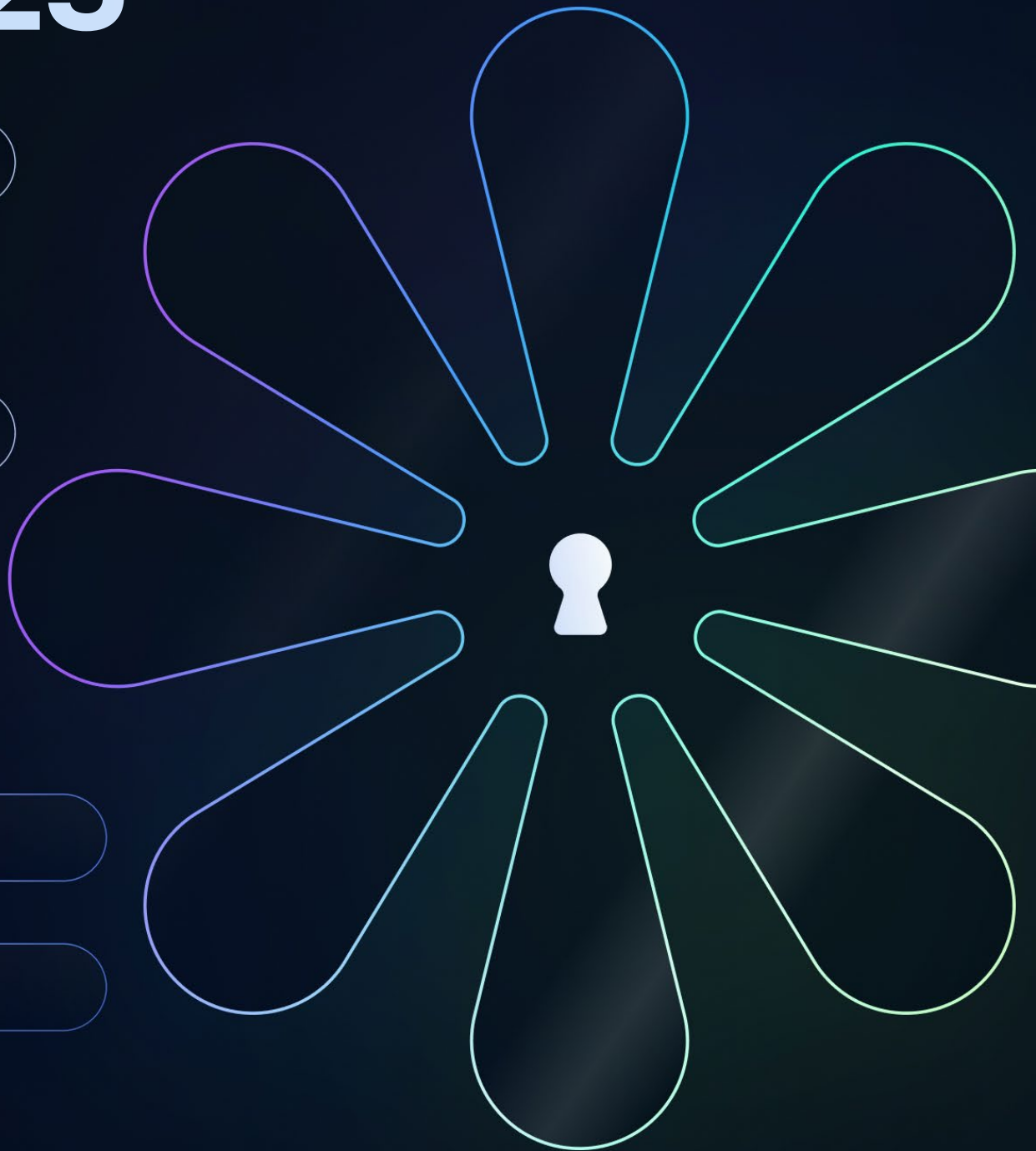




THE STATE OF

Secrets Sprawl 2025



Data analysis by **GitGuardian**

TABLE OF CONTENTS

How Leaky Was 2024	4
AI-Enhanced Detection: Revealing the Full Scope of Credential Exposure	5
58% of All Detected Secrets Are Generic	7
GitHub's Push Protection: A Promising Initiative, But Not a Silver Bullet	9
Private Repositories 8 Times More Likely To Contain Secrets	12
Fastest Growing Services	17
Mapping the SDLC: Where Leaks Happen	18
Collaboration Tools: The Overlooked Frontier of Secrets Sprawl	18
100,000+ Valid Secrets on Docker Hub	21
Copilot increases secrets incidence rate by 40%	25
Detected but Not Fixed: The Alarming Persistence of Exposed Credentials	27
Secrets Managers: Not a Complete Solution	28
Excessive Permissions Make Secret Leaks More Severe	31
Bridging the remediation gap	33
Understanding the Impact: Real-World Risks of Secrets Sprawl	34
Primary Risk Categories and Attack Vectors	34
The Cascade Effect: From Minor Leak to Major Breach	38
Critical Timeline Statistics	39
Risk Amplification Factors	39
About GitGuardian	40
Appendix	41
Definitions	41
Methodology	43

The State of Secrets Sprawl 2025

DATA ANALYSIS BY GITGUARDIAN

23,770,171 ^{+25%}

New secrets detected in public GitHub commits in 2024.

Data analysis by GitGuardian

70%

of valid secrets detected in public repositories in 2022 remain active today

15%

of commit authors leaked a secret

4.6%

of all public repositories contain a secret

35%

of all private repositories contain hardcoded secrets

38%

of incidents in collaboration and project management tools (Slack, Jira or Confluence) were classified as highly critical or urgent, compared to 31% in Source Code Management Systems (SCMs)



#1 APP ON THE GITHUB MARKETPLACE

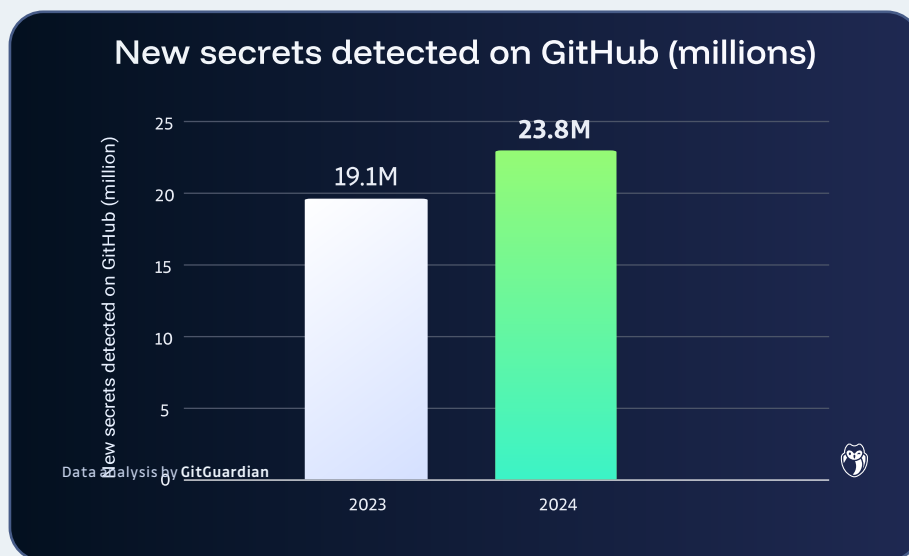
From day one, GitGuardian has been committed to protecting developer environments from secrets sprawl, a dedication that has established us as **the #1 application on GitHub Marketplace**. For over seven years, our real-time scanning of public GitHub events through our [Good Samaritan](#) program has enabled us to proactively notify developers when credentials are exposed. In 2024 alone, we sent **1.9 million pro bono alert emails** to developers who inadvertently leaked sensitive credentials.

How Leaky Was 2024

Long-lived plaintext credentials have been involved in most breaches over the last several years. When valid credentials, such as API keys, passwords, and authentication tokens, leak, attackers at any skill level can gain initial access or perform rapid lateral movement through systems.

In 2024, we found **23,770,171 new hardcoded secrets** added to public GitHub repositories. This figure represents a **25% surge** in the total number of secrets from the previous year. This marks a substantial increase in the number of secrets found and continues the disturbing trend: secrets sprawl is steadily worsening over time.

Despite GitHub's efforts to prevent certain credential leaks during the push stage, which did indeed reduce incidents involving specific secrets (secrets following known patterns for specific services), the platform's measures have not effectively addressed the growing prevalence of generic secrets. It is within this category that we observed the most significant year-over-year surge in plaintext credentials.



The danger of the continued rise of secrets leakage is very real. Over the past 10 years, stolen credentials have been used in 31% of all breaches, according to [Verizon's 2024 Data Breach Investigations Report](#). It is an attacker's favorite way to gain an initial foothold and to move laterally through environments. At the same time, [IBM's Cost of a Data Breach](#) report makes it clear how time-consuming this issue is for the enterprise. Breaches involving stolen or compromised credentials take an average of **292 days** to identify and remediate, more than any other attack vector.

AI-Enhanced Detection: Revealing the Full Scope of Credential Exposure

The 2025 State of Secrets Sprawl report marks a significant milestone in secrets detection, unveiling a more comprehensive picture of the secrets sprawl landscape. For the first time, thanks to our innovative machine learning models, such as the one powering [False Positive Remover](#), GitGuardian can now confidently identify and validate more generic secrets.

Historically, GitGuardian took a conservative stance on generic secrets to avoid a large number of potential false positive results. Our secrets detection engine was intentionally calibrated for high precision, ensuring that when a secret was flagged, it was almost certainly a real secret. Any doubt meant leaving it out.

Our past focus was concentrated on the most commonly used enterprise-specific secrets, such as API keys and service-specific credentials, but these are just the tip of the iceberg. The true magnitude of the secrets sprawl problem lies in the vast ocean of generic secrets, such as usernames & passwords and unstructured credentials.

As an example, here's a Base64 basic auth string:

```
"Authorization": "Basic aW50ZXJuc2hpcDpjZGk="
```

Or an example of a database credential:

```
connect_to_db(host="136.12.43.86", port=8130,  
username="root",  
password="m42ploz2wd")
```

This ML-driven shift not only enables us to find more secrets but also helps us categorize them much more effectively. Doing so we ensure they are genuine secrets, strengthening both [recall and precision](#). The result provides a more accurate, holistic understanding of how and where secrets are spreading.



The Department of The Treasury breach

In December 2024, Chinese state-sponsored attackers [breached the U.S. Treasury Department](#) by exploiting a compromised API key belonging to BeyondTrust, a technical support provider. The API key gave attackers valid credentials to remotely access Treasury workstations and unclassified documents through BeyondTrust’s Remote Support SaaS platform. The incident highlights how a single leaked API key in the supply chain can allow threat actors to bypass security measures and gain unauthorized access to sensitive government systems.

58% of All Detected Secrets Are Generic

This leap in technology not only enhances our detection capabilities but also reveals that generic secrets represent a far more extensive threat than previously recognized.

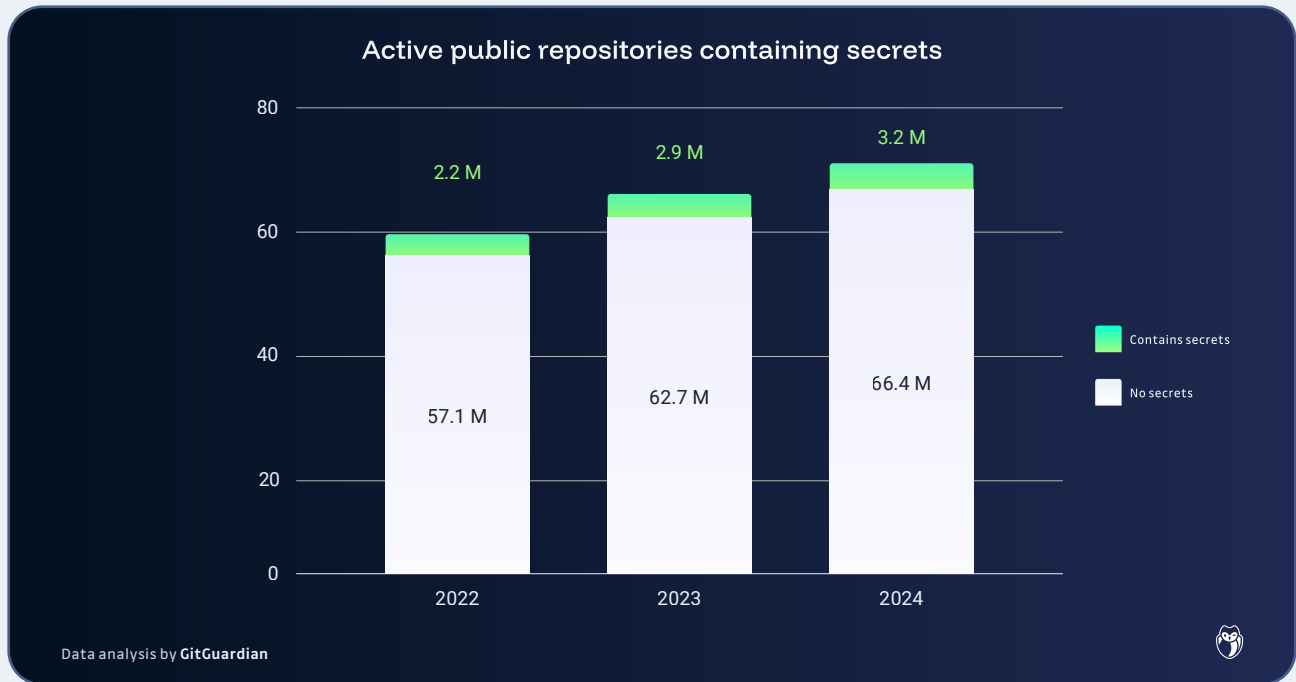
One of the most troubling findings from our analysis is the sharp increase in the number of generic secrets found in public repositories. **In 2024, 58% of all detected secrets** were generic, a jump from 49% in 2023.

Unlike API keys or OAuth tokens that follow recognizable patterns, **generic secrets lack a standardized structure**, making them far more difficult to detect and remediate. These can include:

- Hardcoded passwords embedded in source code
- Database connection strings
- Custom authentication tokens
- Encryption keys stored in plaintext

Since generic secrets do not match predefined patterns, they often bypass automated protection mechanisms like GitHub's built-in secret scanning. This is the fastest-growing and most concerning category of exposed credentials.

GitHub continues to grow, with over 1.4 billion new public commits added in 2024, marking a 20% increase in the previous year's growth. **4.6% of all public repositories in 2024 contained a secret.**



A Story From the Field

Initial repository commits represent a frequently overlooked risk vector. In one documented incident, what appeared to be a routine initial commit (`git add .`) inadvertently included over 21,000 files:

```
$ git add .  
$ git commit -m "Initial commit" && git push
```

```
21986 files changed +3,235,683 -0 lines changed
```

Further investigation revealed that the developer's `.bash_history` had been included in this commit, containing multiple instances of repository cloning operations with embedded GitHub personal access tokens.

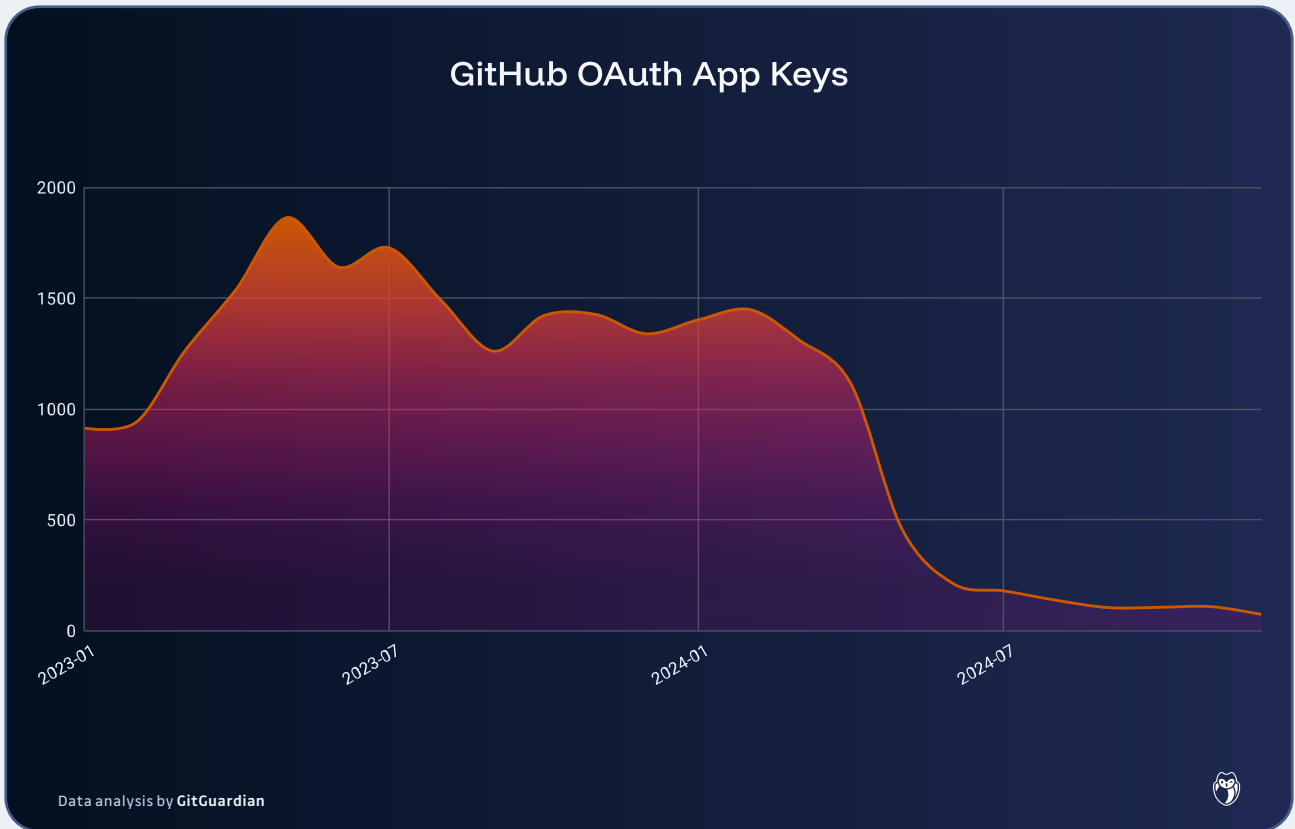
This case demonstrates how generic secrets, particularly those embedded in command histories or configuration files, can be unintentionally exposed through common development workflows.

GitHub's Push Protection: A Promising Initiative, But Not a Silver Bullet

Although the number of specific secrets leaked this past year continued to rise, the rate of increase slowed compared to previous years. This change can be attributed, in part, to GitHub's introduction of Push Protection since our last report. This feature automatically prevents developers from pushing code containing certain known credential patterns to public repositories.

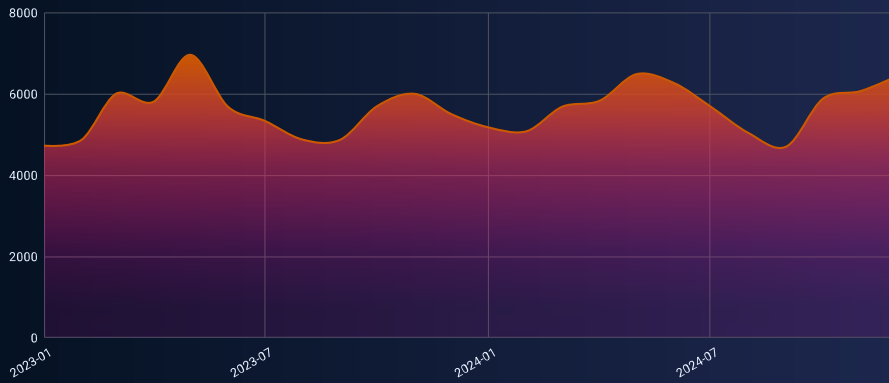


Push Protection has proven highly effective for some specific keys. For instance, following its by-default enablement for free users at the beginning of 2024, we observed a sharp decline in the number of leaked OpenAI secrets and GitHub App keys. The consistent prefixes of these keys, 'sk-' and 'ghu_' respectively, make them easily detectable using simple pattern matching techniques.



However, many other credentials, such as MySQL and MongoDB, lack a standardized prefix, making them more challenging to identify. For these credentials and other patterns not yet included in GitHub's filtering system, we have not witnessed a significant reduction in the number of detected secrets.

MySQL credentials were not impacted by GitHub Push Protection



Data analysis by GitGuardian



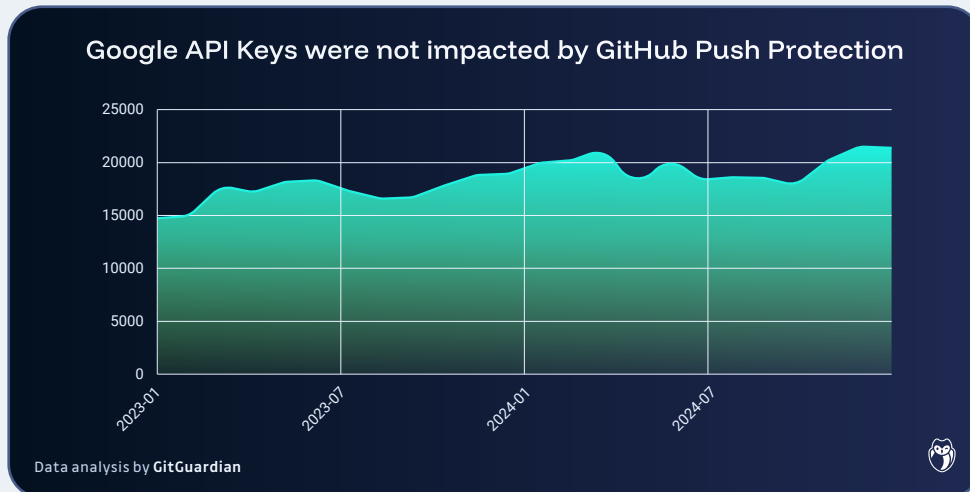
MongoDB Credentials were not impacted by GitHub Push Protection



Data analysis by GitGuardian



Prefixing is not the whole story, though. Even for many secrets that do have a prefix, such as Google API Keys, we have seen an increase, especially at the end of the reporting period.



GitHub's **Push Protection** represents a major step forward in preventing the exposure of hardcoded credentials, but it remains far from a complete solution:

- Push Protection does not detect and stop generic secrets, the fastest-growing secret category.
- Push protection is free for public repositories but is only available for private repositories with a paid plan. If a private repository becomes public, all the hardcoded secrets developers previously committed will be exposed.
- Developers can also bypass push protections with a few clicks, marking them as a "test credential" or simply saying they will fix it later.

Push protection is certainly helping with the education of developers and is showing evidence of starting to curb the issue of API key leakage. Nevertheless, GitHub's solution is limited by its focus on maximizing developer velocity rather than ensuring security.

Private Repositories 8 Times More Likely To Contain Secrets

This year, we compared our findings from public GitHub with anonymized customer data.

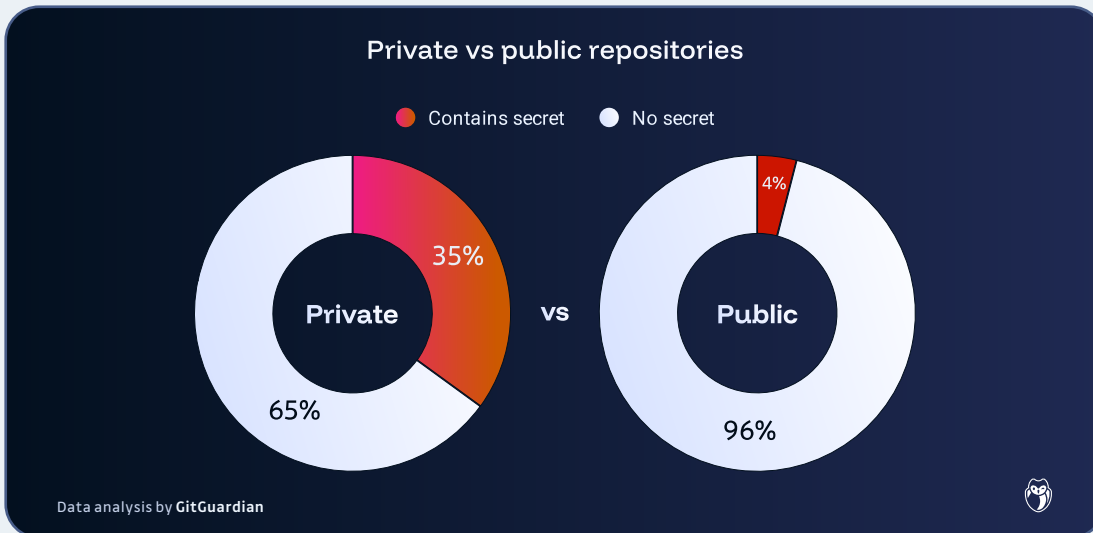
35%



of all private repositories scanned contained at least one plaintext secret.

The data shows **developers treat secrets in public code differently than in private code.**

The trend suggests that organizations may be relying on "security through obscurity," assuming that because their repositories are private, the secrets contained within them are safe.



Attackers who gain access to these internal repositories can easily widen their footprint, moving laterally and with no resistance between systems.



Private repositories can and do become public through misconfigurations, breaches, or accidental exposure. We need to work with developers to ensure they treat all secrets as secret, giving them the right tools to accomplish their mission.

A Story From the Field

Repository metadata represents an often overlooked source of credential exposure. While commit content may be scrutinized for secrets, the metadata itself can contain sensitive information. In one particularly concerning case, a developer had inadvertently configured their email address as their password in their Git configuration.

When examining the commit's API response:

```
{  
  "sha": "fe11116eabd916d818c7f1cb07d0f08eccc761cb",  
  "commit": {
```

```

    "author": {
      "name": "Developer Name",
      "email": "3lGzQ/JdFkdyb!",
      "date": "2024-01-01T06:66:66Z"
    }
  }
}

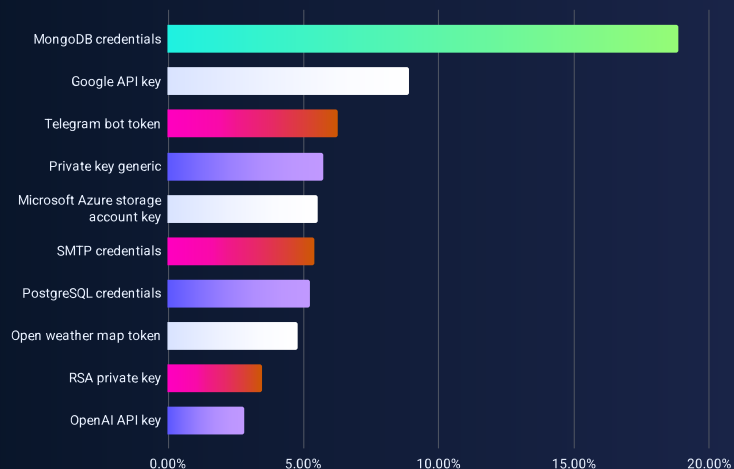
```

The “email” field contained the developer’s actual password rather than an email address. This password was valid and provided access to the developer’s account, demonstrating how even carefully reviewed code can expose credentials through tangential metadata.

Some examples of our findings for certain kinds of secrets

MongoDB, the very popular open-source document database, credentials are again the leakiest secrets this year, making up 18.8% of detected secrets in **public repositories** while only showing up in 3.9% in private repositories. Telegram bot tokens made up 6.3% of the secrets we found in public but a negligible amount in private repositories, which makes sense as most enterprises are not using Telegram as part of their workflow.

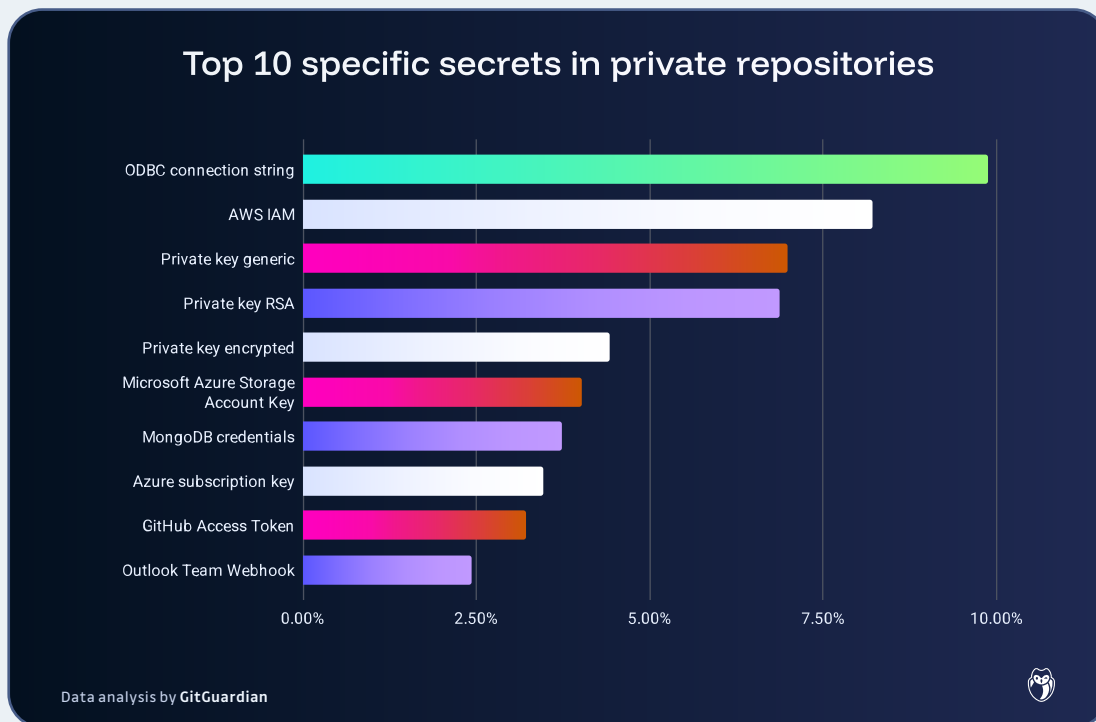
Top 10 specific secrets leaked on public GitHub in 2024



Data analysis by **GitGuardian**



However, when we turn to credentials for paid and enterprise platforms, such as AWS IAM keys, they appear as plaintext in **8% of the private repos** we researched but appear in only 1.5% of public repos. The most common plaintext credentials found were ODBC Connect Strings used with Microsoft SQL Server, making up 10% of all specific secrets detected in private repos but making up less than 0.5% of all secrets found on GitHub publicly.

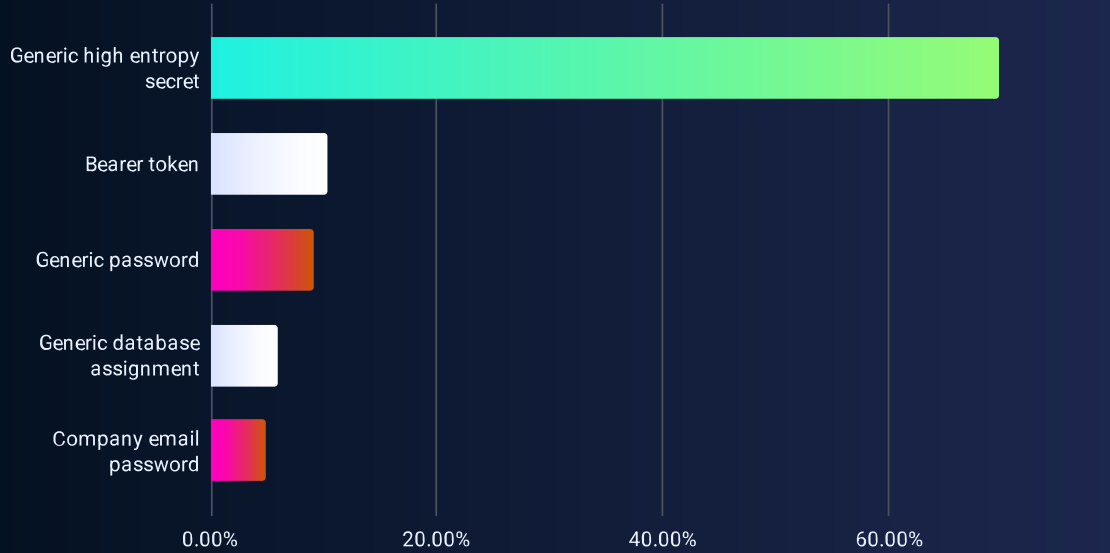


Okta is another interesting secret that rarely gets leaked publicly, showing up only a few hundred times in our scans. However, Okta keys make up a full 1.7% of all the keys found in private.

Generic secrets are proportionally **more prevalent in private repositories** than in public ones. Generic secrets represent **74.4% of all secret types in private repositories**, compared to only **58% in public repositories**.

If we focus on just generic passwords, the story is even more telling of developers relying on the privacy of the repository to protect them. In private repositories, **24% of all generic secrets** we found were generic passwords, versus 9% of all the generic secrets found in public repositories.

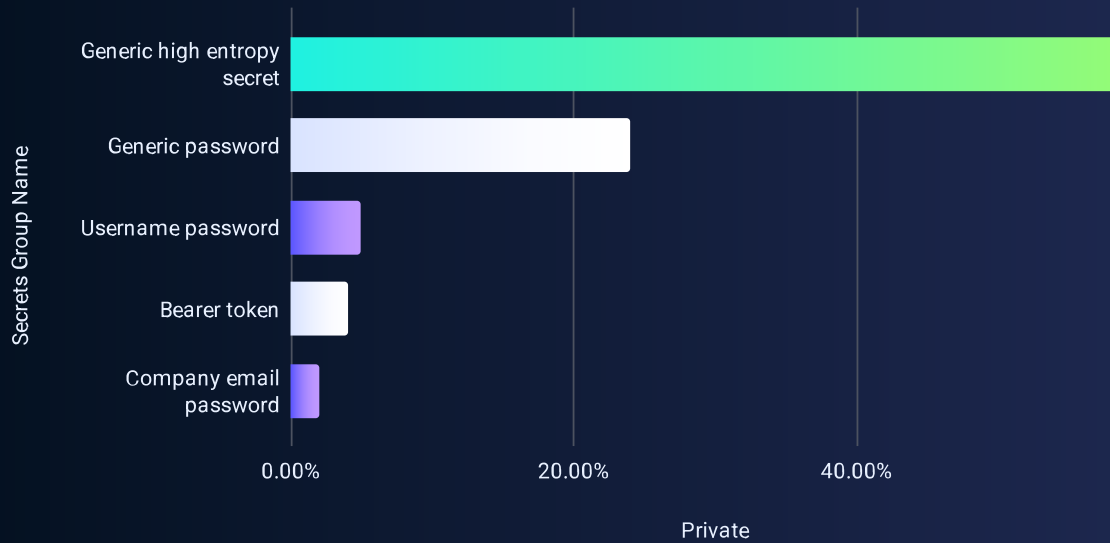
Top 5 generic secrets leaked in public GitHub repositories



Data analysis by GitGuardian



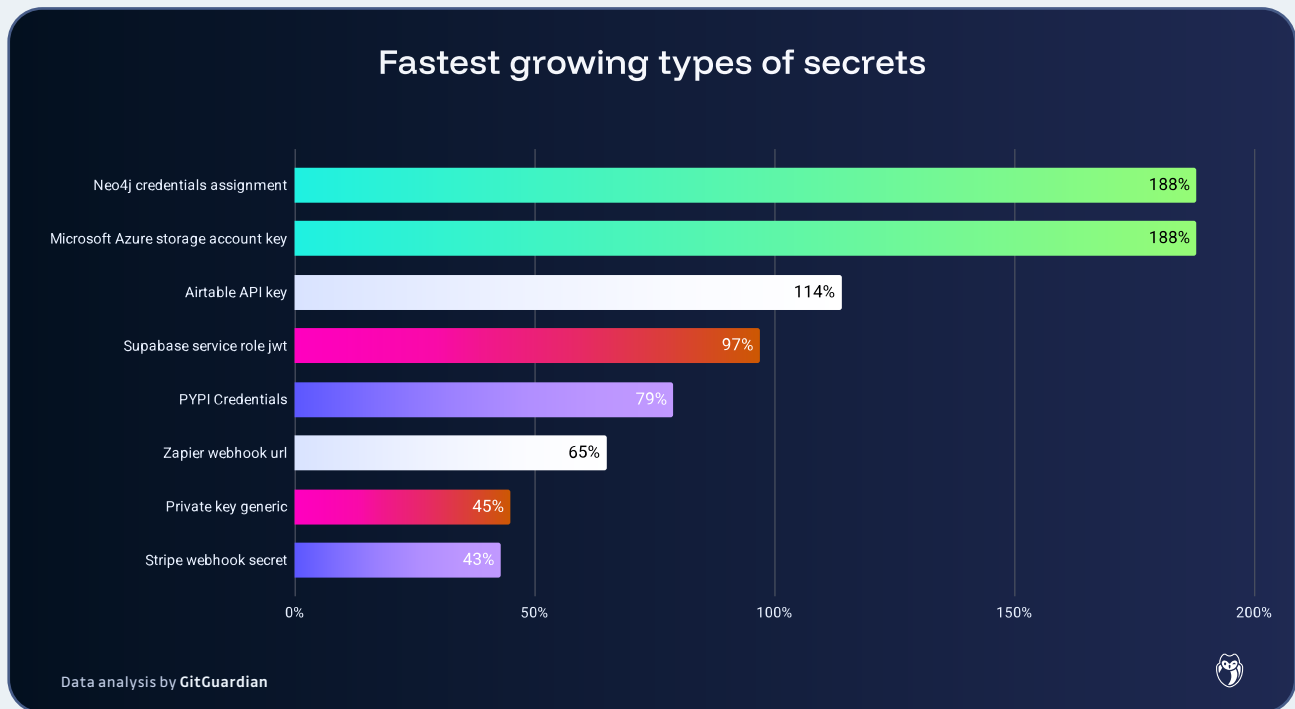
Top 5 generic secrets leaked in private repositories



Data analysis by GitGuardian



Fastest Growing Services



OpenAI and other AI services are not anymore the fastest-growing keys leaked, as they are identified by GitHub Push Protection. **The service with the largest year-over-year growth of leaks is the 188% increase for Neo4j**, the popular open-source graph database used in LLMs to create and manage knowledge graphs, providing context to reduce hallucinations and helping teams better visualize complex data.

We also see a rise in the number of keys for Low-Code/No-Code tools like Airtable, Supabase, and Zapier. This rise correlates with the rise of “shadow IT,” where teams outside of traditional IT and development are introducing new platforms into the enterprise at an accelerating rate. Furthermore, **the increasing adoption of AI agents is fueling this trend, as AI drives the demand for and integration with low-code and no-code platforms, many of which also offer built-in AI capabilities**. Reports from Gartner and other researchers predicted this to be upwards of 50% of technology spending.

We know our findings are only the tip of the iceberg, as the majority of the code produced is not public on GitHub. There is an extremely large volume of code being produced inside organizations, and unfortunately, the data points to the fact people are paying the least attention when it comes to proper secrets management practices.

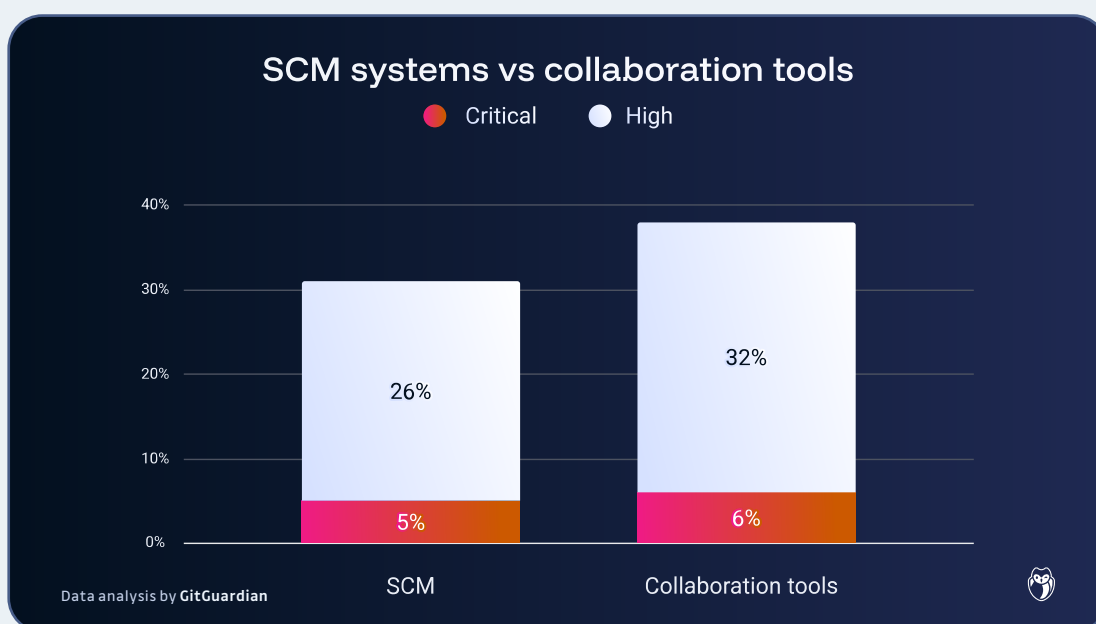
Mapping the SDLC: Where Leaks Happen

For years, source code management tools have been the primary focus of secrets detection. However, secrets appear wherever teams collaborate, often **in collaboration and project** management tools like Slack, Jira, or Confluence.

The rise of **low-code and no-code development** has only worsened the issue. More employees—many without security training—are handling secrets, increasing exposure risks. Unlike Source Code Management tools (SCM), collaboration platforms **lack built-in security controls**, and their **high message volume** makes real-time protection a challenge.

Collaboration Tools: The Overlooked Frontier of Secrets Sprawl

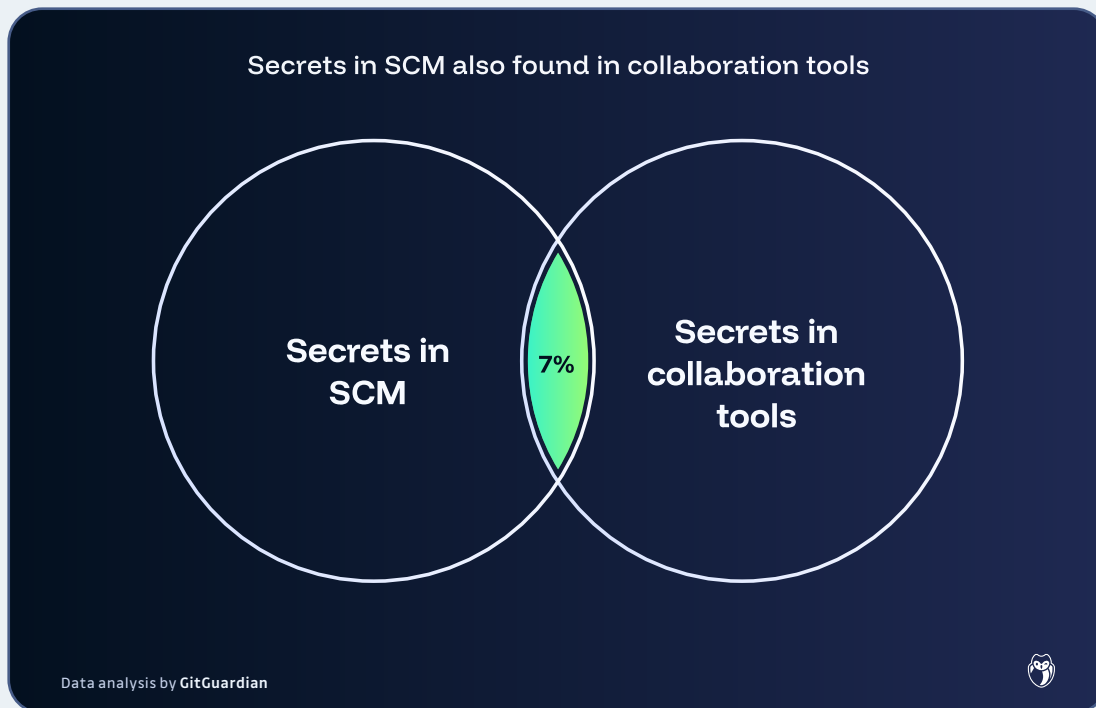
For the past 2 years, GitGuardian has been detecting secrets outside of source code. We observed the number of detected secrets grow quickly as customers expand detection coverage across their productivity platforms.



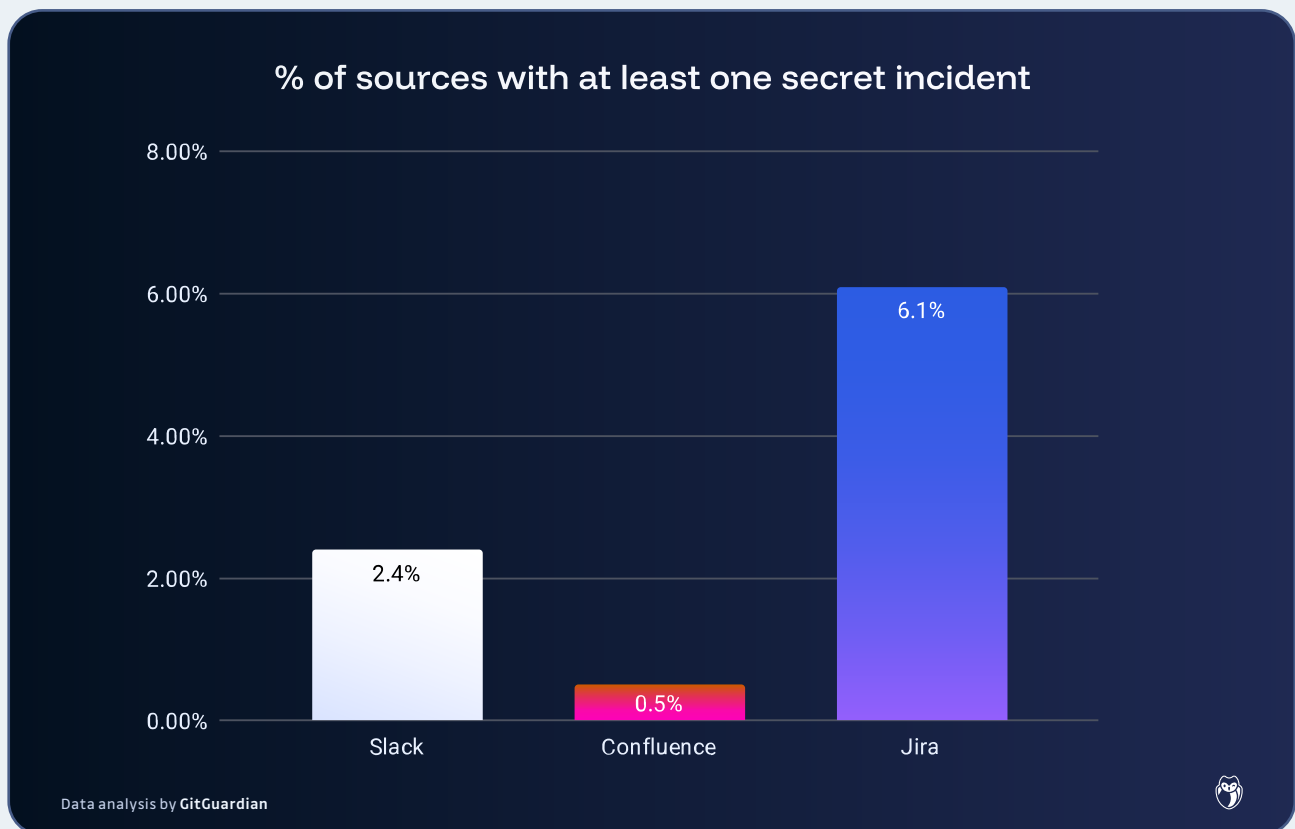
Secrets in collaboration tools are often **more critical than those in SCM tools—38% of incidents were classified as highly critical or urgent**, compared to **31% in SCM**. This happens because:

- **Developers are less cautious** with secrets outside SCM.
- SCM prevention mechanisms (e.g., shift-left security) reduce critical incidents, whereas **collaboration tools lack equivalent safeguards**.
- **Less security-aware employees** frequently handle secrets in these tools.

Secrets in SCM and collaboration tools are almost entirely distinct—only 7% of secrets appear in both. Organizations should address collaboration tools as a different security challenge.



Collaboration Tool	Leak Rate	Average Instances per Customer	Likelihood of Finding Secrets	Key Observations
Slack	2.4% of channels	1,500 channels per workspace	Very High – Most organizations likely have multiple exposed secrets.	Secrets appear frequently, often with multiple leaks per channel due to real-time messaging.
Confluence	0.5% of spaces	400 spaces per customer	Moderate – Some organizations may have a few exposed secrets.	Secrets are less common but still present, likely due to documentation misuse.
Jira	6.1% of tickets	150 projects per customer	High – Many organizations have some exposed secrets.	Highest leak rate. Secrets are frequently shared in tickets, probably for troubleshooting.



While SCM tools have built-in protections, productivity platforms remain a **growing attack surface**. This risk compounds as organizations use many different collaboration tools. Expanding detection capabilities beyond source code **is essential to increase coverage**, which improves the recall and therefore helps prevent breaches.

100,000+ Valid Secrets on Docker Hub

Hardcoded secrets continue to be common practice in cloud environments to simplify testing and deployment. Docker images, being a crucial part of modern cloud infrastructure, are not immune to this problem. Our comprehensive analysis of **15 million public Docker images from Docker Hub, the largest-scale scan of this kind ever conducted**, has revealed a concerning reality: approximately 100,000 valid secrets, including AWS keys, GCP keys, and GitHub tokens belonging to Fortune 500 companies, are currently exposed.

The detailed methodology of this research is described in the methodology section.

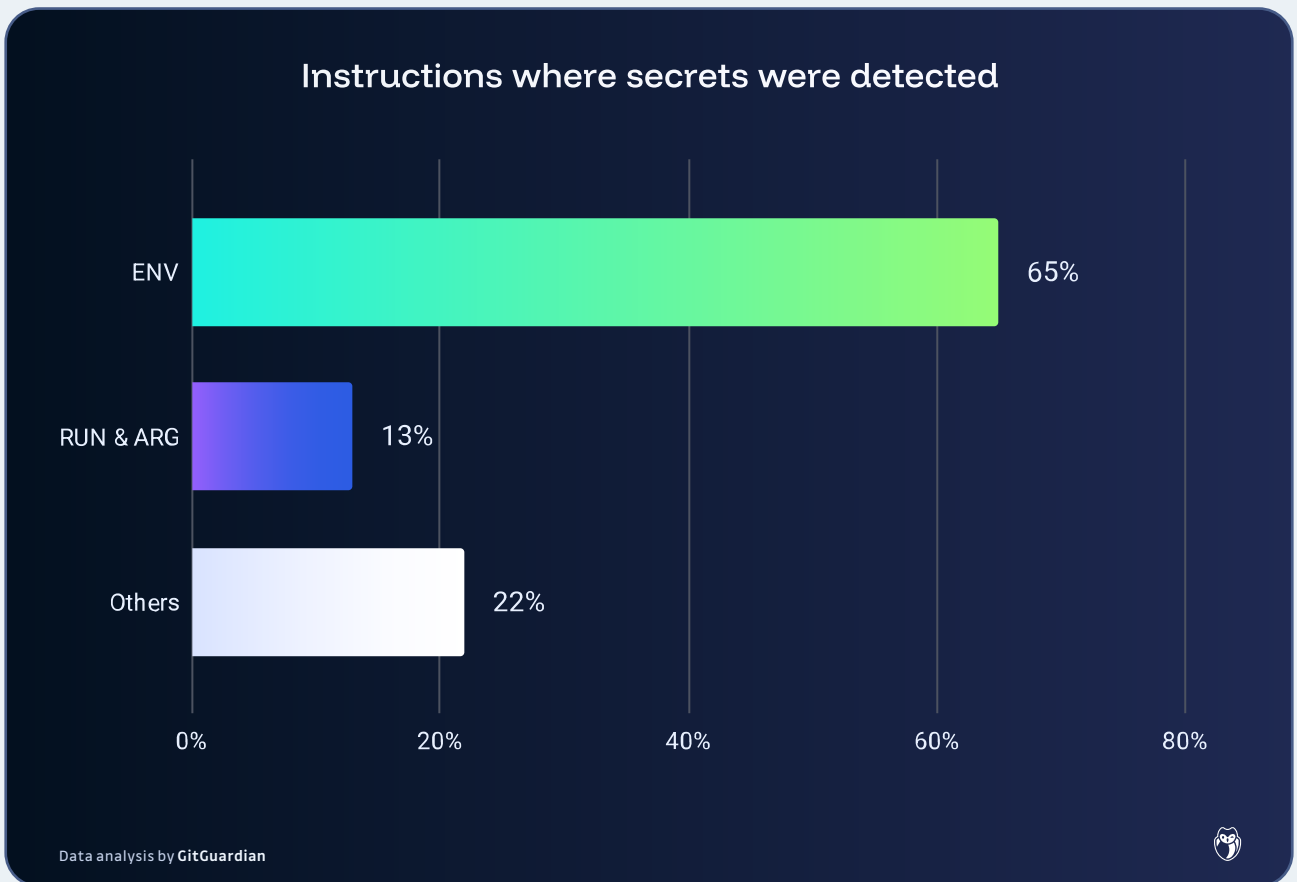
The analysis involved scanning 16 million layers across these images, representing over 30TB of data. This unprecedented investigation uncovered not just the volume but also the nature of exposed secrets on Docker Hub.

The State of Secrets Sprawl in Docker Images

DockerHub analysis revealed that approximately **98% of detected secrets were found exclusively in layers**, with most appearing in layers smaller than 15MB. This finding emphasizes the critical importance of detailed Docker image scanning, as configuration analysis alone proves insufficient for comprehensive security.

When examining the manifests where secrets were detected, a concerning pattern emerged:

- ENV, RUN & ARG instructions were responsible for 78% of secret leaks
- ENV instructions alone accounted for 65% of all leaks



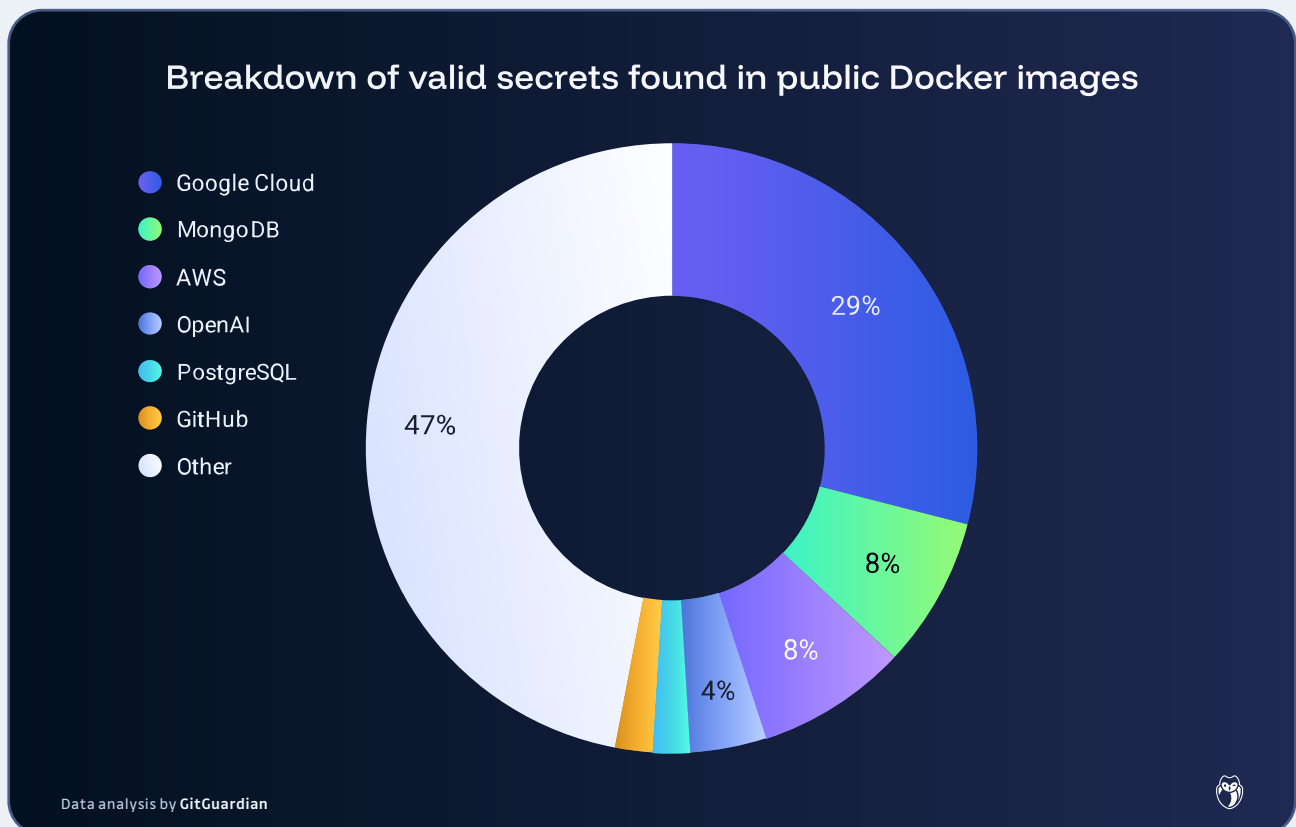
Valid Secrets Analysis

Out of 1,179,475 unique secrets detected, 101,186 were automatically verified as valid, with 97% found exclusively in layers. These secrets provide access to:

- Databases
- AWS infrastructures
- GitHub Enterprise instances
- Artifactory repositories

This represents exposure across 170,227 Docker images.

The nature of valid secrets in DockerHub notably differs from other platforms like GitHub. While platforms such as GitHub have established notification systems for key partners like Microsoft and OpenAI, resulting in quick invalidation of exposed secrets, DockerHub lacks such systematic protection. This gap in security is evident in the discovery of over 7,000 valid AWS keys still active in public images.



Risk Patterns

Among generic secrets, approximately **40,000 RSA private keys** were identified in the dataset.

This research underscores a **critical gap in container security** practices and highlights the need for more robust secret management in containerized environments. The findings suggest that current security measures and awareness around Docker image security need significant enhancement to address the sprawling exposure of sensitive credentials.

A Story From the Field

Docker image layers present unique challenges for secrets management. Analysis of a production image appeared initially secure, but examination of the image history revealed a concerning pattern:

```
$ docker history organization/application-production
# ...
<missing> 6 months ago /bin/bash -o pipefail -c rm -f
.npmrc || : 0B
```

This indicated an attempt to remove a credentials file after build operations. However, extraction of the appropriate layer revealed:

```
//npm.pkg.github.com/:_authToken=ghp_6e*****
registry = https://npm.pkg.github.com/
always-auth=true
```

This demonstrates a critical security flaw: even when files containing secrets are deleted in later build stages, the credentials remain permanently accessible in the underlying layers. Proper secrets management must account for the immutable nature of container layer history.

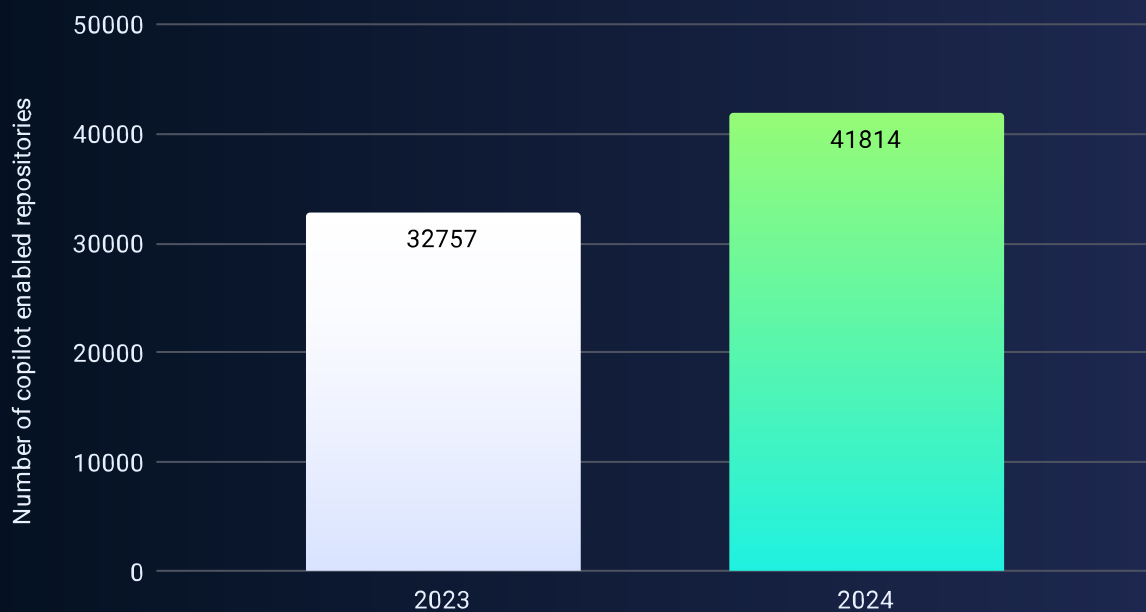
Copilot increases secrets incidence rate by 40%

2024 confirmed the progression of the AI topics. Most providers improved the performances of LLM models and this technology is finding more and more traction across all industries. Application development seems to follow this trend to the point where GitHub now offers Copilot as part of its free offering.

On the usage side, we observe an increase in the number of repositories using Copilot. Between 2023 and 2024, this number increased by 27%.

This tends confirm developers are more and more relying on AI tools to improve their productivity.

Number of repositories with Copilot enabled

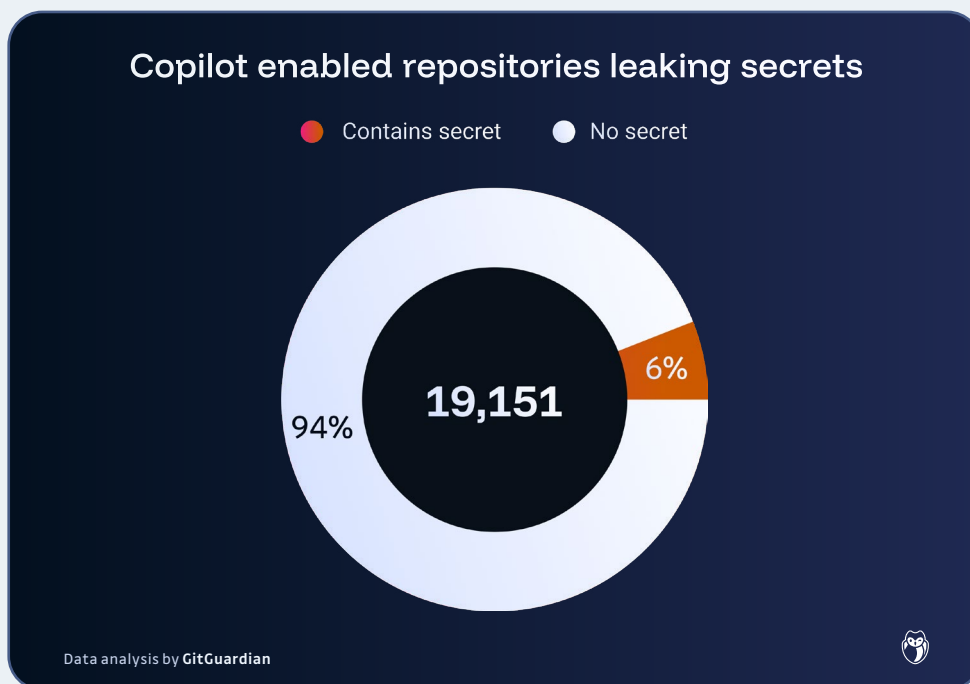


Data analysis by **GitGuardian**



At the same time, researchers across the globe studied the security of the code generated by LLM (examples: [1](#), [2](#) and [3](#)). While no study has specifically quantified the issue of secret leaks, GitGuardian's data supports the general assessment of insecurity.

We conducted an analysis of secrets leaked in repositories where Copilot is allowed and active. In a sample of approximately 20,000 such repositories, over 1,200 leaked at least one secret. This represents 6.4% of the sampled repositories.



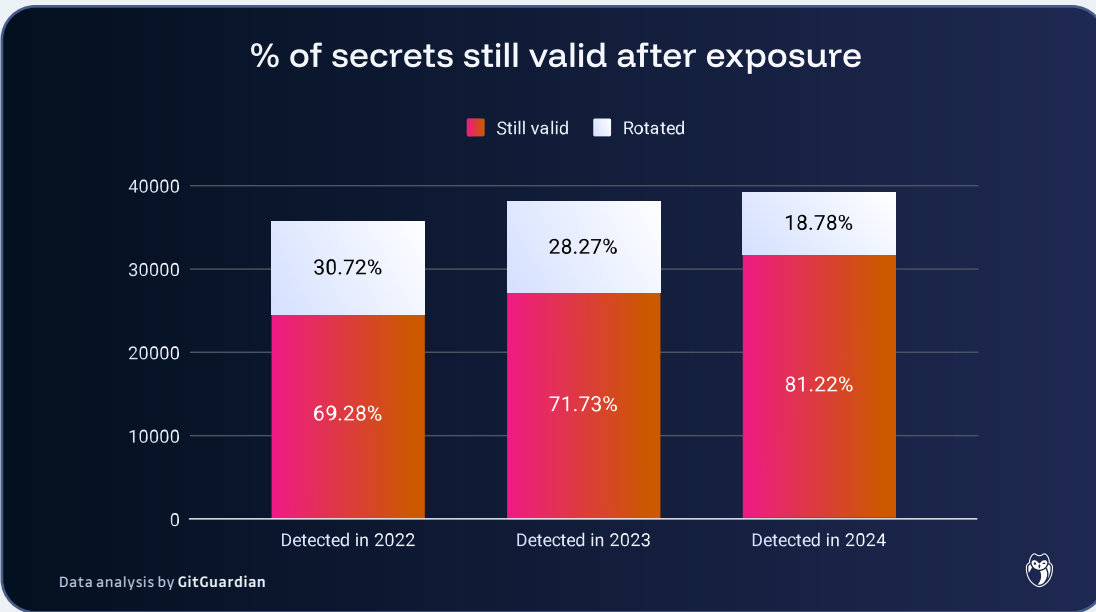
This incidence rate is **40% higher** than what we observe across all public repositories, which stands at 4.6%. Some people may feel the incidence is not that much higher, but in fact it means that AI does not deliver any improvement in terms of security in this matter.

This disparity can be attributed to two factors. First, the code generated by Large Language Models (LLMs) [may inherently be less secure](#). Second, and perhaps more significantly, the use of coding assistants may be pushing developers to prioritize productivity over code quality and security.

Despite the continuous improvement of coding assistants, this data underscores **the ongoing need for robust security controls**, particularly in the area of secret detection.

Detected but Not Fixed: The Alarming Persistence of Exposed Credentials

Detecting a leaked secret is just the first step. The true challenge lies in swift remediation.



Our analysis of over 115,000 initially valid secrets reveals a critical truth: a significant number remain active long after their initial detection. This underscores a fundamental challenge: **while secret detection capabilities have advanced significantly, the remediation process often lags behind.**

This alarming trend persists over the years. The continued validity of these secrets suggests that organizations lack visibility into leaked credentials or fail to enforce lifecycle management practices such as automated expiration and rotation.

This persistence of long-lived secrets is particularly pronounced in the context of **Non-Human Identities (NHIs)**, such as service accounts and API keys.

70%

of valid secrets detected in public repositories in 2022 remain active today



NHIs, often outnumbering human identities, rely heavily on credentials for authentication and access to resources. However, the long lifespan of these credentials significantly increases their vulnerability to exploitation, emphasizing the critical need for organizations to prioritize remediation efforts.

As organizations increasingly adopt automation, cloud computing, and microservices architectures, the number of NHIs will continue to skyrocket. This proliferation of identities will only increase the complexity of managing and securing these credentials, making it even more challenging to ensure timely remediation.

Several key factors contribute to the persistence of long-lived NHI credentials:

- Many NHIs lack proper lifecycle management, leading to long-lived and vulnerable credentials.

*“One organization reported that they want to be alerted when secrets tied to NHIs expire, but their current secrets managers don’t offer this functionality. As a result, they **provision keys with two-year lifespans**, increasing risk.”*

- Static credentials are inherently riskier than dynamic ones.

*“One organization shared that **migrating from static to dynamic secrets** is an objective, but it’s challenging to manage and track across systems.”*

- Organizations often fail to rotate secrets regularly, increasing exposure.

*“A security lead mentioned they aim to **rotate secrets annually**, but this process remains challenging to enforce.”*

- The proliferation of secrets across various environments hinders visibility and control.

*“A security team emphasized the importance of **segregating secrets between production and staging environments** to minimize cross-environment risks.”*

Addressing the above challenges requires a multi-faceted approach.

While organizations are increasingly adopting secrets management solutions and vaults, these tools alone **don’t solve the problem**.

Secrets Managers: Not a Complete Solution

Secrets managers are widely recognized as a best practice for secrets management and storage, but they can only be effective if they are consistently used and if their secrets are not extracted and hardcoded elsewhere.

Secrets sprawl and fragmentation occur as organizations juggle multiple secrets management tools across teams, including those provided by IaaS vendors, leading to a **decentralized environment**. This fragmentation makes it difficult to maintain a unified approach to NHI secrets governance, resulting in **inconsistent security practices** across teams and projects.

Insecure authentication to secrets managers is another risk. If credentials stored on the workload to authenticate with the secrets managers aren't properly secured, it undermines their entire purpose.

Moreover, while they excel at storing secrets securely, they typically do not manage the full lifecycle of credentials, leaving organizations to rely on additional tools. This gap complicates the management of **Non-Human Identities**, especially as organizations scale their infrastructure.

Our 2024 analysis of secrets managers usage across public GitHub repositories reveals a crucial insight: **secret leaks still occur, even in environments where secrets managers are available**.

Key findings from repositories utilizing secrets managers

We examined a sample of **2,584 repositories** where CI/CD configurations indicated the availability of a secrets manager. Alarming, **132 repositories** had leaked at least one secret in 2024, representing **5.1%** of the studied repositories.

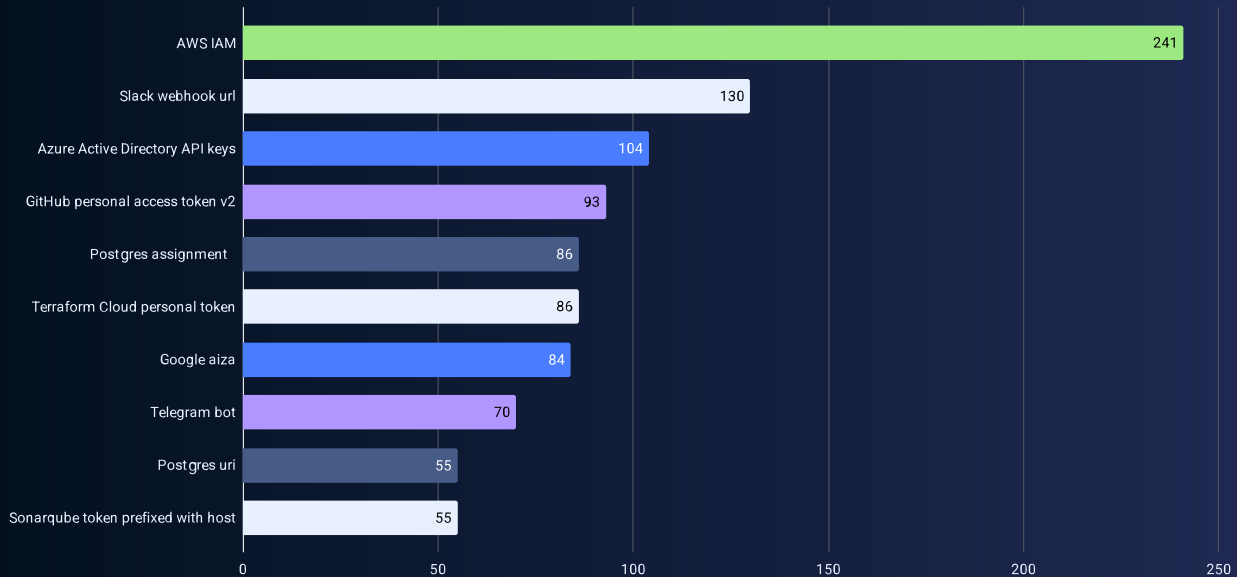


While repositories leveraging secrets managers show a slightly higher leak incidence, these numbers are **not directly comparable**. Such repositories are more likely to handle sensitive information, increasing the risk of exposure. In contrast, many public repositories may never deal with secrets at all.

The nature of leaks in repositories leveraging secrets managers

The types of secrets leaked in these repositories mirror those typically found in corporate environments. **This suggests that even organizations with strong security measures, such as those using secrets managers, are vulnerable to secret leaks.**

Leaked Secret types in repositories leveraging secrets managers



Data analysis by **GitGuardian**



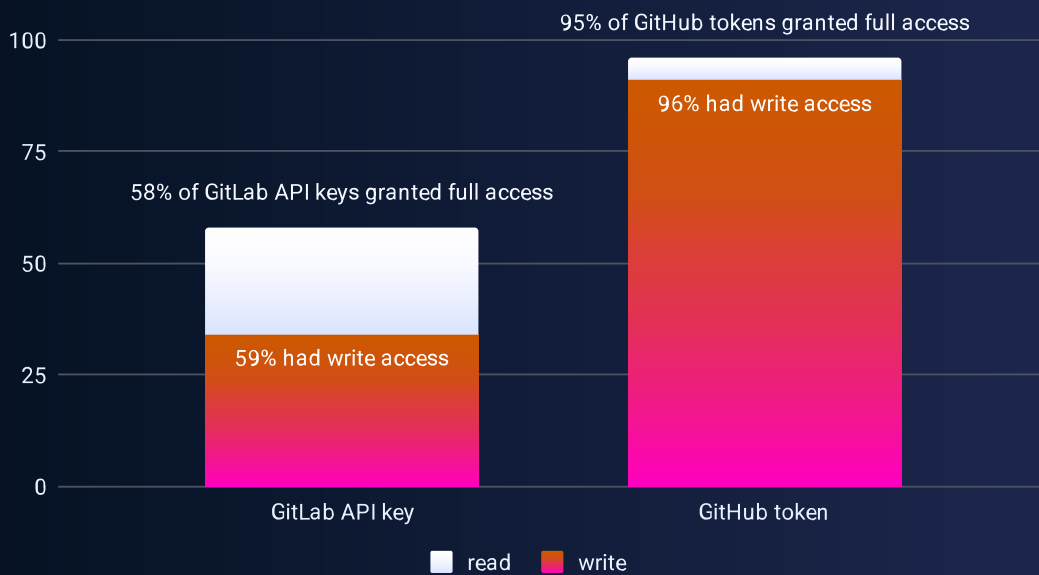
AWS credentials were the most commonly leaked, followed by **Slack webhooks** and **Azure AD API keys**—all tools frequently used in corporate settings.

Excessive Permissions Make Secret Leaks More Severe

Further compounding the issue, our analysis of public data highlights the **mismanagement of permissions** associated with leaked secrets. While examining GitLab and GitHub tokens, we discovered that a significant percentage had **excessive permissions**:

- **99%** of GitLab API keys had either **full access (58%)** or **read-only access (41%)**.
- **96%** of GitHub tokens had **write access**, with **95%** offering **full access** to repositories.

Excessive permissions granted to API keys and tokens



Data analysis by GitGuardian



Excessive permissions significantly amplify the potential impact of a compromised NHI, enabling attackers to move laterally across systems, escalate privileges, bypass security controls, and make it difficult to track or revoke access promptly.

This over-permission often stems from the **challenges developers face in managing permissions** during project work. When under time constraints, it's common for developers to grant all permissions “just to get things done,” with the intention of revisiting and tightening access later—a step that is frequently overlooked. This lack of caution can lead to dangerous exposures, such as leaving high-access keys embedded in code.

Many repositories are deleted or made private after a leak, creating “[zombie leaks](#)” where the secret remains exposed in the deleted repository.

Bridging the remediation gap

While dealing with non-human identities (even in environments utilizing secrets managers) the **remediation gap** remains a consistent challenge. To effectively close this gap, organizations must adopt a **holistic approach** that integrates detection with swift, automated remediation processes:

- **Treat secret leak remediation as a critical security objective** with the same urgency as detection.
- **Provide comprehensive developer training** on secure secrets manager usage and the risks of hardcoding secrets.
- **Integrate secrets discovery and security tools with secrets managers** to automate secret rotation and revocation.
- **Address secrets managers sprawl** by implementing strategies to centralize and consolidate secrets management solutions.
- **Implement tools that discover, track, and monitor NHIs** across the organization, giving deep insights into secret usage and dependencies.
- **Minimize the reliance on static secrets** and embrace principles like least privilege and just-in-time access.
- **Adopt “secretless” approaches** where possible to minimize the reliance on traditional secrets by exploring alternative authentication and authorization mechanisms.

Managing non-human identities and their secrets is an ongoing process that requires constant effort and adaptation.

Understanding the Impact: Real-World Risks of Secrets Sprawl

While the volume of leaked secrets continues to grow, the severity and potential impact of these exposures varies significantly across different types of credentials. Our research reveals that some of the most critical exposures aren't necessarily the most common ones.

Primary Risk Categories and Attack Vectors

There are several distinct categories of secrets that present significant risks when exposed. Each category exhibits unique attack patterns and potential impact levels, requiring specific attention and mitigation strategies.

Package and Container Registry Credentials

Compromising these credentials allows attackers to publish malicious packages or containers, overwrite legitimate artifacts, or access sensitive internal builds, leading to supply chain attacks or service disruptions. We also find credentials in such registries, which could be used to pivot deeper in the targeted infrastructure.

Observed attack patterns:

- Initial access through leaked credentials
- Repository enumeration
- Malicious package deployment
- Supply chain compromise
- Downstream system infection

Examples

- [Attackers built and executed Docker images bypassing detection in place](#)
- [Valid Kubernetes secrets are exposed in public repositories allowing attackers to conduct supply chain attacks](#)

Case Study: Artifactory Token Exposures

A recent investigation into Artifactory token leaks revealed a concerning pattern: while not among the top 10 most commonly leaked secrets, these credentials presented an unusually high-risk profile. Key findings include:

- 60% of leaks occurred in build configuration files
- Most leaked tokens were tied to production environments
- The majority of exposures were found in corporate contexts
- Critical sectors affected included pharmaceutical, energy, defense, and major tech companies

This pattern highlights how the criticality of leaked secrets isn't always correlated with their frequency of exposure.

Cloud Platform Credentials

These credentials often have broad access to cloud resources, including compute, storage, and networking. A compromise could lead to resource abuse such as cryptojacking, data exfiltration, or full service compromise.

Observed attack patterns:

- Resource creation for cryptojacking or similar revenue generation
- Data exfiltration operations
- Complete service compromise
- Infrastructure manipulation

Examples

- [The attacker uses compromised credentials to create computing resources, then use them to mine cryptocurrency](#)
- [The attacker uses compromised credentials to hijack victims' GenAI infrastructure and power their own disreputable LLM applications](#)

Storage Service Access

Access to storage accounts could expose sensitive data, including PII (Personally Identifiable Information), intellectual property, or backups, resulting in compliance violations or significant financial loss.

Observed attack patterns:

- Data exfiltration campaigns to connect to an Azure Blob Storage and enumerate the available containers
- Targeted PII collection operations
- Source code and IP theft
- Ransomware through backup encryption

Examples

- [Using valid credentials, the attacker use legitimate AWS S3 features to encrypt files contained in buckets, then access for a ransom to provide the decryption keys](#)

Source Code Management (SCM) System Credentials

SCM tokens can provide access to private repositories, exposing sensitive code, configurations, and additional secrets. Attackers can also manipulate repositories to inject malicious code.

Observed attack patterns:

- Source code and IP theft
- Supply chain code injection
- Secret mining from codebases
- Repository manipulation and deletion

Examples

- [Attackers stole valid credentials from GitHub, then uploaded a compromised Ultralytics Python module to PyPI](#)

Secrets Management System Credentials

A compromise of secret management tokens (Hashicorp Vault Token, LDAP, Active Directory...) allows attackers to retrieve and misuse all the sensitive secrets stored in a centralized location, amplifying the risk across systems and applications.

Observed attack patterns:

- Mass secret extraction
- Authentication system compromise
- Cross-system privilege escalation
- Organization-wide access exploitation

Database Credentials

Database credentials can provide unauthorized access to critical data. Attackers can steal, modify, or delete information, resulting in data breaches, operational disruptions, or compliance penalties.

Observed attack patterns:

- Data exfiltration campaigns
- Privilege escalation via misconfiguration
- Database schema manipulation
- Credential harvesting from stored data

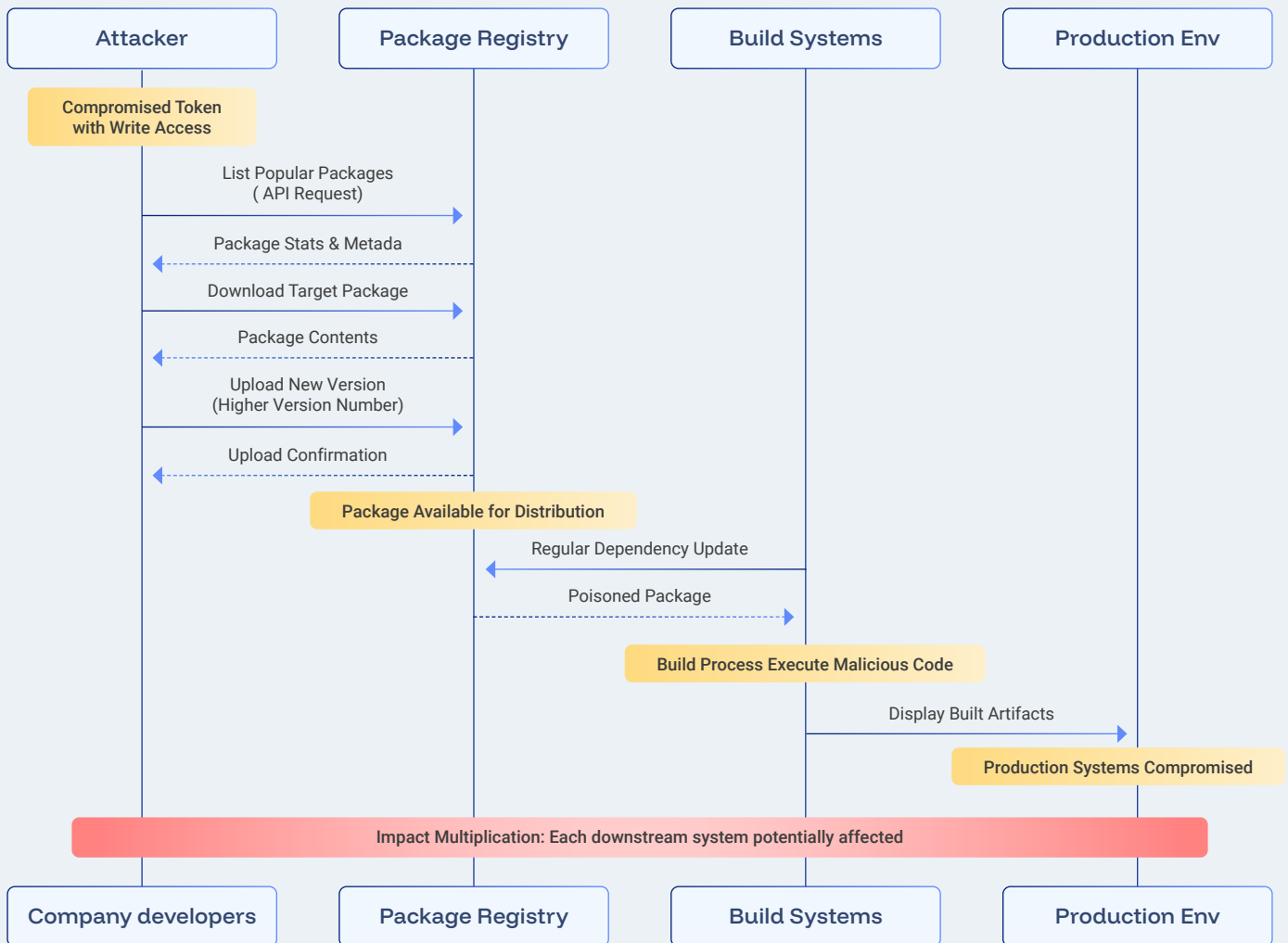
Examples

- [DeepSeek database containing sensitive data such as clients credentials](#)
- [Malware that targets PostgreSQL databases through brute force attacks, ultimately deploying cryptocurrency miners and gaining unauthorized system access](#)

The Cascade Effect: From Minor Leak to Major Breach

There is often a common pattern where seemingly limited access can cascade into critical breaches:

Initial access	Additional credential access	Business and technical impact
<ul style="list-style-type: none"> Discovery of leaked token Permission enumeration Resource mapping 	<ul style="list-style-type: none"> Additional secret discovery in artifacts Access to build configurations Service credential extraction 	<ul style="list-style-type: none"> Cloud resource creation Supply chain manipulation Infrastructure compromise



Critical Timeline Statistics

- 90% of exposed secrets remain valid after 5 days ([State of Secrets Sprawl 2024](#))
- Active exploitation begins within hours of exposure
- Supply chain attacks can affect thousands of downstream systems
- Average time to detection: 3+ months for self-detected leaks

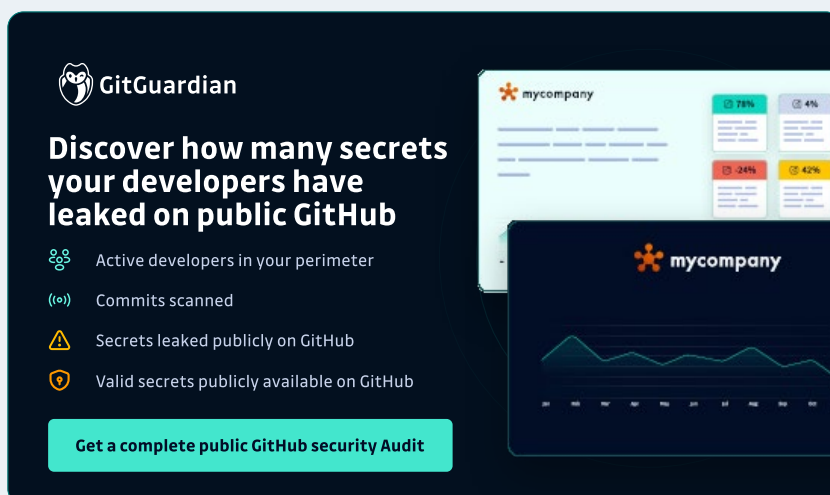
Risk Amplification Factors

DevOps Context	Personal Repository	Supply Chain Multiplication
DevOps tools are close to the production so are more likely to affect the software supply chain or the production environment	Corporate credentials in personal spaces	Single compromise affects multiple organizations
Credentials often have elevated privileges	Limited security oversight	Difficult to track impact scope
Automation requires broad access	Difficult to track and monitor	Complex remediation process
Build processes touch multiple systems		

Secret leaks rarely remain isolated incidents. Instead, they typically serve as entry points for sophisticated attack chains that can compromise entire organizations and their supply chains. This reality demands a shift from simple secret detection to comprehensive secret lifecycle management and rapid incident response capabilities.

About GitGuardian

GitGuardian is an end-to-end NHI security platform that empowers software-driven organizations to enhance their Non-Human Identity (NHI) security and comply with industry standards. With attackers increasingly targeting NHIs, such as service accounts and applications, GitGuardian integrates Secrets Security and NHI Governance. This dual approach enables the detection of compromised secrets across your dev environments while also managing non-human identities and their secrets lifecycle. The platform supports over 450 types of secrets, offers public monitoring for leaked data, and deploys honeytokens for added defense. Trusted by over 600,000 developers, GitGuardian is the choice of leading organizations like Snowflake, ING, BASF and Bouygues Telecom for robust secrets protection.



GitGuardian

Discover how many secrets your developers have leaked on public GitHub

- 👤 Active developers in your perimeter
- 📄 Commits scanned
- ⚠️ Secrets leaked publicly on GitHub
- 📍 Valid secrets publicly available on GitHub

[Get a complete public GitHub security Audit](#)

mycompany

78% 4%

24% 42%

mycompany

Line graph showing trends over time.



1

#1 APP ON THE **GITHUB** MARKETPLACE

Appendix

Definitions

Secret

A secret is any sensitive data we want to keep private. When discussing secrets in software development, we refer to digital authentication credentials that grant access to services, systems, and data. These are most commonly API keys, username and password combos, or private keys. In this report, secrets refer to credentials hard coded in plaintext.

Occurrence

When our detection engine detects a hard-coded secret, it becomes an occurrence. A single incident generally encompasses multiple occurrences: the various locations across files or repositories where the secret was identified. Occurrences map to the magnitude of the sprawl and correlate to the amount of work needed to redistribute the secret after it has been rotated. Occurrences are similar to technical debt.

Detector

At GitGuardian, a detector is a set of rules that filter documents for secrets. Beyond simple regex patterns, detectors are a sophisticated blend of pre- and postvalidation processes engineered for optimal speed, precision, and recall. These steps are performed using a combination of regular expressions and heuristics (e.g., variable assignments) based on contextual information. GitGuardian detectors boast features like automatic deduplication and the ability to recognize prefixed and base64 encoded secrets, such as Kubernetes secrets. GitGuardian developed the vastest library of specific detectors to detect more than 450 types of secrets. You can find the exhaustive list [here](#).

Specific detector

Specific detectors are designed to detect secrets for a specific provider, for example, the AWS IAM detector will only detect AWS IAM secrets. They offer high recall and precision, meaning they will rapidly catch all specific secrets while raising a low number of false alerts.

Generic detector

Generic detectors are designed to catch a broad variety of secrets that cannot be tied to a specific service. For example, the generic password detector aims to catch any strings assigned to a password variable. This broad scope inherently increases their susceptibility to generating false positives, or overlooking true positives, unless meticulous attention is given to fine-tuning their parameters. To strike a balance, GitGuardian's secrets detection engine accepts a false positive rate of approximately 20% as a trade-off for achieving high recall.

Secret incident

A secret incident is a uniquely identified security event determined to impact the organization and necessitates remediation. An incident often has multiple occurrences across files or repositories.

Non-human identities (NHIs)

NHIs are machine-based credentials that allow API integrations and automated workflows, which require machine-to-machine communication. These include API keys, service accounts, certificates, tokens, and roles, which collectively enable the scalability and efficiency required in modern cloud-native and hybrid environments.

Methodology

Study Perimeter

To ensure that the data presented here most accurately represents the state of secrets sprawl, and particularly to eliminate as many generic false positives as possible, filtering was applied to the data collected in 2024.

The filter is applied per detector and determines whether a check is necessary to count the secrets reported by this detector. If so, the secret is only counted if the first occurrence of the secret was valid. Otherwise, the secret is directly included in the count.

This method significantly boosts the precision of some detectors designed to accept a margin of false positives to avoid compromising detection recall. For instance, the GitHub Access Tokens detector employs a broad regex pattern prone to flagging numerous false positives. Hence, this detector underwent filtering. Detectors like AWS keys, which rely on specific prefixes, already exhibit high precision and do not require this additional filtering step.

Beyond the filtering process, we also manually excluded outliers—repositories exhibiting abnormally high leak rates, where a secret might be committed every minute—from this defined perimeter to ensure the integrity and accuracy of our metrics.

[Learn how our machine learning models, such as the one powering FP Remover, identify and validate more generic secrets.](#)

Docker Analysis Methodology

Our investigation into Docker Hub covered 9.3M unique repositories from 3.2M users, focusing on the 5 most recent tags per repository. After filtering through 198M Docker instructions totaling 4.8PB, we analyzed 15M layers (30TB) over three weeks using ggshield, GitGuardian's CLI tool. The detected secrets were validated through automated service checks, API responses, and cross-referencing with public key databases where possible.

Copilot usage analysis

We analyzed a sample of 19,151 public repositories for which we identified Copilot was in use. To find those repositories, we relied on development environment configuration files hosted on the repository.

The Dev Container extension allows the configuration of the Visual Studio Code IDE similar to a Dockerfile. The activated editor extensions can be configured with it.

```
{
  "name": "codespace",
  "customizations": {
    "vscode": {
      "extensions": [
        "GitHub.copilot"
      ]
    }
  }
}
```

We used this file to select repositories when the github.copilot extension was configured. The secrets in those repositories were then analyzed following the methodology for all repositories.

Secrets managers usage analysis

We analyzed a sample of 2,584 public repositories for which we identified a secrets manager was available. To find those repositories, we relied on the CI/CD configuration files when those indicated secrets were pulled from a secrets manager at build time.

We relied on the available configurations for:

- GitHub actions
- Azure Pipeline
- Circle CI

Other CI/CD were also studied with less relevance and volume.

For each of those technologies, we identified how the configuration of the secrets manager occurred for a list of enterprise technologies:

- HashiCorp
- Azure Key Vault
- Conjur
- AWS secret manager
- Akeyless
- Google Cloud Secrets Manager secret manager
- Bitwarden
- Infisical
- Doppler

That way, we were able to identify repositories using each of those providers. For example, repositories using GitHub actions with Hashicorp vault can be identified because they use the [hashicorp/vault-action](#).

The secrets in those repositories were then analyzed following the methodology for all repositories.

Analysis of Leaked Secret Permissions

This study analyzed the permissions associated with leaked GitLab and GitHub API tokens. For both platforms, the study used their respective API functionality to retrieve the scopes of each token, allowing for the determination of read-only versus write access. The results were then tabulated and analyzed to calculate the percentage of tokens with different access levels.



The State of **Secrets Sprawl** 2025

DATA ANALYSIS BY GITGUARDIAN

Learn more at www.gitguardian.com

© 2025 GitGuardian. All Rights Reserved.