A Large-Scale Empirical Study of Secret Keys Leakage in Hugging Face Spaces

Shaoxuan Yun
shaoxuanyun@bupt.edu.cn
Beijing University of Posts and Telecommunications
Beijing, China

Yuchao Zhang*
yczhang@bupt.edu.cn
Beijing University of Posts and Telecommunications
Beijing, China

Zhikun Shi
zhikunshi@bupt.edu.cn
Beijing University of Posts and Telecommunications
Beijing, China

Liu Wang
liuwang@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

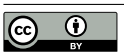
Yi Wang†
yiwang@bupt.edu.cn
Beijing University of Posts and Telecommunications
Beijing, China

Yu Bai
by@bupt.edu.cn
Beijing University of Posts and Telecommunications
Beijing, China

Abstract

Hugging Face Spaces (Spaces) has become a leading platform for hosting AI applications, offering developers seamless integration of Git-based repositories and out-of-the-box web service deployment. However, as its adoption continues to expand, security concerns have come to light. Recent reports have pointed to the issue of accidental exposure of sensitive information, such as API tokens and database credentials, within Spaces-hosted code. Despite these concerns, the research community still lacks a comprehensive understanding of the scope and impact of these security risks. To bridge this gap, we present the first large-scale and systematic empirical study to quantify the extent of secret key leakage in Spaces. We begin by compiling a comprehensive dataset of 313,647 public Spaces repositories created between March 2022 and December 2024. To detect exposed secret keys, we introduce *Secret Reviewer*, an advanced framework that combines static analysis and Large Language Model (LLM)-assisted detection to identify leaked credentials. Applying Secret Reviewer, we identified 9,149 with secret keys leakage vulnerabilities and 11,557 unique keys—76% of which from leading AI service providers such as OpenAI and Groq. Our findings indicate that 30% of leaks were embedded in commit histories, and over 1,000 non-code files contained sensitive data. Further validation revealed that 30% of detected keys remained active, including 140 valid database credentials and 50% of access tokens with write permissions, underscoring significant security risks. Our study sheds light on critical security challenges in AI platform ecosystems and advocates for stronger security practices to mitigate these risks.

*Corresponding author.
†Also with Shandong Key Laboratory of Advanced Computing, Inspur Computer Technology Co., Ltd.



ICSE '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2025-3/2026/04
https://doi.org/10.1145/3744916.3773117

CCS Concepts

• **Security and privacy → Software security engineering**.

Keywords

Secret key leakage, Security, Privacy, Hugging Face Spaces

ACM Reference Format:
Shaoxuan Yun, Yuchao Zhang, Zhikun Shi, Liu Wang, Yi Wang, and Yu Bai. 2026. A Large-Scale Empirical Study of Secret Keys Leakage in Hugging Face Spaces. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3744916.3773117

1 Introduction

With the rapid advancement of artificial intelligence (AI) applications, the demand for convenient and scalable deployment solutions has increased significantly [40]. Hugging Face Spaces (hereafter referred to as Spaces) has emerged as a leading platform for hosting AI applications, providing developers with an efficient and user-friendly environment for quickly deploying and sharing AI models. By leveraging Git-based code repositories and built-in web service deployment, Spaces enables developers to build and showcase AI applications at minimal cost. As of now, Spaces has hosted more than 300,000 public AI web applications, forming a highly efficient AI service pipeline (as illustrated in Figure 1). Through standardized API interfaces, developers can seamlessly integrate third-party model inference services to perform AI tasks such as text generation and image recognition, without the need for complex computing resource management or extensive code adaptation.

However, this convenience also introduces serious security risks. Recent reports [9, 35] have revealed that numerous API access tokens, database credentials, and other secret key information have been inadvertently exposed in Spaces repositories, leading to unauthorized API abuse, user data breaches, and even malicious code execution [20]. Recent industry analyses [31] also documented real-world cyberattacks that exploited leaked API keys, underscoring the growing severity of this threat. Despite these findings, systematic research on secret key leakage in Spaces remains scarce. Compared to traditional platforms, secret key leakage in Spaces may present several distinct security challenges: (1) Since Spaces focuses on AI

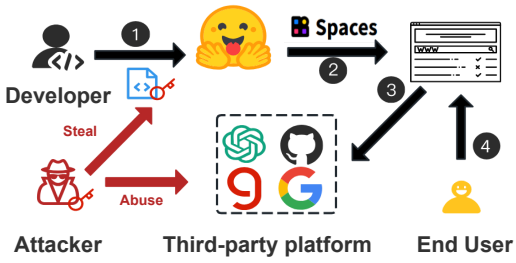


Figure 1: When developers use Spaces, 1. they first create a repository and upload their code to the platform. 2. Subsequently, Spaces deploys and distributes the code based on its contents. 3. In addition to accessing the hosted large language models (LLMs) through API calls, developers can also leverage both Hugging Face and third-party cloud inference APIs. 4. Finally, end-users can interact with various AI applications via web interfaces . However, if user inadvertently hardcodes secret keys in a public Space, attackers can exploit them, leading to API abuse and privacy breaches.

applications and provides built-in web services, the volume, types, and file locations of leaked secret keys may differ significantly from those observed on traditional platforms such as GitHub and GitLab. (2) The rapid expansion of the Spaces ecosystem may have led to inadequate secret key detection and protection mechanisms in its early stages, resulting in variations in the severity of secret key leakage across repositories created at different times. (3) Unlike traditional software, AI applications typically rely on external platforms for services such as model inference and external data access demands [30]. In such cases, access to external platforms invariably requires the use of secret keys (e.g., API keys or authentication tokens), which could further amplify the security risks [22]. However, currently there is no clear statistical evidence to illustrate the landscape and severity of security threats posed by secret key leakage in AI applications. Therefore, a large-scale empirical investigation of Spaces repositories is crucial to understanding and mitigating these security threats.

Our Work. This study conducts the first systematic empirical analysis of secret key leakage in Spaces repositories. Specifically, we investigate Space secret key leakage from three perspectives: prevalence, characteristics, and security risks. Our study addresses the following three research questions (RQs), with key findings summarized below.

RQ1: (Prevalence) *How widespread is secret key leakage in repositories, and which platforms are most affected?* To assess the extent of secret key leakage, we first leveraged the Hugging Face API (HFApi) to retrieve a complete list of all ***public*** Spaces repositories created between March 2022 and December 2024, collecting the data in batches based on repository creation time. Subsequently, we cloned a total of 313,647 public Spaces repositories using Git commands. Our analysis revealed that 9,149 repositories contained secret key leaks, exposing 11,557 unique keys in total. Notably, 76% of these leaked secret keys originated from AI model service providers such as OpenAI and Groq, a significantly higher proportion than previously reported in studies on GitHub repositories [25]. Additionally, we identified over 2,000 leaked keys associated with platforms such as GitHub, database storage services, and other cloud-based tools, further confirming that secret key leakage in the Spaces ecosystem has reached a substantial scale.

RQ2: (Characteristics) *Where are leaked secret keys primarily located, and how has secret key leakage evolved over time?* Our findings indicate that nearly 40% of leaked secret keys are found in commit histories, suggesting that while developers may attempt to remove hardcoded secret keys from their code, remnants in version control records continue to pose security risks. Moreover, almost 50% of secret key leaks were found directly within source code files. In addition, sensitive information was also detected in over 1,000 other file types, such as configuration files and plaintext documents. This suggests that besides direct hardcoding, developers' unintentional uploads of sensitive files further exacerbate security vulnerabilities. Examining leakage trends over time, we observed a gradual improvement in the prevalence of secret key leaks in newly created repositories since March 2023. However, despite this progress, more than 2% of newly created repositories each month still contain exposed secret keys, highlighting the ongoing challenges developers face in managing and securing sensitive credentials.

RQ3: (Security Risks) *How many leaked secret keys remain active, and what are the potential security implications?* Our analysis revealed that nearly 30% of the leaked secret keys found in repositories or commit histories were still valid, indicating that simply removing hardcoded secret keys is insufficient, developers should proactively rotate secret keys to mitigate security risks. Furthermore, we detected over 1,600 valid API tokens from AI model providers, among which 116 tokens still had available credit. If exploited by attackers, these leaks could result in substantial financial losses for affected users. Additionally, 140 leaked database connection credentials were found to be still active, posing risks such as data tampering, deletion, or misuse. More alarmingly, over 50% of active access tokens associated with code hosting platforms had write permissions, meaning that attackers could not only access and read the code but also modify or delete projects. This poses a critical threat to software supply chain security, further emphasizing the need for stringent secret key management practices.

Contributions. In summary, this paper presents the following key contributions:

- **First Systematic Understanding:** We present the first systematic study on secret key leakage in Hugging Face Spaces, offering a comprehensive analysis and quantification of the leaked secret keys. Our study provides a high-level overview of the prevalence, characteristics and security risks associated with secret key leakage in this ecosystem.
- **LLM-Powered Secret Key Detection:** We design and implement *Secret Reviewer*, a framework utilizing LLM to detect leaked secret keys. By incorporating Chain-of-Thought (CoT) prompting, our approach enhances the accuracy of identifying repositories containing exposed credentials. Applying this tool, we systematically assess the prevalence and distribution of secret key leakage in Spaces.
- **Empirical Analysis:** We conduct an in-depth analysis of secret key leakage incidents, including examining the locations of leaked secret keys, their distribution over time, and the affected

platforms, offering valuable insights into the characteristics of secret key exposure.
- **Security Risk Assessment:** Beyond identifying leaked secret keys, we establish a validation pipeline to assess usability and potential security risks. Our findings highlight critical security threats, including the persistence of active credentials and their implications for AI platforms and software supply chain security.

Availability. The source code of *Secret Reviewer* and relevant artifacts are publicly available [1].

2 Background and Related Work

2.1 Hugging Face Space

Recent advances in AI and machine learning have led to significant progress in areas such as natural language processing and computer vision. Hugging Face, known as the "GitHub of AI," has become a leading open-source platform offering models, datasets, and AI application hosting services. In addition to model hosting, Hugging Face provides Spaces, a platform for sharing and deploying ML demo applications [10]. Spaces are Git-based repositories that include code, dependencies, and configurations, and are categorized into three types [11]: 1) **Gradio** and **Streamlit** Spaces for interactive applications, 2) **Docker Spaces** for custom runtime environments, 3) **Static HTML Spaces** for static web pages. As a core component of the Hugging Face ecosystem, Spaces facilitate the open sharing of AI applications. As an integral part of the Hugging Face ecosystem, Spaces play a crucial role in promoting the open sharing of AI applications. However, their public accessibility also exposes source code and configuration files, which may inadvertently leak sensitive credentials such as API tokens or access keys. Thus, investigating security vulnerabilities in Spaces is crucial to improving the overall security of ML application platforms.

2.2 LLM for SE

LLMs are a class of deep learning models trained on large-scale text and image datasets. With advanced natural language understanding and generation capabilities, LLMs enable fluent human-AI interaction [14]. Their applications span various domains, including natural language processing (NLP), text and image generation, significantly improving AI-driven interaction and content creation [18, 42].

Code analysis is the process of analyzing code artifacts, such as source code and configuration files, to detect errors and potential security vulnerabilities. Recent research has explored the application of LLMs to various code analysis tasks, including code structure understanding [28] and program flow analysis [39]. These advancements have demonstrated LLMs' potential in enhancing automated code review and software quality assessment.

LLM frameworks provide software infrastructure for developing and deploying LLM-based applications. These frameworks offer APIs for model interaction and tools for application management and deployment, facilitating the integration of LLMs into software engineering workflows. Notable LLM frameworks include LangChain [21], LlamaIndex [23] and DeepSpeed [5].

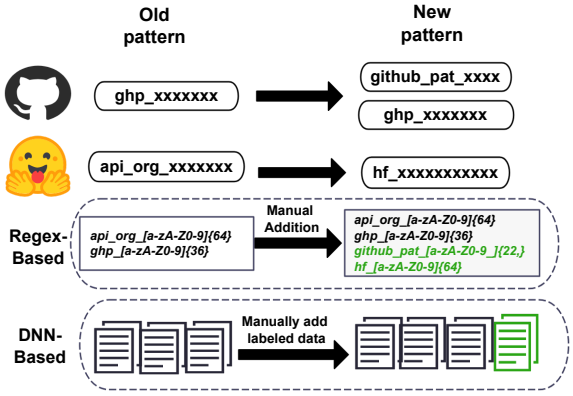


Figure 2: Evolution of Secret Key Patterns and Detection Adaptation Strategies.

2.3 Secret Key Detection

Secret key detection is a security technique designed to prevent unauthorized access and data breaches by automatically identifying exposed API keys, access tokens, and credentials in code, configuration files, or logs [2]. Studies have shown that over 30% of developers have experienced secret key leakage, primarily due to usability issues, lack of training, and insufficient documentation [19, 20]. In addition, Basak et al. [4] summarized 27 practical challenges developers face in secret key management. Rahman et al. [33] further pointed out that organizational misalignments and unclear responsibilities often hinder effective remediation. These studies highlight the importance of secret key detection in enhancing security throughout the software development process.

In existing research, secret key detection methods can be broadly categorized into three types: 1) Regular expression-based methods [16, 25, 29, 36], which rely on predefined regular expressions and entropy-based features to identify secret keys. 2) Deep neural network (DNN)-based methods [24, 34], which use trained deep learning models for secret key matching. 3) Hybrid method [12, 32], which first uses regular expressions to extract potential secret key candidates from files, followed by deep learning models for refined detection. While all types of methods have demonstrated strong capabilities, they remain manual-dependent in real-world scenarios.

First, regular expression-based methods require substantial human effort to maintain rule sets. For instance, as illustrated in Figure 2, the formats of secret keys used by GitHub and Hugging Face have evolved over time. GitHub's Personal Access Tokens (PATs) use different prefix identifiers, such as `ghp_` and `github_pat_`, while Hugging Face's secret key format has also changed from the earlier `api_org_` to the new standard prefix `hf_` [8]. This implies that any expansion or modification of secret key formats necessitates manual updates to the regular expressions to maintain detection accuracy. Similarly, DNN-based methods also require adaptation to such diversity and changes, relying on high-quality labeled data for supervised learning [3]. However, constructing a high-quality training dataset demands significant human effort, further increasing the cost and maintenance complexity [7]. Moreover, a hybrid method that combines regular expressions with deep learning may suffer from both challenges. To address these issues, this paper

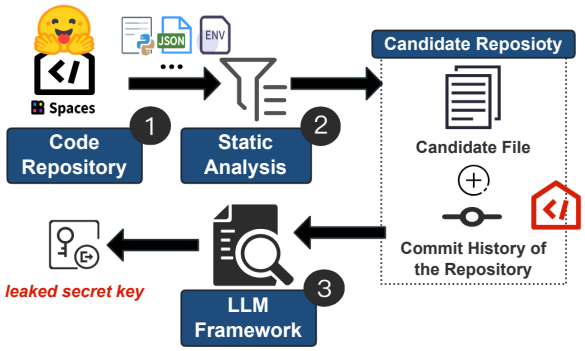


Figure 3: Different phases of the Secret Reviewer

proposes Secret Reviewer, a secret key detection framework based on LLMs. Secret Reviewer eliminates the need for manually written regular expressions or labeled datasets. Instead, it leverages LLMs' extensive pre-trained data and strong contextual reasoning capabilities to enable highly automated secret key detection, thereby enhancing code security more effectively.

3 Approach

In this section, we describe our approach for detecting secret keys. We define secret keys as encryption keys, API credentials, and database connection strings. These sensitive pieces of information, if leaked, can lead to severe security risks, including data tampering, unauthorized access to protected resources, and API abuse. Before discussing the details of the following sections, we will briefly outline the overall strategy. The issues in secret key detection include: 1) Performing secret key detection on every file clearly requires significant computational resources. Therefore, an effective filtering mechanism is necessary to first screen the content of the files before performing precise secret key detection. 2) As discussed in Section 2, the format of secret keys may change as the platform evolves. If we directly use tools from previous works, such as those based on regular expression matching or DNN, these methods often require continuous updates to regular expressions or high-quality labeled data. This process inevitably involves significant human intervention, thereby increasing development costs.

To address these issues, we propose ***Secret Reviewer***, a rigorous automated framework that integrates static code analysis and LLMs to identify leaked secret keys in Spaces repositories. As shown in Figure 3, the framework operates through three phases: (1) Data Collection, (2) Static Analysis, (3) LLM-Based Extraction. Specifically, we first leverage HFApi and git tools to comprehensively collect publicly available Spaces. Next, we conduct static analysis on public Spaces by scanning sensitive files such as `.py` and `.json`, filtering out potentially hardcoded secret keys. In addition, we also scan commit histories to identify secret keys that were previously exposed in plaintext but later removed, thereby ensuring a more comprehensive security assessment. Finally, to extract the exact secret keys from the candidate files and commit records identified during the static analysis phase, we propose an automated extraction method based on LLMs. The candidate information is passed to the LLM in the form of text strings, and through advanced prompt engineering techniques, the model is guided to effectively identify the leaked secret keys.

3.1 Phase1: Data Collection

In this section, we describe our data collection process. Unlike GitHub, Spaces does not offer an API for directly searching file contents. As a result, we are unable to retrieve sensitive information from repositories via remote queries and must download entire repositories for offline analysis. To address this, we leveraged the Hugging Face API (HFApi) to batch-fetch a complete list of all publicly available Spaces repositories created between March 2022 and December 2024. Subsequently, we cloned a total of **313,647** Spaces repositories using Git commands, accumulating over 2 TB of data (including `.git` directories). To the best of our knowledge, this dataset represents the largest collection of Spaces repositories to date. Although this approach increases storage, computation, and network costs during data collection, it greatly enhances data integrity and analytical depth in two key ways: (1) Comprehensive Code Search — Full repository downloads enable efficient local analysis of all code files, bypassing API limitations; (2) Version History Tracking — Retaining `.git` directories allows tracing of historical commits to uncover previously exposed sensitive information.

3.2 Phase2: Static Analysis

In this section, we present our approach for identifying repositories that are likely to contain secret key leaks. Secret keys are typically generated as random strings using specific algorithms and encoding formats. Although manual inspection is the most intuitive way to detect secret keys, it is impractical at the scale of over 300,000 repositories. This motivates us to develop a static filtering mechanism to pre-select files and repositories that are more likely to contain exposed secret keys, which can then be further analyzed by LLMs. Therefore, inspired by previous studies [25, 34, 36], we employ a dual-filtering approach to identify repositories that potentially contain secret keys:

Shannon Entropy-Based Randomness Detection: First, to quantify the randomness of a string in a file or commit history, we use Shannon entropy as a metric. In information theory, Shannon entropy [37] measures the uncertainty or information content of a random variable. When applied to strings, a higher entropy indicates more evenly distributed characters and stronger randomness, which are typical traits of secret keys. Conversely, a lower entropy suggests weaker randomness, making it less likely that the string is a secret key. Therefore, we compute the Shannon entropy of each line in every file and commit history to evaluate its likelihood of containing a secret key. If the Shannon entropy of a line exceeds a predefined threshold, we consider that it potentially contains a secret key. We set the threshold to 5, guided by prior work [38] and confirmed through our own pilot experiments.

Character Sequence Pattern-Based Filtering: Second, while Shannon entropy effectively quantifies the randomness of a string, relying solely on entropy values may lead to a high false positive rate in certain cases, such as HTTP request parameters or hash values. To enhance filtering accuracy, we further employ character sequence pattern analysis. As highlighted in [6, 25], truly random secret keys typically do not contain recognizable character patterns. Therefore, we adopt the three character pattern detection methods proposed in [25], which identify strings containing specific structured patterns, including: a) AAAA-type patterns (e.g., 1111).

b) ABCD-increasing sequences (e.g., abcd). c) DCBA-decreasing sequences (e.g., zyxw). If a string matches any of these predefined patterns, it indicates a structured character arrangement rather than true randomness. Consequently, we do not consider such strings as potential secret keys.

Using the proposed dual-filtering approach, we analyzed both source code and commit histories, identifying 12,819 repositories with potential secret leaks. These candidates were then examined by an LLM for deeper analysis. The dual-filtering process effectively reduces noise and focuses the LLM on high-probability cases, improving both efficiency and detection accuracy.

3.3 Phase3: LLM-Based Extraction

In this section, we introduce an LLM-based approach for secret key extraction. Compared to traditional methods (see Section 2.3), the LLM-based approach demonstrates several significant advantages: **1) Extensive Secret Key Pattern Learning:** Through pretraining, LLMs can learn a wide range of secret key formats across various platforms, including legacy schemes, enabling efficient detection without additional fine-tuning. **2) Dynamic Adaptability:** LLMs can incorporate external databases and continuously updated data sources, allowing them to automatically adapt to emerging secret key patterns without the need for manual rule maintenance. **3) Contextual Reasoning Ability:** LLMs can leverage code context to infer and identify unseen secret key patterns, even if they do not exist in external databases or training data. Therefore, we implemented an LLM-Based secret key extraction module using LangChain [21]. This module assists in secret key auditing by leveraging the filtered information obtained in Phase 2.

```
(1) Indicate the amount of received text and determine whether it corresponds to commit information.
(2) It is important to note that commit information may contain one or multiple entries, separated by a large number of underscores.
(3) Determine whether there is a Secret Key leakage based on the text. If none are found, exit immediately?
(4) If leakage is detected, extract all leaked Secret Keys and report the number of occurrences (some Secret key may be split into two concatenated strings)?
(5) Next, based on the context, determine which real platform or company the leaked Secret Key belongs to (without displaying the context)?
(6) Finally, based on the context, determine the specific functionality that this Secret Key is used for (without displaying the context)?
```

Figure 4: CoT Reasoning for Secret Key Detection

Specifically, we extract the suspected strings and their surrounding 20 lines of context (following [32]) from the candidate files and commits identified in Phase 2, and input them into the LLM as plain text for further analysis. Additionally, to enhance the LLM's response quality, we integrate a Retrieval-Augmented Generation (RAG) mechanism. We collect API-related documentation from various service providers and store it in a vector database, enabling the LLM to generate more informed responses. To further improve reasoning performance, we adopt Chain-of-Thought (CoT) prompting, which has been shown to significantly enhance LLMs' reasoning capabilities and output accuracy [41]. We designed and implemented a series of task-specific reasoning strategies for secret key detection (see Figure 4). In this study, we use GLM-4-plus as the inference model. We set the temperature to 0.0 to ensure deterministic outputs [32]. We allow up to 128k tokens to accommodate long-format secrets (e.g., RSA keys) and reduce the risk of missed detections [32].

Table 1: Comparison of Secret Reviewer(SR), Secret Reviewer without CoT (SR-WCoT), and Truffle Hog (TH)

Metric		SR	SR-WCoT	TH
Leaked Space	Precision	93.5%	85.2%	90.1%
	Recall	100%	97.2%	100%
	FPR	0.53%	1.29%	0.86%
Leaked Secret keys	ACC	100%	91.4%	95.7%
Detection Time	Avg Time	6.9s	6.2s	3.4s

3.4 Effectiveness Evaluation of Secret Reviewer

In this section, we evaluate the effectiveness of our proposed framework in a fully automated setting, without manual intervention. For our experiments, we randomly selected 1,000 code repositories from collected Spaces as the evaluation dataset. To establish a ground truth, we invited 10 human reviewers to independently examine these repositories for potential secret key leaks. The manual inspection revealed that 73 repositories contained leaked secret keys, involving a total of 94 exposed secret keys. Additionally, we selected TruffleHog[1] [36] as the baseline algorithm. We also evaluated a variant of Secret Reviewer without CoT prompting to assess the effectiveness of CoT. We compared the performance of Secret Reviewer across three different dimensions: 1) Leaked Space Identification: Evaluating whether different methods can accurately detect Spaces containing leaked secret keys. We use Precision, Recall, and False Positive Rate (FPR) as evaluation metrics. 2) Leaked Secret Key Detection: Assessing the effectiveness of different methods in identifying leaked secret keys in Space repository. 3) Average Detection Time: Measuring the average time required by each method to analyze a single repository. The experimental results are summarized in Table 1, demonstrating the effectiveness of Secret Reviewer in secret key leakage detection.

The experimental results demonstrate that Secret Reviewer outperforms TruffleHog in detecting secret key leaks. When detecting the leak space, Secret Reviewer achieves an accuracy of 93.5% with a false positive rate of only 0.53%, whereas TruffleHog exhibits a higher false positive rate, making it less effective in this aspect. In addition, both methods achieved a 100% recall rate, meaning that Secret Reviewer can achieve the same results as methods relying on regular expressions, without the need for manual regular expression creation. Besides, in the secret key leakage detection experiment, Secret Reviewer achieved an accuracy rate of 100%, surpassing TruffleHog's 95.7%. This further confirms the performance of Secret Reviewer. Nevertheless, due to the higher computational overhead of LLMs, Secret Reviewer requires a longer detection time compared to TruffleHog. Despite this, its more precise contextual understanding significantly reduces false positives while identifying a greater number of leaked secret keys, ultimately leading to superior detection quality. Moreover, ablation experiments show that without CoT prompting, the LLM exhibits a higher false positive rate of 1.29%, indicating that CoT effectively reduces false positives. In addition, the precision and accuracy metrics in both leaked space and secret key detection demonstrate that CoT prompting significantly enhances the detection performance of Secret Reviewer.

[1]We selected TruffleHog as our baseline because it is the only method officially integrated with Hugging Face Spaces. Therefore, we consider it the state-of-the-art solution for secret key detection in this context.

4 RQ1: Prevalence

4.1 Methodology

Using the approach described in Section 3, we conducted an in-depth analysis of the prevalence of secret key leaks in Spaces. We first provide an overview of the overall secret key leakage in Spaces, followed by a detailed examination of the prevalence of secret key leaks across different platforms. Furthermore, to better illustrate the situation of secret key leakage in Spaces, based on previous study[2, 3, 25] and the characteristics of the Space platform, we categorize keys into the following four major types:

- ***AI Model Provider*** (e.g., OpenAI) – These secret keys are used to authenticate API requests for AI model services, granting access to functionalities such as text generation, image synthesis, and model fine-tuning.
- ***Code Hosting Platform*** (e.g., GitHub) – These secret keys provide access to version control and repository management services, enabling operations such as code cloning, pushing, and repository configuration.
- ***Storage Service*** (e.g., MongoDB) – These secret keys grant access to cloud storage or database management systems, facilitating data retrieval, modification, restore, and administrative control.
- ***Other*** (e.g., SSH, Consumer Keys, OAuth Credentials) – These secret keys includes cryptographic keys and authentication tokens used for secure communication, encryption, and third-party integrations with applications.

4.2 Results

4.2.1 Secret Key Leakage Overview. We applied Secret Reviewer to 313,647 Spaces repositories to detect secret key leakage. The result revealed that 9,149 repositories (approximately 3% of the total) contained leaked secret keys, with a total of 11,557 unique secret keys identified. On average, each affected repository contained 1.26 secret keys. A detailed examination of the data shows that the vast majority (95.6%) of the repositories with leakage issues contained only a single leaked secret key, while the most severely impacted repository exposed six unique secret keys. From the developer perspective, our study involved 6,909 distinct developers. Among them, 3,008 developers (43.5%) were associated with only a single space repository. The presence of secret key leaks across thousands of repositories and developers suggests that secret key management remains a critical security concern in the Spaces ecosystem.

4.2.2 Distribution of Leaked Secret Keys by Platform. We further categorized the leaked secret keys based on different platforms and classified the types of secret keys following the existing study [25] (see Table 2). Our analysis reveals that API key leaks from AI service providers are the most prevalent. Specifically, we identified over 8,000 leaked secret keys from AI service providers, with OpenAI accounting for the highest number (6,959 leaks). This proportion is significantly higher than previously reported in GitHub-based studies [25]. Further investigation shows that these API keys are predominantly used for cloud-based model inference and computational resources, suggesting that while AI providers' APIs lower the barrier to accessing models, they also introduce a higher risk of secret key leakage compared to other services. Additionally, we detected leaked credentials from database platforms and SSH RSA login keys, which pose severe security risks. If obtained by attackers, these secret keys could lead to data tampering, system breaches, and even full platform compromise (see Section 6 for details). Moreover, we identified over 1,800 leaked secret keys from code hosting platforms, which could jeopardize code access permissions and expose repositories to malicious modifications. Furthermore, API key leaks from social media platforms (e.g., Twitter), cloud service providers (e.g., AWS), and payment systems (e.g., Mailgun) present additional security threats, including account hijacking, cloud resource abuse, and unauthorized transactions.

Observation #1: *AI-related API key leaks dominate the detected exposures, which aligns with Spaces' primary use in AI/ML development. Meanwhile, the exposure of database credentials and social media access tokens are also critical concerns, particularly in application integration, data management, and code hosting scenarios, where secret key leakage may lead to severe security vulnerabilities.*

Table 2: Distribution of Leaked Secret Keys Across Different Platforms in Spaces

Type	Name	Category	Count
AI Model Provider	OpenAI	API Key	6,959
	Groq	API Key	1,796
	Replicate	API Key	36
	Nvidia	API Key	20
Code Hosting Platforms	Hugging Face	Access Token	1,761
	Github	Access Token	61
Storage Service	Mongodb	DB Credential	315
	Postgresql	DB Credential	209
Other	RSA	Encryption Key	8
	Openssh(EC)	Encryption Key	52
	PGP	Encryption Key	67
	Twitter	Access Token	79
	Facebook	Access Token	52
	Twilio	API key	23
	Google	OAuth ID/API Key	57
	AWS	Access Key	29
	Stripe	API Key	30
	Mailgun	API Key	3

5 RQ2: Characteristics

5.1 Methodology

In this section, we construct a dual-dimensional analytical framework to systematically investigate the characteristics of secret key leakage incidents within the Spaces platform. Our analysis focuses on two main dimensions: the distribution of file types and temporal trends, aiming to uncover the latent patterns and evolutionary dynamics of such security issues.

5.1.1 File Type Distribution Analysis: To identify potential patterns of key leakage at the file type level, we analyze the distribution of file types in the leakage samples detected by Secret Reviewer. This reveals the high-risk locations and concentrated regions of key leakage across different file types. Specifically, all files containing leaked keys are categorized into five groups: (1) Commit history, (2) Source code files (e.g., `.py`), (3) Configuration files (e.g., `.env`, `Dockerfile`), (4) Text documents (e.g., `.md`, `.txt`), and (5) Data files

(e.g., `.csv`). We map all detected leak instances to these categories and compute the number and proportion of leaks in each type. This categorization offers a comprehensive view of how secret key leakage is distributed across file types on the Spaces platform, reflecting developers' coding practices, handling of sensitive information, and associated security risks.

5.1.2 Temporal Trend Analysis: To further investigate the temporal dynamics of secret key leakage, we develop a time-series trend analysis leveraging the repository timestamp information introduced in Section 3.1 and the leakage data identified by Secret Reviewer. We extract four key indicators: (1) the number of newly created Spaces repositories per month; (2) the number of repositories with key leakage incidents among the newly created repositories in each month; (3) the number of newly discovered unique secret keys among the newly created repositories in each month; and (4) the leak ratio, defined as the proportion of repositories with secret key leaks relative to the total number of newly created repositories in that month. Through this time-series analysis, we not only illustrate the overall trend of key leakage on the Spaces but also identify several critical fluctuation points. These findings provide empirical evidence for assessing the evolution of developers' security awareness and the effectiveness of platform-level safeguards.

5.2 Results

5.2.1 File Distribution of Secret Keys. Firstly, as shown in Table 3, source code files have been identified as the primary source of secret key leaks, with nearly 6,000 leaked secret keys discovered, the majority of which are in `.py` files. This finding contrasts with leak patterns on GitHub, where secret keys are typically exposed in encrypted file types (e.g., `.key` files) [25]. In contrast, on the Hugging Face Spaces platform, leaks are predominantly found in source code files. This discrepancy may be attributed to the platform's built-in web services. As discussed in Section 2, developers can run AI applications simply by writing brief Python scripts. This convenience offered by tools such as Gradio and Streamlit, while facilitating AI application deployment, also significantly increases the risk of secret key leakage.

Secondly, the Git commit history (i.e., the `.git/`) is the largest source of secret key leaks, with 3,883 leaked secret keys identified. This indicates that even if developers delete sensitive information from files, this data can still persist in Git's version history. Attackers can retrieve these exposed tokens using commands like git log or git reflog. In Section 6, we further analyze residual secret keys in historical records and discuss the necessity of cleaning Git history.

Further analysis shows that configuration files are a major source of secret key leaks, with `.env` environment variable files alone exposing 957 keys. This often results from developers neglecting to add `.env` files to `.gitignore`, leading to accidental uploads. Additionally, unlike GitHub, the Space platform's direct code execution may create challenges for some developers, especially beginners. Other structured configuration files, such as `Dockerfile`, `.TOML`, and `.yml`, also contain significant leaks, indicating that secret keys are often unintentionally exposed during containerization or infrastructure configuration, further amplifying security risks.

In addition, token leaks are also found in text and data files. Leaks in Markdown files (138 leaks) and text files (132 leaks) indicate that developers inadvertently included API keys in documentation, `README` files, or explanatory texts. Besides, users may also create various text files with different names, in which secret key leaks can also occur (as shown in 'other' in Table 3). Finally, leaks in CSV data files (32 leaks) suggest potential security vulnerabilities in small-scale data sharing and storage processes.

Observation #2: *The proportion of leaks in code files further confirms the distinctive nature of AI/ML development on the Spaces platform, where developers are likely to embed API keys directly in their code to call external models or data services. Additionally, from the perspective of Git commit history, regardless of whether developers change secret keys or realize a leak and attempt to fix it, these secret keys remain in the commit history, posing a persistent security risk.*

Table 3: Secret Key Leakage Distribution Across Files

Type	Name	Num	Proportion (%)
Commit history	.git/	3883	32.86
Code file	.py	5262	46.73
	.ipynb	305	2.58
	.js	88	0.74
	.sh	44	0.37
	.go	6	0.05
Config file	.env	957	8.10
	.json	275	2.33
	.jsonl	8	0.07
	.yml	118	1.00
	DockerFile	74	0.63
	.toml	59	0.50
	.ini	13	0.11
	.env-sample	13	0.11
Text file	.txt	132	1.12
	.example	21	0.18
	.md	138	1.17
	others	129	1.09
Data file	.csv	32	0.27

5.2.2 Trends in Secret Key Leakage. Figure 5 illustrates the four temporal metrics introduced in Section 5.1 for analyzing secret key leakage trends on the Spaces. The blue line represents the number of newly created Spaces repositories per month, the red line indicates the number of repositories with key leakage incidents among these new repositories, and the yellow line shows the number of newly discovered unique leaked keys. The green line denotes the leak ratio. The first three metrics are plotted on the left axis using a logarithmic scale, while the leak ratio is displayed on the right axis.

From the trends, we observe that the total number of repositories (blue line) is steadily increasing, indicating a growing usage of the Hugging Face Spaces platform. However, the number of repositories with secret keys (red line) and the number of leaked secret keys (yellow line) are also rising, suggesting that while the platform is rapidly developing, the risk of secret key leaks is concurrently increasing. The proportion of leaked repositories (leak ratio, green line) exhibits significant fluctuations. In April 2023, there was a marked spike in the repository leak proportion, exceeding 5%, indicating a sharp rise in secret key leak risks during this period. This peak is likely related to the increased interest in LLMs, such as the

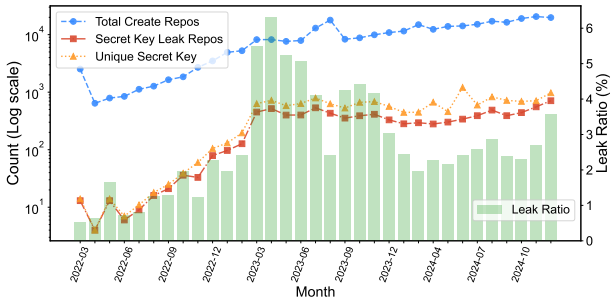


Figure 5: Evolution of Secret key Leakage in Spaces

release of ChatGPT's plugin functionality in April, which attracted more users to engage with LLM applications [17]. Subsequently, the proportion of leaked repositories decreased, although it began to rise again in October 2024. However, it remained below 4%, suggesting developers' security awareness had increased and effective measures were being implemented by the community (such as the collaboration with TruffleHog announced in 2024 [15]).

Observation #3: *The trend analysis indicates that the rapid expansion of the Spaces has indeed contributed to an increase in secret key leak issues, particularly between March 2022 and March 2023, when the leak proportion continuously rose. However, the subsequent trend stabilization reflects the effective security measures the Hugging Face community has implemented to address secret key leak risks.*

6 RQ3: Security Risks

6.1 Methodology

In this section, we discuss the potential risks posed by secret key leakage. We begin by modeling the threat of such leakage, followed by a clarification of our ethical considerations. Finally, we provide a detailed description of the automated process used to verify the validity of the leaked secret keys.

6.1.1 Threat Modeling. In this section, we analyze the security risks associated with secret key leakage in the Spaces environment. Through an investigation of various platforms and by incorporating Microsoft's STRIDE threat analysis model [26], we summarize the potential impacts of different types of secret key leakage into the following three key dimensions: personal information leak, data tampering (including unauthorized access and misuse), and financial loss (see Table 4). Specifically:

- AI Model Provider Secret Keys – Leaked API tokens may expose user profile data, leading to personal information breaches. Attackers may also exploit the API to generate malicious content or abuse API resources, resulting in substantial financial costs for the secret key holder.
- Storage Service Secret Keys – Attackers can leverage compromised credentials to access, modify, or delete sensitive data, leading to data breaches, service disruptions, or ransomware attacks. Additionally, stolen storage resources may be exploited for DDoS attacks, further amplifying security risks.
- Code Hosting Platform Secret Keys – Leaked API tokens may grant unauthorized access to private repositories, exposing proprietary code and sensitive credentials. Attackers can tamper with source code, delete repositories, or conduct supply chain attacks, potentially causing data theft, and financial losses.
- Other Secret Keys – This category includes SSH keys, RSA keys, and OAuth Consumer Keys. The exposure of SSH and RSA keys may enable unauthorized remote access, system compromise, and persistent backdoor implantation. Meanwhile, leaked OAuth Consumer Keys can be exploited by attackers to impersonate legitimate applications, leading to user data breaches and large-scale account compromises.

In Section 5, we observe the presence of numerous secret keys in commit history. Therefore, in the subsequent analysis of security risks, we will address the following aspects separately: (1) secret keys embedded in the **commit history** (.git files), and (2) hardcoded secret keys exposed in **plaintext** within the source code files.

Table 4: Risk Analysis by Service Type

Service Type	Personal Info Leak	Data Tamper	Economic Loss
AI Model Provider	✓	×	✓
Storage Service	✓	✓	✓
Code Hosting Platform	✓	✓	✓
Other	✓	✓	✓

6.1.2 Ethical Considerations. Before introducing our automated verification pipeline, we first clarify the ethical considerations that guided our study. In the process of verifying secret keys, we follow the practices adopted by existing secret detection tools [36] and ethical guidelines [13], utilizing official API interfaces provided by various platforms to verify the validity and access scopes of detected keys. These interfaces are specifically designed for checking whether a key is active and determining its associated permissions. Furthermore, to address potential ethical concerns, we enforced strict data protection measures throughout the verification process: no sensitive information returned by the APIs was recorded, and no actual operations were performed on external systems or services. In addition, we adhered to the principle of least privilege by executing only the essential validation steps, thereby ensuring the integrity of our research while minimizing any potential impact on key owners. Therefore, our evaluation was imited to three types of secret keys: AI model provider, code hosting platform, and storage service. We outline our ethical considerations for each service type:

AI Model Provider (e.g., API Tokens for OpenAI) – These secret keys primarily exist as API keys, enabling developers to access AI inference services for tasks such as text generation, image synthesis, and model fine-tuning. Even if these tokens are not leaked, many are still used in publicly accessible Spaces for inference services. Therefore, testing their validity does not introduce additional security risks. Moreover, we strictly adhere to ethical guidelines and ensure that these API keys are not misused in any way.

Code Hosting Platform (e.g., GitHub API Tokens) – For this category, we only use the official APIs provided by the respective platforms to analyze the types of potentially leaked secret keys. We do not perform any unauthorized deep access or operations.

Storage Service (e.g., MongoDB Credentials) – For this type of secret key, we only verify whether a connection can be established, without attempting login or any subsequent operations. This precaution ensures that no real data leakage or system damage occurs during our testing process.

Other (e.g., SSH Keys, RSA Keys, Consumer Keys) – Testing even a simple connectivity on this type of secret keys could lead to risks such as privacy breaches, reputational damage, or security threats to the secret key holders. To prevent any unintended consequences, we **avoid** conducting any testing on these secret keys.

Besides, this study has been reviewed and approved by the Academic Ethics Review Committee of the first authors institution whose role is equivalent to the Institutional Review Board (IRB) in the United States.

6.1.3 Automated Verification Pipeline. Building upon the aforementioned ethical considerations, we introduce the core concept of our automated verification pipeline to verify the validity of leaked secret keys. As shown in Figure 6, we first evaluate the active status of different types of secret keys using APIs provided by various platforms. For secret keys that remain active, we further evaluate their potential risks: for AI service provider and code hosting platform secret keys, we attempt to retrieve personal information to assess their access scope. Additionally, we validate the functionality of different types of secret keys. Specifically, for AI service provider secret keys, we test whether they can invoke inference services of various models, while for code hosting platform secret keys, we analyze their permission scope, including read, write, and fine-tuning privileges. Notably, for storage service secret keys, we only verify their validity without performing actual login attempts, thereby minimizing any potential impact on real-world environments.

6.2 Results

6.2.1 Model Provider. We first evaluated the leakage of API tokens from different AI model providers, assessing their usability, potential for privacy breaches, and whether the tokens had remaining quotas (denoted as "Available" in the Table 5). The results show that OpenAI and Groq have the largest token leaks. Specifically, OpenAI had 4,236 plaintext tokens and 2,723 tokens exposed in commit hostory, with 840 (19.8%) and 804 (29.5%) still active, respectively. Notably, our investigation of major AI model providers revealed that OpenAI's active API tokens can access personal information (e.g., name, phone number, email), and 49 tokens still have available quotas. If these tokens are exploited by attackers, they could lead to privacy breaches. On the other hand, Groq had 1,397 hard-coded tokens and 399 tokens exposed in .git files, with only 33 (2.4%) and 34 (8.5%) still active, respectively. However, 67 tokens still have remaining quotas, posing a higher potential economic loss risk compared to OpenAI. In contrast, Replicate and Nvidia had fewer leaked tokens, all of which were inactive and had no available quotas, suggesting that their API tokens have likely been revoked or rotated by the owners.

Observation #4: *The existence of leaked secret keys that remain active and have unused quotas highlights the dual risk posed by exposed API tokens from AI model providers. Beyond compromising user privacy, these leaks can lead to direct financial losses, as attackers may exploit the exposed tokens for unauthorized usage.*

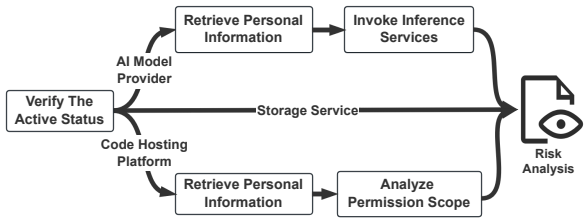


Figure 6: Verification pipeline for secret key leakage risks.

Table 5: Leaked Secret Keys in AI Model Platforms

Platform	Plaintext		Commit History		Available
	Total	Active	Total	Active	
OpenAI	4236	840	2723	804	49
Groq	1397	33	399	34	67
Replicate	22	16	14	10	0
Nvidia	14	0	6	0	0

6.2.2 Code Hosting Platforms. Then, we conducted a retrospective evaluation of leaked access tokens across two types of code hosting platforms (as shown in Table 6). In terms of quantity, we identified numerous exposed tokens on both Hugging Face and GitHub. Specifically, Hugging Face had 1,106 leaked tokens, of which 53 remained valid, while GitHub had 37, with 31 still usable. More alarmingly, despite some tokens being removed from repositories, 41 Hugging Face tokens and 13 GitHub tokens remained valid. This indicates that merely deleting exposed credentials does not eliminate security risks. Without proper revocation, attackers can still exploit these credentials for unauthorized access.

Additionally, the security risks associated with access tokens not only depend on their leakage but are also closely related to their permission levels. We conducted a further permission analysis on currently active tokens. The results reveal that numerous tokens have high-risk write or fine-grained permissions. Specifically, among the leaked tokens from Hugging Face, 55 have write permissions, accounting for more than half of the available tokens, which could lead to model tampering, data contamination, or the injection of malicious code. On the other hand, among the GitHub tokens, 18 have write permissions, allowing attackers to modify or delete repositories, or implant backdoors, thereby launching supply chain attacks. Notably, through manual analysis of the code context, we found that some code does not actually use these write capabilities. Furthermore, GitHub also has 24 tokens with fine-grained permissions. If these tokens are associated with CI/CD pipelines, automated deployments, or infrastructure management, they could pose more serious long-term security risks.

Observation #5: *These findings suggest that leaks from code hosting platforms not only have the potential to cause data breaches but also expose a failure to adhere to the principle of least privilege (PoLP) during development. If tokens are leaked, attackers may gain direct access to modify systems and code, potentially triggering more severe supply chain security incidents, thus escalating security risks.*

6.2.3 Storage service. Finally, we analyzed the leakage of secret keys in storage services (As shown in Table 7). The analysis revealed that a total of 268 MongoDB and 194 PostgreSQL database credentials were exposed in plaintext, with 45.1% of the MongoDB credentials still active. In contrast, only 1.5% of the PostgreSQL credentials remained usable. Additionally, we identified a total of 19 active credentials found in commit history. While the number of leaked storage service credentials is relatively low compared to other types of secret keys, once attackers obtain these leaked credentials, they can directly access, modify, or even delete data

Table 6: Leaked Secret Keys in Code Hosting Platforms

Platform	Plaintext		Commit History		Permission Distribution			
	Total	Active (%)	Total	Active (%)	Read	Write	Fine-Grained	**Total**
Hugging Face	1106	53 (4.79%)	655	41 (3.71%)	38 (40.2%)	**55 (58.2%)**	1 (1.06%)	94
GitHub	37	31 (83.8%)	24	13 (35.1%)	2 (4.45%)	**18 (40.9%)**	24 (54.5%)	44

Table 7: Leaked Secret Keys in Storage Service.

Platform	Plaintext		Commit History		Active
	Total	**Active**	**Total**	**Active**	
MongoDB	268	121	47	18	139
PostgreSQL	194	3	15	1	4

in the databases, potentially leading to large-scale data breaches, data tampering, and service disruptions. Furthermore, attackers may exploit these credentials for persistent control, inject malicious data, or even carry out database encryption ransomware attacks.

Observation #6: *Although storage service leaks are less frequent, the potential risks associated with these leaks can be significant: 1) Once the credentials are verified as valid, attackers can at least gain read access, leading to data breaches, particularly in NoSQL databases like MongoDB, which often lack strict access controls by default. In such cases, attackers may be able to obtain read-write access without additional authentication, significantly increasing security risks; 2) Since these database services typically do not rotate/abandon credentials proactively, if developers fail to find the leak in time, the security threat may persist, potentially going unnoticed for an extended period.*

6.3 Case Studies

To further demonstrate the real-world risks posed by secret key leakage across different platforms, we conducted case studies on three distinct service types discussed in Section 6.2 and performed sampling analysis to validate the prevalence of such risks. It is important to note that, for ethical reasons, no actual leaked content is disclosed in our case studies. In addition, to help researchers better understand the potential impact of secret key leakage, we created test secret keys for different platforms in our open-source repository and provided detailed tutorials [1], enabling controlled and reproducible exploration of these risks. We next describe three types of case studies in detail.

6.3.1 AI Model Platforms. For leaked secret keys associated with AI model provider platforms, we found that beyond financial abuse, these keys can also be used to invoke official user profile query APIs, granting access to users' personal information and leading to severe privacy breaches. Taking OpenAI's API token as an example (as shown below), we confirmed that a leaked key can be used to retrieve sensitive information such as the user's email address and name. To estimate the prevalence of similar risks across all identified leaked keys, we conducted a sampling analysis with a 95% confidence level and a ±4.9% margin of error, in accordance with standard software engineering practices. Specifically, we sampled 326 keys from a total of 1,737 valid leaked secret keys from AI model providers. The results showed that 309 of these keys involved personal information exposure, and 8 remained active with available usage quotas at the time of analysis. The results highlight the serious privacy/finance risks posed by the secret key leak from the AI model provider.

Privacy info exposed through OpenAI API Key

```
"userinfo":{
    "id":"xxx", "object":"user",“created”: xxx,
    "email":"xxx", "name":"xxx","speed_limit":true
    ...     }
```

6.3.2 Code Hosting Platform. For leaked secret keys associated with code hosting platforms, we found that once a leaked key has read permissions, an attacker can access and exfiltrate sensitive content from private repositories. If the key has write permissions, it may further allow tampering with or overwriting repository contents, leading to more severe security risks. Taking Hugging Face (HF) access token as an example (as shown below), our verification through the official API revealed that such token can enumerate all repositories associated with the user, including private ones. With knowledge of the repository name, the token can also be used to download its contents. It is important to note that our testing was conducted using access tokens generated from a self-created account and did not involve any real leaked credentials. Following the sampling practices described in Section 6.3.1, we sampled 103 tokens from a total of 138 active leaked secret keys identified on the platform. Among them, 31 tokens had read permissions, of which 18 could access private repositories; 55 had write permissions, with 38 able to access private repositories; and 17 had fine-grained permissions, 6 of which also granted access to private repositories. The results indicate that a significant portion of leaked access tokens have permissions to access private repositories, posing serious risks of data leakage and integrity compromise.

Private repo exposed through HF access token

```
Token is valid(perimission:read)
username:xxx    org:yyy-yyy,zzz-zzz
Models:                    Datasets:
    xxx/abcd Private = True     xxx/efgh Private =True
        ....                       ....
```

6.3.3 Storage Service. For leaked secret keys associated with storage service platforms, we found that leaked database credentials can be used to directly connect to external databases, enabling unauthorized data access or modification and posing risks such as privacy breaches and data corruption. Taking MongoDB credentials as an example (as shown below), a single-line script is sufficient to connect to the target database, and basic code can be used to perform various data operations. Given the ethical and security concerns associated with accessing or modifying real databases, we did not conduct sampling analysis on leaked credentials from such platforms. However, we verified the potential impact through a self-hosted MongoDB test account. Our controlled experiments suggest that the 143 active DB credentials listed in Table 7 pose significant risks of unauthorized access and data manipulation.

Malicious operations using DB credentials

```
client = MongoClient('DB_Credentials') # Connect
client.list_database_names()           # Read
insert_result = collection             # Write
               .insert_one({"key": "maliciouscontent"})
```

7 Discussion

7.1 Key Takeaways

Our study reveals that secret key leakage is prevalent and persistent in HF Spaces repositories, posing substantial security risks due to exposed and active credentials. Through the validation of different types of secret keys, we conducted an in-depth analysis of the potential security risks posed by each type of key. However, it is important to note that the active secret keys found in the commit history demonstrate that, regardless of whether the secret key was removed from plaintext through the warnings of Hugging Face's official secret key detection tools, or the developer's self-awareness, the security issue of secret key leakage remains. The reason is that, for attackers, even though the plaintext secret key is removed from the code, it may still persist in the history. If the secret key remains active, it can still be exploited by attackers. Therefore, simply deleting the secret key is not enough to fully eliminate the risk. To completely address this risk, in addition to removing the plaintext secret key, developer must also rotated or disabled secret key immediately. This ensures that even if the secret key is discovered by attackers, it cannot be further exploited.

7.2 Suggestions

To address these risks and improve management practices, we propose several suggestion and mitigation measures from two perspectives: developers and the Hugging Face community.

Developer Perspective: Our study reveals that a significant number of secret keys are hardcoded in plaintext within code and configuration files, leading to potential exposure of sensitive information. To mitigate this risk, developers should avoid storing secret keys directly in code and instead utilize environment variables (already supported in Spaces) or Secret Managers for secure storage. Additionally, developers should explicitly ignore sensitive files in `.gitignore` to prevent inadvertent commits containing secret keys. Furthermore, our experiments show that removing secret keys from source code is not enough, as they may still exist in Git commit history. Therefore, we recommend using tools like git filter-repo or BFG Repo-Cleaner to thoroughly sanitize Git history. Lastly, regular rotation of secret keys is essential to minimize long-term exposure and prevent unauthorized access.

Hugging Face Community Perspective: Although Hugging Face has integrated TruffleHog for secret key detection, our findings reveal that this mechanism only blocks Hugging Face-specific API keys, while secret keys from other platforms can still be uploaded. Moreover, while Hugging Face notifies users by email after detecting leaked secret keys, this passive approach can be overlooked, posing ongoing risks. To address this issue, we recommend that Hugging Face adopt GitHub's method of actively blocking commits containing sensitive credentials and requiring explicit user confirmation before proceeding. Additionally, Hugging Face should enhance user security by offering secure coding guidelines during Space creation and integrating a one-click Git history sanitization tool into the UI.

7.3 Generalizability of Our Findings

We conducted a similar empirical study on ModelScope's online deployment platform [27] (referred to as ModelScope Space), which, like Hugging Face Spaces, allows developers to deploy and share interactive AI applications. We collected 4,481 publicly accessible ModelScope Spaces and applied our detection pipeline to identify leaked secret keys. As a result, we identified 8 instances of leaked secret keys, including: two Hugging Face access tokens (from a `.py` and a `.md file`) with one remaining active with fine-grained permissions despite no associated repositories; one GitHub access token found in a `.py` file being inactive; five OpenAI API keys were identified (from `.py`, `.txt`, and `.json` files) with two remaining active and containing personal information although both have exceeded their usage limits. This finding confirms that the risk of secret key leakage is not unique to Hugging Face Spaces, it is also present on other app-hosting platforms that support interactive deployment. It further demonstrates the robustness and applicability of our approach across different deployment ecosystems.

7.4 Limitations

This study has several limitations. First, although we did not perform real-world attacks out of ethical considerations, we provided multiple concrete case studies to demonstrate the practical exploitability of leaked secret keys. These examples offer compelling evidence that such leakage can indeed lead to serious security consequences, even without actively triggering the attack process. Second, our detection pipeline may produce a few false positives. Nonetheless, this primarily affects the overall count of detected secret keys. It does not compromise the validity of those keys that have been successfully verified as functional or active, nor the security risks associated with their exposure, which remain central to our security analysis. In future work, we plan to explore the development of attack simulation tools to better quantify the real-world threats posed by leaked secret keys.

8 Conclusion

Hugging Face Spaces has become a convenient and efficient platform for hosting AI applications, but its ease of use may lead to secret key leakage. In this study, we conduct a large-scale empirical analysis of secret key leakage on the Space platform. Our findings reveal that 30% of the detected secret keys were leaked through commit history, and more than 1,000 non-code files contained sensitive data. Furthermore, we validated the usability of the detected secret keys, showing that 30% of them remained active. Specifically, we identified 140 valid database credentials and found that 50% of the exposed API tokens retained write permissions, highlighting the severe security risks posed by secret key leakage in Space.

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant 2024YFB2906900, the National Natural Science Foundation of China under Grant 62172054, the Beijing Nova Program under Grant 2023140.

References

[1] 2025. Available Artifacts. https://github.com/howardyun/LLM-measure-token-Leakage.

[2] Setu Kumar Basak, Jamison Cox, Bradley Reaves, and Laurie Williams. 2023. A comparative study of software secrets reporting by secret detection tools. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[3] Setu Kumar Basak, Lorenzo Neil, Bradley Reaves, and Laurie Williams. 2023. Secretbench: A dataset of software secrets. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 347–351.

[4] Setu Kumar Basak, Lorenzo Neil, Bradley Reaves, and Laurie Williams. 2023. What challenges do developers face about checked-in secrets in software artifacts?. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1635–1647.

[5] DeepSpeed Contributors. 2021. DeepSpeed: Accelerating Large-Scale Model Training and Inference. https://www.deepspeed.ai/ Accessed: 2025-01-23.

[6] AD Diego. 2017. Automatic extraction of API Keys from Android applications. *PhD thesis, Ph. D. dissertation* (2017).

[7] Zeyad Emam, Andrew Kondrich, Sasha Harrison, Felix Lau, Yushi Wang, Aerin Kim, and Elliot Branson. 2021. On the state of data in computer vision: Human annotations remain indispensable for developing deep learning models. *arXiv preprint arXiv:2108.00114* (2021).

[8] Hugging Face. 2023. Security Tokens. https://huggingface.co/docs/hub/security-tokens. Accessed: 2023-10-01.

[9] Hugging Face. 2023. *Space Secrets Disclosure.* https://huggingface.co/blog/space-secrets-disclosure

[10] Hugging Face. 2025. *Spaces.* https://huggingface.co/docs/hub/spaces Accessed: 2025-03-06.

[11] Hugging Face. 2025. *Spaces Overview.* https://huggingface.co/docs/hub/main/en/spaces-overview Accessed: 2025-03-06.

[12] Runhan Feng, Ziyang Yan, Shiyan Peng, and Yuanyuan Zhang. 2022. Automated detection of password leakage from public github repositories. In *Proceedings of the 44th International Conference on Software Engineering*. 175–186.

[13] OWASP Foundation. 2021. Vulnerability Disclosure Cheat Sheet. https://cheatsheetseries.owasp.org.

[14] Jie Gao, Simret Araya Gebreegziabher, Kenny Tsu Wei Choo, Toby Jia-Jun Li, Simon Tangi Perrault, and Thomas W Malone. 2024. A taxonomy for human-llm interaction modes: An initial exploration. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–11.

[15] Luc Georges and Hugging Face Team. [n. d.]. Hugging Face partners with Truffle-Hog to Scan for Secrets. https://huggingface.co/blog/trufflesecurity-partnership. Published: September 4, 2024. Accessed: 2025-07-19.

[16] Gitleaks. 2025. Gitleaks. https://github.com/gitleaks/gitleaks.

[17] Valentin Hartmann, Anshuman Suri, Vincent Bindschaedler, David Evans, Shruti Tople, and Robert West. 2023. Sok: Memorization in general-purpose large language models. *arXiv preprint arXiv:2310.18362* (2023).

[18] Hamit Kavas, Marc Serra-Vidal, and Leo Wanner. 2024. Using Large Language Models and Recruiter Expertise for Optimized Multilingual Job Offer–Applicant CV Matching. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*. 8696–8699.

[19] Alexander Krause, Jan H Klemmer, Nicolas Huaman, Dominik Wermke, Yasemin Acar, and Sascha Fahl. 2022. Committed by Accident: Studying Prevention and Remediation Strategies Against Secret Leakage in Source Code Repositories. *arXiv preprint arXiv:2211.06213* (2022).

[20] Alexander Krause, Jan H Klemmer, Nicolas Huaman, Dominik Wermke, Yasemin Acar, and Sascha Fahl. 2023. Pushed by Accident: A {Mixed-Methods} Study on Strategies of Handling Secret Information in Source Code Repositories. In *32nd USENIX Security Symposium (USENIX Security 23)*. 2527–2544.

[21] LangChain. 2025. LangChain: Building Applications with Large Language Models. https://python.langchain.com/ Accessed: 2025-01-10.

[22] Ziyang Li, Aravind Machiry, Binghong Chen, Mayur Naik, Ke Wang, and Le Song. 2021. Arbitrar: User-guided api misuse detection. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1400–1415.

[23] Jerry Liu. 2022. LlamaIndex: A Data Framework for LLM Applications. https://github.com/run-llama/llama_index Accessed: 2025-02-1.

[24] Nikolaos Lykousas and Constantinos Patsakis. 2023. Tales from the Git: Automating the detection of secrets on code and assessing developers' passwords choices. In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 68–75.

[25] Michael Meli, Matthew R McNiece, and Bradley Reaves. 2019. How bad can it git? characterizing secret leakage in public github repositories.. In *NDSS*.

[26] Microsoft. 2021. Threat modeling: STRIDE approach. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats Accessed: 2025-07-03.

[27] ModelScope Team. 2025. ModelScope Studios: Introduction. https://modelscope.cn/docs/studios/intro.

[28] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[29] Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. 2018. Finding clues for your secrets: semantics-driven, learning-based privacy discovery in mobile apps.. In *NDSS*.

[30] Karen Ka Yan Ng, Izuki Matsuba, and Peter Chengming Zhang. 2025. RAG in health care: a novel framework for improving communication and decision-making by addressing LLM limitations. *NEJM AI* 2, 1 (2025), AIra2400380.

[31] Erdal Ozkaya. 2025. DeepSeek Cyberattack: A Comprehensive Analysis & Security Guide. https://www.linkedin.com/pulse/deepseek-cyberattack-comprehensive-analysis-security-guide-ozkaya-ibese/. accessed 2025-07-05.

[32] Md Nafiu Rahman, Sadif Ahmed, Zahin Wahab, SM Sohan, and Rifat Shahriyar. 2025. Secret Breach Detection in Source Code with Large Language Models. *arXiv preprint arXiv:2504.18784* (2025).

[33] Md Rayhanur Rahman, Nasif Imtiaz, Margaret-Anne Storey, and Laurie Williams. 2022. Why secret detection tools are not enough: It's not just about false positives-an industrial case study. *Empirical Software Engineering* 27, 3 (2022), 59.

[34] Aakanksha Saha, Tamara Denning, Vivek Srikumar, and Sneha Kumar Kasera. 2020. Secrets in source code: Reducing false positives using machine learning. In *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 168–175.

[35] Lasso Security. 2023. *1,500 Hugging Face API Tokens Were Exposed, Leaving Millions of Meta Llama, Bloom, and Pythia Users for Supply Chain Attacks.* https://www.lasso.security/blog/1500-huggingface-api-tokens-were-exposed-leaving-millions-of-meta-llama-bloom-and-pythia-users-for-supply-chain-attacks

[36] Truffle Security. 2025. TruffleHog. https://github.com/trufflesecurity/trufflehog.

[37] Claude E Shannon. 1951. Prediction and entropy of printed English. *Bell system technical journal* 30, 1 (1951), 50–64.

[38] SonarSource. 2024. Hard-coded credentials should not be used. https://sonarqube.inria.fr/sonarqube/coding_rules?open=typescript%3AS6418&rule_key=typescript%3AS6418.

[39] Chengpeng Wang, Wuqi Zhang, Zian Su, Xiangzhe Xu, Xiaoheng Xie, and Xiangyu Zhang. 2024. LLMDFA: Analyzing Dataflow in Code with Large Language Models. *Advances in Neural Information Processing Systems* 37 (2024), 131545–131574.

[40] Stephen John Warnett and Uwe Zdun. 2024. On the understandability of MLOps system architectures. *IEEE Transactions on Software Engineering* (2024).

[41] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[42] Lifan Yuan, Yangyi Chen, Ganqu Cui, Hongcheng Gao, Fangyuan Zou, Xingyi Cheng, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023. Revisiting out-of-distribution robustness in nlp: Benchmarks, analysis, and llms evaluations. *Advances in Neural Information Processing Systems* 36 (2023), 58478–58507.