2022].

[32] "Google App Engine," https://cloud.google.com/appengine, [Online; accessed March 7, 2022].

[33] M. R. Rahman, N. Imtiaz, M.-A. Storey, and L. Williams, "Why secret detection tools are not enough: It's not just about false positives-an industrial case study," *Empirical Software Engineering*, vol. 27, no. 3, pp. 1–29, 2022.

[34] "Git Update Index (Notes)," https://git-scm.com/docs/git-update-index\#_notes, [Online; accessed June 15, 2022].

[35] "Manage Secrets & Protect Sensitive Data," https://www.vaultproject.io, [Online; accessed March 3, 2022].

[36] "Azure Key Vault," https://azure.microsoft.com/en-us/services/key-vault, [Online; accessed March 3, 2022].

[37] "Ignoring a previously committed file," https://www.atlassian.com/git/tutorials/saving-changes/gitignore, [Online; accessed March 12, 2022].

[38] "Smudge and clean your git working directory," https://www.atlassian.com/git/tutorials/saving-changes/gitignore, [Online; accessed March 12, 2022].

[39] "Gitignore Templates," https://github.com/github/gitignore, [Online; accessed February 13, 2022].

[40] Michael DeHaan, "Ansible Vault," https://docs.ansible.com/ansible/latest/cli/ansible-vault.html, [Online; accessed March 2, 2022].

[41] "Chef Vault," https://docs.chef.io/workstation/chef_vault, [Online; accessed March 2, 2022].

[42] "Heroku Config Vars," https://devcenter.heroku.com/articles/config-vars, [Online; accessed March 2, 2022].

[43] Monica, Diogo, "Why you shouldn't use ENV variables for secret data," https://diogomonica.com/2017/03/27/why-you-shouldnt-use-env-variables-for-secret-data, 2017, [Online; accessed February 15, 2022].

[44] "Git Filter Repo," https://github.com/newren/git-filter-repo, [Online; accessed March 29, 2022].

[45] "Git Filter Branch," https://git-scm.com/docs/git-filter-branch, [Online; accessed March 7, 2022].

[46] "BFG Repo Cleaner," https://rtyley.github.io/bfg-repo-cleaner, [Online; accessed March 7, 2022].

[47] "Removing sensitive data from a repository," https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/removing-sensitive-data-from-a-repository, [Online; accessed February 13, 2022].

[48] "git-secret," https://github.com/sobolevn/git-secret, [Online; accessed February 23, 2022].

[49] "git-crypt," https://github.com/AGWA/git-crypt, [Online; accessed February 23, 2022].

[50] "AWS Key Management Service," https://aws.amazon.com/kms, [Online; accessed March 3, 2022].

[51] "Git Submodules," https://git-scm.com/book/en/v2/Git-Tools-Submodules, [Online; accessed February 15, 2022].

[52] Cimpanu, Catalin, "Hacker gains access to a small number of Microsoft's private GitHub repos," https://www.zdnet.com/article/hacker-gains-access-to-a-small-number-of-microsofts-private-github-repos, 2020, [Online; accessed February 15, 2022].

[53] "Git Hooks," https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks, [Online; accessed February 23, 2022].

[54] "Git Flag (Assume-unchanged)," https://git-scm.com/docs/git-update-index\#Documentation/git-update-index.txt---no-assume-unchanged, [Online; accessed February 23, 2022].

[55] "Git Flag (Skip-worktree)," https://git-scm.com/docs/git-update-index\#Documentation/git-update-index.txt---no-skip-worktree, [Online; accessed February 23, 2022].

[56] "Keeping your organization secure," https://docs.github.com/en/enterprise-cloud@latest/organizations/keeping-your-organization-secure, [Online; accessed March 3, 2022].

[57] "TruffleHog," https://github.com/trufflesecurity/truffleHog, [Online; accessed February 23, 2022].

[58] "Gitrob," https://github.com/michenriksen/gitrob, [Online; accessed February 23, 2022].

[59] "The Mapping of Answers to Question Category," https://figshare.com/s/1532991322add36e2eb5, [Online; accessed August 31, 2022].

[60] "Git Filter Branch Safety," https://git-scm.com/docs/git-filter-branch#SAFETY, [Online; accessed January 27, 2023].

[61] "Nine DevSecOps secret scanning tools to keep the bad guys at bay," https://www.cybersecasia.net/tips/nine-devsecops-scanning-tools-to-keep-the-bad-guys-at-bay, [Online; accessed Jan 7, 2023].
[62] "Foursquare API (Search for Venues)," https://developer.foursquare.com/reference/v2-venues-search, [Online; accessed April 15, 2022].
[63] "Foursquare API exposing secret in javascript," https://stackoverflow.com/questions/32559855, [Online; accessed June 15, 2022].
[64] "What is the safest way to store user secrets in a .NET Core application?" https://stackoverflow.com/questions/47316330, [Online; accessed June 15, 2022].
[65] Rick Anderson and Kirk Larkin, "Safe storage of app secrets in development in ASP.NET Core," https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets?tabs=windows&view=aspnetcore-6.0, [Online; accessed June 15, 2022].
[66] "Best practices for securely using API keys," https://support.google.com/googleapi/answer/6310037, [Online; accessed June 14, 2022].
[67] "How do I securely use Google API Keys," https://stackoverflow.com/questions/39625587, [Online; accessed June 14, 2022].
[68] "Continuous deployment for Azure Functions," https://docs.microsoft.com/en-us/azure/azure-functions/functions-continuous-deployment, [Online; accessed June 11, 2022].
[69] "How to configure Connection string in continuous deployment on Azure functions," https://stackoverflow.com/a/46790625/4299527, [Online; accessed June 14, 2022].
[70] "Apple FairPlay Streaming," https://developer.apple.com/streaming/fps/FairPlayStreamingOverview.pdf, [Online; accessed June 29, 2022].
[71] Cimpanu, Catalin, "Nissan source code leaked online after Git repo misconfiguration," https://www.zdnet.com/article/nissan-source-code-leaked-online-after-git-repo-misconfiguration, 2021, [Online; accessed April 12, 2022].
[72] "The National Institute of Standards and Technology (NIST)," https://www.nist.gov/, [Online; accessed June 14, 2022].
[73] K. S. Murugiah Souppaya and D. Dodson, "Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities," https://csrc.nist.gov/publications/detail/sp/800-218/final, [Online; accessed June 10, 2022].
[74] "Stack Overflow Creative Commons Data Dump," https://stackoverflow.blog/2009/06/04/stack-overflow-creative-commons-data-dump, [Online; accessed June 11, 2022].
[75] "Academic Papers Using Stack Overflow Data," https://stackoverflow.blog/2010/05/31/academic-papers-using-stack-overflow-data, [Online; accessed June 11, 2022].
[76] "Attribution Required," https://stackoverflow.blog/2009/06/25/attribution-required, [Online; accessed June 11, 2022].
[77] "CodeProject," https://www.codeproject.com, [Online; accessed March 16, 2022].
[78] "Coderanch," https://coderanch.com, [Online; accessed March 16, 2022].
[79] "Is it completely safe to publish an ssh public key?" https://security.stackexchange.com/questions/150540, [Online; accessed June 15, 2022].
[80] "Medical Data Leaked on GitHub Due to Developer Errors," https://threatpost.com/medical-data-leaked-on-github-due-to-developer-errors/158653, [Online; accessed Jan 15, 2022].
[81] "No need to hack when it's leaking," https://www.databreaches.net/wp-content/uploads/No-need-to-hack-when-its-leaking.pdf, [Online; accessed Jan 15, 2022].
[82] H. Assal and S. Chiasson, "'think secure from the beginning' a survey with software developers," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–13.
[83] J. Xie, H. R. Lipford, and B. Chu, "Why do programmers make security errors?" in *2011 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. IEEE, 2011, pp. 161–164.
[84] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: Why do java developers struggle with cryptography apis?" in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 935–946.
[85] I. Pashchenko, D.-L. Vu, and F. Massacci, "A qualitative study of dependency management and its security implications," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1513–1531.
[86] A. Saha, T. Denning, V. Srikumar, and S. K. Kasera, "Secrets in source code: Reducing false positives using machine learning," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2020, pp. 168–175.
[87] Z. Y. Ding, B. Khakshoor, J. Paglierani, and M. Rajpal, "Sniffing for codebase secret leaks with known production secrets in industry," *arXiv preprint arXiv:2008.05997*, 2020.
[88] A. Rahman and L. Williams, "Different kind of smells: Security smells in infrastructure as code scripts," *IEEE Security & Privacy*, vol. 19, no. 3, pp. 33–41, 2021.
[89] A. Rahman, E. Farhana, and N. Imtiaz, "Snakes in paradise?: Insecure python-related coding practices in stack overflow," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 200–204.
[90] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.