Why secret detection tools are not enough: It's not just about false positives - An industrial case study

Md Rayhanur Rahman[1] · Nasif Imtiaz[1] · Margaret-Anne Storey[2] · Laurie Williams[1]

Abstract

Checked-in secrets in version-controlled software projects pose security risks to software and services. Secret detection tools can identify the presence of secrets in the code, commit changesets, and project version control history. As these tools can generate false positives, developers are provided with mechanisms to bypass the warnings generated from these tools. Providing this override mechanism can result in developers sometimes exposing secrets in software repositories. *The goal of this article is to aid software security practitioners in understanding why' secrets are checked into repositories, despite being warned by tools, through an industrial case study of analysis of usage data of a secret detection tool and a survey of developers who bypassed the tool alert.* In this case study, we analyzed the usage data of a checked-in secret detection tool used widely by a software company and we surveyed developers who bypassed the warnings generated by the tool. From the case study, we found that, despite developers classified 50% of the warning as false positive, developers also bypassed the warning due to time constraints, working with non-shipping projects, technical challenges of eliminating secrets completely from the version control history, technical debts, and perceptions that check-ins are low risk. We advocate practitioners and researchers to investigate the findings of our study further to improve secret detection tools and related development practices. We also advocate that organizations should insert secondary checks, as is done by the company we studied, to capture occasions where developers incorrectly bypass secret detection tools.

Keywords Secret detection tool · Hardcoded secrets · Secrets in repositories · Credentials in repositories

---

This article belongs to the Topical Collection: *Software Engineering in Practice*

✉ Md Rayhanur Rahman
    mrahman@ncsu.edu

Extended author information available on the last page of the article.

1 Introduction

Software developers use secrets (also known as credentials) for performing authentication among software artifacts. Secrets refer to information, such as user identifiers, passwords, and API tokens, SSH, and encryption keys used for authentication to remote services. Although security best practices suggest the use of secret stores, such as Amazon AWS KMS (Amazon 2020), the use of these practices may be difficult for new or inexperienced developers. Alternatively, developers may insert the secret in the text which can result in secrets being stored in a version control system (VCS), either inadvertently or expressly for ease of sharing and distribution. The presence of plaintext secrets in repositories leaves the software vulnerable to security breaches. The presence of plaintext secrets in software artifacts is also reported as a common software security weakness (CWE-798: Use of Hard-coded Credentials) (MITRE 2020).

Secret key leakage in VCS repositories is a persistent problem that needs to be addressed (Meli et al. 2019). For example, in 2020, approximately 200K patient records were leaked due to a development mistake persisting hard-coded secrets of software artifacts in public repositories (Montalbano 2020). In 2019, Starbucks API keys were found in a public GitHub[1] repository which could have been used to gain unauthorized access to Starbucks' internal systems (Ilascu 2020; Kumar 2020). Meli et al. (2019) found approximately 200K API keys and tokens checked into public GitHub repositories, which also substantiate the case of checked-in secrets in repositories.

Software engineering researchers (e.g., Meli et al. 2019; Rahman et al. 2019; Rahman et al. 2020; Rahman et al. 2019) and tool vendors have developed automated tools to check for secrets in repositories, which can be used to alert developers prior to checking in a secret. Automated tools, such as TruffleHog[2] and GitLeaks[3], can detect the presence of plaintext secrets in repository branches and version history. GitHub has also added features[4] to help prevent fraudulent use of accidentally committed secrets.

Despite the availability of these automated tools, developers continue to check secrets in repositories. Overall, these phenomena indicate that the availability of secret detection tools may not be enough to prevent developers from checking in secrets. There could be other contributing factors. For example, not all developers use these tools consistently throughout their development activities. Moreover, secret detection tools can also generate a high number of false positives (Saha et al. 2020) which may cause developers to lose confidence in the use of the tool. Research efforts are underway to lower false positive rates (e.g., Sinha et al. 2015; Saha et al. 2020; Ding et al. 2020). However, our experience indicates that true positive warnings reported by tools might also be bypassed and secrets might be persisted in the VCS. Hence, in this case study, our focus is on understanding the actions, behavior and decisions of developers responding to secret detection.

The goal of this article is to aid software security practitioners in understanding why secrets are checked into repositories despite being warned by tools through an industrial case study of analysis of usage data of a secret detection tool and a survey of developers who bypassed the tool alert. In this study, we ask these following research questions (RQs):

– **RQ1:** How often do developers bypass the warning generated by secret detection tools?

---

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

[2]https://trufflesecurity.com/trufflehog

[3]https://github.com/zricethezav/gitleaks

[4]https://docs.github.com/en/github/authenticating-to-github/removing-sensitive-data-from-a-repository

–   **RQ2:** What is the concentration of checked-in secrets in source code artifacts?
–   **RQ3:** What is the distribution of checked-in secret activity among the developers inside the organization?
–   **RQ4:** Why do developers check in potential secrets despite tool warnings?

Overall, answers to these RQs could help practitioners identify factors that are responsible for persisting secrets in VCS despite tool warnings. These factors could help practitioners improve secret detection tools and related development practices. To answer these RQs, we study an internal secret detection tool used broadly and consistently within a software company named XTech[5](see Section 3.1). Throughout this article, we refer to this tool as the XTech Secret Detection Tool (XSDT) (See Section 3.2). The tool checks for the existence of plaintext secrets in artifacts. The tool is applied as part of a *git*[6] commit and can block developers from pushing changes to a repository.[7] When blocked, developers have the choice of extracting the offending file from the push and committing the other files into git, or continuing to check-in by explicitly bypassing the XSDT warning.[8] The detected secrets or the offending file(s) can be subsequently deleted, including its git history, or the file can remain in git. Analysis of plaintext string literals to identify secrets is complex and can generate false positives. Here, false positive refers to the incident where the XSDT tool identifies a string literal as a potential secret, however, that string literal has nothing to do with authentication/authorization. Due to these false positives, XSDT offers developers a bypass mechanism. However, XTech has a secondary process to capture the occurrence of true secrets that developers have checked in to git. The focus of the analysis in this article is the developer activity on the first security check and identifying the potential factors that may influence the developers to bypass the tool warning.

We analyze data from XSDT usage and its associated VCS over the period from June 1, 2019 to May 31, 2020. We then conduct a survey on a set of developers who have recently (at the time of the survey) bypassed XSDT warnings and checked in potential secrets in repositories.

We list our contributions as follows:

1.   A quantitative analysis of

   (a)   developer actions regarding checked-in secrets (Section 5.1),
   (b)   source code artifacts associated with developer actions of checking-in secrets (Section 5.2),

---

[5]We anonymize the actual name of the company in this study. XTech is a surrogate name for the actual company

[6]https://git-scm.com/

[7]XTech addresses the problem of exposed plaintext secrets in engineering artifacts in multiple ways, starting with the required company-wide training to build security awareness and communicate relevant XTech policies. Developers are advised to avoid checking in any actual secret to an engineering artifact, **no matter what the perceived risk is**. In the event that a secret is exposed due to developer error or another reason, teams must invalidate and replace the secret within a time window set by the project's security experts. To help prevent exposure and drive remediation in the case of error, security scans are integrated in multiple phases of the engineering process, including code authoring, peer code review, and automated build pipelines. XTech additionally maintains a centralized system that scans all developer changes on a regular basis. All these systems may block a process (such as a check-in or build) or schedule a remediation effort as a work item or servicing ticket

[8]All bypassed secrets will continue to be tracked and reported by complementary downstream XSDT scan systems that are not part of this study

2. An analysis on distribution of developers inside the organization who checked in secrets (Section 5.3),
3. an evaluation of developer rationales for bypassing secret detection warnings (Section 6),
4. recommendations on how to reduce the occurrence of checked-in secrets (Section 8).
5. dataset used for conducting the study[9].

The rest of the article is organized as follows. In Section 2, we discuss the related studies. In Sections 3 and 4, we discuss the XSDT tool and our methodology. We report our findings in Sections 5 and 6. Finally, in Sections 7, 8 and 9, we include additional discussions, recommendations, limitations, and threats to the validity of our study, followed by the conclusion in Section 10.

2 Related Work

Software engineering researchers have focused on investigating the presence of secrets in VCS repositories. Meli et al. (2019) conducted the first large-scale, longitudinal study of checked-in secrets over software project repositories (13% of all public Github repositories). This study demonstrated that the problem of checked-in secrets is vast: *thousands* of new, unique keys are being leaked each day.

Researchers have also led to improvements in the performance of secret detection tools (Sinha et al. 2015), including false positive reduction and introducing detectors for additional languages. Viennot et al. (2014) published a set of regular expressions for detecting API keys of seven service providers, including Amazon AWS, Facebook, LinkedIn and Twitter. Saha et al. (2020) created a generalized framework to detect secrets with a low false positive rate through a combination of an extensive regular expression list and machine learning models. Ding et al. (2020) use known production secrets as a source of ground truth for detecting secret leaks in code and differential code revisions with a reduced false positive rate.

Rahman et al. (2019, 2020) proposed static analysis tools (SLIC and SLAC) to detect security smells, including hardcoded secrets and empty passwords, in infrastructure-as-code (IaC) scripts such as Puppet, Ansible, and Chef. Rahman et al. (2019) conducted a study of 5,822 code snippets shared on GitHub Gist and found 714 empty or hardcoded passwords, demonstrating that developers may inadvertently propagate secrets while intentionally sharing code. Bunyakiati and Sammapun (2019) examined approaches for managing and handling secrets when developing mobile applications, including the development of pedagogy, tool support and design patterns for secret management and handling. Meanwhile, researchers also focused on developers' perceptions of static analysis tool use. For example, Johnson et al. discussed the challenges that cause miscommunication and confusion among developers while using program analysis tools (Johnson et al. 2016).

Overall, these studies discuss the prevalence of checked-in secrets, secret detection tools, and the associated challenges developers face while using the tools. However, there has been little research to understand the behavior of developers and the decisions they make related to checking in secrets. This study addresses this gap and provides insights on developer behaviour when using secret detection tools.

---

[9]the dataset are included as supplementary information files

3 XTech and XTech Secret Detection Tool (XSDT)

3.1 XTech

We conduct our study on a US-based software company having the anonymized name of XTech. We present information about XTech in this subsection without providing specific details to protect the anonymity of the company. Software and services development is the primary business activity of this company for more than ten years. Most of their products are developed in-house. The number of full-time software developers/engineers is more than one thousand, and they are from various countries and cultures working both in in-house and remote capacities.

3.2 XTech Secret Detection Tool (XSDT)

Once a secret in an engineering artifact is pushed to a VCS, the secret is exposed to all engineers having access to the repository. A complete fix requires purging the secret entirely from VCS history, invalidating and replacing the secret. To reduce this remediation cost, the XTech Secret Detection Tool (XSDT) analyzes all check-ins (also known as a *push* in *git* terminology) submitted by developers to the VCS. XSDT is integrated with the internal VCS services that store the majority of XTech project code. XSDT automatically scans each check-in and blocks persisting them to the VCS if one or more potential secrets are detected. Developers are notified when their submission is blocked using the same mechanisms that report errors provided by the VCS (such as error messages shown in *git bash*). In this study, we refer to the notification generated by XSDT as a *warning*. Developers can choose one of the following *actions* when they receive an XSDT warning:

– **Remove:** remove the secret from the code artifact and commit history.
– **Abandon:** remove the offending file from the check in, or abandon working on the branch where the potential secret is embedded[10].
– **One-time bypass:** include a special *string* in the commit message after which XSDT will allow the developer to bypass its scan and persist the check-in to the repository[11].
– **Permanent bypass:** suppress the XSDT warning by putting a special *string* in a specified metadata file or in the files where the potential secrets are embedded. When subsequently scanning the artifact, XSDT will detect the special strings and will not fire warnings for the suppressed potential secrets. Thus, developers can permanently bypass XSDT warnings for a specific occurrence of a pattern in the code containing potential secrets. Because permanent bypasses are rendered in the source or as part of a separate metadata file under source control, they serve as documentation for the security review process.

The potential developer actions can be categorized into two action types:

– **prevention** of the potential secrets detected by XSDT from being persisted to the VCS
– **bypass** of the XSDT warning

---

[10]XSDT considers a branch abandoned if there is no further activity on the warning within the next three days
[11]All bypassed secrets will continue to be tracked and reported by complementary downstream XSDT scans that are not a part of this study.

Removing potential secrets or **abandoning** the check-in process fall into the **prevention** category. **One-time bypass** or **permanent bypass** fall into the **bypass** category. The final outcome of a XSDT warning, thus, is one of the following:

– **Blocked.** No potential secret is checked in to the repository because the developer removes the blocked code pattern (code patterns containing potential secrets) or abandons the check-in. The outcome is the elimination of risk, if any, associated with the warning

– **Exposed.** A potential secret is checked in to the repository if the developer chooses a one-time bypass or permanent bypass. Risk is increased for these actions (in the case of a true positive) because the secret is exposed to anyone with access to the repository. The cost to remediate is also significantly increased, as the exposed secret (in the case of a true positive) must be invalidated and replaced, rather than simply removed from the commit history.

XSDT can be deployed to comprehensively scan all artifacts in a code base. When analyzing check-ins on pushes, however, XSDT reporting is intended to be constrained to potential secrets that are newly introduced by a developer (to prevent the corresponding increase of risk due to any new exposure). XSDT should not, by design, report on the presence of potential secrets that already existed in the VCS. These secrets, when true positives, are already compromised and should be invalidated and replaced. XTech implements other scanning systems as a supplemental security activity that reports XSDT warnings for already checked-in code. This secondary process captures secrets that developers bypass in the original check. This study focuses solely on the initial XSDT scanning of incremental changes to code.

4 Methodology

The research methodology of our study is embedded in a close collaboration among researchers and organizational members to gain insight on the problem of checked-in secrets, how the secret detection tools can help mitigate the problem, and what actionable solutions can be derived to motivate the developers to stop checking in secrets. The first and second authors participated in this collaboration along with two organizational members.

The usage data of the XSDT and associated VCSs are collected and stored as snapshots in internal databases. This usage data can be a source for insights on how developers use the tool and factors that contribute to the checked-in secret problem. As all developers are XTech employees and the XSDT tool is consistently applied to nearly all check-ins, their feedback on these warnings are solicited through a survey which provides a diverse and valuable source of information on how to potentially improve the tool and to mitigate security problems. We discuss our methodology regarding the usage data analysis and survey in Sections 4.1 and 4.2 respectively.

4.1 Usage Data Analysis

The data we used in our study consist of the usage data of XSDT and the VCSs with which the tool is integrated. This data is captured from the XSDT and VCS logs. XSDT produces telemetry on every push where the telemetry includes these following information: (i) file(s) url, (ii) repository, (iii) organization, (iv) commit identifier, (v) branch, (vi) user identifier, (vii) session identifier, (viii) alert identifier, (ix) timestamps, (x) developer action associated

with the alert.. Telemetry is analyzed to create a view of a developer session (which may consist of multiple pushes to finalize a contribution to a branch and/or resolve a detection). The data is stored in internal databases which are updated approximately every 10 minutes. Metadata for repositories, branches, and pushes are also ingested from internal VCS APIs, processed, and eventually stored in databases as well. We queried the following database tables: *warning*, *commit*, *push*, *repository*, and *authors*. Information related to XSDT warnings, developer action, timestamps, files, commits, versions, projects, developers, teams, and organization are queried from this data to answer our research questions (RQ1 - RQ3), as reported in Section 5.

4.2 Developer Survey

To answer RQ4, we conducted a survey with developers who recently bypassed a XSDT warning. We designed the survey in an iterative manner, seeking feedback on early drafts from both the XSDT product team and its parent team about which questions should be included and how they should be worded. The survey was also refined based on five informal interviews conducted with developers who have used the XSDT tool. In these interviews, we inquired about their perceptions of the effectiveness of the XSDT tool for identifying checked-in secrets, and how they deal with the secrets identified by the tool during development.

We conducted two pilot surveys to improve and refine the survey. By coding open-ended answers in the pilot surveys, we identified closed-answer responses for the survey questions to facilitate faster survey completion and to quantify the survey results. During the pilot surveys, we also learned that developers occasionally felt nervous responding to the survey and worried they may be blamed for introducing security vulnerabilities in their project code. Hence, we made the final survey anonymous.

We sent the final survey to full-time developers who had bypassed warnings in the last seven days. We only considered developers bypassing in last seven days because our experience with the pilot surveys indicated that if we ask developers about bypass activity which occurred many days ago, such as two weeks ago, the possibility of getting noisy response from the developers increases. For example, the developer may forget the exact reason of bypassing the secret. Hence, we chose seven days to limit the effect of recall bias. The developers who participated in the final survey did not participate in any other prior pilot surveys.

We sent each recipient a URL pointing to the potential checked in secret so that the developer could revisit the details of the warning. The expected time to complete the survey was less than five minutes. Although in the company, there were other employee types such as interns and contractors, we only considered full time employees to maintain consistency among the surveyed developers.

In Table 1, we list the questions and closed-answer response options for the survey, where we refer to the questions as Sx and the answers as Ax.y. The objective of the survey was to understand if the developers felt the bypass was related to a deficiency in the tool (i.e., a *tool-related* factor) or for development practices (i.e., a *development-related* factor. The first question (S1) asks the developers if they believe the bypassed XSDT warning is false positive, and if not, whether they have already or will in the future remove the identified secret. If the developers respond that the warning is false positive (A1.1), the survey then asks them to provide further explanations about the false positive warning in S2. If developers respond that they will not go back and remove the secret later (A1.2), the survey prompts them to share the reasons behind their decision (S3). If the developers state that they will

Table 1 Survey questions and closed-answer responses

Q. Id	Question	A. Id	Answer	Precond.
S1*	Please help us understand whether the secret scanning result that you recently bypassed was accurate (a true positive) or a false positive.	A1.1	It was not an actual credential[†].	None
		A1.2	It was an actual credential and we will not go back to remove it.	
		A1.3	It was an actual credential and we have already removed it.	
		A1.4	It was an actual credential that we intend to remove.	
		A1.5	I was blocked for something that already existed in the code (which I did not introduce).	
		A1.6	*Open-ended answers*	
§2	Please tell us more about the false positives or other issues.	A2.1	*Open-ended answers*	A1.1
S3*	Please help us understand why the blocked secret is not worth fixing. Check all that apply.	A3.1	It is not a production credential (for example, it is a credential for a non-shipping prototype).	A1.2
		A3.2	The credential doesn't protect any data, service, etc., with significant security value.	
		A3.3	The credential has additional protections (e.g. it is encrypted in the file, is password-protected, etc.).	
		A3.4	It expires soon or has already been invalidated.	
		A3.5	*Open-ended answers*	

Table 1 (continued)

Q. Id	Question	A. Id	Answer	Precond.
S4*	Please help us understand the considerations that prevented you from removing the secret immediately. Check all that apply.	A4.1	Removing the credential now would break services. etc., that belong to other teams.	A1.4
		A4.2	We were not sure what secret management/other solution we can use to resolve the problem.	
		A4.3	We understand how to remove the credential but doing so now would put our deadlines at risk.	
		A4.4	We have other exposed credentials in our project and the costs to remove them all immediately are too high.	
		A4.5	I attempted to remove the credential but it was too difficult to do in the time that I had.	
		A4.6	Open-ended answer	
S5	If you have any comments on the problem of detecting and removing exposed credentials or suggestions for improving this tool, please let us know.	A5.1	Open-ended answer	None

Questions marked with an asterisk (*) are mandatory

† in this survey, credential is a synonym for secret

remove the secret later (A1.4), then the survey probes for the considerations that prevented them from removing the secret immediately (S4). In S3 and S4, a respondent can provide multiple answers. We did not probe respondents further when they chose A1.3 and A1.5. Finally, S5 asks for optional additional feedback on the tool. We consider A1.1 as well as A1.5 to represent tool-related issues, and A1.2-A1.4 to relate to development-related issues. We report our findings on the survey in Section 6.

5 Findings from XSDT Usage Data Analysis

In this section, we report our findings on RQ1 - RQ3. These findings are derived from XSDT usage data analysis. Overall, the usage data from 06.01.2019 to 05.31.2020 contains information on 43M commits, 18M pushes, 98M scanned files, and 75K repositories. Throughout this entire section, we report the findings on potential secrets identified by XSDT as our analysis did not include a confirmation of true positives in the data set. We will discuss the false positive rate as attested by developer feedback in Section 6.1.

5.1 RQ1: How Often do Developers Bypass The Warning Generated by Secret Detection Tools?

We report developer actions responding to XSDT warnings that a check-in to a repository contains one or more potential secrets. From Table 2, we observe that XSDT warnings block potential secrets or are bypassed at similar rates. Developers removed the potential secrets 26.2% of the time and abandoned the check-in 22.4% of the time, respectively. As a result, 48.6% of the time, XSDT was successful[12] in *blocking* the potential secret from being stored on the server and security risk was reduced. On the other hand, developers used a one-time bypass 44.2% of the time and a permanent bypass 7.2% of the time. Overall, in 51.4% of the cases, the tool believes potential secrets were *exposed* in the repositories[13].

Our observation regarding the 44.2% bypass rate leads us to further investigate why developers are inclined to bypass XSDT warnings. Hence, we calculate the number of subsequent check-in attempts (or pushes) made by developers until the XSDT tool generates no subsequent warning on the corresponding commit changeset. In Fig. 1, we report the average attempt counts for each of the action types. We also report the median, 25th, 75th and 95th percentile in the table. We observe that the average count of pushes when removing secrets from the source code is 6.6, while abandonment occurs after an average of 2.5 pushes. The average attempt count of one-time bypass and permanent bypass are 2.6 and 0.6. The average attempt counts of the actions suggests that developers needed the most attempts to remove the secret, on average. The average and median of permanent bypass suggest that a subset of developers use permanent bypass proactively when they introduce potential secrets for the first time.[14]

---

[12]The success of preventing secrets from getting into VCS is indicated by the warning generated (detecting a string literal as a secret).

[13]If these are actual secrets, there will be additional engineering costs required to invalidate and replace the secrets.

[14]Proactive bypasses may indicate circumstances when a developer is running XSDT against their local changes in advance of pushing to the VCS, and has already therefore reviewed and produced permanent bypass data for false positives.

Table 2 Actions taken by developers when prompted by XSDT warning

Action Type	Action	Count	Percentage	Total	Outcome
Prevention	Remove	37K	26.2	48.6%	Blocked
	Abandon	32K	22.4		
Bypass	One Time Bypass	63K	44.2	51.4%	Exposed
	Permanent Bypass	10K	7.2		

Compared to the 25th percentiles and 75th percentiles, the attempt counts at 95th percentiles are much higher for the remove action compared to other actions, approximately two times higher than the count of one-time bypass, and five times higher than that of permanent bypass. In Fig. 2, we also report the box-plot of the attempt counts where we observe that there are long tails denoting alerts associated with very higher attempt counts (eg. 400). Such higher number of attempts are associated with alerts where a commit changeset detected to have secrets came from a mirrored repository Overall, we observe that developers make several attempts to check-in after first getting blocked, and then appear to give up and abandon the branch or bypass the warning. This seems to indicate that developers may have challenges in removing or fixing the issue.

5.2 RQ2: What is the Concentration of Checked-in Secrets in Source Code Artifacts?

We sort the files scanned by XSDT into five categories based on file extensions to understand if developer behavior resolving XSDT warnings varies by file type. Configuration files contain package deployment and run-time configuration information. Program files contain the source code. Script files contain instructions written in scripting languages or shell programs. Key files comprise keys and certificates. Document files contain text, web, and word
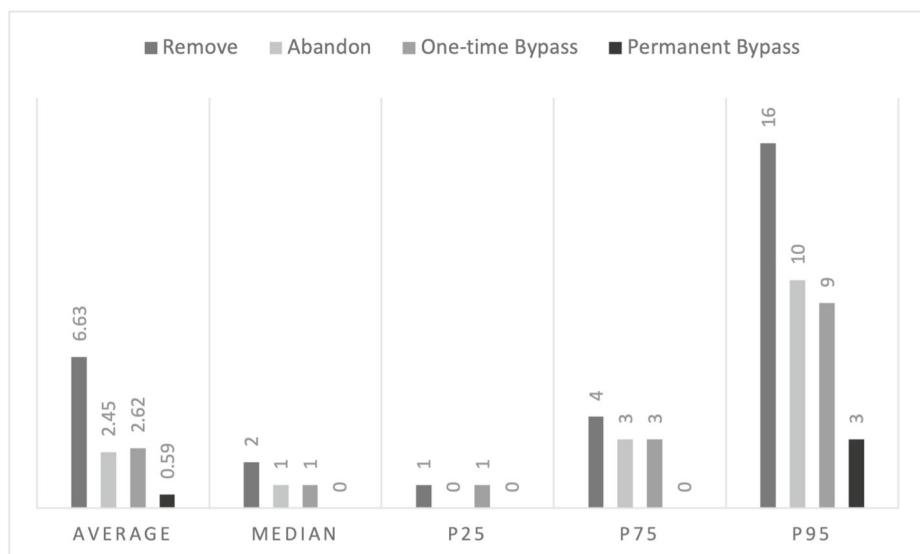

Fig. 1 Count of check-in attempts by the developers between the first XSDT warning and final action

Fig. 2 Boxplot of the count of check-in attempts by the developers between the first XSDT warning and final action
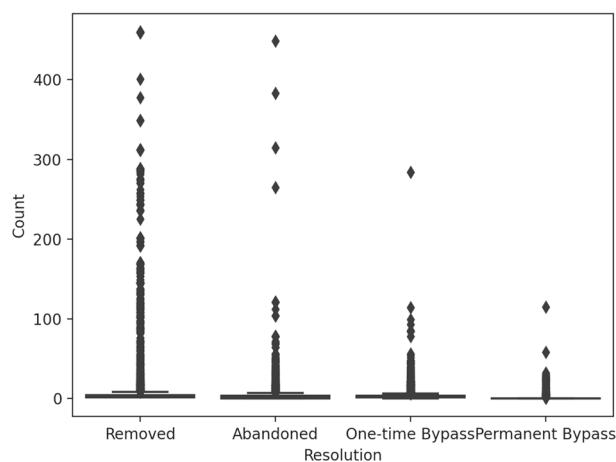
processing content. However, repositories may contain other file types, such as *.jpg* or *.ico*, that are not scanned by XSDT as those file types are not expected to contain secrets. In our study, non-scanned file types comprised 27% of the contents across all repositories.

In Table 3, we report the analysis on artifact types containing potential secrets. In the table,

– α is the percentage of the quantity of files of a certain type with XSDT warnings relative to the quantity of all files containing XSDT warnings,
– β is the percentage of the quantity of files of a certain type relative to the quantity of all the files in the repository scanned by XSDT,
– γ is the relative ratio of issues among the total scanned files which is equivalent to α/β, and
– Δ is the difference between the percentages of blocked and exposed potential secrets in each artifact type. For example, if the bypass rate of an artifact type is b, then the fix rate will be $f = 1 - b$. The value of Δ is $b - f$. The Δ value is the measure that denotes whether a specific artifact type is associated with more exposure of potential secrets than prevention. A negative value of Δ indicates that developers are preventing more exposure in the corresponding artifact type than permitting it. A positive value of Δ indicates that developers are bypassing more XSDT warnings on the artifact type than removing the potential secrets.

From Table 3, we observe that 62.1% of the potential secrets containing files are configuration files, even though configuration files account for 14.6% of the total files scanned.

Table 3 Analysis on Program Artifacts found having secrets

Artifact types	α	β	$\gamma = \alpha/\beta$	Δ
Configuration Files	62.1	14.6	4.3	4.2
Program Files	21.4	34.4	0.6	0.8
Script Files	9.8	9.4	1.0	-2.6
Key Files	3.6	0.1	36.0	15.4
Document Files	3.0	41.5	0.1	-9.2

Configuration files are often used to manage specific assets (e.g., data, service) and often contain multiple properties relevant to asset security. The γ, having a value of 4.3, indicates that configuration files are overrepresented (by a factor of 4.3) in the set of potential secret-containing files based upon their prevalence in the full set of files. Key files are the most overrepresented ($\gamma = 36.0$), accounting for 3.6% of the potential secret-containing files even though key files are only 0.1% of the population. However, key files are different from other artifact types as these files are themselves secret (and generally should not be checked in to a repository at all). Hence, they are low in number among all scanned files but overrepresented in XSDT warnings.

The Δ value of the configuration and program files suggests that developers can be more prone to bypassing and therefore exposing potential secrets to VCSs for these file types. These observations may reflect tool accuracy issues when scanning these file types or indicate there is a particular time and technical complexity specific to removing secrets, which motivates developers to bypass warnings. Overall, there are workarounds to avoid using secrets in the configuration and program files, such as using environment variables or secret managers. Still, program files ($\alpha=21.4$) have the second greatest concentration of potential secrets. Developers might be prone to bypass if the coding patterns in program files do not already include the use of a secret manager. Configuration files ($\alpha=62.1$) are most likely to expose potential secrets in repositories given the fact that configuration files typically contain multiple secrets relevant to their associated services or software.

Developers might also bypass warnings due to the increased costs to remove all secrets in the file when XSDT detects a newly introduced potential secret. The Δ value for key files is the highest ($\Delta = 15.4$) compared to other types. Developers may password-protect key files, motivating the check-in of a key file even after detection by XSDT. Development teams may also generate self-signed certificates which are used strictly for testing and not a security purpose. Negative Δ values for script and document files suggest that developers eliminate the potential secrets more often than exposing them in VCSs.

5.3 RQ3: What is the Distribution of Checked-in Secret Activity Among the Developers Inside the Organization?

In Fig. 3, we report the Pareto property for the count of potential secrets exposed due to bypass actions of unique developers who were blocked by at least one XSDT warning over the study period. The Pareto property states that 20% of the causes are responsible for roughly 80% of the effects (Box and Daniel Meyer 1986). In the figure, the x-axis shows the percentage of unique developers, and the y-axis shows the percentage of total exposed secrets from the accounts. Observing the Pareto property, we identify that 85% of the potential secret exposures originate from 20% of the developers, which indicate that some developers may work more with aspects of the system that relate to secret-containing files, have less security awareness, work on a team unusually subject to time pressures, are maintaining consistency with existing (potentially insecure) patterns in code, work on projects that disproportionally trigger tool false positives, or other factors. However, these factors are speculated from organizational experience in XTech and are needed to be tested which is out of scope for this study.

5.4 Summary of Findings for RQ1 - RQ3

–   51.4% of the time, potential secrets reported by XSDT were eventually exposed on the
    VCS, the rest of those were blocked. However, the survey discussed in Section 6.1, will
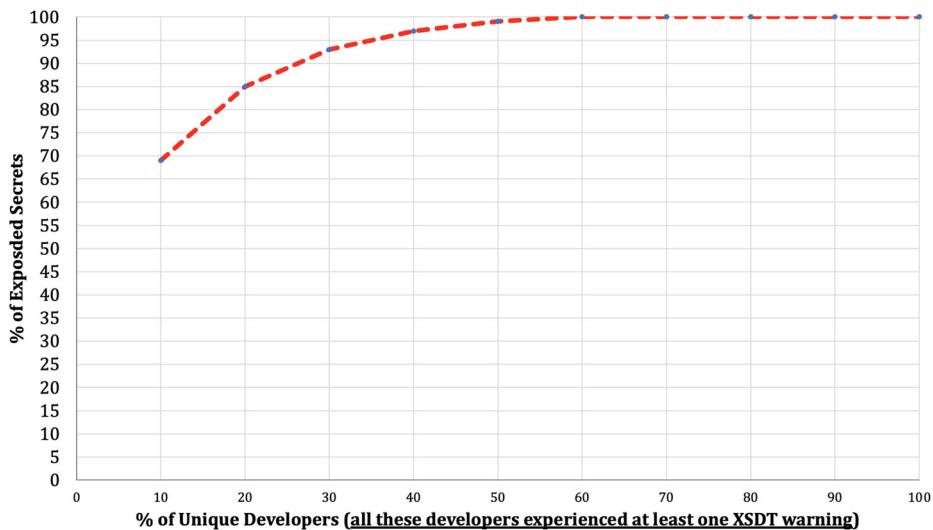
Fig. 3 Pareto property for secret exposure from **unique developers** who triggered at least one XSDT warning

demonstrate that at least 50% of the time these exposures are false positives from the developers' point of view.

– Removing the secrets required multiple push attempts from the developers, indicating that the developer might have difficulties with removing the secrets from the artifacts.

– Developers are most likely to attempt to check in potential secrets in key and configuration files.

– Numerically, the greatest number of potential secrets are exposed in configuration files and program files.

– The majority (85%) of bypasses are contributed by 20% of the developers who faced at least one XSDT warning, indicating that there might be a set of developers who are subject to a high amount of XSDT warnings due to various untested but potential factors (such as technical debt, backward compatibility, security awareness).

6 RQ4: Why do Developers Check in Potential Secrets Despite Tool Warnings?

In this section, we report the results of our survey which asked why developers checked in a potential secret despite receiving a warning from XSDT. We sent the survey to 451 full-time developer employees at XTech and received responses from 113 developers (a response rate of 25.1%). The average completion time for the survey was 4 minutes.

6.1 Developer Perceptions on True and False Positive Warnings

The first question of the survey was *Please help us understand whether the secret scanning result that you recently bypassed was accurate (a true positive) or a false positive*. This

question was mandatory. We report the closed-answer response counts in Table 4 and list our observations below:

Tool-related, false positive (A1.1, 50 respondents, 44%) The detected secret is not an actual secret, and hence the developer bypassed the warning and considered the warning a false positive. For example, one developer states, "It was a test input to verify end to end flow in test pass". In Question S2, developers provide more information on false positives, as discussed later in this section.

Development-related, true positive, has been or will be removed, (A1.3, 16 respondents, 14%), and A1.4, 9 respondents (8%) The warning is accurate and although the developers bypassed the warning, they had already removed the secret at the time of the survey or intend to remove the credential in the future. One developer expressed, "I deleted the entire branch, created a new one".

Development-related, true positive, will not be removed (A1.2, 16 respondents, 14%) Developers bypassed a warning despite acknowledging the warning was accurate and stating they will not remove the secret in the future. For example, one developer wrote, "It was an actual credential, but only for unit test purposes, so we will not remove it". In Sections 6.3 and 6.4, we will discuss more survey results related to bypassing activity.

Tool-related, legacy secret (A1.5, 7 respondents, 6%) The detected potential secrets were already exposed in the repositories, and the developers did not introduce them. One person explained, "I rearranged the code to sort config keys in alphabetical order. I didn't introduce any new secret."

Finally, 13% of the survey respondents chose option A1.6 and provided open-ended answers. We summarize their input, classified as either tool related (TR) (73%) or development related (DR) (27%), in Table 5.

In summary, we observe 50% false positive warnings being generated from the tool (A1.1, and A1.5). We also learned that XSDT warnings, which are bypassed, can be associated with tool-specific factors (such as emitting false positives) or development-specific factors (such as perceiving true positives as low risk or being subject to unusual time pressure). Secret detection tools are known to generate false positive warnings (Saha et al. 2020). However, from Table 4, we observe that developers can bypass a warning which is not false positive and could be related to development-related factors. These development factors will be discussed in Sections 6.3 and 6.4.

Table 4 Response counts from the closed-answer options for S1

Id	Answer	Count(%)
A1.1	It was not an actual credential	50 (44%)
A1.2	It was an actual credential and we will not go back to remove it	16 (14%)
A1.3	It was an actual credential and we have already removed it	16 (14%)
A1.4	It was an actual credential that we intend to remove	9 (8%)
A1.5	I was blocked for something that already existed in the code (and which I did not introduce)	7 (6%)
A1.6	*Open-ended Answer*	15 (13%)

Table 5 Summary of open-ended responses on classifying true vs. false positives (A1.6)

Category*	Description	Example	Count(%)
Dummy secret for testing (TR)	Data used to drive security test cases.	"It was a test input to verify end to end flow in test pass"	5 (33%)
Secret for test asset (DR)	Actual secrets used to secure test assets.	"It was an actual credential, but only for unit test purposes, so we will not remove it"	2 (13%)
Legacy secret (TR)	Potential secret was already exposed, not newly introduced.	"It was a possible credential pulled from a public open source GitHub repo into our private fork"	2 (13%)
False positive (TR)	Detection was a false positive with insufficient detail to classify as placeholder, dummy, etc.	"It was a variable name or string containing the word Password only"	2 (13%)
Secret placeholder (TR)	Detection was a placeholder, i.e., a value that is expanded or transformed at run-time.	"Not an actual credential, but I didn't include the "placeholder" text to clarify that"	1 (7%)
Not a secret (TR)	The detected code is not an actual, dummy or placeholder secret.	"The GUID flagged looked like a secret but is not a secret"	1 (7%)
Developer error (DR)	A potential secret was bypassed unintentionally.	"I actually made a mistake. I had reset to the first change I had made, rather than the change before. I thought [XSDT] was failing to see that I had reset, so I had it bypass."	1 (7%)
No better alternative (DR)	The developer did not perceive a better solution for securing the secret.	"It was an actual credential, and unfortunately the alternate ways to pass in the password to [...] apart from plaintext is no more safer than it"	1 (7%)

*For XTech's internal use, the categories are interpreted as follows. *Dummy secret for testing, False positives, Secret placeholder,* and *Not a secret* are treated as A1.1, *Secret for test asset* as A1.2, *Developer error, No better alternative* as A1.3, *Legacy secret* as A1.5

Table 6 Summary of open-ended responses on false positives (S2)

Category*	Description	Example	Count(%)
Dummy secret for testing (TR)	Data used to drive security test cases.	"It is for a unit test, not an actual credential"	14 (28%)
Not a secret (TR)	The detected code is not an actual, dummy or placeholder secret.	"I checked a test json which contains few guids and I was not able to check in the code without bypassing the scanning"	11 (22%)
Secret placeholder (TR)	Detection was a placeholder, i.e., a value that is expanded or transformed at run-time.	"It was a template for a connection string which did not have the actual password and just had a placeholder"	11 (22%)
Secret name was flagged (TR)	An identifier for a secret misinterpreted as a secret value.	"The actual credential is stored in a key vault. The value that was flagged was the name of the key vault entry that held the credential."	4 (8%)
Implementing security features (TR)	Code constructs that comprise security functionality were misconstrued as secrets.	"The code looked at the first character of a (user-supplied) secret to determine whether or not it starts with a hyphen."	3 (6%)

Table 6 (continued)

Category*	Description	Example	Count(%)
Secret for test asset (DR)	Actual secrets used to secure test assets.	"It was the [...] local emulator secret."	2 (4%)
Legacy secret (TR)	Potential secret was already exposed, not newly introduced.	"I used NPM to install the package [...] whose test code contains this credential"	2 (4%)
Testing XSDT behavior (DR)	Dummy code constructs pushed by users to test XSDT scanning behavior.	"I was testing if we lose history of the [XSDT] commit message"	1 (2%)
Secret still visible in history (DR)	Detected secret was not completely removed from VCS commit history.	"The credential were removed from the PR , but still [XSDT] kept complaining about it"	1 (2%)
Unknown	Developer did not provide an explanation.		1 (2%)

*The *Test secret* and *Already exposed* categories are interpreted as A1.2 and A1.5, respectively, for internal analysis at XTech

6.2 Observations on *False Positives*

In Question S2, 50 respondents who classified the XSDT warning by selecting A1.1 provided more details about the false positives through an open-ended response. We read the open-ended answers and identified seven categories of XSDT warnings that developers have stated are *not an actual secret* or false positives. We present these categories in Table 6. Each classification is suggestive of improvements specific to minimizing false positives in the category. XSDT has a feature, for example, that allows developers to entirely avoid 'Secret placeholder' false positives by encoding specific values in the contents of the placeholder. XSDT may need to be improved to make this capability more discoverable (we see more evidence that feature discoverability is a general problem in Section 6.5). Minimizing 'Dummy secret for testing' related false positives through heuristics is also possible. Augmenting analysis with machine learning might reduce detections that clearly do not comprise a secret (whether actual or which have the intentional 'shape' of one).

6.3 The Warnings are not Worth Fixing

The third question of the survey (S3) is *Please help us understand why the blocked secret is not worth fixing*. The question is mandatory for those that selected A1.2 in S1, and the developer can select multiple answer options for this question. We report the answers, response count, and percentages in Table 7. We describe our observations here:

Not a production credential, (A3.1, 8 respondents, 36%) Developers may not feel motivated to remove the secret after triggering an XSDT warning when they are working with artifacts that do not ship in the production system, as the secrets are not perceived as an immediate security risk.

No significant security value (A3.2, 6 respondents, 27%) A developer may perceive a secret as low risk if the secret does not protect any significant data, service or other software assets. Hence, the developer may choose to bypass the warning.

Additional protection, (A3.3, 2 respondents, 9%) The XSDT warning may also be bypassed if the secrets have additional protections. For example, we observed a high percentage of key files being exposed in repositories. Those key files, however, may be password-protected and thus developers felt less motivation to remove them.

Expired/invalidated secrets, (A3.4, 1 respondent, 5%) Developers may choose to invalidate or expire secrets rather than remove them. One respondent asserted that expired/invalidated secrets are not worth fixing.

Table 7 Responses on why the warnings are not worth fixing (S3)

Id	Answer	Count(%)
A3.1	It is not a production credential (for example, it is a credential for a non-shipping prototype)	8 (36%)
A3.2	The credential doesn't protect any data, service, etc., with significant security value	6 (27%)
A3.3	The credential has additional protections (e.g. it is encrypted in the file, is password-protected, etc.)	2 (9%)
A3.4	It expires soon or has already been invalidated	1 (5%)
A3.5	*Open-ended answers*	5 (23%)

We also categorized the five open-ended answers provided by the developers to S3, A3.5. Two responses added clarifying details to supplement the checked answers. Three open-ended answers were provided with no other answers checked. Two of the remaining answers suggested that A3.1 could have been selected. For example, one developer said, "These are test certs that we need to run our tests". The final answer suggested that A3.2 could have been checked, as the developer reported, "The VMs do not have any access to any network resources".

In summary, we observe that if developers' perception of the risk associated with a checked-in secret is low, they are less motivated to remove secrets and may bypass warnings. All secrets are not of equal value and secret detection tools are indifferent to the value of the asset protected by a secret. The developer likely has an understanding of this, having an automated mechanism to identify the value of the protected asset would be beneficial. Secrets for low risk assets can still be dangerous in the hands of an attacker, as they can often be used as a foothold for lateral movement that leads to sensitive assets.[15]

6.4 Why Secrets Cannot be Promptly Removed

The fourth question of the survey (S4) is *Please help us understand the considerations that prevented you from removing the secret immediately*. This question is mandatory and the developer can provide multiple responses. We report the answers, response count, and percentage in Table 8.

The most reported disincentive for prompt removal is a lack of awareness of an appropriate solution that would enable removing the secrets (A4.2, 4 respondents, 27%). Moreover, 20% (A4.3, 3 respondents) indicate that although developers did understand what solution to apply, doing so would have compromised a project deadline. Similarly, 20% (A4.5, 3 respondents) indicate that the developer tried to remove the credential after getting the warning, but faced difficulties in fixing in a reasonable amount of time. Overall, from these three observations, lack of time and technical complexity are the two factors that need to be considered for prompt removal. On the other hand, 13% (2 respondents) from both A4.1 and A4.4 suggest that developers also tend to bypass the XSDT warning when the repository contains existing technical debt (other exposed secrets) or when they feel that restructuring the code to remove the secret would introduce bugs into other, off-project systems. In summary, time constraints, workload pressures, and technical challenges contribute to the developers' motive to bypass XSDT warnings rather than immediately removing true positives.

6.5 Additional Insights

The final question of the survey was optional and open-ended to seek further comments on the challenge of detecting and removing exposed credentials, or suggestions for improving the tool. We received 37 open-ended responses. We report the main comments we received below, some of them add to or reinforce our earlier observations.

---

[15]XTech's policy is that actual secrets which are persisted to engineering artifacts should be invalidated within a time window set by the project's security experts. All bypassed secrets will continue to be tracked and reported by complementary downstream XSDT scan systems that are not part of this study.

Table 8 Responses on why secrets cannot be promptly removed (S4)

Id	Answer	Count(%)
A4.1	Removing the credential now would break services. etc., that belong to other teams.	2 (13%)
A4.2	We were not sure what secret management/other solution we can use to resolve the problem.	4 (27%)
A4.3	We understand how to remove the credential but doing so now would put our deadlines at risk.	3 (20%)
A4.4	We have other exposed credentials in our project and the costs to remove them all immediately are too high.	2 (13%)
A4.5	I attempted to remove the credential but it was too difficult to do in the time that I had.	3 (20%)
A4.6	*Open-ended answer*	1 (7%)

Appreciation for XSDT. Nine developers expressed that that they appreciate the XSDT warnings and find the tool useful. For example, one developer wrote, "I think your detection system is doing great. It's better to have false positive than true negatives in cases like this." Overall, these comments indicate agreement with the overall effectiveness of the tool in (and the overall importance of) minimizing the occurrence of checked in secrets

Request for features that already exist. Eight developers suggested the addition of features that already exist in the tool. Seven of these suggestions requested a suppression capability that already exists (the permanent bypass mechanism described in Section 3). For example, one developer said "For my case, it's actually useful if I can flag the keys to be skipped". One developer requested that XSDT output a pointer to supporting documentation for suppression: "Add a reference to some documentation to learn about the different options to mark parts of code to skip validation (for instance a comment)".

Clarification of bypass instance. Eight developers used the open-ended comment field to provide additional details on the specific detection that was bypassed. One developer noted: "This detection worked as intended, but the key was a part of a PR and has been removed with another commit in the same PR".

Requests for improved accuracy. Seven developers commented on a need to improve tool accuracy. For example, one developer states, "It seems like it's identified as a potential credential because it's composed by numbers, letters and symbols, which is not the case a lot of times."

Requests for improved guidance. Four developers requested expanded or improved guidance for resolving XSDT warnings. For example, one developer remarked: "Perhaps there should be better guidance on how to review these kinds of cred scanners detections". Another developer commented: "using a standard of examples for credentials would probably help both the exposed credential finder, and remove the friction on commits".

Request for new XSDT feature. One developer requested a feature that does not exist in XSDT today, stating: "you can add manager approvals for these commits to be pushed".

6.6 Summary of RQ4 Findings

– From Table 4, we observe that developers classified 50% of the warning as false positives and bypassed the warning. The rest of the warnings are bypassed for other

development-related factors. Hence, focusing on the accuracy of the secret detection tool may not be able to solve the secret check in phenomenon alone.

– Developers classified 50% of the warnings as false positives. From Table 2, we observe that 48.6% of the potential secrets did not get exposed. From Table 4, we observe that 14% of the respondents introduced and do not intend to remove a true positive. Adjusted to include all XSDT warnings, 92.8% of detections are therefore initially blocked or will be removed after exposure (conforming to XTech policy), the remaining 7.2% require additional follow-up initiated by supplemental security scans which are not the focus of this work. These additional checks, comprising at least two more layers of scanning at later points in the workflow, are an opportunity for XTech's processes to identify and eliminate the exposures that were inadvertently committed to the VCS. These additional systems are not the focus of this work

– Developers may bypass a warning if they think the corresponding secret is not protecting a sensitive asset and hence, the security risk is low. XSDT cannot determine the value of asset being protected by the secret but the developers have this understanding.

– Developers may view pushing actual secrets for non-production environments such as test services and prototypes and proof-of-concept projects as low risk. XTech's policy is that actual secrets which are persisted in engineering artifacts should always be invalidated within a time window set by the project's security experts.

– Developers may experience technical challenges removing secrets. Developers also indicated that they are more likely to bypass if their deadline is at risk.

– Developer decisions to remove the secrets or bypass the warning may also depend on development considerations of the projects they are working on. For example, services or software of partner teams with shared dependencies may break if secrets are invalidated without coordination across the organization. The additional time and effort needed to apply the changes contributes to the developers' decisions when responding to XSDT warnings.

7 Discussion

1. **Why developers are bypassing? What do we know now?** From RQ1 - RQ4, we observe that developers can bypass a tool warning for these potential factors:

   (a) False positive warnings;
   (b) Difficulty in removing the secrets from artifacts;
   (c) Working with projects where developer has to deal with existing coding pattern which may trigger warnings;
   (d) Developer may have less security awareness;
   (e) Developer may be subject to workload/time constraints;
   (f) Non-production projects, such as prototype projects, may contain bypassed secrets; and
   (g) Developer may consider a secret not risky to be exposed in a repository as the asset protected by the secret is of low value

   These factors should not be considered an exhaustive list of potential factors. These identified factors may also be correlated with each other. (For example, less security awareness may have a correlation with perceived value of an asset protected by the bypassed secret.) Apart from the tool accuracy factor, the rest of the factors should be

tested against a large user study. Our identified factors are derived from one industrial case study and hence, all of these factors would benefit from additional evaluation from multiple organizations' development practices.

2. **Could there be any relation between developers' perception and the identified factors?** Developers perception on the security aspects of the system is expected to have an impact on the identified factors we stated above. We did not study how developers *evaluate* the value of an asset, security risks associated with the asset and trade-off between accepting the risk (checking in) and mitigating the risk (removing the secret). Hence, the accuracy of their risk assessment can be studied as well. For example, one developer may think that a secret for authenticating a local virtual machine is not worth attention (pushing the secrets in organizational repository so that no public access is possible), which should be a very low security risk. However, such secrets are not invisible from insider attacks. For designing effective secret detection tools and development practice around it, how developers evaluate the risk should thoroughly be tested to gain a holistic understanding of the factors that can influence developers to bypass.

8 Recommendations and Future Work

Based upon our engineering and organizational experience as well as quantitative and qualitative studies of this issue at XTech, we make the following recommendations:

Incorporating secondary checking.   The developers prefer tools that can verify code early in the development and deployment process, ideally while they are editing the code. At this early stage in the development process, tools may be less accurate because the tool may not incorporate the context of the code in its analysis. As a result, tools often provide bypass mechanisms. As shown in our study, allowing developers to bypass warnings can result in secrets being exposed. To address this issue, our results suggest that teams, such as is done by XTech, institute a secondary process for reviewing bypass decisions, providing a mechanism to correct invalid actions.

Study factors that lead to delayed removal of secrets: We observe a significant percentage of bypass decisions for true positives due to time pressure or burdensome costs to immediately remove the secrets. A deeper understanding of what factors are most prevalent would help prioritize investments for improving motivation, increasing ability and lowering costs to immediately remove secrets from code.

False positive reduction: Other researchers (Saha et al. 2020) show that secret detection tools may report false positives. We also show this issue and we were able to show that this is a complex problem. A particular concern identified by our work is the problem of identifying test data and placeholders, i.e., patterns that have the actual 'shape' of a secret but which contain dummy values or values that are transformed at runtime into actual secrets. Machine learning may assist with this problem and generally improve the effectiveness of secret detection tools, as suggested by others (Saha et al. 2020).

Study on secret distribution: Our results indicate that the majority of secret detection tool results in clustering around a relatively small percentage of people, file types, repositories, etc. 1130 repo, 23570 files, 7704 devs Hence, the efficiency and effectiveness of investments made to lower true positive introduction rates could be improved by understanding where exposure is most concentrated.

Security risk awareness: Developers perceive that checking in secrets for test and other non-production assets is low risk and acceptable. However, persisting any actual secret to a VCS creates the risk of data leakage to public or insider threats. Developers appear to underestimate the risk associated with these secrets. Attackers are skilled at using a foothold provided by a 'low risk' secret for subsequent lateral movement that can lead to higher-value assets.

Enhanced asset risk metadata: Developers have knowledge of the value and longevity of an asset protected by a secret. Secret detection tools are indifferent to the relative value of these assets. Having an automated mechanism to categorize the value of the protected asset would be beneficial.

Purging secrets. Purging secrets from VCS history is not currently a straightforward task. VCSs should offer features that facilitate easier removal of sensitive data from history.

Earlier detection: In order to lower costs (such as the cost of purging secrets from history), scanning can be moved from the push phase to an earlier phase, such as local commit. This feature can also be supported by integrated development environments to report warnings at code authoring time.

Studying the factors responsible for bypassing: For future work, the potential factors for bypassing the tool warning can be studied across multiple organizations and developers with different skill-set. The correlation among these factors can also be studied.

Studying the risk evaluation process: Researchers and practitioners can also consider studying how developers evaluate the risk associated with the secrets, how they estimate the value of the asset protected by the secret, and the potential correlation between the perceived risk and actions taken by the developers on the tool warning.

Studying whether *some* types of projects are more prone to have exposed secrets Certain classes of projects may be more prone to exposing secrets. For example, projects having backward compatibility constraints, interoperability with third party constraints, or specific technical debts may have more exposed secrets. We advocate researchers and practitioners to investigate whether there are such attributes of projects that can lead towards exposing more secrets by developers.

9 Limitations and Threats to Validity

1. **XSDT telemetry and success of the tool:** The findings from RQ1 - RQ3 were derived from telemetry of XSDT usage. We did not verify whether every detected secret by XSDT is really a secret or not (which can only be verified by the corresponding teams/developers working with the project and hence, this verification requires significant amount of time which is out of scope for this study). The success of the tool was also measured solely from the tool's point of view as verifying individual exposure or block of secrets is also out of scope for this study. From security point of view, any potential secret checking-in getting blocked can be considered as a lowered or mitigated potential security risk.

2. **Untested hypothesises:** From the findings of RQ1-RQ4, we suggested several hypothesises (such as high attempt count implicates difficulty in removing the secret; or a specific set of developers are subject to get higher rate of XSDT warning due to

technical debt, workload pressure) which are speculated but not tested. We consider this as a limitation of our study, however, we advocate future researchers to test these hypothesises to unearth more findings on how to make secret detection tools more effective and efficient.

3. **Survey focus:** We sent the survey only to developers who exhibited a specific behavior (one-time bypass). We did not investigate what factors prompted developers to take other actions such as remove or abandon. A complete view would include developers who disposed of warnings in other ways (removal, branch abandonment, or permanent bypass). However, the scope of this study was solely focused on the bypass activity of the developers to identify factors that could lead to the further enhancement of the XSDT tool and development practices around it. We also advocate future researchers to investigate the set of developers who took different actions for different alerts (i.e. removed secrets on an alert however, bypassed another alert).

4. **Subjective responses:** The security perception and expertise of the developers we survey may vary among employees and teams. All survey responses are subjective and will vary from person to person. We also did not perform an independent analysis on whether survey responses are correct. We identify this as a source of bias. However, when we sent the survey emails to the developers, we also sent them the url of the commit changesets where they chose the bypass actions which helps them recall the reason they bypassed.

5. **Survey population:** Moreover, we only considered full time employees as survey respondents. We exclude any other type of employees such as contractors and interns in the survey population. Their perceptions may have been useful.

6. **Response bias:** We achieved approximately 25.1% response in the survey however, 74.9% of the audience did not respond to the survey, which may have led to a response bias.

7. **Recall bias:** Although, we choose to send the survey to the developers who bypassed the warning in the last 7 days, the responses could still have a recall bias.

8. **Generalizability:** The findings from the survey should not be generalized as the response count is *small* and several tables such as Tables 5 and 8 represents data for a very low (n = 15) number of developers.

9. **Developer perception:** The survey responses came from multiple developers having different skillset, background and responsibilities. Security risk perception regarding checking-in a secret or the perceived value of an asset protected by the potential secret can also vary from developer to developer. Moreover, the development factor related to bypassing the warning or not going back to resolve the warning may also be a subject to bias. Independent confirmation of the survey responses was out of scope for this study. A deeper understanding of what factors are most prevalent for making decisions (remove secrets or bypass warnings) could have made our findings more generalizable. We advocate future researchers to investigate these factors further.

10. **Covid-19 pandemic:** Finally, the entire study was conducted during the COVID-19 pandemic (Wikipedia 2021), and hence developers may have adopted different ways of collaborating with each other, new development activities, and lifestyle choices. All of these factors could introduce a bias in their responses.

11. **Others:** Finally, although we iteratively created the survey and pilot tested it, the questions and closed-option responses may have introduced additional bias.

10 Conclusion

Despite using secret detection tools, the tendency to check-in secrets is still evident among developers. In this work, we analyzed the usage data of XSDT, an internal secret detection tool used in XTech, and surveyed the developers who bypassed the XSDT warnings. We found that developers bypassed the tool warnings even if the detections made by the tool were accurate. From the survey, we observe that, developers may bypass the tool warning based on their perception of (i) the risk associated with checking-in secrets; (ii) effort; (iii) the time required to remove the secret; and (iv) technical debts. Based on these observations, we provide actionable suggestions that may help mitigate the checked-in secret problem. Overall, we find that, a secret detection tool, despite being accurate half of the time in detecting secrets in software artifacts, may not be able to mitigate the problem alone as there could be other potential factors. We advocate for further investigation on developers' perception of risk associated with the checking-in secrets, technical debt incurred for removing secrets/restructuring artifacts and any other potential associated factors which can have an impact on developers' decision making while using the tool.

Data Availability Uploaded as supplementary metadata files

Availability of data and material In order to protect the identity and privacy of the company, we cannot unfortunately share the raw data for this study

Code Availability N/A

Declarations

Conflict of Interests N/A

References

Amazon (2020) Aws key management service https://aws.amazon.com/kms/
MITRE (2020) Cwe-798: Use of hard-coded credentials. https://cwe.mitre.org/data/definitions/798.html Accessed 16 Sep 2020
Meli M, McNiece M, Reaves B (2019) How bad can it git? characterizing secret leakage in public github repositories Networked and Distributed Systems Security Symposium (NDSS)
Montalbano E (2020) Medical data leaked on github due to developer errors. https://threatpost.com/medical-data-leaked-on-github-due-to-developer-errors/158653/, Accessed 16 Sep 2020
Ilascu I (2020) Starbucks devs leave api key in github public repo. https://www.bleepingcomputer.com/news/security/starbucks-devs-leave-api-key-in-github-public-repo/, Accessed 16 Sep 2020,
Kumar V (2020) Jumpcloud api key leaked via open github repository. https://hackerone.com/reports/716292, Accessed 16 Sep 2020

Rahman A, Parnin C, Williams L (2019) The seven sins: Security smells in infrastructure as code scripts. In: 2019 IEEE/ACM 41St international conference on software engineering (ICSE), pp 164–175

Rahman A, Rahman MdR, Parnin C, Williams L (2020) Security smells in ansible and chef scripts: A replication study. ACM Transaction of Software Engineering and Methodology

Rahman MdR, Rahman A, Williams L (2019) Share, but be aware: Security smells in python gists. In: 2019 IEEE International conference on software maintenance and evolution (ICSME), pages 536–540

Saha A, Denning T, Srikumar V, Kasera SK (2020) Secrets in source code: Reducing false positives using machine learning. In: International conference on COMmunication systems NETworks (COMSNETS), pp 168–175

Sinha VS, Saha D, Dhoolia P, Padhye R, Mani S (2015) Detecting and mitigating secret-key leaks in source code repositories. In: 2015 IEEE/ACM 12Th working conference on mining software repositories, pp 396–400

Ding ZY, Khakshoor B, Paglierani J, Rajpal M (2020) Sniffing for codebase secret leaks with known production secrets in industry

Viennot N, Garcia E, Nieh J (2014) A measurement study of google play. SIGMETRICS Perform Eval Rev 42(1):221–233

Bunyakiati P, Sammapun U (2019) On secret management and handling in mobile application development life cycle: a position paper. In: 2019 34Th IEEE/ACM international conference on automated software engineering workshop (ASEW), pp 77–80

Johnson B, Pandita R, Smith J, Ford D, Elder S, Murphy-Hill E, Heckman S, Sadowski C (2016) A cross-tool communication study on program analysis tool notifications. In: ACM SIGSOFT International symposium on foundations of software engineering, pp 73–84

Box GEP, Daniel Meyer R (1986) An analysis for unreplicated fractional factorials. Technometrics 28(1):11–18

Wikipedia (2021) Covid-19 pandemic https://en.wikipedia.org/wiki/COVID-19_pandemic

Md Rayhanur Rahman has started his Ph.D. program in the department of Computer Science at North Carolina State University. His research interest is in the area of Software Engineering and Software Security. Currently he is working on the research domain of mining the cyber threat intelligence artifacts.

Nasif Imtiaz is a Ph.D. student at North Carolina State University. His current research interest lies in the area of software engineering, software security, and software supply chain security. Contact him at simtiaz@ncsu.edu.

Dr. Margaret-Anne Storey is a Professor of Computer Science and Co-Director of the Matrix Institute for Applied Data Science at the University of Victoria. She holds a Canada Research Chair (Tier 1) in Human and Social Aspects of Software Engineering and is a member of the Royal Society of Canada's College of New Scholars, Artists and Scientists.

Laurie Williams is a Distinguished University Professor and co-director of the Secure Computing Institute at North Carolina State University. Her research area is software security. Laurie received her PhD in Computer Science from the University of Utah. Contact her at lawilli3@ncsu.edu.

Affiliations

Md Rayhanur Rahman[1] · **Nasif Imtiaz[1]** · **Margaret-Anne Storey[2]** · **Laurie Williams[1]**

Nasif Imtiaz
simtiaz@ncsu.edu

Margaret-Anne Storey
mstorey@uvic.ca

Laurie Williams
lawilli3@ncsu.edu

[1]   North Carolina State University, Raleigh, NC, USA

[2]   University of Victoria, Victoria, BC, Canada