International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

Context-Aware Detection of Hard-Coded Credentials in Software Repositories using a Hybrid Approach

Yashshri Deshmukh
K J Somaiya Institute of Technology, Sion, Maharashtra

Madhura Phadke
Department of Computer Engineering, K.J. Somaiya Institute of Technology, Mumbai, India

Abstract - Hard-coded secrets including API keys, passwords, access tokens, and private keys still constitute a serious threat to security in software repositories. Secret detection methods usually utilize regular expressions and entropy heuristics that can effectively find secrets but have difficulty differentiating between genuine secrets and harmless pieces of code that have the same pattern as a secret. In this paper, we propose a context-aware hybrid method for the detection and classification of hard-coded credentials that combines pattern-based extraction of candidate secrets with semantics and machine learning. Candidate secrets are extracted with the help of regular expression patterns defined beforehand, whereas the source code context around these secrets is encoded using GPT-2 embeddings. The context representations obtained in this way are further analyzed by means of Multi-Layer Perceptron (MLP) through the process of binary credential detection and fine-grained credential classification. Our approach is experimentally evaluated based on SecretBench data. The achieved results show accuracy of 94.74%, precision of 94.93%, recall of 94.51% and F1-score of 94.72%. Our findings indicate that encoding of context representations with pattern-based candidate extraction is effective in identifying hard-coded credentials.

Software Security, Hard-Coded Credentials, Secret Detection, Source Code Analysis, Context-Aware Detection, GPT-2, Machine Learning.

1. INTRODUCTION

Software repositories have become an essential component in contemporary software development process that allows developers to collaborate, version control their software, and continuously deliver their products via services such as GitHub or GitLab. On the other hand, repositories containing source codes may store sensitive data such as passwords, API keys, access tokens, and private keys if these credentials were accidentally included directly in the source files. Such approach is related to the category of vulnerability known as CWE-798: Use of Hard-Coded Credentials and poses a security threat to software systems by making them vulnerable to unauthorized access, data leakage, and other risks (**b2?**). Modern repository hosting solutions such as GitHub have implemented functionality aimed at secret scanning in order to prevent any potential credential leakage (**b14?**).

Despite the existence of various secret detection systems, the traditional methods usually employ predetermined regular expression patterns and entropy based heuristics (**b5?**; **b13?**). Those approaches are computationally inexpensive and can be used in order to scan big repositories; however, they perform analysis of structure and properties of a string instead of considering the context in which it is located. As a result, non-credential information such as configuration parameters, identifiers, test data, and other code snippets might be incorrectly identified as credentials (**b6?**; **b8?**). The high level of false positives will lead to the developer's alert fatigue and decrease of trust towards automation security systems (**b4?**; **b11?**). Recent developments of LLMs have brought new prospects for introducing contextual data into the process of source-code analysis. GPT-2 and BERT models are able to produce representations that capture the connection between tokens and their context (**b1?**; **b3?**; **b12?**). Such representations could be useful for complementing the classical pattern-based analysis with additional contextual information required to distinguish potential credentials from other pieces of source-code content.

On the basis of this observation, this work proposes a novel approach to automated hard-coded credential detection that combines pattern-based candidate generation with contextual representation and machine learning. Initially, regular expressions are applied to extract potential credential candidates from the source code. Then, the context of each candidate is processed to obtain its semantic representations with the use of GPT-2 embeddings. Finally, these representations are used as input for the Multi-Layer

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

Perceptron (MLP) classifier responsible for credential detection and categorization. Thus, the proposed approach combines the efficiency of pattern-based candidate generation and contextual analysis.

Evaluation of the approach was performed with the use of samples obtained from the SecretBench dataset. The key objective of the evaluation is to determine the ability of the hybrid approach to separate credential-containing samples from the others and categorize the detected credentials. The experimental results show the high performance of pattern-based extraction combined with contextual representation.

The following are the main goals of the research:

- Identification of potential hard-coded credentials by means of predefined regular-expression patterns as an initial candidate generation method.
- Context incorporation into the process of representation with the help of GPT-2-based semantic embeddings.
- Classification and categorization of credential candidates with the use of MLP-based machine learning.
- Evaluation of the approach with the use of SecretBench samples and traditional classification metrics.

The rest of this paper describes existing hard-coded credential detection methods and the corresponding research gap, as well as the methodology, implementation, experiments, results, and conclusions.

2. RELATED STUDIES AND RESEARCH GAP

Methods used in hard-coded credential detection have evolved from traditional pattern matching approaches to more sophisticated approaches involving machine learning, program analysis, and semantic representations. Current approaches vary in their capability in analyzing the context, computational requirements, scalability, and detection performance. In this section, current approaches are reviewed and discussed, and the limitations leading to the development of the proposed approach are identified.

2.1 Current Detection Approaches

Current approaches for detecting hard-coded credentials can be generally categorized based on the kind of information that is used in detection.

1. **Heuristic and Pattern Matching Approaches:** Heuristic and pattern matching approaches mainly utilize regular expression and entropy heuristics to detect strings that have credentials properties. These approaches like Gitleaks and TruffleHog are highly efficient and scalable for scanning an entire repository, however, due to their poor contextual analysis, they generate a large amount of false positives (**b2?**; **b5?**; **b13?**).

2. **Machine Learning Approaches:** Machine learning approaches formulate credential detection as a classification problem and use various features such as characters distribution, string properties, and patterns. Although these approaches provide higher detection precision than traditional heuristics, they depend on feature engineering and may suffer from difficulties in detecting previously unseen credential patterns (**b6?**; **b15?**).

3. **Flow-aware and Contextual Analysis Approaches:** SEAGULL is a technique using static analysis in order to analyze credential interaction with variables, functions, and flows. These techniques can provide richer information context for detection; however, they require detailed analysis of the programs, which could incur high computational costs (**b4?**; **b11?**). AssetHarvester, on the other hand, uses relationship analysis between secrets and software assets to improve the contextual analysis, although at additional costs.

4. **LLM-Based Approaches:** Contextual representations generated by transformer-based architectures like BERT and GPT-2 enable recognition of the semantic connections in source-code text. These approaches are capable of offering better context awareness in the recognition of credentials in comparison with other code parts; however, their computation cost can be higher than that of pattern-based approaches (**b1?**; **b3?**; **b12?**).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

2.2 Comparison of Available Approaches

The comparative analysis of existing approaches to hard-coded credentials detection is presented in Table 1, where the main recognition method, data source used in the analysis, and main results of the study are mentioned. The proposed approach is included for comparison with pattern-based candidate extraction along with context awareness.

Comparative Study of Representative Methods for Detecting Secrets

Method	**Approach**	**Datasets**	**Key Findings**
SecretHunter (**b13?**)	Regex + Heuristics	Public Git Repositories	Efficient scanning with high recall but limited contextual evaluation can lead to false-positive detections.
SEAGULL (**b11?**)	Flow-aware static analysis	Open-source projects	Evaluates credential usage through program-flow information, but requires greater computational effort for large-scale analysis.
AssetHarvester (**b9?**)	Static Analysis	Software artifacts	Identifies relationships between secrets and software artifacts, improving contextual analysis at the cost of additional program analysis.
LLM-based (**b3?**)	Large Language Models	Code repositories	Provides semantic and contextual representations for credential detection, but requires more computational resources than simple pattern-based methods.
Proposed Method	Regex + GPT-2 + MLP	SecretBench	Combines pattern-based candidate extraction with contextual representations and machine learning to improve credential detection while reducing false positives.

2.3 Research Gap

Current approaches show the trade-off between computational efficiency and context-awareness. Pattern- and heuristic-based approaches are ideal for scanning due to their lightness, but their poor understanding of source-code context leads to false positives (**b5?**; **b13?**). Machine learning approaches can benefit from additional features, however, the performance of such techniques depends on the quality of the features extracted (**b6?**; **b15?**).

Flow-aware and LLM-based approaches bring more contextual data and allow to distinguish genuine credentials from source code fragments better. Still, they require more computational resources or more complicated program analysis, hence, such methods might not be scalable for large repositories (**b4?**; **b11?**; **b12?**). Thus, the current approaches either focus on efficiency of the candidates extraction or context-awareness, but not both at the same time.

In other words, the research gap is the need for a detection approach that will retain efficiency of pattern-based candidates extraction together with context-awareness. This work fills this gap through the development of a hybrid detection technique that combines regular-expression-based candidates extraction and GPT-2 context embedding with MLP classifier.

2.4 Research Contributions

To cover the limitations of the previous works and fill the research gap, the contributions of this work include:

- **Multi-stage detection pipeline:** Staged detection approach combining regular-expression-based candidates extraction and subsequent ML validation increases credential identification and reduces false positive detections.
- **Context Windowing Approach:** Definition of the context window around each candidate allows extracting relevant semantic and syntactic information from the source-code environment.
- **Hybrid MLP-LLM Architecture:** Combination of GPT-2 embeddings and MLP classifier helps to introduce context-awareness to the detection approach without using computationally expensive LLM classification stage.
- **Fine-grained Credential Classification:** Besides binary detection, the proposed approach allows to perform fine-grained categorization of the detected credentials.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

3. PROPOSED FRAMEWORK

The design of this system consists of three phases; candidate extraction through regex, contextual embedding generation through GPT-2, and credential classification through MLP. These phases are carried out in a step-by-step manner where candidates are extracted initially from the source code repository, and then analyzed based on contextual information. The overall flow of this proposed framework is illustrated in Fig. 1.

Conceptual Flow of the Proposed Secret Detection Framework

3.1 Components of the Framework

The proposed framework is based on a series of functional components that analyze potential credential candidates in successive order. As shown in Fig. 1, the process starts with the repository analysis and goes through candidate extraction, context generation, embedding creation, and finally to credential classification.

1. **Candidate Extraction:** At this stage, the source-code files are analyzed using predefined regular expressions. This component extracts strings which are likely to be hard-coded credentials as a preliminary stage before applying contextual analysis.

2. **Context Window Generation:** A context window of 200 characters around each extracted candidate is obtained from the source code. The context may contain useful information such as variable names, assignment, functions, comments, and configurations.

3. **Contextual Embedding Generation:** Each extracted context of the candidate is then processed using GPT-2 to create contextual embeddings. The created embeddings encode semantic and syntactic relationships in the source-code content.

4. **Credential Classification and Categorization:** The obtained embeddings are used by an MLP classifier for determining whether the candidate falls into the *Secret* or *Non-Secret* classes. If the secret class is selected, the candidates will be further classified into credential types like API keys, passwords, authentication tokens, and private keys.

Table 2 shows the main components of the proposed framework and their tasks.

Functional Components of the Proposed Framework

Module	Technology Component	Key Responsibility
Candidate Extraction	Regex Patterns	Identification of potential credential candidates
Context Extraction	200-Character Window	Collection of relevant surrounding source-code information
Feature Extraction	GPT-2	Generation of contextual representations
Classification	MLP Classifier	Binary detection and credential type categorization

3.2 Credential Detection Process

The first step in detecting credentials involves searching for candidates within the source-code repository through regex patterns. The aim of this step is to reduce the amount of source-code content to be processed by filtering out those strings which match the defined credential-related patterns.

Each candidate found is then analyzed along with its context in the source code. A window of 200 characters is used to collect extra data that can be helpful for differentiating the actual credential from the other string with a similar structure. The gathered context is then processed by GPT-2 to extract contextual embeddings.

The next step involves passing the embeddings to the MLP classifier to perform binary prediction. The classification here means predicting whether a candidate is *Secret* or *Non-Secret*. Those candidates that are classified as secrets are further categorized to find their specific credential type, e.g., API keys, passwords, authentication tokens, and private keys.

Thus, the output of the framework includes two types of information – the result of the detection of credentials and their categories.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

4. METHODOLOGY

The methodology employs the processing pipeline shown in Fig. 1. The description includes details of implementation and experiment procedures that are employed to process code examples, detect possible credentials, form contextual representations, and classify the credentials. The methodology under consideration suggests a multistage technique that combines pattern-based filtering and context-based analysis to separate real credentials from non-sensitive strings.

4.1 Experimental Data

The experiments were performed on the basis of SecretBench dataset, which is a publicly available benchmark dataset for the software secrets detection problem (**b7?**). The dataset consists of the candidate credentials collected from open-source GitHub projects and contains both real secret and non-secret examples in different programming languages and formats. The variety of the credential type and code example helps to verify context-aware secret detection approaches.

The main features of the dataset used in the work are shown in Table 3.

SecretBench Dataset Statistics

Dataset Property	**Value**
Total Credential Candidates	97,479
Verified Secrets	15,084
Programming Languages	49
File Formats	311
Regex Rules Used for Candidate Extraction	761
Dataset Split	Train / Validation / Test

The data was split into training, validation, and test datasets to facilitate the construction and evaluation of the detection model. The large size of the dataset and many candidate credentials together with different programming environments make the dataset appropriate for evaluating the suggested detection framework.

The data consists of samples that can be used to construct a binary classification of *Secret* and *Non-Secret* samples. Also, the secret samples are verified and include several types of credentials like API keys, passwords, authentication tokens, access secrets, and private keys.

4.2 Data Preprocessing

Source-code samples were preprocessed in order to have a consistent input while keeping necessary information for credentials detection. Text normalization, removing redundant whitespaces and line endings normalization were done during preprocessing. Only source-code tokens needed for detecting credentials and understanding their context were kept.

These operations will help to remove redundant information from the input while keeping semantic and syntactic information necessary for contextual analysis.

4.3 Candidates Detection and Context Construction

During the first stage of processing, candidate detection is done using regular-expression patterns predefined by the researcher. In total, 761 regular-expression patterns extracted from common secret detection rules were used for finding the strings which can be candidates for credentials (**b2?**; **b5?**). The patterns are directed on finding identifiers and values related to credentials like API keys, passwords, authentication tokens, access secrets, and private keys.

The goal of this stage is efficient reduction of the volume of code that needs further contextual analysis. Although regex-based detection can generate false positives, it is an effective way for the initial detection of sensitive strings (**b13?**).

Examples of credential-related patterns are presented in Table 4.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

Examples of Candidate Extraction Patterns

Credential Type	Example Identifier
API Key	AKIA...
Password	mypassword
Authentication Token	token123
Secret	secret123
Private Key	—–BEGIN...

A fixed context window of 200 characters is taken from the surrounding code snippet around each candidate. This context window size provides a trade-off between keeping enough contextual information while minimizing the computational cost of embedding generation.

The contextual information may include variable names, assignments, comments, function calls, and configuration options. Such information helps distinguish real credentials from dummy values, test data, and other non-sensitive strings (**b1?**; **b3?**). Taking into account both semantic and syntactic cues from the surrounding code, the contextual window provides additional information for downstream representation generation and classification tasks.

4.4 Context Representation Using GPT-2

Each extracted context is tokenized by the GPT-2 byte-level tokenizer and fed into the pretrained GPT-2 model (**b1?**; **b3?**). Token level contextual representations are generated by the last transformer layer of the model, encoding relationships among tokens of the surrounding source-code snippet.

A mean pooling operation is then applied to the generated token-level embeddings in order to generate a fixed-length representation that will be used in the classification step. Compared to conventional handcrafted features, contextual embeddings provide more information about the source-code context and help to better distinguish between credentials and non-sensitive strings (**b12?**).

4.5 Classification Strategy

Generated contextual embeddings by GPT-2 are given to a Multi-Layer Perceptron (MLP) classifier (**b1?**). The classifier learns patterns in the embedding space and makes predictions by classifying the input into *Secret* and *Non-Secret* classes (**b6?**; **b15?**).

The model is trained on labeled samples and fine-tuned with the Adam optimizer under the cross-entropy loss function. The architecture of the MLP classifier is described in Table 5.

MLP Classifier Configuration

Parameter	Configuration
Input Features	GPT-2 Embeddings
Hidden Layers	2
Activation Function	ReLU
Regularization	Dropout
Optimizer	Adam
Loss Function	Cross-Entropy Loss
Output Classes	Secret / Non-Secret

4.5.1 Mathematical Representation

Let C represent the contextual window extracted around a candidate credential. The GPT-2 model generates a contextual representation E, as defined in Equation (1).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

$$E = GPT2(C)$$

where E denotes the embedding vector representing semantic and syntactic information extracted from the surrounding source-code context.

The binary prediction produced by the classifier is expressed in Equation (2).

$$y_b = MLP(E)$$

where $y_b \in \{Secret, Non\text{-}Secret\}$.

For candidates predicted as secrets, an additional classification stage is used to assign a credential category, as represented in Equation (3).

$$y_m = MLP_{multi}(E)$$

where y_m represents categories such as API Key, Password, Authentication Token, Private Key, or Generic Secret.

The use of contextual embeddings representation allows the MLP classifier to generate predictions using information from the context of the candidate source code environment. This process is thus used as a filtering approach for reducing the number of false positives while maintaining accurate credential detection.

4.6 Credential Category Identification

Secrets that fall into the secrets category are then mapped to their corresponding credential categories. Such categories include API keys, passwords, authentication tokens, access secrets, and private keys (**b8?**; **b10?**).

This extra level of classification helps to have more information on detected credentials as well as to detect the particular type of sensitive information that is embedded into the source code.

4.7 Implementation of Framework

The suggested framework was developed in Python, using Hugging Face Transformers and PyTorch. For contextual embeddings' generation, GPT-2 was employed, while MLP was applied for classification purposes.

Data preparation, candidate identification through regexes, context windows' extraction, embedding generation, model training, predictions, and credential category identification are included into the implementation.

4.8 Model Training and Performance Assessment

For training the classification model, a dedicated training dataset was used, while the validation dataset was used for the model development stage. Model performance assessment was done via the independent test dataset.

Performance assessment was done based on Accuracy, Precision, Recall, F1-Score, and False Positive Rate.

5. EVALUATION METRICS

The performance of the suggested framework was analyzed based on the results from the confusion matrix. There are four possibilities for the prediction results: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

Table 6 shows the confusion matrix that is used to show the connection between the actual and predicted classes.

Confusion Matrix Components

Actual / Predicted	**Secret**	**Non-Secret**
Secret	TP	FN
Non-Secret	FP	TN

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

True Positive corresponds to a credential that has been classified as a secret, while True Negative denotes a non-secret string that has been accurately classified as such by the classifier. False Positive means that a benign string has been misclassified as a secret, while False Negative refers to a secret that has not been detected by the classifier.

On the basis of the above classification results, the following metrics were calculated to evaluate the proposed framework: Accuracy, Precision, Recall, F1-Score, and False Positive Rate. The Precision metric was used to determine the reliability of identified credentials and the reduction of false positives. Recall was used to measure the ability of the proposed framework to detect actual secrets, while F1-Score served as a compromise metric between Precision and Recall. Accuracy corresponded to the overall accuracy of the classification results, while False Positive Rate reflected the proportion of benign strings that had been misclassified as secrets.

6. EXPERIMENTAL RESULTS AND ANALYSIS

The proposed framework combining GPT-2 contextual embeddings and MLP classifier was evaluated in order to evaluate its efficiency in detection of hard-coded credentials in source-code repositories. The efficiency of the proposed framework was assessed in terms of classification performance measured through standard metrics like Accuracy, Precision, Recall, and F1-Score. The experimental results reflect the contribution of contextual information in detection of potential credentials among non-sensitive strings having similar characteristics.

6.1 Overall Classification Results

Classification performance of the proposed framework is shown in Table 7.

Performance Evaluation Metrics

Metric	Value (%)
Accuracy	94.74
Precision	94.93
Recall	94.51
F1-Score	94.72

Results have been obtained for consistent performance on all the stated metrics. For instance, a value of Precision of 94.93% shows that the framework does an excellent job in minimizing the occurrence of misclassifications. Similarly, the value of Recall of 94.51% shows the effectiveness of the model in identifying a significant number of credentials.

6.2 Comparison with Existing Detection Frameworks

Comparison of the existing detection method with the proposed one has been done qualitatively in Table 8. The comparison has shown the significance of context-based analysis in overcoming the drawbacks of those frameworks that work based on string pattern or entropy.

Comparison of Detection Approaches

Method	Characteristics
Regex-Based Detection	Provides fast repository scanning but may generate false-positive detections because of limited contextual understanding.
Entropy-Based Detection	Identifies high-entropy strings effectively but can incorrectly classify benign values as credentials.
Proposed Framework (GPT-2 + MLP)	Combines pattern-based candidate identification with contextual analysis to improve classification reliability and reduce false-positive alerts.

6.3 Interpretation of Results

Based on the experiment results, it is possible to conclude that contextual information is an important factor in credential recognition within software repositories. Since the model makes use of GPT-2 embeddings, it is able to encode not only syntactic but semantic information from the source-code context, which becomes another source of evidence to classify candidate strings.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

Moreover, the presented approach allows integrating computational efficiency in candidate identification and contextual classification processes. While the former relies on the language model and therefore requires processing the whole repository, the latter involves performing language modeling only on those parts of repository content, which have been recognized as candidates based on the usage of regex patterns.

Apart from binary detection, the approach provides information on the type of detected credentials, which could include API keys, passwords, authentication tokens and private keys. This information could be useful for developers and security specialists to have a better understanding of detected credentials and to prioritize actions aimed at their removal or replacement.

6.4 Possible Applications and Deployment

The proposed framework can be used in several cases related to software engineering and security. It could be used for security analysis of source-code repositories when the repositories are analyzed for credential exposure before releasing the software. As a result, it would be possible to avoid the problems related to the presence of exposed credentials and reduce the risk of unauthorized access because of hard-coded sensitive information.

Moreover, the framework could be used to enhance secure code review practices to detect credentials in the code by means of automation. Finally, this methodology could be used to automate the process of analyzing the presence of credentials in software in the context of CI/CD.

6.5 Limitations and Future Work

Even though the results presented here have shown promising classification capabilities, more validation can be done by running multiple experiments in order to get more statistical support as far as the reliability of the framework is concerned. It will be possible to do more work involving the confusion matrix, FPR, and analysis of each type of credential like API keys, passwords, authentication tokens, and private keys.

Future research will involve assessing the performance of the framework in respect of different types of credentials and the extent to which the framework can lower false positives.

7. CONCLUSION

A hybrid context-aware solution for the detection of hard-coded credentials in the source code of software repositories was presented in this paper. The proposed framework consists of two stages – regex-based candidate identification and GPT-2 contextual representations with an MLP-based classifier.

The multi-step structure of the framework allows identifying possible credential strings quickly and further analyzing them based on contextual information available in the vicinity of their occurrences in the source code.

The experimental findings confirm the importance of introducing semantic and contextual information into the process of credential detection. Having contextual information makes it possible to distinguish real credentials from non-credentials that have the same structural properties.

The combination of the pattern-based candidate filtering and contextual classification ensures a trade-off between efficiency and accuracy of candidate identification and prediction, respectively.

The proposed solution can be applied in various scenarios including repository security scanning, secure CI/CD practices, DevSecOps workflows, and automatic source-code security assessment. All these applications could lead to earlier detection of exposed credentials and mitigation of the risks of unintentional credential exposure.

Though the achieved results look promising, further evaluation based on multiple runs and statistical analysis is required for a more complete assessment of the proposed solution. Further research may also include studying the performance of the solution on individual credential categories and in other repository contexts.

ACKNOWLEDGMENT

The author would like to thank Madhura Phadke for valuable guidance and contribution to this research.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

REFERENCE

[1] M. Meli, M. R. McNiece, and B. Reaves, "How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories," in *Proc. NDSS*, 2019. doi: 10.14722/ndss.2019.23418.

[2] GitHub, "Secret Scanning: Detecting and Preventing Leaked Credentials in Repositories," GitHub Security Documentation, 2024.

[3] Z. Y. Ding, B. Khakshoor, J. Paglierani, and M. Rajpal, "Sniffing for Codebase Secret Leaks with Known Production Secrets in Industry," *arXiv preprint arXiv:2005.09757*, 2020.

[4] E. Wen, J. Wang, and J. Dietrich, "SecretHunter: A Large-Scale Secret Scanner for Public Git Repositories," in *Proc. TrustCom*, pp. 123–130, 2022. doi: 10.1109/TrustCom56396.2022.00028.

[5] V. Srikumar, A. Saha, T. Denning, and S. K. Kasera, "Secrets in Source Code: Reducing False Positives Using Machine Learning," in *Proc. COMSNETS*, 2020. doi: 10.1109/COMSNETS48256.2020.9027350.

[6] S. K. Basak, J. Cox, B. Reaves, and L. Williams, "A Comparative Study of Software Secrets Reporting by Secret Detection Tools," *arXiv preprint arXiv:2301.09752*, 2023.

[7] R. Han, H. Gong, S. Ma, J. Li, C. Xu, E. Bertino, S. Nepal, Z. Ma, and J. Ma, "A Credential Usage Study: Flow-Aware Leakage Detection in Open-Source Projects," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 722–734, 2024. doi: 10.1109/TIFS.2023.3326985.

[8] R. Han, H. Gong, S. Ma, J. Li, C. Xu, E. Bertino, S. Nepal, Z. Ma, and J. Ma, "SEAGULL: A Flow-Aware Framework for Credential Leakage Detection in Open-Source Projects," in *Proc. ICSE*, 2023.

[9] C. Biringa and G. Kul, "Detecting Hard-Coded Credentials in Software Repositories via LLMs," *arXiv preprint arXiv:2506.13090*, 2025.

[10] M. N. Rahman, S. Ahmed, Z. Wahab, S. M. Sohan, and R. Shahriyar, "Secret Breach Detection in Source Code with Large Language Models," in *2025 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Honolulu, HI, USA, 2025, pp. 207–217. doi: 10.1109/ESEM64174.2025.00021.

[11] Y. Huang, Y. Li, W. Wu, J. Zhang, and M. R. Lyu, "Your Code Secret Belongs to Me: Neural Code Completion Tools Can Memorize Hard-Coded Credentials," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 2515–2537, 2024. doi: 10.1145/3660818.

[12] GitGuardian, "State of Secrets Sprawl Report," GitGuardian Security Report, 2023.

[13] S. K. Basak, K. V. English, K. Ogura, V. Kambara, B. Reaves, and L. Williams, "AssetHarvester: A Static Analysis Tool for Detecting Secret-Asset Pairs in Software Artifacts," in *Proc. ICSE*, pp. 1–13, 2025. doi: 10.1109/ICSE55347.2025.00067.

[14] S. K. Basak, L. Neil, B. Reaves, and L. Williams, "SecretBench: A Dataset of Software Secrets," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, Melbourne, Australia, 2023, pp. 347–351. doi: 10.1109/MSR59073.2023.00053.

[15] R. Feng, Z. Yan, S. Peng, and Y. Zhang, "Automated Detection of Password Leakage from Public GitHub Repositories," in *Proc. ICSE*, pp. 175–186, 2022. doi: 10.1145/3510003.3510150.