A Hybrid Framework for Credential Exposure Detection in Source Code Repositories and CI/CD Logs

Dr. Rekha K S[1], Professor and Deepika N[2], PG Student

[1,2] Department of Computer Science And Engineering, SJCE, JSS Science and Technology University, Mysuru, Karnataka, India
rekhaks@jssstuniv.in, deepuot7@gmail.com

Abstract. The leakage of credentials from code repositories and CI/CD logs still poses a significant security threat in the modern software engineering world. Credential leakage occurs through APIs, passwords, access tokens, and cloud credentials in source code and deployment logs. The current approaches of detecting such leakage mostly depend on pattern recognition and entropy analysis, resulting in potential false positives and noncontextual leaks. As a solution to such shortcomings, this paper presents a hybrid approach that incorporates static repository scanning, dynamic log scanning, machine learning classifiers, and prioritization techniques. Repository scans are conducted using regular expression and entropy-based measures, whereas CI/CD logs undergo scan tests for runtime credential leakages. Logistic Regression, Random Forest, Support Vector Machine, and Naive Bayes classifiers are used to evaluate the detected candidates. Fusion of the detected secrets is done using a fusion technique and prioritized based on risk levels. The system generates visualizations through a dashboard. The experimental study reveals that the fusion of the rule-based detection and machine learning improves detection of secrets and facilitates the monitoring of credentials leaks from both repositories and CI/CD logs.

Keywords: Secret Detection, Credential Leakage, CI/CD Security, Repository Scanning, Risk Scoring, Machine Leaning.

1 Introduction

Source code repositories and CI/CD pipelines are the integral components of today's software development process. These environments often include credentials like API keys, passwords, access tokens, cloud secrets, and private keys required for application services interaction. Despite their critical role in software functionality, the accidental exposure of these credentials is one of the most common security problems. Credentials may be included into the repository, placed in configuration files, hardcoded in the code, and even revealed in build and deployment logs [1], [2].

Credentials exposure can result in unauthorized access, service takeover, data breach, and money loss. Various empirical studies reveal that exposed credentials continue being alive for a considerable time period after discovery which increases the potential risk of credential use [1], [8], [15]. Using cloud services, DevOps techniques, and automated deployment has added many more locations for credentials exposure [7], [8].

The secret scanning technology is widely used to address these vulnerabilities. Almost all modern technologies use repository mining, scanning, and secret detection technologies to find exposed secrets. These technologies can accurately detect exposed secrets that fit into known templates. However, the random strings that mimic the format of secrets can result in false positive detection. Also, some secrets exposed during runtime operations cannot be found using repository mining [2], [11], [13].

Some recent studies have considered machine learning and methods based on large language models for better accuracy in secret detection [5], [6]. Such methods use the context for distinguishing between real credentials and non-sensitive string values, thus increasing the accuracy of detection. The risk-based approach has also recognized the need for priority based on the consequences of detected secrets [6].

The combination of components such as repository scanning, log analysis, machine learning classification, and risk-based prioritization can provide better coverage of the problem domain than any one of the above-mentioned detection techniques separately. The goal is to detect exposed secrets in the repositories and CI/CD logs, combining rule-based and machine learning methods to increase secret detection accuracy, and to monitor the process using risk-based prioritization and dashboard reporting.

2 Related Work

Detection of secret leakage in software repositories has been thoroughly studied due to the importance of the problem for software security and cloud technologies. Scientists have offered different approaches for detecting secret leaks, avoiding false positives, and ranking leak detection.

Meli et al. [1] performed a large-scale research on secret leakage by means of analyzing public GitHub repositories. Thousands of exposed credentials were revealed and it was found that a significant number of secrets stay active after their discovery. This research highlights the seriousness of the problem but it mainly focuses on measurement of the problem instead of improving detection.

Lykousas and Patsakis [2] developed a technique for automatic discovery of secrets in source code repositories and investigated developers behavior regarding password choice. It was found that mining of repositories can detect a large number of secrets. But the research was limited to repositories only and did not investigate CI/CD environments.

Saha et al. [3] presented a machine learning technique aimed at minimizing the number of false positive results in the detection of secrets in the source code. According to the results obtained, the addition of contextual features improves the differentiation between real and harmless pieces of information, solving problems associated with the rule-based approach to scanning.

RiskHarvester by Basak et al. [4] focused on risk assessment and remediation planning, as opposed to secret detection from multiple sources.

Biringa and Kul [5] conducted a research into the use of large language models to detect hard-coded credentials in software repositories. The research showed that using large language models increases secret classification accuracy as compared to the traditional rule-based detection, especially when dealing with obfuscated or complex exposure scenarios.

Similarly, Rahman et al. [6] assessed secret breach detection using large language models to separate genuine sensitive data from benign code. The research proved that contextual AI classification can detect credential exposure in more complicated structures in the source code than traditional scanning can reveal.

Abueshareik and Katbeh [7] studied the impact of AI-based security scanning on DevSecOps scenarios. According to their results, intelligent scanning approaches could be used for improving the process of security evaluation and automation of detection workflows.

Zhou et al. [8] analyzed cases of secret leakage in practice and identified different types of leaked credentials in both repositories and CICD logs by buiding a hybrid analytical system. In specialized software environments, Shi et al. [9] ran an empirical study on credential leakage in mini-application ecosystems and revealed extensive leaking caused by poor practices of application development.

Alfieri et al. [10] studied security and privacy awareness among GitHub users, while Lykousas and Patsakis [11] focused on developer behavior in terms of password selection. Krause et al. [12] explored methods of prevention and mitigation of credential exposure in software repositories and discovered the high frequency of accidental leakage, as well as the need for the detection of leaks and enhanced developer awareness to minimize the number of leakage cases.

Feng et al. [13] proposed an automatic tool for the detection of password leakage in publicly available GitHub repositories. The analysis conducted by the researchers identified many examples of leaked credentials and confirmed the efficiency of automated repository scanning as a technique for finding secrets. Basak et al. [14] assessed reporting performance of several detection tools and noted differences in the scope of their detection capability and the degree of reporting reliability, indicating that there is no method for the comprehensive identification of various types of secret disclosure. Rabzelj and Sedlar [15] analyzed the exploitation of leaked credentials in honeypots, which emphasizes the importance of rapid secret detection and fast response.

There have been some papers covering repository analysis, secret classification, risk assessment, and credential leakage analysis independently of each other. In contrast to that, repository analysis, CI/CD logs analysis, machine learning-based classification, hybrid decision fusion, risk scoring, and monitoring through dashboards have not received much attention as a set of interrelated aspects. This void calls for an integrated solution of the above mentioned components.

3 Proposed Framework

The system utilizes a hybrid detection technique to identify exposed credentials in source code repositories and CI/CD logs. The system includes three modules used to analyze repository files, runtime logs, and machine learning datasets feature extraction to provide input for hybrid fusion engine to generate the output of the detection

algorithm. Identified secrets are ranked based on risk scoring scheme and it provided visibility using a dashboard built using Streamlit. Our system architecture in shown in figure 1.

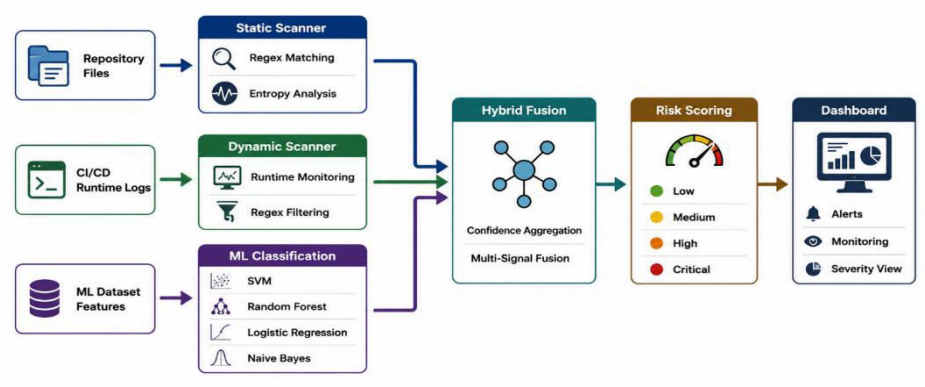

Fig. 1. Architecture of the Hybrid Secret Detection Framework

There are six components in the architecture: Static Scanner, Dynamic Scanner, Machine Learning Module, Hybrid Fusion Engine, Risk Scoring Module, and Dashboard Module. Static Scanner performs repository analysis using regular expressions and entropy analysis. Dynamic Scanner performs analysis of CI/CD logs for credential exposure at runtime. Machine Learning Module performs analysis of extracted features using several different techniques; the results generated in each of these three process are provided as independent inputs to hybrid fusion engine. Fusion Engine integrates detection outputs from the different modules to improve decision accuracy. Risk Scoring Module ranks detected secrets according to severity level ranging from Low to Critical. Dashboard Module provides monitoring information, secret distribution, severity statistics, and alert visualizations.

4 Methodology

Our implementation includes the dataset creation, static repository analysis, dynamic log analysis, machine learning classification, hybrid decision fusion, and dashboard based monitoring process. The workflow used for implementing the same is explained below.

4.1 Dataset Generation

A synthesized dataset was created consisting of both repository and CI/CD logs containing samples of secret and non-secret information. This dataset contained source code files, configuration files, environment files, JSON files, YAML files, and log files. Various kinds of credentials such as API keys, passwords, access tokens, cloud credentials, and private keys were included.

4.2 Static Repository Analysis

Repository files were checked against pre-defined regular expressions to extract known credential formats. Entropy analysis was performed next to discover random secrets which may not match any pre-defined patterns. The context-based filtering was carried out to reduce the number of false positives based on surrounding security-related keywords.

4.3 Dynamic Log Analysis

Log files from CI/CD process were analyzed to discover exposed credentials during build, test, and deploy operations. This involved checking for authentication credentials, tokens, passwords, and cloud credentials. Further analysis of suspicious strings was carried out through entropy and pattern matching algorithms.

4.4 Machine Learning Classification

The datasets created during the data preparation process are used to train and validate Logistic Regression, Random Forest, Support Vector Machine, and Naive Bayes classifiers. These classifiers classify if a given input feature is a real secret or a harmless feature. The outputs of these classifiers are given as an independent detection signal to the hybrid fusion engine.

4.5 Hybrid Fusion and Risk Scoring

Detection signals from static scanners, dynamic scanners, and machine learning classifiers are combined through a fusion process. When more than one component determines that a given candidate is a secret, then there is increased confidence on detection. The outcomes are ranked based on different severity levels which include low, medium, high, and critical.

4.6 Dashboard Monitoring

The Streamlit dashboard is used for displaying the detection signals such as severity distribution, secret type distribution, monitoring alerts, and statistics.

5 Results And Discussion

The proposed framework was tested on repository files and CI/CD log dataset which included both secret and non-secret samples. Evaluation was carried out using Accuracy, Precision, Recall and F1-score. The evaluation aimed to determine the effectiveness of static scanning, dynamic log analysis, machine learning classification and hybrid detection approaches.

5.1 Detection Performance

The Table 1 below shows the performance of each of the detection modules and hybrid framework.

Table 1. Performance of Detection Modules

Detection Module	Accuracy	Precision	Recall	F1-Score
Static Scanner	0.8619	0.3245	0.3154	0.3199
Dynamic Scanner	0.9042	1.0000	0.0695	0.1300
SVM	0.9461	0.8756	0.5548	0.6792
Hybrid Framework	0.8973	0.5008	0.6961	0.5825

The static scanner had moderate performance but was able to detect secrets matching the defined patterns. However, it had some extra false positives since some random strings were also found to have properties of secret exposures. The dynamic scanner had perfect precision since all the detected credentials were true positive exposures. However, the recall of the dynamic scanner was low since the analysis only took into account runtime logs.

The SVM classifier had the highest accuracy and F1-score (94.61%) and (67.92%) respectively. This shows the highest classification capabilities among the classifiers. Hybrid framework had the highest recall value (69.61%) showing that it can identify more exposures than other modules due to the use of combined evidence from detection modules.

The findings show that different methods have different capabilities of detecting different types of secret exposures.

5.2 Evaluation of Machine Learning Models

Four machine learning models were considered based on the prepared train and test data.The evaluation of each models is shown in Table 2.

Table 2. Performance Characteristics of Machine Learning Models

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.9369	0.9704	0.3993	0.5658

Random Forest	0.9387	0.9247	0.4401	0.5964
SVM	0.9461	0.8756	0.5548	0.6792
Naive Bayes	0.8288	0.3304	0.6459	0.4372

From all the tested models, the Support Vector Machine (SVM) proved to be the most advantageous in terms of precision and recall and showed the best performance in total. Logistic Regression and Random Forest showed high precision but were not able to find enough secret disclosures. Naive Bayes had better recall but a higher number of false positives, which decreased its precision.

Having considered this information, the SVM model was selected as the main classifier to be incorporated into the hybrid detection system.

5.3 Dashboard Analysis

Streamlit dashboard allows centralizing the analysis of detected credential exposures and facilitates security monitoring using the severity assessment, visualization of secret types and alerting mechanism.

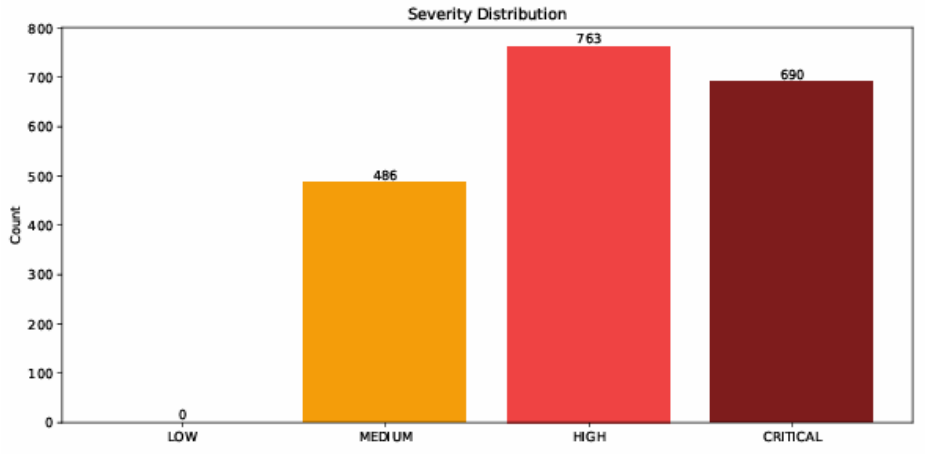

Fig. 2. Severity Distribution of Detected Secrets

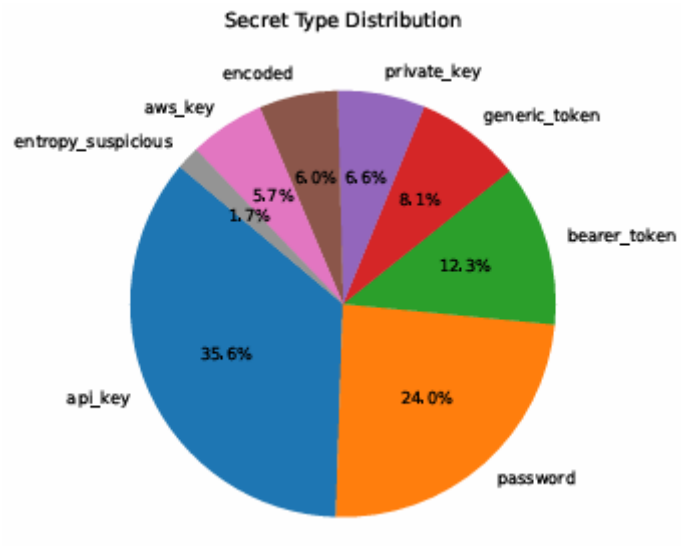

Fig. 3. Distribution of Detected Secret Types

In the course of evaluation, 13,551 inputs were analyzed and included 7,313 repository files and 6,238 CI/CD log entries. 1,939 detected potential credential exposures were found and consisted of 1,785 repository files and 154 log files with secrets, which gives an overall detection rate of 14.31%.

Figure 2 presents the severity distribution generated by the risk scoring module. Among all detected exposures, 486 findings had Medium severity, 763 had High severity and 690 had Critical severity. There were no Low severity detections. Many High and Critical severity findings imply that a great amount of detected credentials can create security risks in case of their exposure.

Figure 3 presents the distribution of secret types. 755 detections belonged to API keys, 509 – to passwords, 260 – to bearer tokens, 171 – to generic tokens, 141 – to private keys, 127 – to encoded credentials, and 122 – to AWS keys. There were also smaller categories of entropy-suspicious strings (37), tokens (30) and Base64-encoded secrets (14). Dominance of API keys and passwords indicates that authentication-related credentials remain the most often exposed artifacts among repository files and CI/CD log files.

The alert notifications were generated by the monitoring module for suspicious files and runtime log entries. 1,249 findings were considered high-risk exposures, whereas 690 detections caused blocking of the deployment because of Critical severity level. The alerts allow focusing on the most dangerous credential leaks and prioritization of remediation efforts.

The results obtained from the dashboard prove the utility of detection, risk scoring and visualization in the common workflow. In case of detection modules, they find exposed credentials, while the dashboard provides additional information and allows focusing on security findings.

6 Conclusion

The problem of credentials leakages from repositories and CI/CD logs represents a significant threat to security since leaked secrets may be used to gain access to various applications, cloud services, and even development environments. As a solution, a hybrid framework for credential leakage identification was created based on static analysis of repository content, dynamic analysis of CI/CD logs, classification with machine learning approach, and prioritization using the risk score. It was proven that this framework allows detecting different kinds of credentials both in repository files and CI/CD logs as well as improves the visibility due to using dashboard-based monitoring tools. The results have shown that each method used has its unique advantages, and combining their results makes the process more effective. Prioritizing findings by means of risk scoring and alerts allows selecting those which need more attention due to their possible impact. This framework achieved all stated goals as it identified credential exposures, included both rule-based and machine learning techniques, and allowed monitoring using severity analysis. Future work can focus on working with real github repositories and live CI/CD logs supporting other additional types of secrets.

References

1. M. Meli, M. R. McNiece, and B. Reaves, “How bad can it Git? Characterizing secret leakage in public GitHub repositories,” in Proc. Network and Distributed System Security Symposium (NDSS), 2019, doi: 10.14722/ndss.2019.23418.
2. N. Lykousas and C. Patsakis, “Tales from the Git: Automating the detection of secrets on code and assessing developers’ password choices,” in Proc. IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2023, pp. 68–75, doi: 10.1109/EuroSPW59978.2023.00013.
3. A. Saha, T. Denning, V. Srikumar, and S. K. Kasera, “Secrets in source code: Reducing false positives using machine learning,” in Proc. International Conference on Communication Systems and Networks (COMSNETS), 2020, pp. 168–175, doi: 10.1109/COMSNETS48256.2020.9027350.
4. S. K. Basak, T. Pardeshi, B. Reaves, and L. Williams, “RiskHarvester: A risk-based tool to prioritize secret removal efforts in software artifacts,” arXiv preprint arXiv:2502.01020, 2025.
5. C. Biringa and G. Kul, “Detecting hard-coded credentials in software repositories via LLMs,” Digital Threats: Research and Practice, vol. 6, no. 3, 2025, doi: 10.1145/3744756.
6. M. N. Rahman, S. Ahmed, Z. Wahab, S. M. Sohan, and R. Shahriyar, “Secret breach detection in source code with large language models,” arXiv preprint arXiv:2504.18784, 2025.
7. A. Abueshareik and A. Katbeh, “Empirical evaluation of AI-based security scanning and SonarQube in DevSecOps pipelines,” 2025.
8. J. Zhou, Z. Zhang, L. Ying, H. Chai, J. Cao, and H. Duan, “Hey, your secrets leaked! Detecting and characterizing secret leakage in the wild,” in Proc. IEEE Symposium on Security and Privacy, 2025, doi: 10.1109/SP61157.2025.00122.
9. Y. Shi et al., “The skeleton keys: A large scale analysis of credential leakage in mini-apps,” in Proc. Network and Distributed System Security Symposium (NDSS), 2025, doi: 10.14722/ndss.2025.230273.

10. C. Alfieri, J. Di Rocco, P. Inverardi, and P. T. Nguyen, “Exploring user privacy awareness on GitHub: An empirical study,” Empirical Software Engineering, vol. 29, no. 6, 2024, doi: 10.1007/s10664-024-10544-7.
11. N. Lykousas and C. Patsakis, “Decoding developer password patterns: A comparative analysis of password extraction and selection practices,” Computers & Security, vol. 145, 2024, doi: 10.1016/j.cose.2024.103974.
12. A. Krause, J. H. Klemmer, N. Huaman, D. Wermke, Y. Acar, and S. Fahl, “Committed by accident: Studying prevention and remediation strategies against secret leakage in source code repositories,” arXiv preprint arXiv:2211.06213, 2022.
13. R. Feng, Z. Yan, S. Peng, and Y. Zhang, “Automated detection of password leakage from public GitHub repositories,” in Proc. 44th International Conference on Software Engineering (ICSE), 2022, pp. 175–186, doi: 10.1145/3510003.3510150.
14. S. K. Basak, J. Cox, B. Reaves, and L. Williams, “A comparative study of software secrets reporting by secret detection tools,” in Proc. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2023, pp. 1–12, doi: 10.1109/ESEM56168.2023.10304853.
15. M. Rabzelj and U. Sedlar, “Beyond the leak: Analyzing the real-world exploitation of stolen credentials using honeypots,” Sensors, vol. 25, no. 12, 2025, doi: 10.3390/s25123676.