A Comparative Study of Software Secrets Reporting by Secret Detection Tools

Setu Kumar Basak*, Jamison Cox†, Bradley Reaves‡ and Laurie Williams§
North Carolina State University, USA
Email: *sbasak4@ncsu.edu, †jcox3@ncsu.edu, ‡bgreaves@ncsu.edu, §lawilli3@ncsu.edu

Abstract*—*Background: **According to GitGuardian's monitoring of public GitHub repositories, secrets sprawl continued accelerating in 2022 by 67% compared to 2021, exposing over 10 million secrets (API keys and other credentials). Though many open-source and proprietary secret detection tools are available, these tools output many false positives, making it difficult for developers to take action and teams to choose one tool out of many. To our knowledge, the secret detection tools are not yet compared and evaluated.** ***Aims: The goal of our study is to aid developers in choosing a secret detection tool to reduce the exposure of secrets through an empirical investigation of existing secret detection tools. Method:*** **We present an evaluation of five open-source and four proprietary tools against a benchmark dataset.** ***Results:*** **The top three tools based on precision are: GitHub Secret Scanner (75%), Gitleaks (46%), and Commercial X (25%), and based on recall are: Gitleaks (88%), SpectralOps (67%) and TruffleHog (52%). Our manual analysis of reported secrets reveals that false positives are due to employing generic regular expressions and ineffective entropy calculation. In contrast, false negatives are due to faulty regular expressions, skipping specific file types, and insufficient rulesets.** ***Conclusions:*** **We recommend developers choose tools based on secret types present in their projects to prevent missing secrets. In addition, we recommend tool vendors update detection rules periodically and correctly employ secret verification mechanisms by collaborating with API vendors to improve accuracy.**

I. Introduction

GitGuardian measured the exposure of secrets in GitHub repositories for the last three years and reported in March 2023 that secrets sprawl continued accelerating in 2022 by 67% compared to 2021, exposing more than 10 million secrets [1]. In addition, they discovered that one out of 10 GitHub code authors exposed at least one secret in 2022. Secrets (such as API keys and access tokens) are indispensable for software as secrets are needed for third-party service integration, such as payment systems. However, developers leak secrets in plain text in the version control systems (VCS) and application packages [2], [3]. In September 2022, an attacker took over Uber's internal tools and applications by leveraging hard-coded admin credentials in their PowerShell scripts [4].

To prevent secrets from leaking in VCS, several open-source and proprietary tools such as Gitleaks and SpectralOps are available. However, these tools generate many false positives. Chess and McGraw [5] state that a high percentage of false positives may lead to 100 percent false negatives because people stop using the tool. This phenomenon is called *alert fatigue* [6]. In addition, a tool will be unsound if it allows false negatives to escape to reduce false positives. As a result, developers face challenges in selecting secret detection tools. To our knowledge, no research has been conducted yet evaluating and comparing existing secret detection tools.

The goal of our study is to aid developers in choosing a secret detection tool to reduce the exposure of secrets through an empirical investigation of existing secret detection tools.

In this study, we analyzed existing open-source and proprietary secret detection tools and provided answers to the following research questions:

- **RQ1:** How do the secret detection tools perform in detecting secrets in terms of precision and recall?
- **RQ2:** What features are offered by the secret detection tools to aid in preventing secrets exposure?

We selected five open-source and four proprietary tools and compared the tools against a benchmark dataset of 818 repositories. We analyzed the tools report and evaluated how tools perform in detecting secrets. In addition, we analyzed the features offered by the tools in preventing the exposure of secrets and identified future research needs for secure software secret management. We have also made a dataset of the false positive secrets reported by the tools publicly-available for future researchers to aid in expediting research on the accuracy of the tools [7]. We summarize our contributions as follows:

- A first comparative study of the existing open-source and proprietary secret detection tools and a qualitative analysis of the reports generated by the tools;
- A categorization of the features provided by the secret detection tools to aid in preventing secrets exposure; and
- A dataset of false positive secrets reported by the tools.

The rest of our paper is structured as follows: Section II, III and IV introduce the benchmark dataset, selection process of tools, and the methodology to compare and evaluate the tools result, respectively. We discuss the findings and implications of our work in Section V and VI. Section VII discusses the ethics, followed by the limitation of our paper. We discuss the related work in Section IX and conclude in Section X.

II. Benchmark Dataset

To compare the secret detection tools, we selected *SecretBench* [8], a publicly-available benchmark dataset of software secrets. We accessed the dataset using Google Cloud Storage (Bucket Name: *secretbench*) [9] and Google BigQuery (Dataset ID: *dev-range-332204.secretbench.secrets*) [10]. A detailed description of the dataset is given below:

978-1-6654-5223-6/23/$31.00 ©2023 IEEE

Repositories: The dataset has been curated from the Google BigQuery Public Dataset of GitHub [11] using 761 regular expression patterns of different types of secrets. The dataset consists of 818 public GitHub repositories.

Secrets: The dataset consists of 97,479 labeled plain-text secrets (labeled as true and false) extracted from 818 repositories. The secrets were manually labeled by the two authors of SecretBench [8]. Among the 97,479 candidate secrets, 15,084 are true secrets. In addition, among the true secrets, 4,457 are unique since the same secret can have multiple instances in a repository (multiple commits and files).

Categories: The secrets of the dataset are categorized into eight categories. The number of total candidate secrets and true secrets of the eight categories are presented in Table I. The top three categories based on the number of true secrets are: "Private Key", "API Key and Secret" and "Authentication Key and Token". The candidate secrets of the "Other" category are random strings and non-exploitable IDs such as GitHub commit IDs which are mostly false positives (99.29%).

TABLE I: The eight categories of secrets in SecretBench.

Category	True Secrets	Total Secrets
Private Key	5,789	8,584
API Key and Secret	4,529	5,162
Authentication Key and Token	3,569	5,833
Other	524	66,690
Generic Secret	334	439
Database and Server URL	162	9,970
Password	150	705
Username	27	96

Programming Languages: The dataset repositories comprised source codes of 49 programming languages. The top five programming languages based on the number of repositories are Shell (459), JavaScript (414), Python (312), Java (180), and Ruby (172). The number in the parenthesis denotes the number of repositories that used the specific language.

File Types: The dataset consists of 311 file types in which secrets have been found. All the 311 file types and the number of true secrets present in these file types can be found in the GitHub repository of SecretBench [12]. The top five file types based on the number of true secrets are presented in Table II.

TABLE II: SecretBench's top five file types on true secrets.

File Type	Description	True Secrets
txt	Text File	2,935
toml	Configuration File	1,985
js	Javascript file	1,583
html	Hypertext Markup Language File	1,337
pem	Privacy Enhanced Mail Format File	813

Secrets Metadata: The dataset provides secrets metadata, such as repository name, file path, commit id and start line of where the secrets are matched. We used the metadata to compare the tool-reported secrets, as discussed in Section IV.

III. Secret Detection Tools

In this section, we explain the selection process of secret detection tools; provide a brief description of each tool; how we installed each tool; and how we scanned the benchmark repositories using each tool.

A. Selection of Secret Detection Tools

To find the existing open-source and proprietary secret detection tools, we searched both the web and academic literature. We constructed a set of the following search strings: *(secret OR credential OR password) AND (detection OR scanning OR digger) AND (tool OR utility)*. For web search, we used the Google Search Engine and selected the top 100 results for each search string according to the Google Search Engine's Page Rank algorithm. The stopping criteria of 100 for each search string has been set based on the guideline of grey literature search in prior works [13]. Similarly, for academic literature search, we searched the top five scholar databases recommended in the computing science domain [14], [15], [16], [17], [18]. We identified 20 tools from the search result and applied the following selection criteria to choose the secret detection tools for our study.

1) **Accessible:** The tool can be installed into a local system or accessed via subscription from the tool vendors.
2) **Scans Git Repositories:** The tool can scan Git repositories since our dataset contains Git repositories.
3) **Active:** The tool's repository has shown activity for the last two years. We checked the last commit date in the repository of the open-source tools.
4) **Flags Secrets:** The tool flags individual secrets instead of flagging only secret-containing suspicious file names.
5) **Reports Plain Text Secret:** The tool reports secrets in plain text as we must compare the secrets with our benchmark dataset.

Based on the above selection criteria, we excluded 11 tools. After each tool, we provide in parenthesis the criteria we used to exclude a tool using the enumerated criteria listed above: Credential-Digger [19] (1), Credscan [20] (1), Cycode [21] (1), detect-secrets [22] (5), git-all-secrets [23] (3), git-hound [24] (5), gitrob [25] (3), Gittyleaks [26] (3), repo-security-scanner [27] (4), SecretHunter [28] (1) and Saha et al. Tool [29] (1). Ultimately, we selected 9 secret detection tools, of which 5 tools are open-source and 4 tools are proprietary.

B. Tools Description

For the selected secret detection tools, we provide a) a brief description of the tool, b) how we installed the tool, and c) the scanning technique employed for finding secrets in benchmark repositories. Since each tool provides configuration options for detecting secrets, we installed and ran the tools with recommended configurations by contacting the tool vendors or by obtaining suggested configurations in the product documentation to get higher accuracy.

git-secrets: git-secrets [30] developed by AWS-Labs [31] is an open-source tool. We installed Version `1.3.0` of the tool using HomeBrew. In addition, as a pre-requisite to scan for secrets in the repositories, we installed two git hooks (`git secrets --install` and `git`

`secrets --register-aws`) separately for each repository. We used the `--scan-history` flag (`git secrets --scan-history &> report.txt`) to scan the entire Git history and outputted the secrets in a text file.

Gitleaks: Gitleaks [32] is an open-source tool written in Go. We installed Version `8.2.7` of the tool using HomeBrew and scanned the repositories using the `detect` command (`gitleaks detect -v --source=repo_dir --report-path=report.json`). The verbose flag (`-v`) has been used to retrieve metadata information of the matched secret, and we extracted the secrets in JSON files.

Repo-supervisor: Repo-supervisor [33] is an open-source tool written in JavaScript. We downloaded the binary release (Version `3.2.0`) and installed Node Package Manager (NPM) dependencies (`npm ci && npm run build`). The tool operates in two separate modes. The first mode allows to scan GitHub pull requests through webhooks, and the second mode works from the command line, where it scans local repository directories. We performed the latter by executing the `cli.js` file (`JSON_OUTPUT=1 node ./dist/cli.js repo_dir`) and extracted the output in JSON file.

TruffleHog: TruffleHog [34] is an open-source tool developed by Truffle Security [35] and written in Go. We installed Version `3.18.0` of the tool using HomeBrew. We scanned the repositories with `--regex` and `--entropy` flags enabled (`trufflehog git --regex --entropy file://repo_dir`) and downloaded the JSON report.

Whispers: Whispers [36] is an open-source tool written in Python. The tool supports different formats for structured text parsing, such as YAML and XML. The tool parses the source code in key-value pairs, where the key is the field name and the value is the potentially hard-coded secret assigned to the given key. We installed Version `2.1.5` of the tool using `pip3`. To scan the repositories, we executed the `whispers repo_dir > report.json` command and extracted the output in JSON files.

Commercial X: Since the proprietary tool vendor would not allow their identity to be disclosed in the paper, we refer to them as "Commercial X". In addition to scanning GitHub repositories, the tool can find secrets in images and non-searchable PDFs. The tool can be integrated with Slack, JIRA, and Google Drive to find any secrets exposure. We contacted their team and provided the snapshot of 818 repositories of our benchmark. They ran their tool on those repositories and provided us with the scan report. We parsed the scan report and outputted the secrets with the metadata in a CSV file.

ggshield: ggshield [37] has been developed by GitGuardian [38]. We installed the tool (Version `1.14.3`) using HomeBrew. Though the tool is open-sourced in GitHub, the tool requires an API key for scanning a repository since ggshield internally uses GitGuardian's public API [39] through py-gitguardian [40] client to scan and detect secrets. We contacted GitGuardian to get an API key (API Quota Limit: 8 Million) and set the key in the local environment variable to scan all the benchmark repositories. We executed the `scan repo` command (`ggshield secret scan repo repo_dir --show-secrets --json -v -o report.json`) for searching secrets in each repository. The `--show-secrets` flag has been used to extract the secrets in non-redacted form, and the found secrets are outputted in a JSON file.

Github Secret Scanner: GitHub has an integrated secret scanner [41] to scan for secrets in the repositories. The "Secret Scanner" settings can be enabled from the "Code security and analysis" option in GitHub. To scan the repositories of the benchmark dataset, we forked each repository into the first author's GitHub account. We enabled the "Secret Scanner" settings for each repository. As soon as we enabled the settings, the scanner was triggered and displayed the detected secrets under the "Security/Secret scanning alerts" tab of the specific repository. We wrote a Python script to extract each repository's secrets in a CSV file using GitHub Rest API [42].

SpectralOps: SpectralOps [43] is a proprietary tool. To scan repositories in a local environment, we created a Spectral account and contacted the Spectral support team to gain access for seven days. We received a Spectral Data Source Name (DSN) key and saved it in the local environment. The tool provides three scanning modes: "Developer", "Security" and "Audit" based on different precision and recall rates. The Spectral team recommended using the "Security" mode for better precision and recall. We ran the scan command (`spectral scan --all --forensic --ok --show-match --include-tags base,audit --with-branches --json report.json`) and outputted secrets in JSON files. The base and audit tags are used for "Security" scan mode, and `--forensic` flag retrieves the secret's metadata.

C. Machine Configuration

We installed eight tools in two Mac instances except for the GitHub Secret Scanner and Commercial X. The configuration of the instances are as follows: Instance 1 (OS: Monterey version 12.3.1, RAM: 64 GB, Persistent Disk: 1 TB) and Instance 2 (OS: Monterey version 12.6.2, RAM: 32 GB, Persistent Disk: 1 TB). We used two Mac instances to speed up the scanning process since the benchmark dataset contains large repositories with a large commit count. After scanning with each tool, we wrote Python scripts to extract the secret with additional metadata from the JSON and text files and outputted in CSV files. The extracted results are used for analysis and comparison, as discussed in Section IV.

IV. Analyzing Tool Results

In this section, we explain the secret and tool metadata we analyzed and how we filtered and compared the tool results to answer our research questions.

A. Secret Metadata

Below, we discuss the metadata information related to secrets we processed to answer our research questions.

Commit ID: A commit id in Git is a unique SHA-1 hash created whenever a new commit is recorded. The commit id

helps to identify the exact commit reference where the secrets have been found during comparison.

File Path: The file path is the file's location in the repository where the secret has been found. We normalized the file path as it contained either the computer root folder location where the tool has been installed or the repository directory. For example: Repo-supervisor outputs the file path as "<Repo_dir>/conf/file.py" while Spectralops outputs as "/Users/<User_name>/<Repo_dir>/conf/file.py". We extracted the file path as "conf/file.py" for comparison.

Line Number: The line number denotes the line in the file where the secret has been matched, which helps to identify if the same secret is present in multiple places of the same file.

Plain Text Secret: The plain text secret is the tool-reported hard-coded secret in the source code. However, some tools report secrets along with the source code context. For example, git-secrets outputs the function or variable declaration where the secret is used (bitly_token <- bitly_auth(key = "xxxxxx")). The "xxxxxx" is the secret where bitly_auth and bitly_token are the function and variable name, respectively. As a result, matching reported secrets with the benchmark through automation is difficult. In addition, manual inspection is impractical due to the large number of reported secrets by the tools. However, we observed patterns such as "key=", "token=" and "id:" in the reported secret text. We removed non-alphanumeric characters, such as brackets and space, from the string and extracted the secret by only taking the string part after the pattern. We used these normalized secrets for comparison.

Alert Count: The alert count is the total number of alerts reported by each tool which indicates the amount of audit effort required by the practitioners. Tools such as SpectralOps and ggshield provide the number of alerts in the respective reports. For tools that do not provide the number of alerts in the report, we calculated the total number of alerts using a Python script by iterating through each report.

B. Filter and Compare Tool Alerts

We observe that tools provide non-secret alerts, such as alerts for suspicious files and dangerous functions. For example, Whispers flags suspicious files, such as database.sql file, and dangerous functions, such as exec and eval. In the output, the tool provides a rule identifier for different types of alerts, such as secret and api-key for secrets; file-known for suspicious files; and system for dangerous functions. We filtered the non-secret alerts using the rule identifiers. We also filtered secrets committed after November 25, 2022, since the benchmark dataset contains secrets introduced before that date. For example, the GitHub secret scanner scans the repository's latest snapshot (February 25, 2023) since the tool can not scan a local repository. We retrieved the commit date of each commit using GitHub Rest API [42]. We filtered any secrets introduced after November 25, 2022, for a fair comparison of the tools with the benchmark.

Next, we compare the secret of each repository reported by the tool with the secrets of the same repository in the benchmark. We mark the secret reported by the tool as true positive (TP) if the secret is matched. Otherwise, we mark the secret as a false positive (FP). However, we are unable to match different types of secrets with exact string comparison for all the tools though we normalized the secrets. Below, we discuss the different scenarios of the secret match and how we calculated the match for each.

Jaro-Winkler Similarity: After normalizing the secrets for source code context, we observe that additional source code as a suffix can be present. For example, git-secrets outputs secrets with additional source code context ("analytics_configuration": {key: "xxxxxxxxxxxx", type: "Traffic"}). The secret is "xxxxxxxxxxxx" and after normalizing, we got "xxxxxxxxxxxxtypeTraffic" where the string part "typeTraffic" is not part of the secret. As a result, we cannot perform an exact match of the secret with the benchmark. To address this scenario, we used Jaro-Winkler Similarity [44] for string comparison, a variant of the Jaro Distance metric [45]. The Jaro–Winkler similarity employs a prefix scale that rewards strings that match from the beginning with high scores [44]. The Jaro–Winkler algorithm provides a similarity score between [0,1] where 0 represents two entirely dissimilar strings and 1 represents identical strings. We used the jaro_winkler_similarity function of jellyfish [46] package in Python to calculate the similarity. We found the similarity score of "xxxxxxxxxxxx" and "xxxxxxxxxxxxtypeTraffic" is 0.82. We termed two secrets a match if the similarity score equals or exceeds 0.7. We set the cut-off similarity score of 0.7 by randomly sampling secrets and observing the score with the benchmark.

Gestalt Pattern Match: We observe that a secret can contain additional context in the middle, especially for multi-line secrets. For example, private keys are generally present as multi-line in the source code. Tools output these private keys differently, making it difficult to perform an exact match with the benchmark. Figure 1 shows three different outputs of the same secret. Tool A outputs the "Proc-Type" and "DEK-Info" properties along with carriage return ("\r") and line feed ("\n"), which is the same as the benchmark. However, Tool B excludes the "Proc-Type" and "DEK-Info" properties in the output, and Tool C includes the properties but outputs the secrets in a single-line instead of a multi-line without "\r\n". To address this scenario, we used the Gestalt pattern matching algorithm [47] after removing non-alphanumeric characters from the secret and making the secret single-line. The algorithm calculates the similarity score by finding the longest common substring and then recursively finding the number of matching characters in the non-matching regions on both sides of the longest common substring [47]. As a result, we could match a secret even if the secret does not contain the middle context (the properties of the private key). We used the SequenceMatcher function of difflib [48] package in Python to calculate the Gestalt similarity score. We termed two secrets a match if the similarity score equals or exceeds 0.6. Similar to the Jaro-Winkler similarity, we set the

Fig. 1: Different outputs of the same secret by three tools.

cut-off similarity score of 0.6 by randomly sampling secrets and observing the score with benchmark secrets.

We marked a secret reported by a tool as TP if the secret equals or exceeds the cut-off similarity score of either the Jaro-Winkler or Gestalt algorithm. To check whether the combination of algorithms correctly matches tool-reported secrets with benchmark and label automatically, we randomly selected 100 unique reported secrets from each tool and manually inspected the label calculated by the algorithms. The combination of both algorithms correctly labeled 97% of the secrets.

Recall Cases: We observe that the same secret can be present in multiple commits, multiple files, and different lines of the same file of a repository. As a result, finding and removing all instances of a secret from the source code is necessary. However, every tool does not provide all the metadata related to secrets, such as the commit id, file path, and line number, as shown in Table V. As a result, we calculated the recall of each tool in two cases to have a fair comparison. Case 1 of recall denotes when the secrets of the benchmark are found exactly in the same commit, file, and line number of the tools report, and Case 2 denotes that the secrets of the benchmark are found at least in the repository, irrespective of the metadata. For Case 1, we matched each tool's reported secrets with all the benchmark secrets for a repository. If a secret of the benchmark matches the tool-reported secret but does not match the metadata, then we mark the secret as a false negative ("FN"). However, for Case 2, we matched the unique secrets of the benchmark for a repository with the reported secrets of the tools. If a secret of the benchmark matches the tool-reported secret but does not match the metadata, we still mark the secret as true positive ("TP") since the secret is at least found in the repository. We could not calculate Case 1 for Repo-supervisor and SpectralOps as these tools do not provide either commit id or line number, thus calculating F1-score using precision and Case 2 of recall.

C. Tool Metric

Below we discuss the tool metric we calculated to answer our research questions.

Scan Time: Scan time helps to understand how quickly secrets will be identified to remediate any secrets exposure. Running each tool multiple times on all 818 benchmark repositories is impractical since scanning takes a long time. Hence, we calculated the scan time on a sample set of repositories of our benchmark to calculate the efficiency of the tools. First, we curated the sample set of 15 repositories as follows:

- **Repository Size:** The largest, smallest and median size of a repository in the benchmark is 5,658.22 MB, 0.04 MB, and 37.42 MB, respectively. We selected a random sample of 6 repositories based on the repository size: 4 repositories with repository sizes greater than the median and 2 repositories less than the median.
- **Commit Count:** Since a repository of a larger size can have a low number of commit counts, and vice-versa, we also included repositories in the sample set based on the commit count. The benchmark repository's highest, lowest and median commit count is 425,699, 22, and 1,200, respectively. We selected a random sample of 6 repositories based on the commit count: 4 repositories with a commit count greater than the median and 2 repositories less than the median.
- **Programming Language:** The sample set should have at least 1 repository for each of the top 5 programming languages of the benchmark (see, Section II). We randomly selected 3 additional repositories since 2 languages were already present in the above-selected 12 repositories.

Next, we ran each tool 5 times on each of the 15 repositories, calculated the total scan time using the `time` [49] package of Python, and calculated the average scan time.

Popularity: Since the open-source tools publish their source code in a public repository, we can measure the tool's popularity among the developers. Developers can *fork* the open-source tools repository in GitHub. The fork count of a repository indicates a higher chance of attracting potential contributors to the project. Developers can also *star* a repository when they want to appreciate the project and *watch* when they want to be notified of all the activities (bug fixes, new features) of the project. We used each open-source tools repository's fork, star, and watch count as a proxy to calculate the tool's popularity instead of considering a single metric. Previous studies [50], [51] have also used these metrics to calculate the popularity of a repository. To verify the rank correlation among fork, star, and watch count, we calculated the Spearman's rho (ρ) [52] using Kaggle's GitHub repository dataset [53]. We observed a significant correlation between star and fork (ρ = 0.71), watch and fork (ρ = 0.60), and watch and star (ρ = 0.55) counts. To calculate the popularity score for each tool, we normalized the fork, star, and watch counts using min-max normalization [54] and calculated the average of the counts.

V. Results

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

A. RQ1: How do the secret detection tools perform in detecting secrets, in terms of precision and recall?

Below we discuss a) the precision, recall, and F1-score of each tool; b) the overlap of secrets reporting by the tools; c) a comparison of the scan time and popularity of the tools;

and d) an analysis of the false positives and false negatives reported by the tools.

Precision, Recall and F1-score: Table III presents the precision, recall and F1-score of each tool. The column "Precision (Total Alerts, TP)" denotes the precision of each tool in detecting secrets. The numbers in parenthesis denote the total number of alerts reported by the tool and the count of true positives detected by the tool, respectively. The columns "Recall - Case X (TP, FN)" present the recall of each tool, where X denotes the two cases as discussed in Section IV-B. The numbers in parentheses denote the number of true positives and false negatives found by the specific tool, respectively. Low precision indicates more false positives causing the tool to be unusable and low recall indicates more false negatives causing a missed opportunity to be alerted of a secret. The column "F1 Score" denotes the F1-score of each tool, the harmonic mean of precision and recall (Case 2) as discussed in Section IV-B. Below, we discuss our observations related to precision, recall, and F1-score.

TABLE III: Precision, Recall, F1-Score, Scan Time (ST), and Popularity Score (PS) of each tool.

Tool	Precision (Total Alerts, TP)	Recall - Case 1 (TP, FN)	Recall - Case 2 (TP, FN)	F1 Score	ST (min.)	PS
git-secrets	0.05 (94491,4907)	0.04 (671,14413)	0.21 (956,3501)	0.08	6.71	0.92
Gitleaks	0.46 (45932,21047)	0.86 (12954,2130)	0.88 (3901,556)	0.60	46.29	0.85
Repo-supervisor	0.02 (181310,3652)	X	0.17 (751,3706)	0.04	0.32	0.04
TruffleHog	0.06 (90982,5426)	0.31 (4736,10348)	0.52 (2323,2134)	0.11	8.52	0.87
Whispers	0.01 (416516,2448)	0.01 (122,14962)	0.38 (1707,2750)	0.02	0.91	0.00
Commercial X	0.25 (86607,21674)	0.22 (3255,11829)	0.48 (2151,2306)	0.32	X	X
ggshield	0.19 (167046,32277)	0.23 (3536,11548)	0.46 (2068,2389)	0.26	228.94	0.06
GitHub-scanner	0.75 (1721,1292)	0.03 (408,14676)	0.36 (1606,2851)	0.48	54.48	X
Spectralops	0.01 (1547994,4777)	X	0.67 (2979,1478)	0.02	50.03	X
	Highest	Second Highest	Third Highest			

- We observe that based on the precision, the top three tools are GitHub Secret Scanner (75%), Gitleaks (46%), and Commercial X (25%), respectively. Among the nine tools, five tools have a precision score of less than 7%.
- Based on recall, we observe that Gitleaks is the top tool in both cases (Case 1: 86% and Case 2: 88%) and the second-best based on precision. In addition, TruffleHog has the second-best recall in Case 1 (31%) and third-best in Case 2 (52%) though the precision is only 6%.
- We observe that based on F1-score, the top three tools are Gitleaks (60%), GitHub Secret Scanner (48%), and Commercial X (32%).
- Though GitHub Secret Scanner is the top tool based on precision, the recall score is low (6%), indicating the tool misses many secrets. In contrast, SpectralsOps is the third-best based on recall (68%), with a precision score of only 1%. Thus, our findings indicate that no current tool has the coveted high precision and high recall scores.
- Recent research [55], [29] utilizes machine learning (ML) to reduce false positives. However, Commercial X and SpectralOps, which employ ML to detect secrets, have lower precision scores 25% and 1%, respectively.

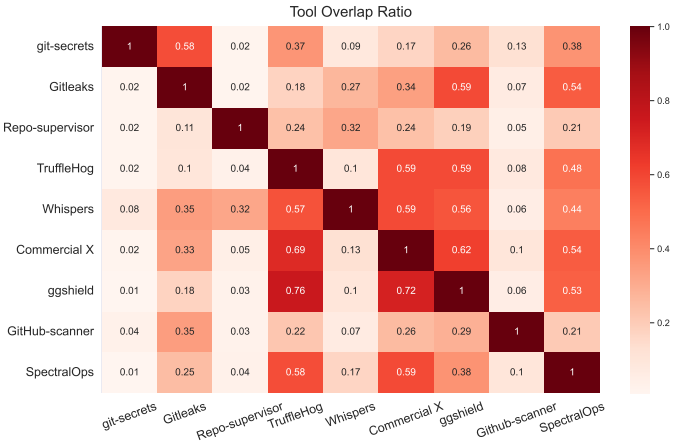


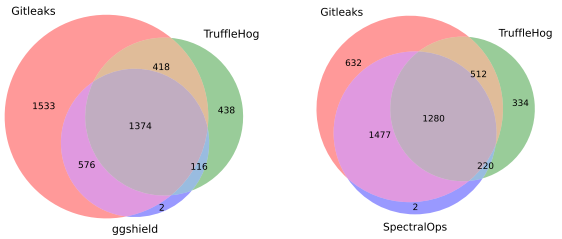
Fig. 2: Overlap ratio of secrets reported by each tool.

Since the secrets of our benchmark dataset are categorized into eight categories, such as "Private Key" and "API Key and Secret", we calculated the recall score per category for each tool. As a result, we identified which tool performs best in which category of secrets to aid developers in choosing tools based on the category of secrets present in their code. Table IV presents the recall score of each tool for the eight categories in two cases (Case 1 and Case 2). The numbers in parentheses denote the number of true positives and false negatives found by the tool for a specific category. We observed that Gitleaks and TruffleHog are the top two tools in most categories. However, SpectralOps has the second-best recall score for categories such as "Private Key" (Case 2) and "Generic Secret" (Case 2), whereas ggshield has the second-best recall for "Username" (Both cases). In addition, SpectralOps has the second-best recall score for categories such as "API Key and Secret" (Case 2), whereas GitHub Secret Scanner has the second-best recall for "Database and Server URL" (Case 2).

Tool Overlap: We measured how much unique true positive (TP) secrets one tool reported overlap with another to identify which tools output similar secrets. The heatmap of Figure 2 depicts the overlap ratio between each pair of tools. For a pair of tools (A, B), the heatmap shows how many unique TP secrets reported by tool A are also reported by tool B. For example, 76% of the unique TP secrets reported by ggshield are also reported by TruffleHog. However, only 18% of the unique TP secrets reported by ggshield are reported by Gitleaks. The Venn diagrams in Figure 3 show the non-overlap unique TP secrets among Gitleaks, TruffleHog, and ggshield (*Top three tools based on recall (Case 1)*) and among Gitleaks, SpectralOps, and TruffleHog (*Top three tools based on recall (Case 2)*). Figure 3a shows that Gitleaks and TruffleHog outputs 1533 and 438 non-overlap unique TP secrets, respectively. Similarly, as shown in Figure 3b, we observed that Gitleaks and TruffleHog outputs 632 and 334 non-overlap unique TP secrets, respectively. As a result, our findings substantiate the necessity of not relying on a single tool to identify all the secrets present in a repository.

TABLE IV: Recall of each tool for eight secrets categories.

Category	Case		git-secrets	Gitleaks	Repo-supervisor	TruffleHog	Whispers	Commercial X	ggshield	GitHub-scanner	Spectralops
Private Key	Case 1	Recall (TP,FN)	0.00 (4,5785)	0.96 (5585,204)	X	0.39 (2284,3505)	0.00 (28,5761)	0.37 (2133,3656)	0.38 (2227,3562)	0.00 (22,5767)	X
	Case 2		0.28 (782,2018)	0.99 (2759,41)	0.16 (441,2359)	0.59 (1648,1152)	0.55 (1546,1254)	0.57 (1606,1194)	0.53 (1470,1330)	0.33 (914,1886)	0.77 (2166,634)
API Key and Secret	Case 1		0.05 (233,4296)	0.86 (3917,612)	X	0.18 (802,3727)	0.01 (26,4503)	0.12 (547,3982)	0.14 (624,3905)	0.05 (205,4324)	X
	Case 2		0.09 (55,586)	0.75 (478,163)	0.17 (111,530)	0.33 (211,430)	0.11 (68,573)	0.34 (220,421)	0.44 (284,357)	0.38 (241,400)	0.55 (352,289)
Auth Key and Token	Case 1		0.09 (338,3231)	0.71 (2539,1030)	X	0.36 (1274,2295)	0.01 (46,3523)	0.08 (286,3283)	0.10 (378,3191)	0.04 (125,3444)	X
	Case 2		0.14 (74,463)	0.56 (299,238)	0.24 (127,410)	0.58 (308,229)	0.09 (49,488)	0.27 (143,394)	0.33 (176,361)	0.48 (258,279)	0.43 (233,314)
Generic Secret	Case 1		0.17 (57,277)	0.96 (321,13)	X	0.09 (29,305)	0.04 (12,322)	0.31 (105,229)	0.44 (148,186)	0.01 (4,330)	X
	Case 2		0.14 (18,114)	0.94 (124,8)	0.11 (15,117)	0.14 (18,114)	0.08 (10,122)	0.42 (56,76)	0.58 (76,56)	0.42 (56,76)	0.61 (81,51)
DB and Server URL	Case 1		0.00 (0,162)	0.34 (55,107)	X	0.93 (150,12)	0.01 (2,160)	0.43 (69,93)	0.51 (83,79)	0.26 (42,120)	X
	Case 2		0.08 (5,61)	0.41 (27,39)	0.29 (19,47)	0.98 (65,1)	0.11 (7,59)	0.60 (40,26)	0.59 (39,27)	0.67 (44,22)	0.53 (35,31)
Password	Case 1		0.07 (11,139)	0.70 (105,45)	X	0.32 (48,102)	0.04 (6,144)	0.38 (57,93)	0.26 (39,111)	0.00 (0,150)	X
	Case 2		0.08 (5,55)	0.85 (51,9)	0.15 (9,51)	0.17 (10,50)	0.18 (11,49)	0.57 (34,26)	0.15 (9,51)	0.23 (14,46)	0.55 (33,27)
Username	Case 1		0.85 (23,4)	0.85 (23,4)	X	0.85 (23,4)	0.00 (0,27)	0.00 (0,27)	0.26 (7,20)	0.00 (0,27)	X
	Case 2		1.00 (2,0)	1.00 (2,0)	0.00 (0,2)	1.00 (2,0)	0.00 (0,2)	0.00 (0,2)	0.5 (1,1)	0.00 (0,2)	0.00 (0,2)
Other	Case 1		0.01 (5,519)	0.78 (409,115)	X	0.24 (126,398)	0.00 (2,522)	0.11 (58,466)	0.06 (30,494)	0.02 (10,514)	X
	Case 2		0.07 (15,204)	0.74 (161,58)	0.13 (29,190)	0.28 (61,158)	0.07 (16,203)	0.24 (52,167)	0.06 (13,206)	0.36 (79,140)	0.41 (89,130)
									Highest	Second Highest	Third Highest



(a) (b)

Fig. 3: Venn diagram for overlap of unique true positive secrets among top three tools based on recall. Subfigure (*a*) depicts the overlap of Gitleaks, TruffleHog, and ggshield. Subfigure (*b*) depicts the overlap of Gitleaks, SpectralOps, and TruffleHog.

Scan Time: The column "ST" of Table III shows the time taken by each tool in minutes to scan the sample set of repositories. We could not calculate the scan time of Commercial X as the tool vendor has conducted the scanning, and the report does not contain any scan time. The top three tools based on scan time are Repo-supervisor, Whispers, and git-secrets, which took 0.32, 0.91, and 6.71 minutes, respectively. However, these tools have relatively low precision and recall scores indicating that tools did not scan the source code thoroughly. In contrast, the top two tools based on precision - GitHub Secret Scanner and Gitleaks took 54.48 and 46.29 minutes, respectively. However, we observe that tools having higher scan times do not always yield high precision and recall scores. For example, ggshield took the highest amount of time (4.8 hours) among all the tools, but the precision and recall were relatively low. We identified that Gitleaks, GitHub Secret Scanner, and SpectralOps showed a balance between scanning time and either high precision or recall.

Tool Popularity: The column "PS" of Table III presents the popularity score of each tool. We could only calculate the popularity score of the five open-source tools and one proprietary tool, ggshield. The source code of ggshield is open-sourced in GitHub, except for their proprietary scanning API implementations. Based on the PS score, the top three tools are git-secrets (0.92), Gitleaks (0.87), and TruffleHog (0.85), respectively. Though git-secrets is the most popular among the developers, the precision and recall are relatively low. In contrast, Gitleaks and TruffleHog are popular among developers having relatively high precision or recall scores.

Analysis of False Positives: Since we observed a high false positive rate by the tools, we inspected a random sample of 50 false positives from each tool to identify the types of false positive secrets. Below, we discuss our observations related to the false positive secrets and the detection rules triggering the false positives.

1. Generic Regular Expressions (regex): Tools use generic regex to detect secrets that trigger false positives. Below we discuss the generic regex for different types of secrets.

1.1 API Keys and Tokens: Tools, such as Whispers, employ generic regex (`.*[A-Za-z0-9_]+(key|token)$`) for finding API keys and tokens. The regex treats any string having a "key" or "token" at the end as an API key or token. As a result, placeholder API keys or tokens such as "`testkey`" and "`sampletoken`" are output as secrets. However, tools such as Gitleaks and GitHub Secret Scanner identify API keys and tokens by applying regex for specific API keys and tokens. For example, the regex employed for the Stripe API key is `(?i)(sk|pk)_(test|live)_[0-9a-z]{10,32}`. However, the regex matches "`sk_live_111111111111`", a dummy API key, and outputs as a Stripe API key.

1.2 Password: To detect passwords, generic regex such as `(passwords?|passwd|pass|pwd)_?[0-9]*$` is used. As a result, strings such as "`testpassword`" or a UNIX command ("`pwd`") are detected as passwords.

1.3 Cryptographic Key: According to a study by Meli et al. [2], cryptographic keys are the most exposed secrets in the source code. However, tools employ generic regex `(.*[-]{3,}BEGIN (RSA|DSA|EC|OPENSSH)? ?(PRIVATE)? KEY[-]{3,}.*)` to identify cryptographic keys, thus reporting false positives. For example, a template string such as "`---BEGIN RSA KEY---`" with no

following RSA key characters matches as a secret.

2. Ineffective Entropy Calculation: We observe that tools employ Shannon entropy [56] to identify possible secrets. Though the core Shannon entropy algorithm is correct, differentiating secrets from false positives is not always effective. For example, TruffleHog computes the entropy of "2b95710rD1e6287e69Z8f2E24373449d879b70c7601B3x9" and "ThisIsAReallyLongString" as 4.08 and 4.11 respectively, thus having higher entropy score for the latter [57]. As a result, the dummy string is termed as a secret. We also observed substantial instances of GitHub commit ids, such as "0e2b3d4e3dec5f38ae95f62519eb2736f73c0b", outputted as secrets because of ineffective entropy calculation.

3. Insufficient Filters/Prefix Regex: We observe that tools apply filters for HTML attributes and CSS selectors. For example, Repo-supervisor applies regex to prevent false positives such as "input[val='test']" and "button[value='submit']" [58]. However, the filters are insufficient as we observed strings such as "shape=rect;rounded=1" and "child{margin-bottom:10px;}" are still marked as secrets. In addition, tools apply prefix regex to ensure that at least one of the specified keywords related to the API key and token are within some characters (e.g., 40 characters) of the capturing group. For example, if a Strava API key is found by regex "`[0-9a-z]{40}`", then the specified prefix regex checks whether the keyword "strava" is present within 40 characters of the capturing group [59]. However, checking with prefix keywords does not always prevent false positives. For example, TruffleHog applies regex `((?:glpat|)[a-zA-Z0-9=_]{20,22})` with "gitlab" as prefix keyword to identify GitLab tokens. However, for a string such as "https://docs.gitlab.com/gitlab-basics/add-file.html#add-a-file-using-the-command", TruffleHog treats "add-a-file-using-the-" as a token since the string matches the regex and the prefix keyword is present within 40 characters.

Analysis of False Negatives: Since we observed a low recall score by the tools, we inspected a random sample of 50 false negatives from each tool. Below, we discuss the reasons behind the low recall score.

1. Faulty Regex: We observe that tools miss secrets because of employing faulty regular expressions. For example, Whispers employ regex `(.*[A-Za-z0-9_]+(key|token)$)`, which expects a secret will have a "key" or "token" word at the end. However, the "key" or "token" word can be present at the start of the context of the secret (`api_key="xxxx"`) or even not present at all, thus unable to capture secrets.

2. Skip Specific File Types: We observe tools skip specific file types while scanning. For example, ggshield does not scan HTML files to prevent false positives [60]. However, we observed that secrets are present in the HTML files either inside the HTML tags or in the JavaScript code embedded in the HTML files in a `<script></script>` tag. In addition, the HTML file type is in the top five file types containing secrets in the benchmark dataset (Table II).

3. Insufficient Ruleset: We observe that tools do not have sufficient rulesets for all secret types. For example, TruffleHog does not have detectors for IGDB [61] and Mashape API [62] keys. As a result, since TruffleHog matches prefix keywords for a specific key, these API keys are not captured. We also observe that tools do not periodically add/update rules for detecting secrets. For example, the rules of the tools such as Whispers were last updated on August 25, 2021.

False Positive Secrets Dataset: We created a dataset of the false positives reported by the tools to expedite the research on improving the accuracy of the tools. The dataset is stored as a relational structured data in Google BigQuery (Dataset ID: dev-range-332204.fpsecretbench), and users can run SQL queries to access the dataset. However, the dataset may contain sensitive information, such as mislabeled true positives since the applied string-matching algorithms may mislabel the tool-reported secrets (Section IV-B). As a result, we will distribute only to fellow researchers and tool developers who should email the authors to access the dataset [7].

B. RQ2: What features are offered by the secret detection tools to aid in preventing secrets exposure?

Tools provide features to aid developers in preventing the exposure of secrets. We categorized the features into seven categories. Table V presents the features offered by each tool, which we discuss as follows.

F1: Pre-commit Hook Integration: Pre-commit hook is a VCS mechanism that can be used for any validation before a commit is pushed. Secret detection tools can be integrated into a pre-commit hook to prevent leaking secrets. The tools will scan the source code of the current commit and reject the commit if any secret is found. Developers can employ this feature in accordance with "shifting left" on security [63].

F2: CI/CD Integration: Secret detection tools offer integration with continuous integration and continuous deployment (CI/CD) pipelines such as GitHub Actions [64], Travis [65], and CircleCI [66]. As a result, if a secret is found in the deployment package, the deployment can be rejected.

F3: Custom Rule: Tools support adding custom rules, thus allowing developers to devise rules to detect known secrets. Tools allow adding custom detectors using regex or keywords for scanning secrets. In addition, tools support custom rules for ignoring secrets. If a dummy secret is knowingly committed in the source code, developers can devise rules to ignore that secret to reduce the false positive warnings from the tools.

F4: Secret Verification: If any potential secret is detected, the tool verifies the validity of the secret by calling the endpoint provided by the respective API vendor to reduce false positives. For example, TruffleHog's AWS credential detector [67] performs a "GetCallerIdentity" API call against the AWS API to verify if the credential is active. In addition, if the secret is validated, GitHub Secret Scanner notifies the repository administrators and owners through email.

F5: Remediation Steps: Tools provide remediation workflows when a secret is detected to revoke and rotate the secret quickly. Tools assign the detected secret to the developer who leaked the secret. The developer can resolve the secret alert either by revoking the secret or marking it as a false positive.

Tools also use developer feedback to improve their algorithm to reduce false positives. In addition, tools also provide suggestions, such as removing the secret from Git history and reviewing access logs to nullify the threat completely.

F6: Infrastructure as Code (IaC) Script Scan: Scanning for secrets in IaC script is essential as Rahman et al. [68] identified hard-coded secret is the most occurring security smell within IaC scripts. SpectralOps and ggshield provide support for scanning secrets in IaC scripts.

F7: Non-source Code Scan: Developers can expose secrets in screenshots added as images in a repository and non-searchable PDFs shared for tutorials. These secrets can not be captured using regular regex matches. However, Commercial X employs Object Character Recognition (OCR) to detect secrets in images and non-searchable PDFs.

VI. Discussion and Recommendations

Below we discuss our findings and make recommendations.

Developers should employ tools based on the type of secrets present in the project. Table III shows that tools miss secrets as the recall (Case 2) varies between 17% and 88%. However, if developers know the secret types present in the project, selecting tools based on secret types can yield higher recall. For example, for "Database Server and URLs" category, the recall (Case 2) score of TruffleHog is 98% (Table IV), whereas the overall recall (Case 2) score is 52% (Table III).

Tool vendors should update detection rules periodically. According to the State of APIs Report from Rapid [69], API types are expanding, and API adoption is on the surge, with 63% of developers relying more on APIs in 2022. However, we observe that tools do not update the detection rules for API keys and tokens. For example, the rules of Whispers were last updated on August 25, 2021. We recommend tool vendors to update detection rules periodically to prevent missing secrets.

Tool vendors should correctly employ secret verification by collaborating with API vendors. We find that tools verify the found secrets with the API endpoints (F4). As a result, tools show relatively higher precision by reducing false positives. For example, before the verification option was enabled (`--only-verified`), TruffleHog's precision was 6%, outputting almost 100K alerts for our benchmark. In contrast, the precision changed to 90% when the verification was enabled and outputted only 611 secrets. However, verification methods are not 100% correct as we observe 10% false positives. For example, the tool tagged dummy server URLs such as "http://dyn.example.com:password@dyn.dns.he.net" as secrets. In addition, TruffleHog does not report an active secret if the API endpoint is unreachable [70]. We also find that GitHub has a secret scanning partner program [71] where API vendors can join in scanning their API keys and tokens in GitHub repositories and receiving notifications for quick remediation. However, only 66 API vendors have joined the program [72]. Therefore, we recommend that API vendors collaborate with tool vendors in correctly employing secret verification to prevent the exposure of secrets.

Tool vendors should develop automated technology to revoke and rotate secrets as remediation steps quickly. We find that tools provide remediation workflows when a secret is detected (F5). However, currently, the workflow is a manual process where the leaked secret is assigned to the developer to revoke and rotate the secret. In addition, developers have to sanitize the Git history by themselves using history sanitizing tools such as BFG repo-cleaner [73]. However, recent research [74] shows that malicious actors take only one minute to start making calls with the leaked API keys. Therefore, we suggest that tool vendors develop an automated workflow that the organization can employ in their system. The organization can mark the used secrets, and if a secret is reported that are among the used secrets, the workflow will automatically revoke and rotate the secrets. In addition, the workflow will sanitize the Git history without developers' manual effort, deploy new artifacts if needed, and review access logs to find any breaches.

VII. Ethics and Data Protection

Since the dataset of false positive secrets may contain mislabeled true positives, we will distribute the dataset selectively. To prevent unethical use, researchers and tool developers will sign a data protection agreement with us. Following that, we will use their email addresses to grant them access to our dataset from Google BigQuery. In addition, we have redacted/obfuscated example secrets presented in our paper.

VIII. Threats to Validity

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

Tool Selection: Our study's list of tools is not exhaustive. Though we have chosen the tools based on the selection criteria mentioned in Section III-A, we could not access proprietary tools such as Cycode [21] and CredScan [20]. As a result, we do not claim the findings we have in Section V to be generalizable for all tools.

Benchmark Dataset: Our selection of benchmark dataset is susceptible to bias. Basak et al. [8] curated SecretBench using open-source tools Gitleaks and TruffleHog, which also poses bias to the result of these two tools. However, they manually inspected and labeled each extracted secret using the tools. Out of 97,479 reported secrets of these two tools, 15,084 are true secrets. We used the true secrets to compare the tools of our study. SecretBench also has the drawback of only extracting secrets from GitHub repositories rather than from other VCSs, such as GitLab and BitBucket. Since SecretBench is the only publicly-available dataset, we could not compare the tools with another benchmark dataset to mitigate the potential bias.

Secrets Matching: We employed two string matching algorithms, Jaro-Winkler Similarity, and Gestalt Pattern Match, to match a secret with the benchmark for some tools. The similarity cut-off scores for both the algorithm we chose poses a threat to internal validity. However, we randomly selected 100 unique reported secrets from each tool and found that the combination of both algorithms' cut-off scores correctly labeled 97% of the secrets.

TABLE V: Seven categories of features and additional secrets metadata provided by each tool.

Tool	Tools Feature							Secrets Metadata		
	Pre-commit Hook	CI/CD Integration	Custom Rule	Secret Verification	Remediation Steps	IaC Script Scan	Non-source Code Scan	Commit ID	File Path	Line No.
git-secrets	✓		✓					✓	✓	✓
Gitleaks	✓		✓					✓	✓	✓
Repo-supervisor	✓	✓							✓	
TruffleHog	✓	✓	✓	✓	✓			✓	✓	✓
Whispers			✓					✓	✓	✓
Commercial X		✓	✓		✓		✓	✓	✓	✓
ggshield	✓	✓	✓		✓	✓		✓	✓	✓
GitHub-scanner			✓	✓	✓			✓	✓	✓
Spectralops	✓	✓	✓	✓	✓	✓			✓	✓

Precision for Each Secret Category: We have the category of a secret and the number of secrets in a category of benchmark dataset. As a result, we could calculate the recall of each category by checking if the secrets of the specific category of the benchmark are present in the tool-reported secrets. However, we could not calculate the precision for a category since the tool can output false positives, which requires manual inspection for categorization.

IX. Related Work

The root causes of the widespread leakage of secrets in software artifacts have been studied in prior work [2], [75], [68], [76], [77]. Researchers have found that the most prevalent insecure practice adopted by developers causing secret leakage is hard-coded secrets in software artifacts. In 2019, Meli et al. [2] studied a 13% snapshot of public GitHub repositories and found over 100K hard-coded secrets in the source code. Within Infrastructure as Code (IaC) scripts, Rahman et al. [68] studied a recurring coding pattern known as "security smells" which are indicators of security flaws that can result in potential security breaches. They investigated 5,232 IaC scripts extracted from 293 open-source repositories. They found seven security smells and the hard-coded secret is the most occurring security smell with 1,326 occurrences. In addition, hard-coded secrets have also been found in GitHub Gists that are used to share code snippets among developers. Rayhanur et al. [75] investigated 5,822 publicly available Python Gists and found 689 instances of hard-coded secrets in the code snippets. All of these prior works suggest that hard-coded secrets have been leaking in different forms in software artifacts.

To prevent secret leakage in software artifacts, researchers have suggested developers follow secure practices for secret management [78], [79]. Basak et al. [78] conducted a grey literature review of Internet artifacts, such as blog articles, and identified 24 practices comprised of both developer and organization practices. They suggested using VCS scan tools to prevent accidental commit of secrets. In another work, Basak et al. [79] investigated the questions related to checked-in secrets in Stack Exchange (SE) and the solutions posted by the SE users to mitigate the challenge. They identified that the SE users have also suggested using VCS scan tools to prevent accidental secrets leakage. However, in 2021, Rahman et al. [80] conducted a developer survey in XTech company (Anonymized) and found that developers bypass the alerts of scan tools as the tools generate a lot of false positives. Recent research [55], [29], [81] utilizes ML algorithms to reduce false positives in secret detection. Saha et al. [29] employed a Voting Classifier (a combination of Logistic Regression, Naive Bayes, and SVM) to distinguish real secrets from false positives. Feng et al. [55] applied deep neural networks to uncover the intrinsic characteristics of textual passwords and detect real passwords by reducing false positives.

At present, many open-source and proprietary secret detection tools are available. However, developers face difficulty choosing one tool out of many because of a high number of false positives. As far as we know, no research has been conducted yet evaluating and comparing the existing secret detection tools. In this work, we concentrated our research efforts on evaluating and comparing 9 secret detection tools.

X. Conclusion

We investigated five open-source and four proprietary secret detection tools against a benchmark dataset containing 818 GitHub repositories. We found that the top three tools based on precision are: GitHub Secret Scanner (75%), Gitleaks (46%), and Commercial X (25%), and based on recall are: Gitleaks (88%), SpectralOps (67%) and TruffleHog (52%). We also provided tools performance based on secret type to aid developers select the best tools for their use cases. Our manual analysis of the reported false positives indicates that generic regex and ineffective entropy calculation are the reasons for high false positives. We also analyzed the false negatives and found that faulty regex, skipping file types, and insufficient rulesets for secret detection are the reasons for low recall. In addition, we provided a dataset of false positives to expedite the research in secret detection. We also categorized the features offered by the secret detection tools to aid in preventing the exposure of secrets. We recommend developers choose tools based on secret types present in the project to prevent missing secrets. In addition, we recommend future research on developing an automated technology for quick remediation of the exposed secret.

Acknowledgment

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

REFERENCES

[1] "The State of Secrets Sprawl 2023," https://www.gitguardian.com/state-of-secrets-sprawl-report-2023, [Online; accessed April 12, 2023].

[2] M. Meli, M. R. McNiece, and B. Reaves, "How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories." in *NDSS*, 2019.

[3] 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 April 25, 2023].

[4] M. Jackson, "Uber Breach 2022 – Everything You Need to Know," https://blog.gitguardian.com/uber-breach-2022, [Online; accessed April 10, 2023].

[5] B. Chess and G. McGraw, "Static analysis for security," *IEEE Security & Privacy*, vol. 2, no. 6, pp. 76–79, 2004.

[6] Hadjy, Paul, "What Is Alert Fatigue? 4 Ways to Mitigate It and Prevent Burnout," https://learn.g2.com/alert-fatigue, [Online; accessed April 12, 2023].

[7] "False Positive Secret Dataset," https://github.com/setu1421/FPSecretBench, [Online; accessed July 02, 2023].

[8] S. K. Basak, L. Neil, B. Reaves, and L. Williams, "SecretBench: A Dataset of Software Secrets," *arXiv e-prints*, p. arXiv:2303.06729, 2023.

[9] "Google Cloud Storage," https://cloud.google.com/storage, [Online; accessed March 24, 2023].

[10] "Google BigQuery," https://cloud.google.com/bigquery, [Online; accessed April 12, 2023].

[11] "GitHub on BigQuery: Analyze all the open source code," https://cloud.google.com/blog/topics/public-datasets/github-on-bigquery-analyze-all-the-open-source-code, [Online; accessed March 17, 2023].

[12] "SecretBench File Types," https://github.com/setu1421/SecretBench/tree/main/Metadata, [Online; accessed June 26, 2023].

[13] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Information and Software Technology*, vol. 106, pp. 101–121, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584918301939

[14] "ACM Digital Library," https://dl.acm.org, [Online; accessed April 11, 2023].

[15] "SpringerLink," https://link.springer.com, [Online; accessed April 11, 2023].

[16] "IEEE Xplore," https://ieeexplore.ieee.org/Xplore/home.jsp, [Online; accessed April 14, 2023].

[17] "DBLP," https://dblp.org, [Online; accessed April 13, 2023].

[18] "ScienceDirect," https://www.sciencedirect.com, [Online; accessed April 11, 2023].

[19] S. Lounici, M. Rosa, C. Negri, S. Trabelsi, and M. Önen, "Optimizing leak detection in open-source platforms with machine learning techniques," 01 2021, pp. 145–159.

[20] "CredScan," https://secdevtools.azurewebsites.net/helpcredscan.html, [Online; accessed April 10, 2023].

[21] "Cycode: The Application Security Platform," https://cycode.com, [Online; accessed April 10, 2023].

[22] "detect-secrets," https://github.com/Yelp/detect-secrets, [Online; accessed April 14, 2023].

[23] "git-all-secrets," https://github.com/anshumanbh/git-all-secrets, [Online; accessed June 26, 2023].

[24] "git-hound," https://github.com/tillson/git-hound, [Online; accessed June 26, 2023].

[25] "Gitrob," https://github.com/michenriksen/gitrob, [Online; accessed June 23, 2023].

[26] "Gittyleaks," https://github.com/kootenpv/gittyleaks, [Online; accessed June 26, 2023].

[27] "repo-security-scanner," https://github.com/techjacker/repo-security-scanner, [Online; accessed April 14, 2023].

[28] E. Wen, J. Wang, and J. Dietrich, "Secrethunter: A large-scale secret scanner for public git repositories," in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2022, pp. 123–130.

[29] 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.

[30] "git-secrets," https://github.com/awslabs/git-secrets, [Online; accessed April 15, 2023].

[31] "Amazon Web Services - Labs," https://github.com/awslabs, [Online; accessed April 15, 2023].

[32] "Gitleaks," https://github.com/gitleaks/gitleaks, [Online; accessed April 15, 2023].

[33] "Repo-supervisor," https://github.com/auth0/repo-supervisor, [Online; accessed April 13, 2023].

[34] "TruffleHog," https://github.com/trufflesecurity/trufflehog, [Online; accessed April 14, 2023].

[35] "Truffle Security," https://trufflesecurity.com, [Online; accessed April 14, 2023].

[36] "Whispers," https://github.com/Skyscanner/whispers, [Online; accessed April 13, 2023].

[37] "ggshield," https://github.com/GitGuardian/ggshield, [Online; accessed April 13, 2023].

[38] "GitGuardian: Git Security Scanning & Secrets Detection," https://www.gitguardian.com, [Online; accessed March 27, 2023].

[39] "GitGuardian API," https://api.gitguardian.com/docs, [Online; accessed April 13, 2023].

[40] "py-gitguardian: GitGuardian API Client," https://github.com/GitGuardian/py-gitguardian, [Online; accessed April 13, 2023].

[41] "Github Secret Scanner," https://docs.github.com/en/code-security/secret-scanning, [Online; accessed April 16, 2023].

[42] "Getting started with the REST API," https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api?apiVersion=2022-11-28, [Online; accessed March 27, 2023].

[43] "SpectralOps," https://spectralops.io, [Online; accessed April 13, 2023].

[44] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage." 1990.

[45] M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 414–420, 1989.

[46] "Python jellyfish package," https://pypi.org/project/jellyfish, [Online; accessed March 28, 2023].

[47] P. E. Black, "Ratcliff/obershelp pattern recognition," *Dictionary of algorithms and data structures*, vol. 17, 2004.

[48] "SequenceMatcher of difflib package," https://docs.python.org/3/library/difflib.html#module-difflib, [Online; accessed March 28, 2023].

[49] "Python time package," https://docs.python.org/3/library/time.html, [Online; accessed March 28, 2023].

[50] J. Zhu, M. Zhou, and A. Mockus, "Patterns of folder use and project popularity: A case study of github repositories," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2652524.2652564

[51] H. Borges and M. Tulio Valente, "What's in a github star? understanding repository starring practices in a social coding platform," *Journal of Systems and Software*, vol. 146, pp. 112–129, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218301961

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

[53] "Kaggle's GitHub Repository Dataset," https://www.kaggle.com/code/pelmers/explore-github-repository-metadata, [Online; accessed March 12, 2023].

[54] "Feature Scaling," https://en.wikipedia.org/w/index.php?title=Feature_scaling&oldid=1075231919, [Online; accessed April 18, 2023].

[55] R. Feng, Z. Yan, S. Peng, and Y. Zhang, "Automated detection of password leakage from public github repositories," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 175–186.

[56] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[57] "Improving TruffleHog's Entropy Calculation," https://github.com/trufflesecurity/trufflehog/issues/168, [Online; accessed April 24, 2023].

[58] "Repo-supervisor: CSS Filters," https://github.com/auth0/repo-supervisor/blob/3d6252571318a24c1ffbaba48e024d79e44f9ac0/src/filters/entropy.meter/pre.filters/css.selectors.js, [Online; accessed April 24, 2023].

[59] "TruffleHog: Prefix Regex," https://github.com/trufflesecurity/trufflehog/blob/main/pkg/detectors/strava/strava.go, [Online; accessed April 13, 2023].

[60] "GitGuardian: Secrets Detection Engine," https://docs.gitguardian.com/secrets-detection/quick_start#how-it-works, [Online; accessed April 10, 2023].

[61] "IGDB API Docs: Getting Started," https://api-docs.igdb.com/#getting-started, [Online; accessed April 10, 2023].

[62] "Mashape API Documentation," https://rapidapi.com/rokity/api/mashape, [Online; accessed April 10, 2023].

[63] "DevOps tech: Shifting left on security," https://cloud.google.com/architecture/devops/devops-tech-shifting-left-on-security, [Online; accessed April 23, 2023].

[64] "GitHub Actions," https://github.com/features/actions, [Online; accessed April 20, 2023].

[65] "Travis CI," https://www.travis-ci.com, [Online; accessed April 12, 2023].

[66] "CircleCI," https://circleci.com, [Online; accessed April 15, 2023].

[67] "TruffleHog AWS Detector," https://github.com/trufflesecurity/trufflehog/blob/main/pkg/detectors/aws/aws.go, [Online; accessed April 15, 2023].

[68] 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.

[69] "2022 State of APIs," https://stateofapis.com/, [Online; accessed April 25, 2023].

[70] "TruffleHog: Verified secrets - unreachable website," https://github.com/trufflesecurity/trufflehog/issues/1112, [Online; accessed April 10, 2023].

[71] "GitHub Secret scanning partner program," https://docs.github.com/en/code-security/secret-scanning/secret-scanning-partner-program, [Online; accessed April 10, 2023].

[72] "GitHub Secret scanning patterns," https://docs.github.com/en/code-security/secret-scanning/secret-scanning-patterns#supported-secrets, [Online; accessed April 10, 2023].

[73] "BFG Repo Cleaner," https://rtyley.github.io/bfg-repo-cleaner, [Online; accessed March 17, 2023].

[74] Hercz, Tibor, "What happens when you leak AWS credentials and how AWS minimizes the damage," https://xebia.com/blog/what-happens-when-you-leak-aws-credentials-and-how-aws-minimizes-the-damage, [Online; accessed April 10, 2023].

[75] 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.

[76] I. Koishybayev, A. Nahapetyan, R. Zachariah, S. Muralee, B. Reaves, A. Kapravelos, and A. Machiry, "Characterizing the security of github CI workflows," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2747–2763. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/koishybayev

[77] 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.

[78] S. K. Basak, L. Neil, B. Reaves, and L. Williams, "What are the practices for secret management in software artifacts?" in *2022 IEEE Secure Development Conference (SecDev)*, 2022, pp. 69–76.

[79] S. K. Basak, L. Neil, B. Reaves, and L. Williams, "What Challenges Do Developers Face About Checked-in Secrets in Software Artifacts?" *arXiv e-prints*, p. arXiv:2301.12377, 2023.

[80] 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.

[81] E. Wen, J. Wang, and J. Dietrich, "Secrethunter: A large-scale secret scanner for public git repositories," in *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2022, pp. 123–130.