THE STATE OF

SECRETS SPRAWL 2024

GitGuardian

TABLE OF CONTENTS

The State of **Secrets Sprawl 2024**

DATA ANALYSIS BY GITGUARDIAN

12,778,599 +28%

NEW secrets detected

IN PUBLIC GITHUB COMMITS IN 2023

IT, education, science & tech services were the industries **most affected** by secrets sprawl in 2023

See Industry leaks

UNIQUE SECRETS DETECTED

3,698,686 +25%

More than 90% of the secrets remain valid **5 days after being leaked**

See What Happens After a Public Leak?

1.8 Million

pro-bono alert emails

7 out of 1K

commits exposed at least one secret

>1 in 10

commit authors leaked a secret

3 Million

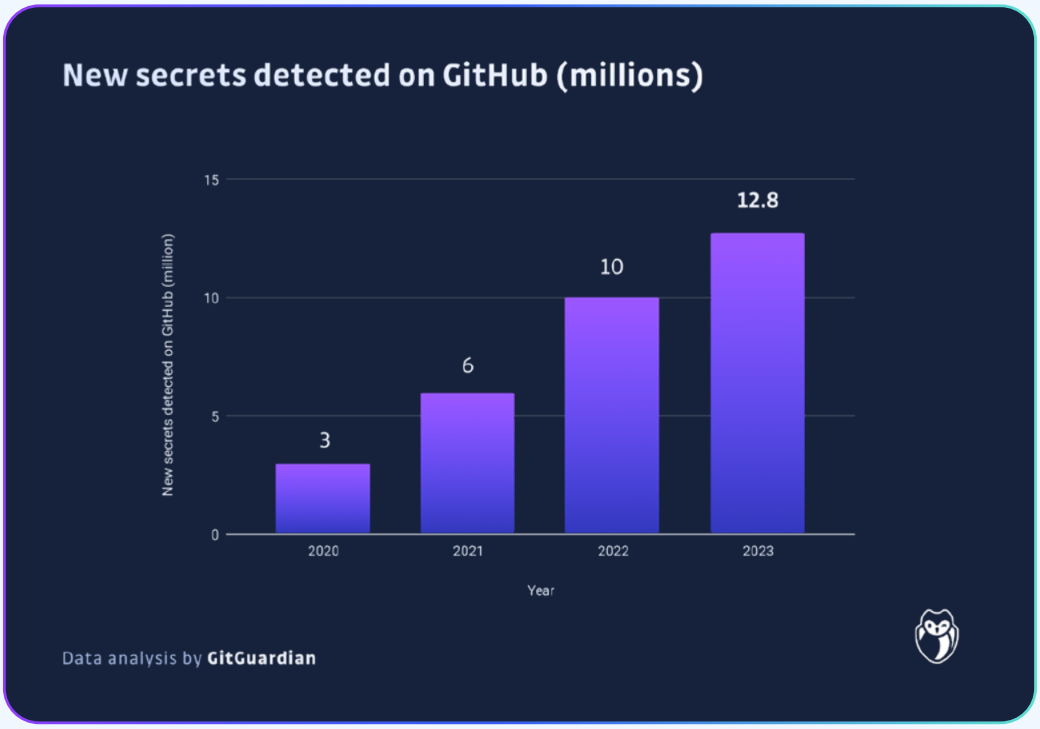
repositories leaked a secret

GitGuardian

Foreword

It is not a secret. Hard-coded credentials have long been a primary cause of security incidents in the software world. Yet, with the growing complexity of digital supply chains, secrets sprawl is the Achilles' heel for organizations of all sizes and security postures.

GitGuardian has been at the forefront of identifying and reporting hard-coded secrets for the past four years. Remarkably, the incidence of publicly exposed secrets has quadrupled in this time, with a **staggering 12.8 million occurrences** detected on GitHub.com in the last year alone—a 28% increase from 2022.



New secrets detected on GitHub (millions)

Data analysis by **GitGuardian**

> "[In 2023] for the first time, compromised credentials took the top spot in root causes [of attacks]. In the first six months, compromised credentials accounted for 50% of root causes, whereas exploiting a vulnerability came in at 23%."
>
> Verizon's 2023 Data Breach Investigations Report

> “49% of breaches by external actors involved Use of stolen credentials, while Phishing made up 12% of external attacks. Attackers used the Exploit vulnerability technique in 5% of breaches.”
>
> Sophos’2023 Active Adversary Report

The proliferation of 50 million new code repositories on GitHub, a 22% increase from last year, amplifies the risk of both accidental exposures and deliberate malicious acts. This reality underscores the vital need for companies to track and manage the exposure of their sensitive information. Too many remain vulnerable to breaches without awareness or means to mitigate them.

Our research sheds light on a concerning trend: **90% of exposed valid secrets remain active for at least five days after the author is notified.** This finding emphasizes a crucial lesson in code security: **while detecting vulnerabilities is critical, the real challenge lies in remediation.** Security, we believe, must be a shared responsibility across all stages of the Software Development Life Cycle (SDLC), not just the domain of specialized teams. Raising awareness about these seemingly minor lapses is essential for mitigating supply chain risks.

When 507 IT decision-makers were asked, “Have you ever been impacted by, or heard of secrets (API keys, username and passwords, encryption keys, etc.) leaking within your organization?” the responses highlighted widespread concern:

- **75% of respondents reported experiencing a secret leak.**
- **60% reported leaks impacting the company or its employees.**
- **47% identified “Hard-coded secrets” as key risk points in their software supply chain.**

Voice of Practitioners: the State of Secrets in Appsec

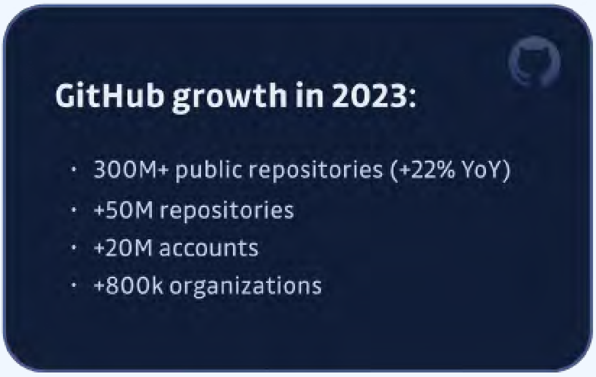
As our world becomes increasingly digital, and as secrets continue to underpin the trustworthiness of digital systems, closing the remediation loop is imperative for securing a safer digital future.

How Leaky Was 2023?

All the metrics featured in this report have been meticulously filtered to ensure they precisely depict the current state of secrets sprawl. For a comprehensive understanding of our methodology, please refer to the appendix.

4.6% of active repositories leaked a secret in 2023
On 67,243,678 active repositories, 3,066,304 leaked a secret.

More than 1 in 10 commit authors leaked a secret
Out of the 14,978,367 distinct authors who contributed, 1,749,398 (11.7%) leaked a secret.



GitHub growth in 2023:

- 300M+ public repositories (+22% YoY)
- +50M repositories
- +20M accounts
- +800k organizations

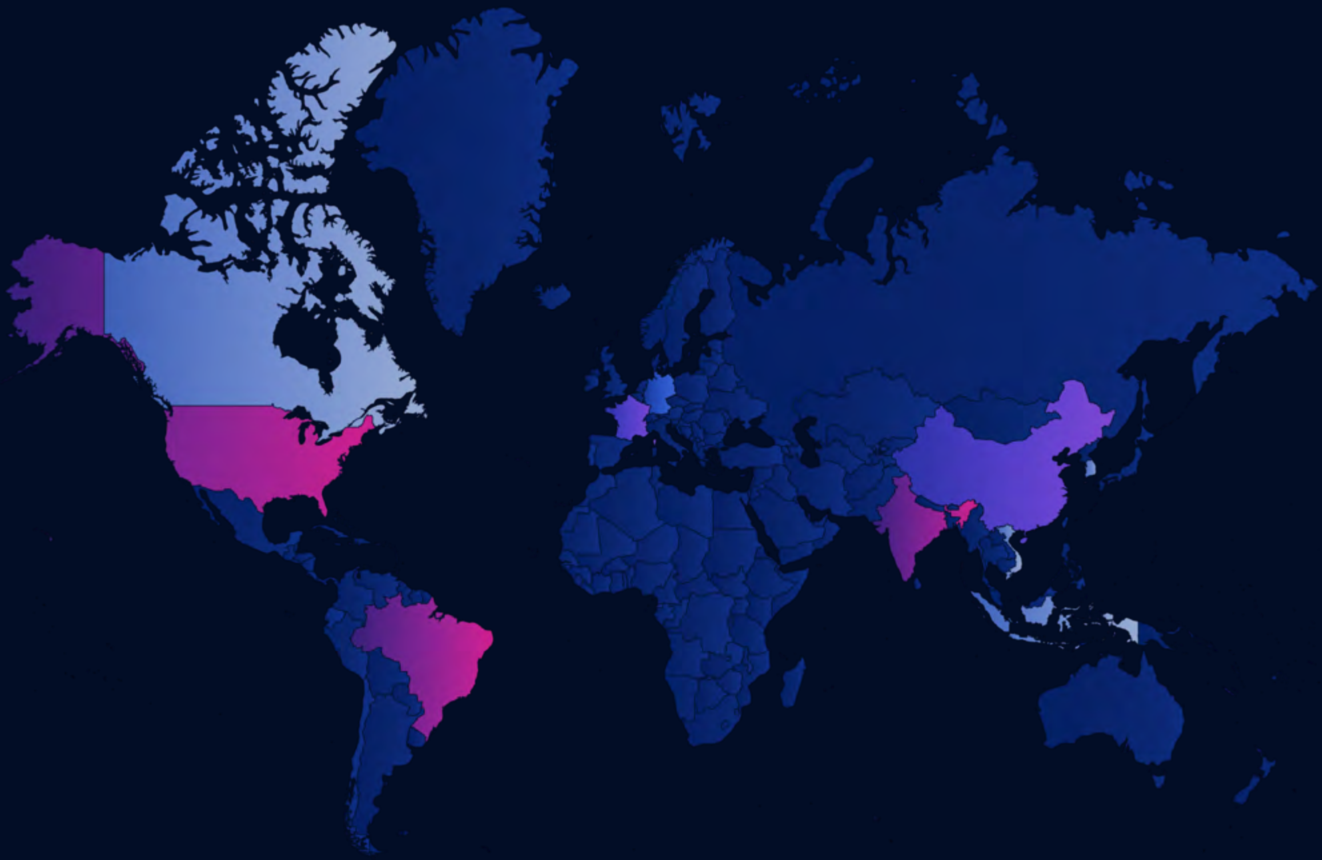
1.8M pro-bono alert emails (+23.5%)
GitGuardian sent 1,854,834 pro-bono alert emails following the detection of an exposed secret for the first time. Out of these, only 33,242 secrets were revoked within 5 days. GitGuardian stops tracking the status of secrets after these 5 days.

2023 Map of Leaks

THE 10 COUNTRIES WITH THE MOST LEAKS

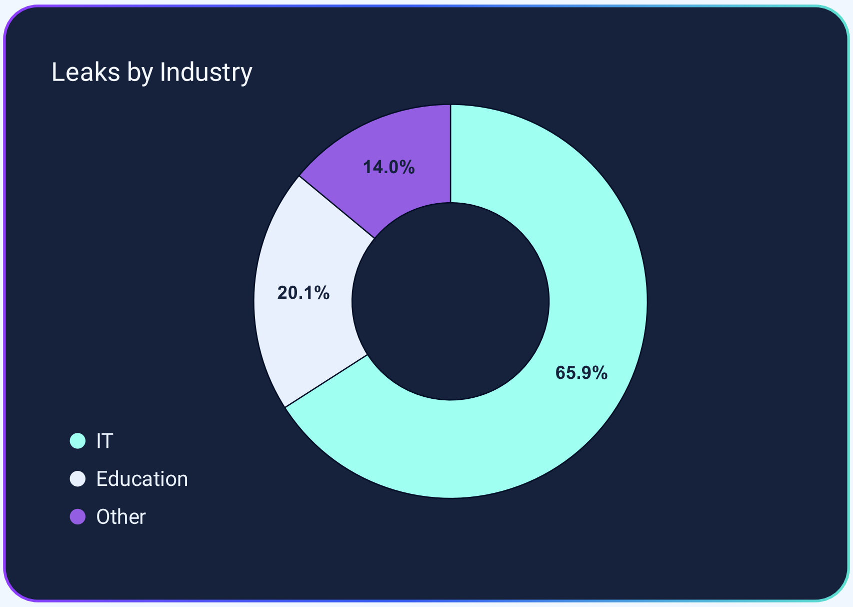
For GitHub profiles mentioning location.

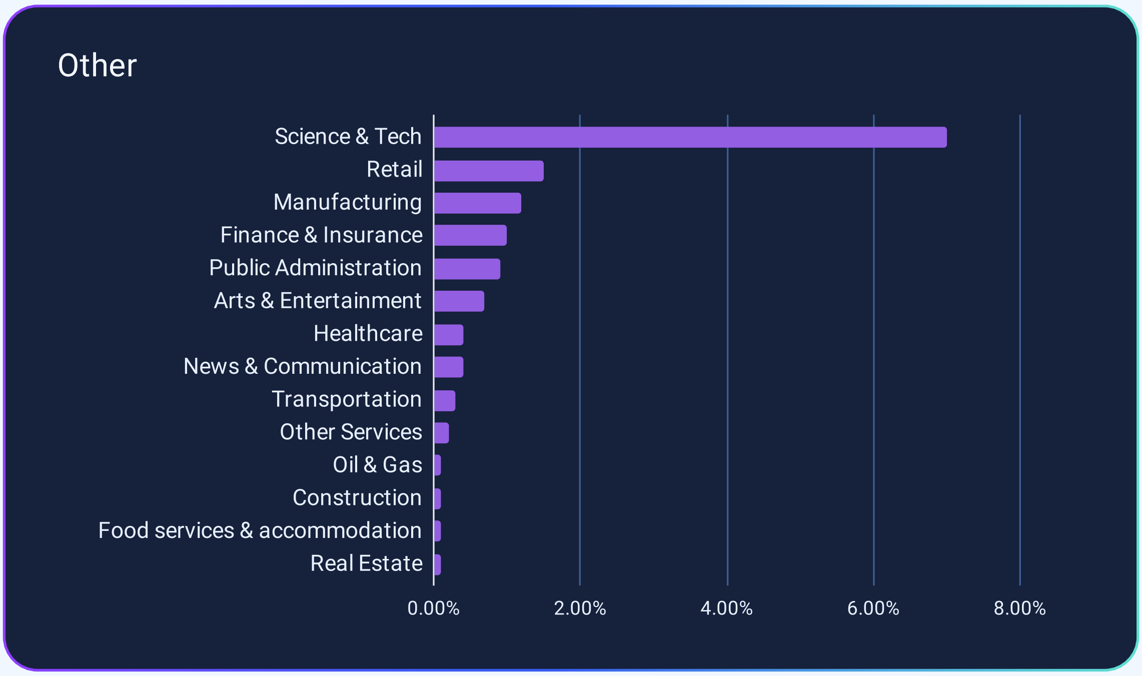
01	=	India	06	NEW	Canada
02	︽	United States	07	NEW	Vietnam
03	︽	Brazil	08	NEW	Indonesia
04	︾	China	09	︾	South Korea
05	︽	France	10	︾	Germany



Industry Leaks

Leveraging its advanced proprietary algorithm, GitGuardian has successfully traced many secret occurrences back to their respective companies.
This capability extends to incidents where secrets were leaked outside the company-owned repositories. Through this sophisticated mapping process, GitGuardian provides insightful data on the prevalence of leaks across different industries:





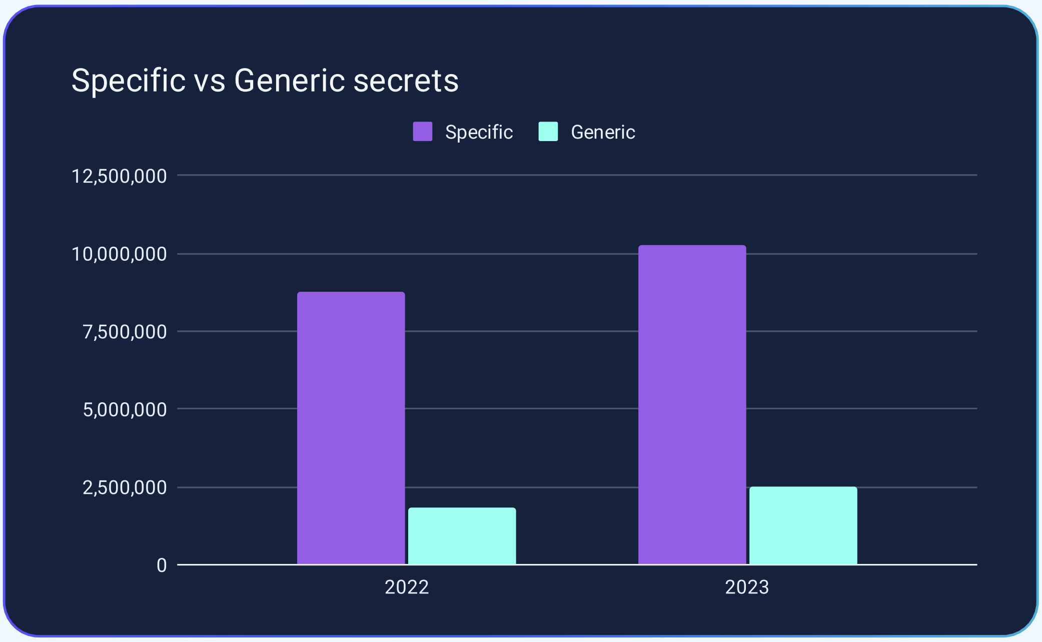
Unsurprisingly, the IT sector, which includes software vendors, accounts for 65.9% of all detected leaks.
Following IT, the education sector is responsible for 20% of the leaks, reflecting the growing digitization and reliance on technology within academic institutions.
The remaining 14% of leaks are evenly divided between the Science & Technology fields and other industries.

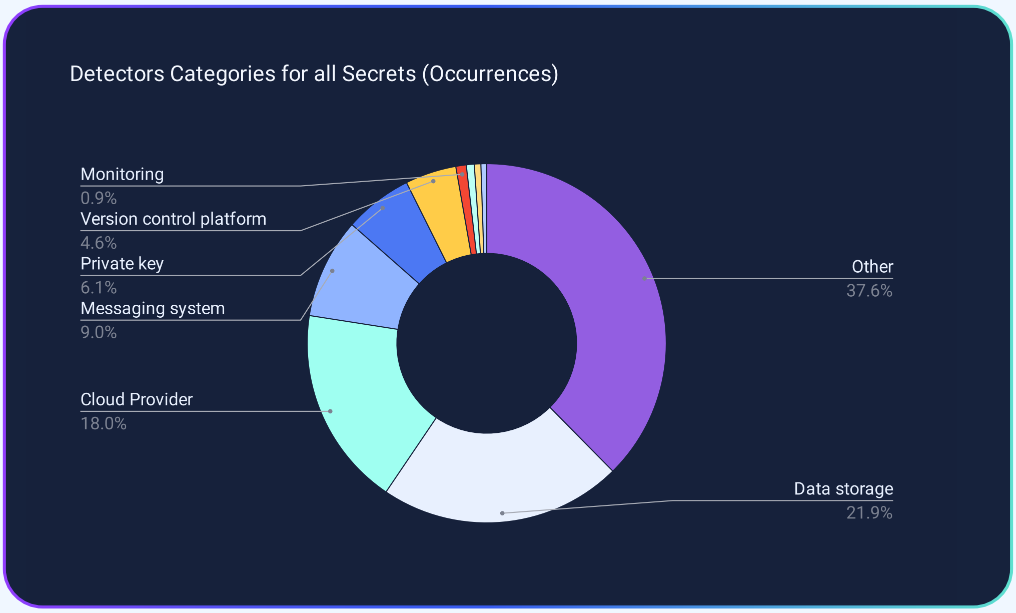
This distribution highlights a crucial aspect of our modern economy: digitization is pervasive, and virtually every industry now operates with a significant online presence, often utilizing platforms like GitHub for development and collaboration.

Consequently, the potential for secrets to be exposed on such platforms is a universal challenge, extending far beyond the realms traditionally associated with high cybersecurity risks.

Secrets Detectors

GitGuardian differentiates specific secrets from generic ones (see the definitions in the appendix). In 2023, specific occurrences were up **16.7%**, while generic occurrences were up **37%**.

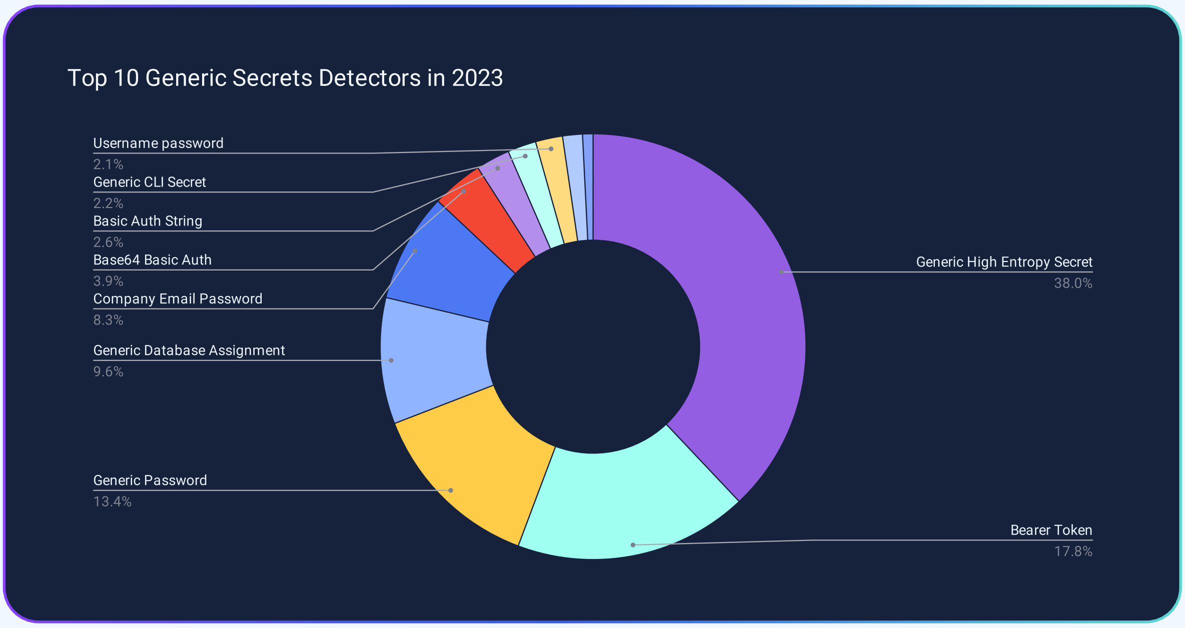




GENERIC DETECTORS

For the purposes of this report, we intentionally omitted a certain percentage of generic secrets. Consequently, the ratio of specific to generic detectors does not precisely represent the prevalence of generic secrets in our overall detection results.

> Indeed, as **Pierre Lalanne**, Engineering Manager at **GitGuardian**, notes: "Overall, generic detectors account for 45.4% of all the secrets we detect."

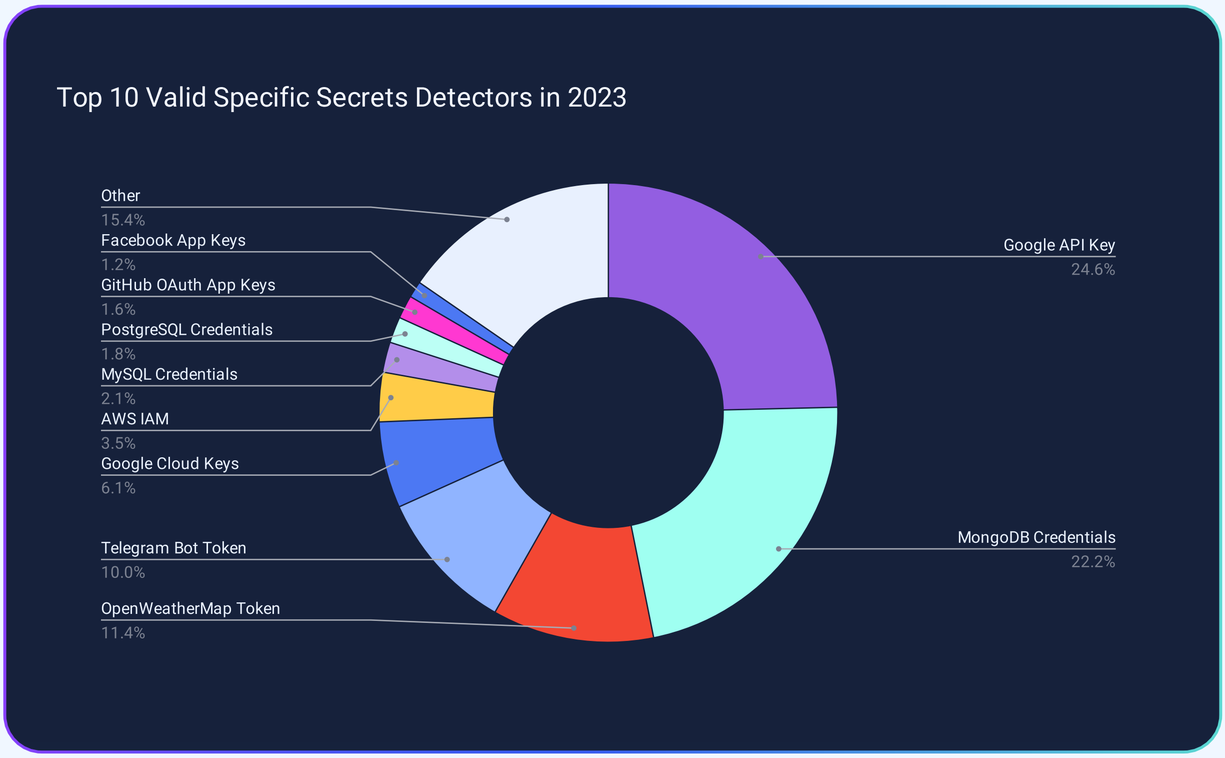


💡 Leveraging a promising machine learning algorithm, the GitGuardian AI/ML team was able to categorize Generic High Entropy and Generic Password secrets.

Read more in Powering Secrets Detection with AI: GitGuardian’s Approach

SPECIFIC DETECTORS

Looking specifically at specific detectors, we found **over 1M valid Google API secrets (occurrences), 250K Google Cloud, and 140K AWS IAM.**



Top 10 Valid Specific Secrets Detectors in 2023

🦉 **Fun Fact:**

Ironically, GitHub became its own cautionary tale when it inadvertently leaked its RSA SSH host key for GitHub.com right on GitHub.com! GitHub caught it quickly and replaced the key to ensure operations remained secure. It just goes to show, nobody’s immune to the occasional slip-up, not even the experts!

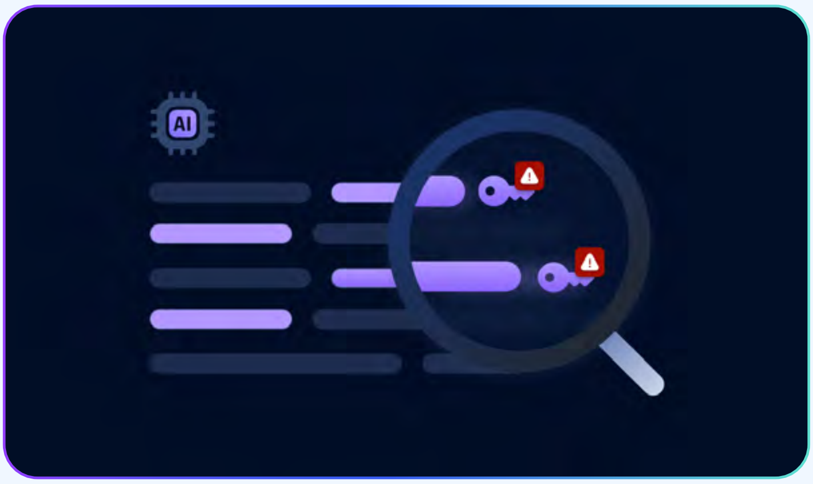
A UNIQUE CASE: OPENAI KEYS

OpenAI, the powerhouse behind the widely recognized ChatGPT, DALL-E, and the newer Sora generative AI tools, has seen its services soar in popularity well beyond the tech world. It's hardly surprising that API keys for programmatically accessing these tools have proliferated across GitHub, becoming a common sight.

> In 2023, GitGuardian observed a **1212x increase** in the number of OpenAI API key leaks from previous year, unsurprisingly making them the highest-growth detector.

As we move into 2024, the landscape of AI services has dramatically expanded, with key players from both established tech giants and innovative startups competing to deliver advanced inference capabilities across diverse formats, including text, audio, and video. These services also extend to ancillary offerings like embeddings (pre-processed inputs for Large Language Models) and persistence solutions for storing these embeddings in **vector databases**.

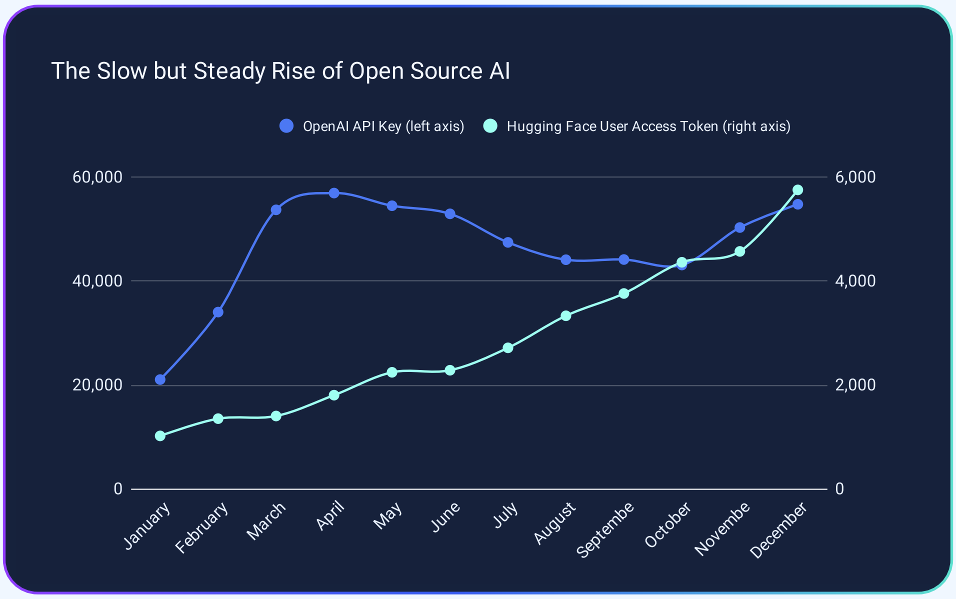
Prompted by this evolution, we explored the idea of using the frequency of secret leaks as an indicator of a service's popularity.



Focus: GenAI Secrets Leaks

OpenAI remains the most utilized AI service, evidenced by the alarming average of **46,441 API key occurrences leaked each month in 2023.** This figure significantly surpasses the leakage rates of other AI services we monitored throughout the year, such as Cohere, Claude, Clarifai, Google Bard, Pinecone, and Replicate.

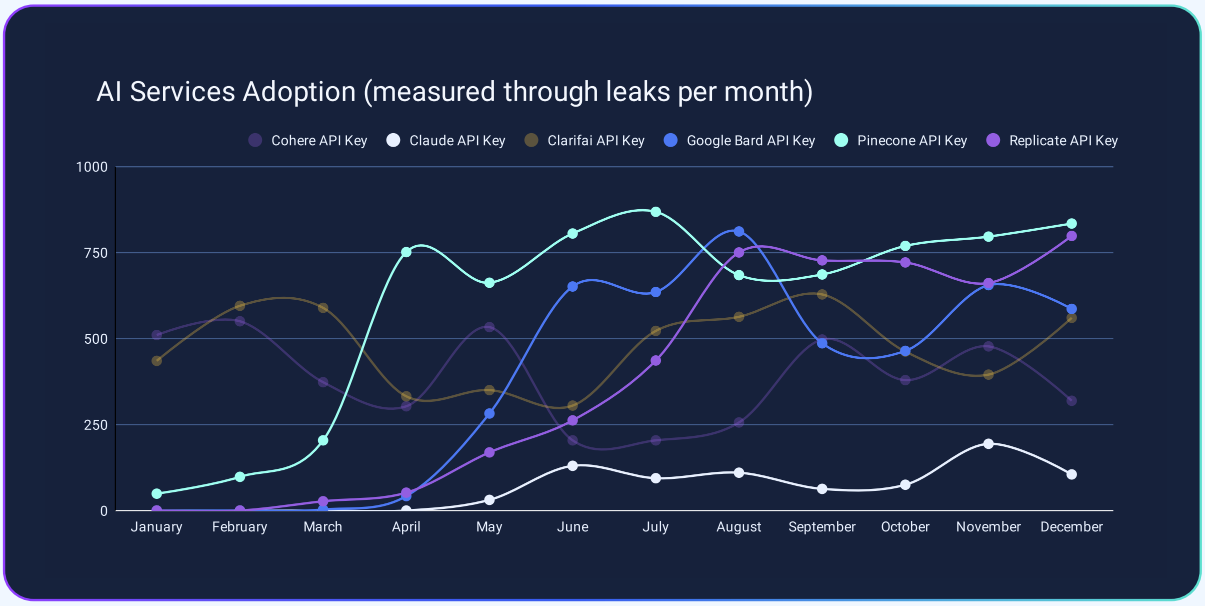
The leaks curve shows a sharp increase at the start of the year, reaching a peak in April, followed by a gradual decline from May to October. While this reduction is a positive development for application security, it may inversely reflect the perceived popularity of OpenAI during this period.



While OpenAI leads by a wide margin in the number of leaks, more and more tokens used to access HuggingFace open-source models have been seen on GitHub month after month, hinting at a growing interest in open-source AI among developers, albeit still overshadowed by OpenAI's dominance.

Our analysis also highlights the rapid penetration of services like Gemini (Google's ChatGPT alternative, formerly known as Bard and introduced at the end of March), Pinecone (a vector database service), Replicate (AI models-as-a-service), and to a lesser extent, Claude (an AI assistant by Anthropic), Cohere, and Clarifai.

By assessing the spread of secret leaks, it is easy to discern the rising use of these AI services.



GitGuardian proactively tracks the emergence of new secret types on GitHub, continually updating its detection tools to match the latest trends in secrets sprawl. This vigilant approach ensures its scanning engine remains effective against evolving security threats.

If you're a software provider, we encourage you to reach out for the development of bespoke detectors tailored to your service's specific needs:

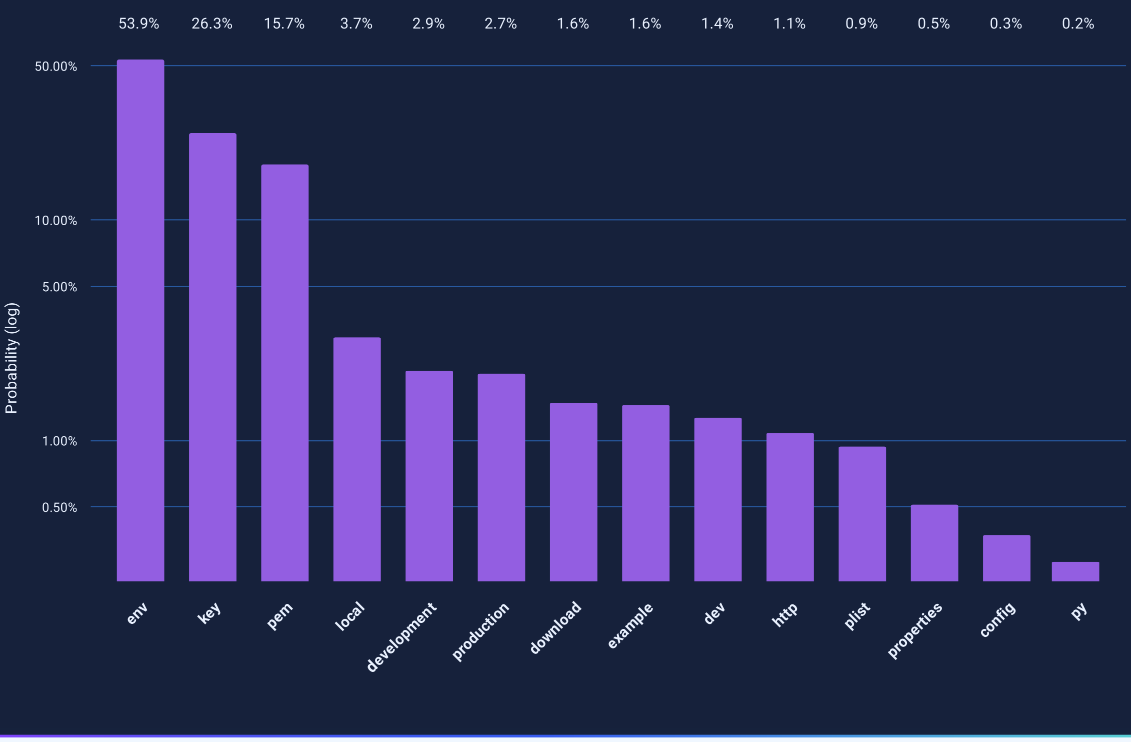
Request a custom detector from GitGuardian for your service.

Ranking File Extensions by Their Leakiness

To better understand which file types are most likely to leak secrets, we adopted a risk-based approach. The core of this method involves calculating the likelihood that a file with a specific extension will expose a secret. Two key pieces of information were leveraged for this analysis: the prevalence of each file extension on GitHub and the frequency with which each type appears in incidents involving leaked secrets.

The result is a **probabilistic risk score** for each file extension (see the Methodology section for a deeper dive).

File extensions probability to expose a secret



For example, this score indicates that whenever GitGuardian scans **an ".env" file, there's a 54% chance that it will result in the detection of a secret.** This metric gives us a clearer picture of which file types warrant closer scrutiny for potential security risks.

Best Practices: Correctly using .env files as a first layer of secrets protection

> "Compromised credentials are a gift that keeps on giving (your stuff away)"
> The 2023 Active Adversary Report from Sophos

What Happens After a Secret Leaks?

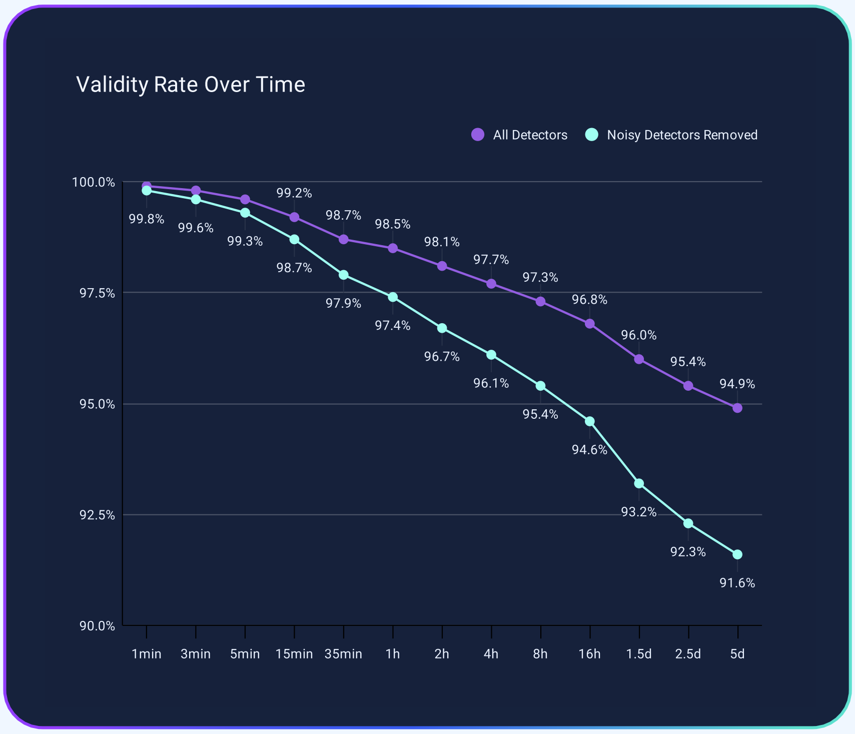
When someone exposes a secret on public GitHub, they should consider it compromised. **The author must revoke the secret quickly to reduce the impact of the incident.**

GitGuardian offers a free service that notifies the author when it confirms a secret: the pro-bono alerting system. The system checks the secret's validity up to 13 times based on a predetermined schedule. The last check happens about 5 days after the first detection.

Remediation Efforts

In 2023, GitGuardian monitored how well authors fixed leaks. The tracking started when the first valid occurrence of a secret was detected and ended five days later.

> The common trend is troubling:
>
> **More than 90%** of the secrets remained valid **5 days** after being leaked.



These curves display the progress of secret validity over time after detection. The perimeter is restricted to secrets for which the first occurrence was found valid, which amounts to 644,947 unique secrets detected in 2023 (not all secrets can be checked for validity). For each one, GitGuardian's pro-bono alerting system emailed the commit author.

GitGuardian found that **after 5 days, 91.6% of the secrets remained valid**.

To ensure a more balanced analysis, we excluded 3 detectors that accounted for 50% of the observations (namely Google Cloud Keys, Google API keys, and MongoDB Credentials).

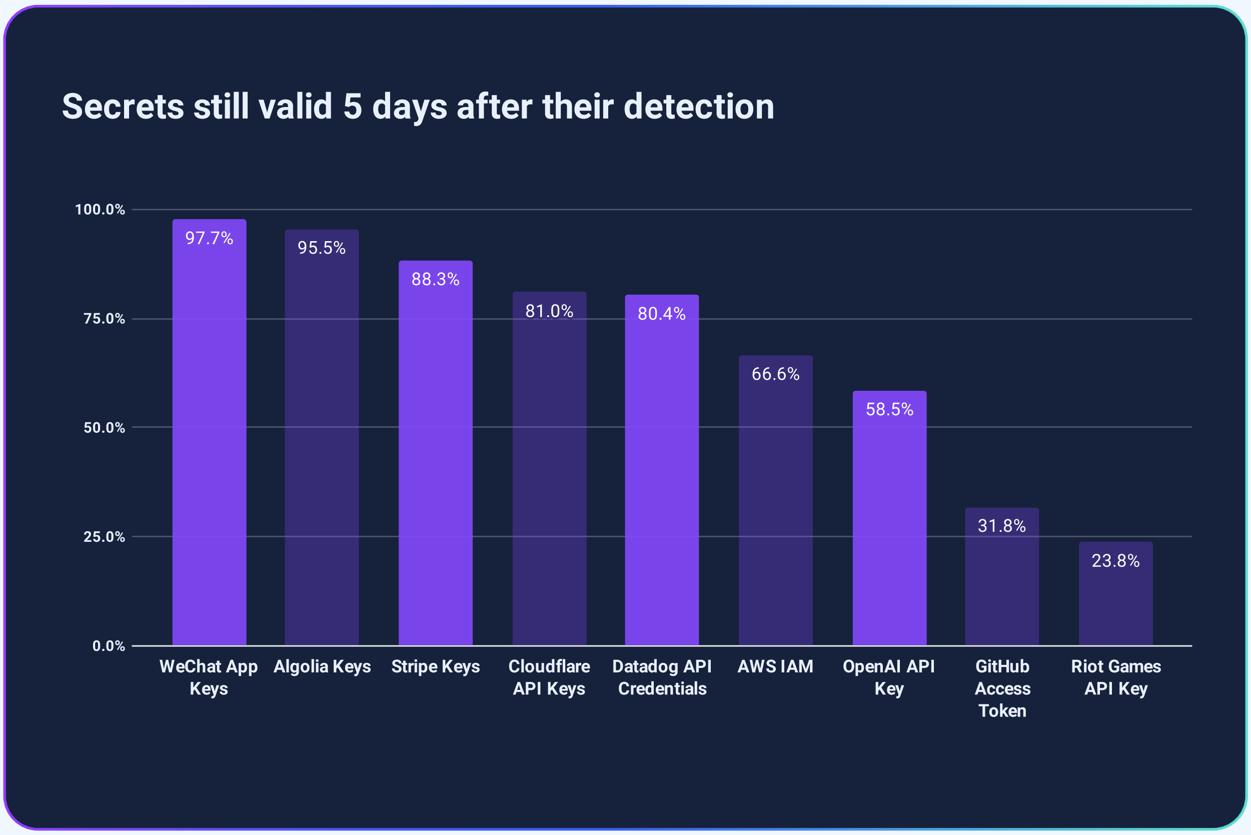
> Also, **just 2.6% of the secrets** were revoked within **1 hour** of notification via email.

This finding is crucial for grasping the full scope of the secrets sprawl issue. While the majority of security initiatives focus on detecting these leaks, the actual bottleneck lies in remediation. Simply alerting developers falls short; what's truly essential is providing them with the necessary guidance and support to rectify their mistakes effectively.

In other words, it's all about being able to answer, "What happens after a secret leaks?"

Revoked secrets

Not all types of secrets are fixed (revoked) at the same rate. We noticed some significant differences around which secrets were likely to be fixed within 5 days after their detection (selected sample of detectors with more than 400 observations):



This analysis reveals that leaked WeChat App and Algolia keys are the most likely to remain exposed for over 5 days. Conversely, developers are more concerned about the risks of leaking Stripe or Cloudflare API keys, as these would be prime targets in credential-harvesting campaigns.

A noteworthy finding is that **58.5% of OpenAI API keys are still valid 5 days after being leaked**. Given the substantial volume of these leaks throughout 2023 (46K per month) and the emerging threats posed by the misuse of generative AI, the security of OpenAI keys warrants urgent attention. Likewise, only 33.4% of valid AWS IAM keys are revoked within the initial five days.

An interesting observation is that WeChat, Stripe, Datadog, and OpenAI, all participants in GitHub's partner program for streamlining the secret detection and reporting process, **still experience a high rate of unresolved leaks**.

How Toyota Customer Data was Compromised with a Credential Exposed for 5 Years

On October 7, 2022, Toyota, the Japanese-based automotive manufacturer, revealed they had publicly exposed a credential allowing access to customer data for nearly 5 years. The code was accessible in a public GitHub repository from December 2017 through September 2022. While Toyota says they have invalidated the key, anyone who found that credential could access the server, gaining access to the data of 296,019 customers.

This situation highlights that automated detection is a necessary but insufficient layer of protection. With a valid secret exposed for that long, threat actors can compromise resources, data, and move laterally across the supply chain. Fixing vulnerabilities should be the primary focus of a secrets security program.

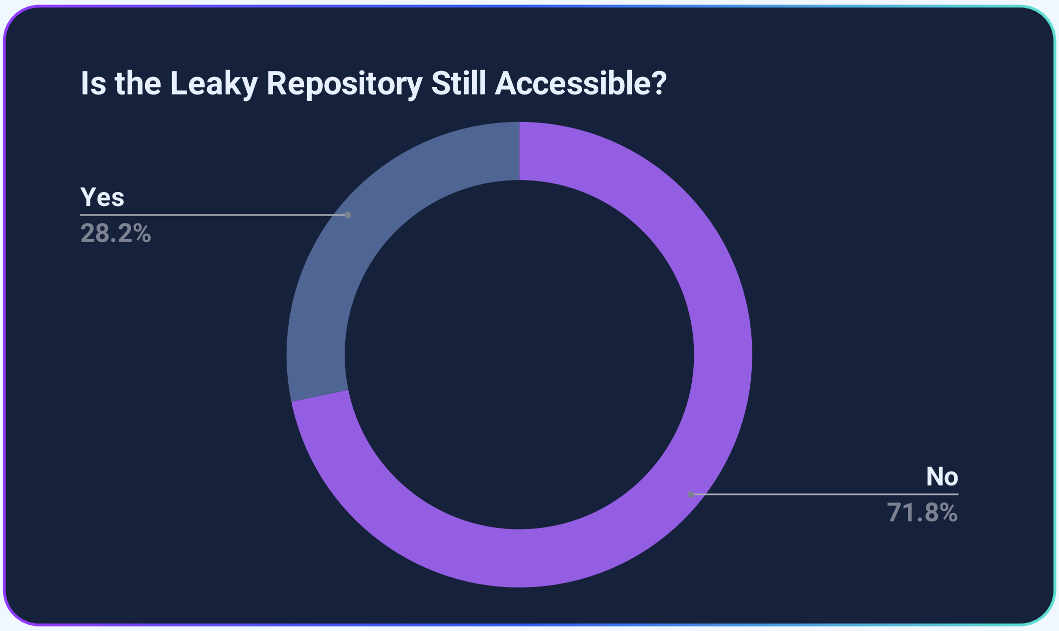
🦉 Fun Fact:

Only 24% of Riot Games keys were still active after five days vs. 95.5% for Algolia! **Could gamers be a secret weapon against secrets sprawl?**

Zombie Leaks: a Hidden Threat

Repository owners often react to leaks by either deleting the repository or making it private, cutting off public access to the leaked information. However, this approach can lead to one of the riskiest scenarios for an organization: a **"zombie leak"**.

A zombie leak occurs when a secret is exposed **but not revoked**, remaining a potential attack vector. The commit author may believe that deleting the commit or repository is sufficient, overlooking the crucial revocation step.



A study involving a random sample of 5,000 erased secret occurrences 5 days after their exposure revealed that **only 28.2% of the repositories were still accessible today**. This suggests that the remaining repositories were likely deleted or privatized, a common response to leaks.

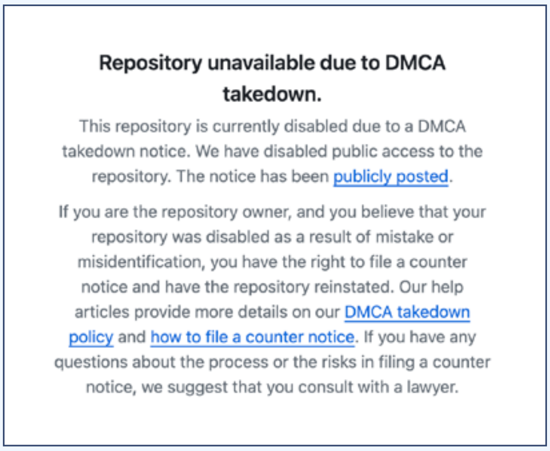
Although we cannot tell when these repositories disappeared, many of the disappearances were probably caused by what seemed most likely to hide the issue rather than effective remediation.

It's important to remember that numerous threat actors continuously monitor and mirror public GitHub activity in real-time. Any sensitive information exposed, even briefly, should be considered compromised. **For secrets, this means that merely hiding the leak is ineffective and can create a false sense of security.**

[What to do if you expose a secret: the secret leak remediation cheat sheet]

DMCA Takedown Notices: a Last Resort to Stop Leaks?

DMCA takedown notices are a process for any copyright owner in the U.S. to demand the removal of content that infringes on their rights. As a “safe harbor,” GitHub must process DMCA requests when infringing code is posted on the platform. For that, a dedicated repository is used to archive all DMCA takedown notices and counter-notices they receive. These notices can mention one or multiple repositories simultaneously.



Repository unavailable due to DMCA takedown.

This repository is currently disabled due to a DMCA takedown notice. We have disabled public access to the repository. The notice has been publicly posted.

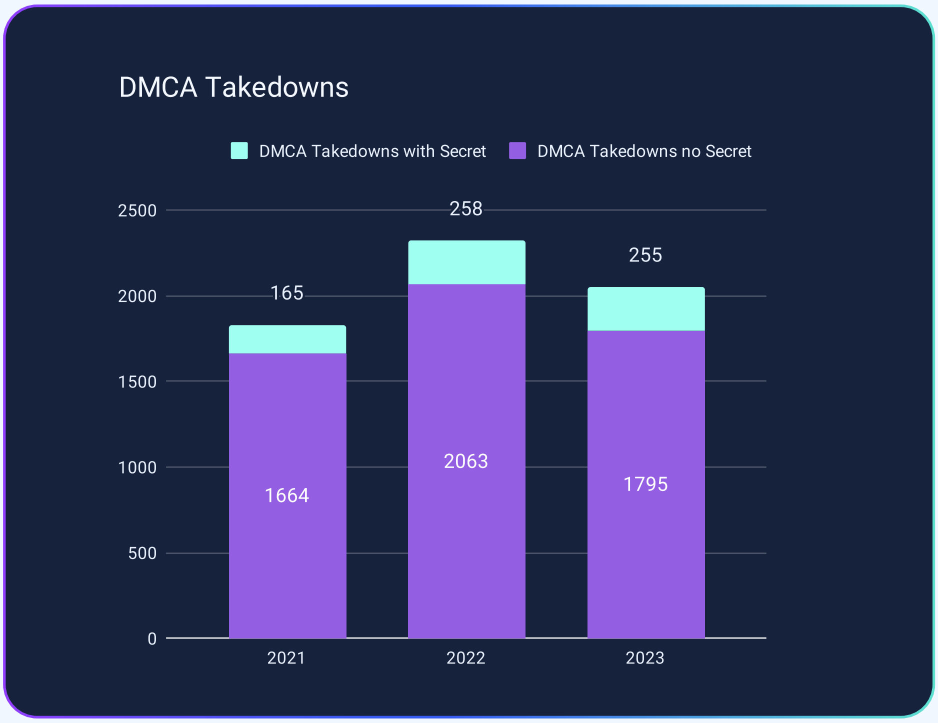
If you are the repository owner, and you believe that your repository was disabled as a result of mistake or misidentification, you have the right to file a counter notice and have the repository reinstated. Our help articles provide more details on our DMCA takedown policy and how to file a counter notice. If you have any questions about the process or the risks in filing a counter notice, we suggest that you consult with a lawyer.

⚖️ The Digital Millennium Copyright Act (DMCA)

Is a U.S. copyright law established in 1998. But GitHub is used worldwide. So GitHub’s DMCA policy also takes international copyright laws into account. As such, GitHub will also respond to takedown notices that are compliant with equivalent laws in other jurisdictions.

Given that leaks frequently occur outside an organization’s control, **often in personal GitHub** accounts, DMCA notices are mainly employed to manage such external repositories. Data points to an increasing use of DMCA notices as a last-ditch effort to remove repositories that inadvertently expose secrets.

In 2023, GitHub received 2,085 DMCA takedown notices, and 2,050 of the repositories mentioned in these notices were taken down: **of these, 255 (12.4%) exposed at least one secret**. This proportion was 9% in 2021 and 11.1% in 2022. Between 2021 and 2023, the percentage of takedown requests that involved a repository with an exposed secret increased by 37.8%.



It's worth noting that the public disclosure of such requests can inadvertently highlight problematic repositories to malicious actors. This visibility makes it a double-edged sword, suggesting it should be used as a last-resort solution due to the risk of drawing unwanted attention to sensitive content.

AI for Secrets Detection

The year 2023 marked the breakthrough of Generative AI, significantly impacting various professional fields with rapid adoption facilitated by user-friendly chats and developer-friendly APIs. Developers, as we have seen, are at the forefront of this new wave, and there is no doubt that this powerful technology, in the hands of both good and bad actors, will have an outsized impact on cybersecurity.

Could foundational models effectively replace human-developed engines for tackling secrets sprawl?

This chapter explores whether LLMs models could serve as an alternative to traditional secret detection tools. It examines the operational scale, cost and time efficiency, and overall performance of general-public AI in detecting secrets. That leads to an in-depth look at GitGuardian's AI-driven approach to enhancing the detection and management of sensitive information.

How Good Can LLMs Be at Detecting Secrets?

Large Language Models (LLMs) are advanced artificial intelligence systems designed to understand, generate, and work with human language. They comprise multiple orders of magnitude more parameters than traditional machine learning models and are tuned on unprecedentedly vast amounts of data.

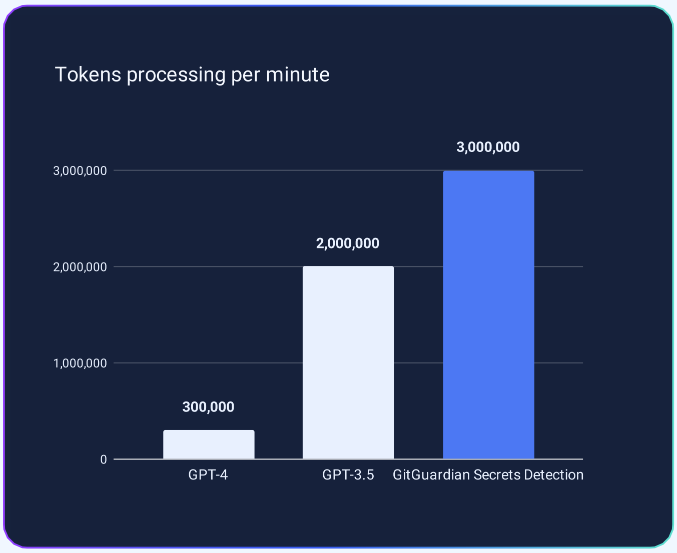
As of today, many such LLMs have been made available to the public, e.g. through APIs as is the case for GPT-4, with the possibility to specialize them by providing task-specific instructions as done with "custom GPTs."

In the context of source code, LLMs models are competent at understanding and generating code for a variety of programming languages as well, making them a natural candidate for secret detection.

OPERATING SCALE

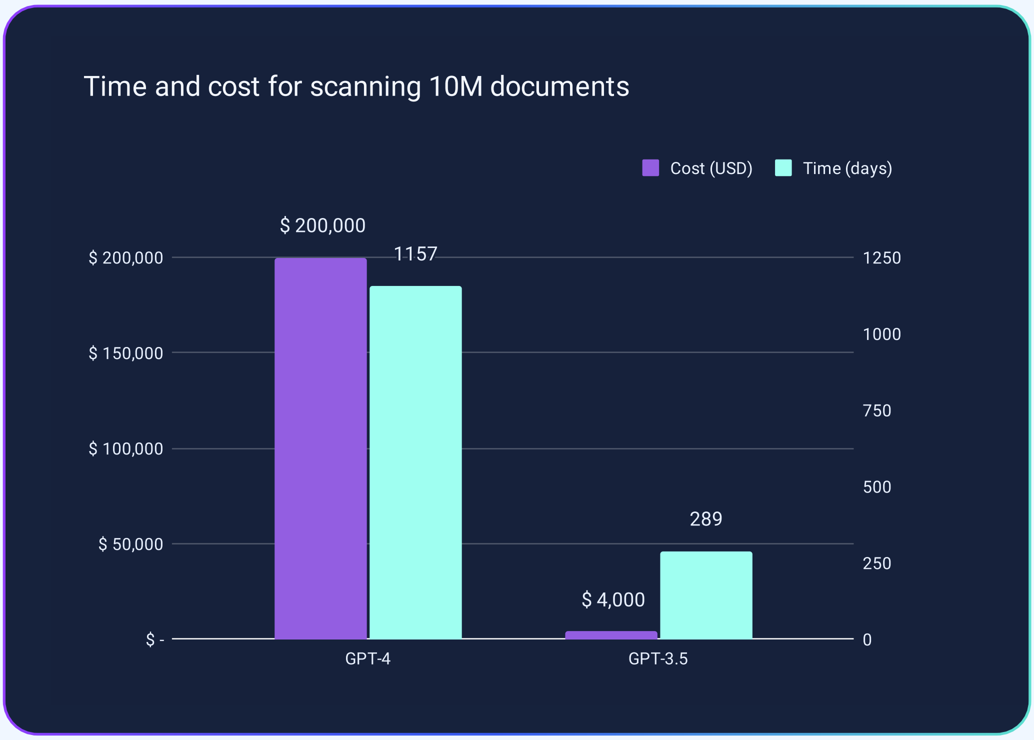
GitGuardian scans ~10M public documents per day, or ~116 per second. Assuming an average document size of 500 tokens (a very conservative approach), this would be equivalent to **3M tokens being processed every minute** on average, illustrating GitGuardian's operating scale.

This volume challenges the rate limits set by OpenAI for their foundational models:



TIME & COST FACTORS

Then comes the price and time consideration. With an average price per document of 0.02$ and an average time of 10 seconds using GPT-4 (respectively ~0.0004$ and ~2.5s/doc for GPT-3.5), here is what would be needed to scan 1 day of data processed by GitGuardian:
Note: the observed latency difference between the models is based on an average derived from our empirical testing, and has been documented by other users.


Time and cost for scanning 10M documents

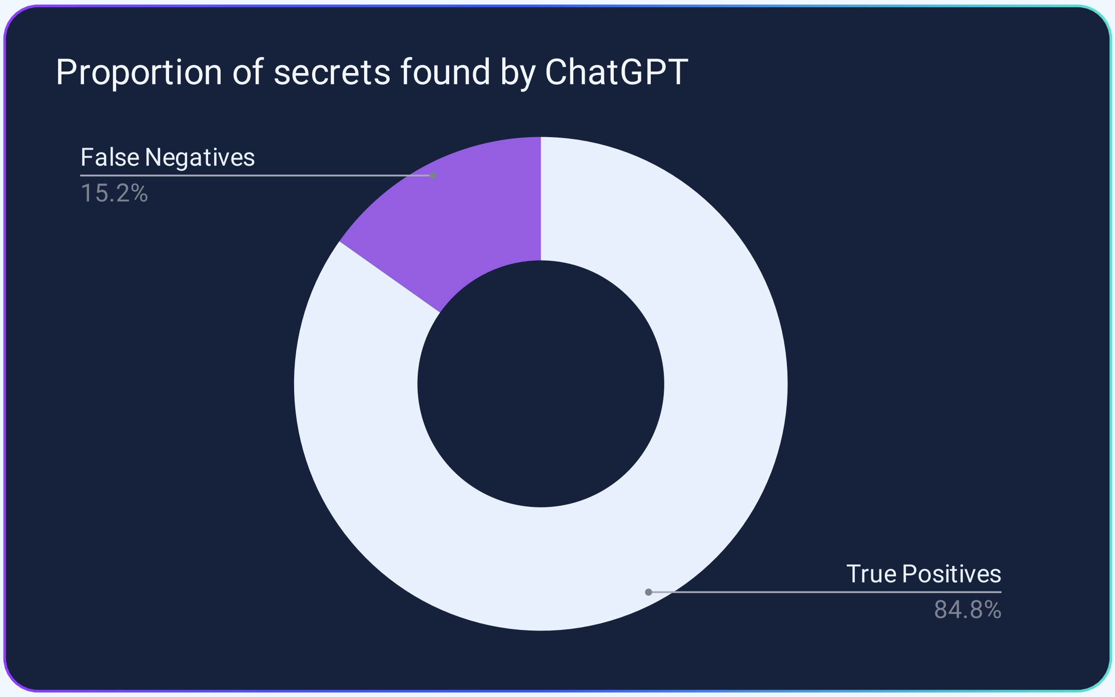
PERFORMANCE

Assessing ChatGPT's recall

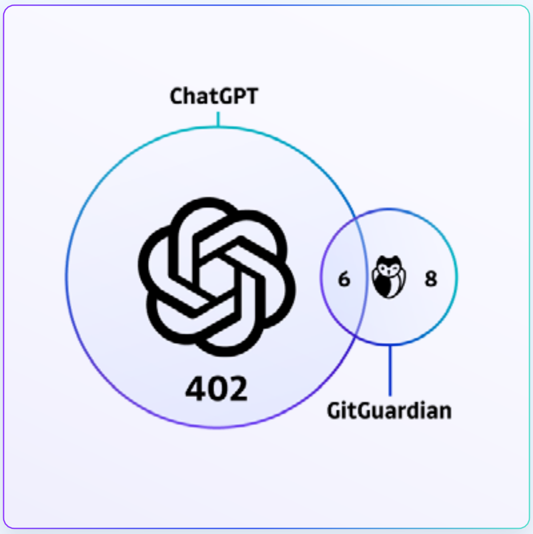
To evaluate ChatGPT's efficiency in accurately identifying valid secrets as true positives (assessing its recall capability), we conducted a test with 1,000 known valid secrets.
We prompted ChatGPT (full prompt detailed in the Methodology section) to detect these secrets. The outcome revealed that **ChatGPT failed to recognize 15.2% of the secrets, demonstrating less-than-ideal recall**. This finding is particularly concerning given that the test focused on specific secrets; the recall rate is likely to be even lower for generic secrets.

Assessing ChatGPT's precision

To assess ChatGPT's precision in identifying hard-coded secrets, we collected 1,000 random Python documents from GitHub and tasked ChatGPT with detecting secrets. We then cross-referenced ChatGPT's findings against GitHuardian's. The examination yielded the following distribution of detected secrets: ChatGPT exclusively identified 402, GitGuardian exclusively found 8, and both tools agreed on 6 instances.



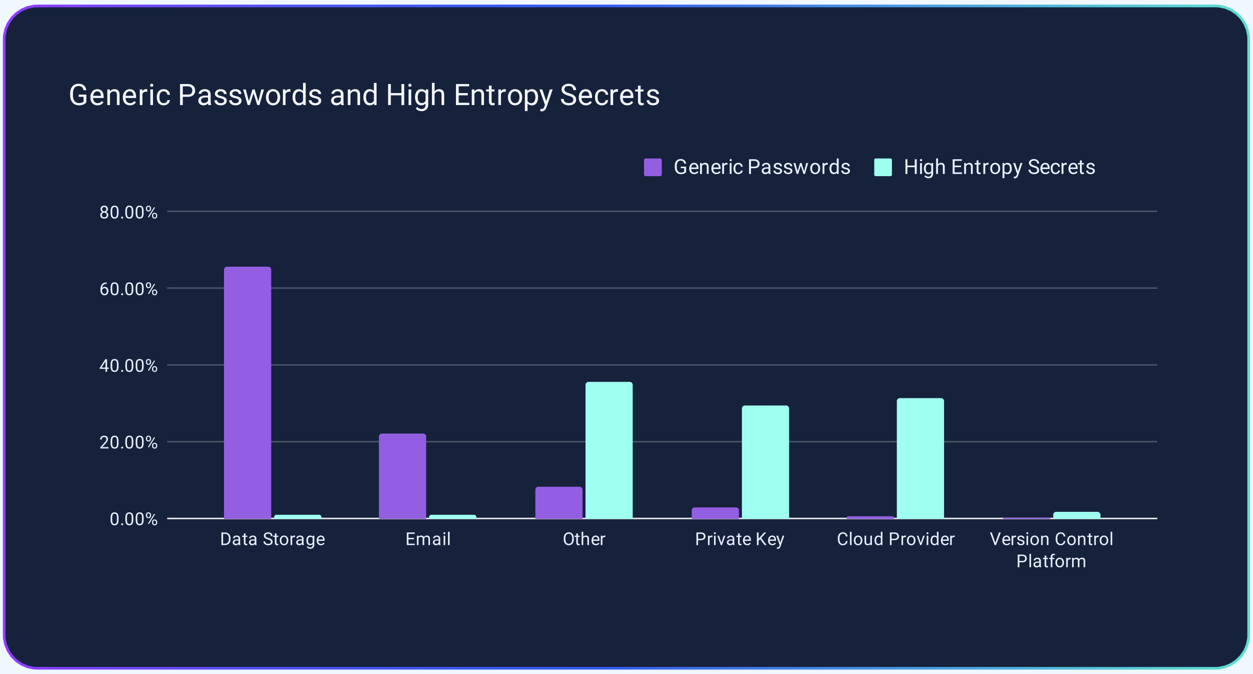
ChatGPT is flagging an excessive number of files. A manual review of the documents confirms this, revealing that alerts for secrets were triggered in simple cases, such as common placeholders or even IP addresses. This suggests a high propensity for generating false positives.



While refining the prompt could potentially improve the results (at a cost), in this particular experiment, **ChatGPT's precision was significantly low**.

Powering Secrets Detection with AI: GitGuardian's Approach

Make no mistake: AI represents a revolutionary shift in the field of secrets detection. However, our conviction lies in the power of combining AI with our specialized secrets detection engine to unlock true value. We outline several use cases where integrating AI can significantly enhance our capabilities.

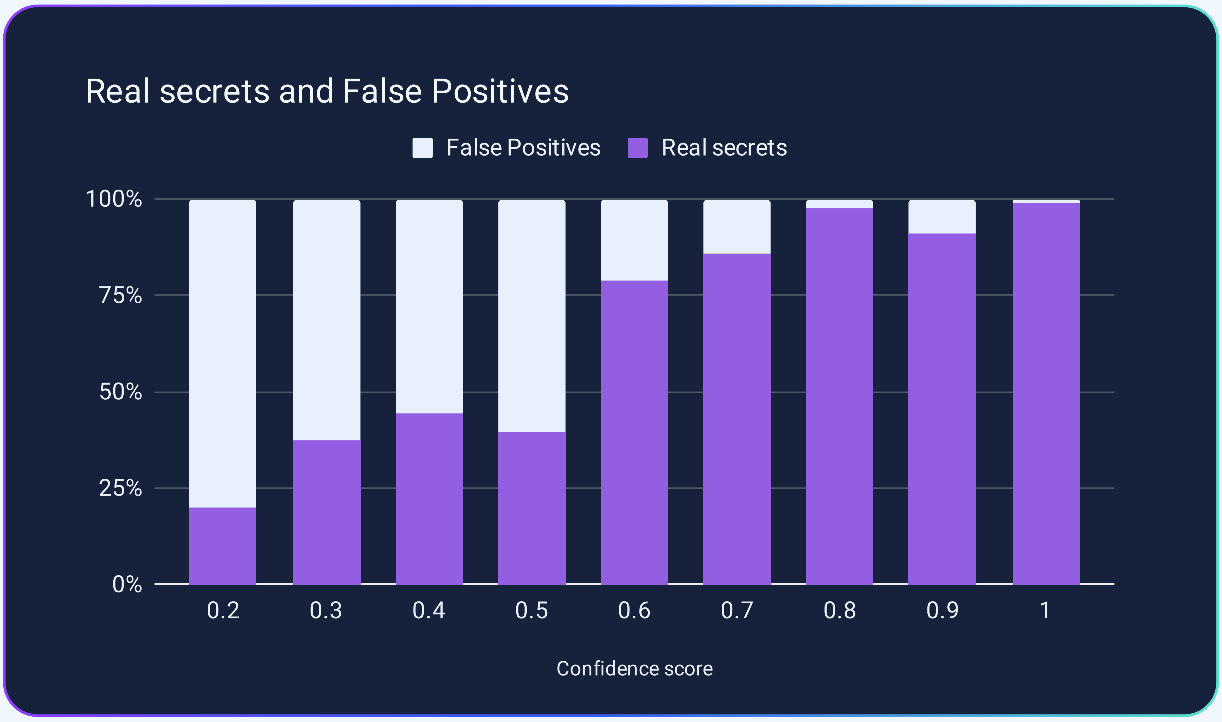


CATEGORIZE GENERIC SECRETS

Generic secrets, not associated with specific services, present unique challenges in secrets detection: the lack of contextual information and validity checkers makes it difficult to offer incident status visibility or context-tailored remediation guidelines. GitGuardian is deeply committed to advancing this area, leveraging AI to enhance the contextualization of leaks. An initial step towards enriching our insights involves using AI to classify generic secrets, providing valuable perspectives. For instance, when categorizing generic passwords and high-entropy secrets, we found that 66.7% of exposed passwords are related to database storage, suggesting potential remediation paths. Similarly, 31.5% of high-entropy strings are associated with cloud services.

And this is only the beginning. GitGuardian is developing machine learning techniques to identify crucial details like the service targeted by the secret, its environment, the incident's severity, and the secret's active usage within the codebase, thereby significantly bolstering our remediation framework.

IMPROVING PRECISION AND RECALL FOR GENERIC SECRETS

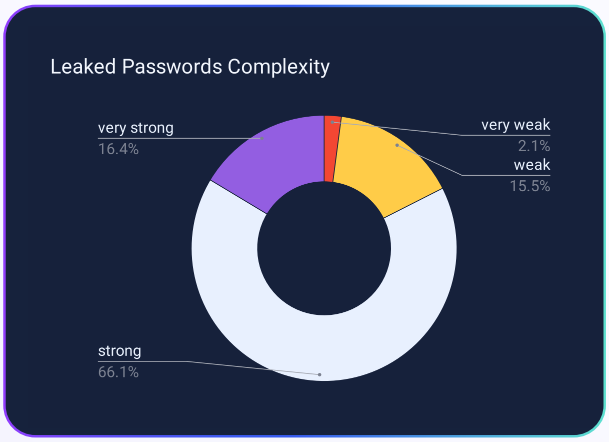


To enhance application security through machine learning, GitGuardian's ML team is developing a model to accurately score the likelihood of a genuine generic secret versus a false positive. This model is particularly effective, as test results show distinct score distributions for true and false positives: Such clear differentiation allows for setting a threshold that significantly reduces false positives without substantially impacting the detection of true positives, showcasing the model's capability to refine secrets detection. The reduction of false positives, in turn, reduces wasted response efforts by our customers.

🦉 Fun Fact:

It turns out that leaked passwords on GitHub might just be the most authentic survey of password habits out there.

When we peeked into 5,000 leaked passwords, we discovered a nice tidbit: the majority are cryptographically secure. **However, reality check – about 18% wouldn't resist a serious cracking attempt.** Leaks inadvertently offer a real-world glimpse into our passwords' strengths and weaknesses!



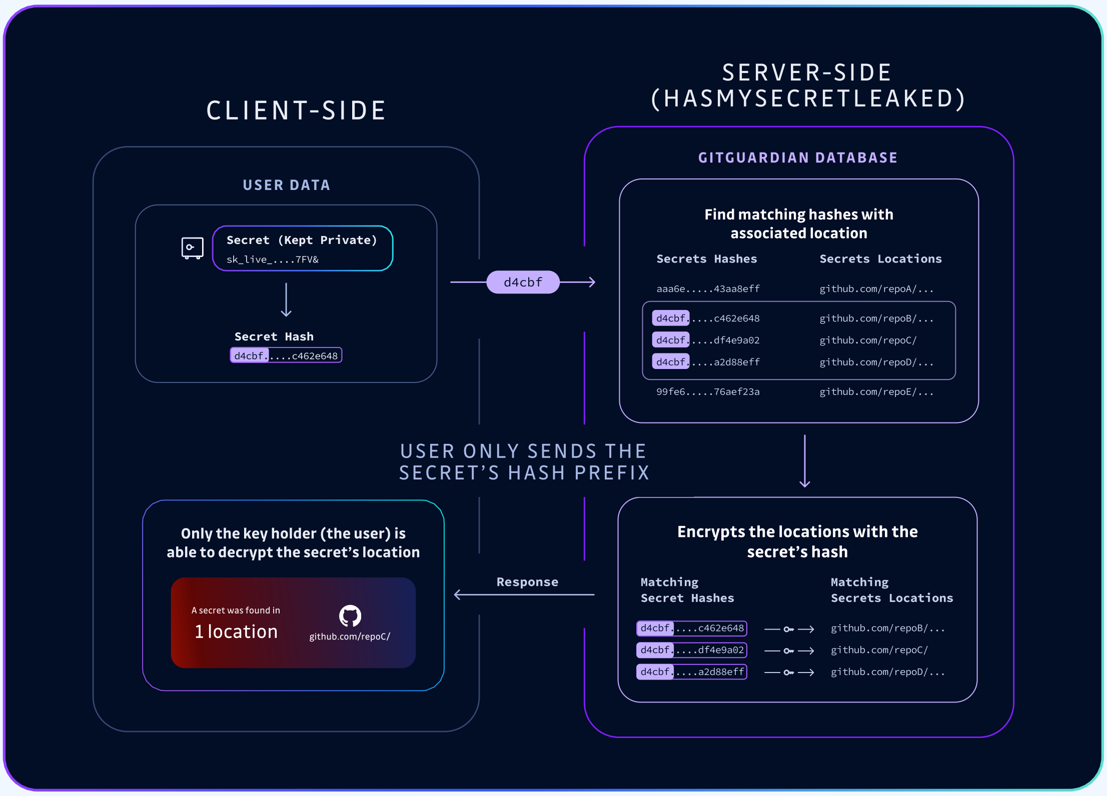
Are You Sure to Know Where Your Secrets Are?

Unveiling Secret Exposures with HasMySecretLeaked

HasMySecretLeaked is a pioneering free service designed to allow security practitioners to proactively check if their secrets have been exposed on GitHub.com. It leverages GitGuardian's comprehensive database, which includes over 20 million records of leaked secrets found across GitHub since 2017. The service is built on a trustless model, ensuring that users' secrets remain secure throughout the process, as **GitGuardian never accesses the secrets directly.**

The protocol behind the service is a hash-based security mechanism that has been enhanced with a series of protective layers (see the diagram below) to ensure that only the owner of a secret can know if this secret has been leaked, and where.

Additional measures, such as adding peppering and rate-limiting, are implemented to prevent misuse and enumeration attacks, making **HasMySecretLeaked** a secure and privacy-preserving tool for querying secret leaks.


CLIENT-SIDE
USER DATA
Secret (Kept Private)
sk_live_....7FV&
Secret Hash
d4cbf.....c462e648
d4cbf
USER ONLY SENDS THE SECRET'S HASH PREFIX
Only the key holder (the user) is able to decrypt the secret's location
A secret was found in
1 location
github.com/repoC/
Response
SERVER-SIDE (HASMYSECRETLEAKED)
GITGUARDIAN DATABASE
Find matching hashes with associated location
Encrypts the locations with the secret's hash

Secrets Hashes	Secrets Locations
aaa6e.....43aa8eff	github.com/repoA/...
d4cbf.....c462e648	github.com/repoB/...
d4cbf.....df4e9a02	github.com/repoC/
d4cbf.....a2d88eff	github.com/repoD/...
99fe6.....76aef23a	github.com/repoE/...

Matching Secret Hashes	Matching Secrets Locations
d4cbf.....c462e648	github.com/repoB/...
d4cbf.....df4e9a02	github.com/repoC/
d4cbf.....a2d88eff	github.com/repoD/...

To test the hypothesis that secrets leaked in private repositories are also leaked on public GitHub, we conducted a study on a perimeter comprising 403,571 leaked secrets, querying HasMySecretLeaked to know if these were also leaked on GitHub.

> **The result:**
>
> 3.11% of the secrets leaked in private repositories were also exposed in public repositories.

This fact hints at a well-known saying: **"Security through obscurity is no security at all."** Applied to our case, it dismantles the idea that relying on the privacy of source code as a security layer is a valid strategy.

These “private yet public” leaks have been publicly exposed 3.48 times on average, and 99% were found in source code files (less than 1% in GitHub issues, Pull Request descriptions, or GitHub Gists).

Extended Attack Surface: Secrets Sprawling in PyPI

PACKAGE REPOSITORIES

Open-source isn’t just about open-source code, it’s also about open distribution. Open-source ecosystems rely heavily on package hosting and management, with central repositories like NPM, Maven, and PyPI being crucial for software development.

These platforms offer easy access to millions of code libraries, streamlining the integration of functionality into projects “without reinventing the wheel.” **As part of the software supply chain, open-source packages make up an estimated 90% of the code run in production today.**

Historically, package repositories have been overlooked as a critical area for supply chain security. Yet, the recent surge in malicious packages posing as legitimate, popular code libraries has raised awareness about the risks associated with unmonitored package submissions.

> As recently as December 2023, ESET Research identified no less than 116 malicious packages within PyPI, spread across 53 projects. These harmful packages have been downloaded more than 10,000 times by unsuspecting victims. The malware found in these packages featured a backdoor enabling remote command execution, data exfiltration, and the ability to take screenshots.

This shift in perspective is driving the demand for enhanced security protocols to safeguard an essential component of the software industry. Much like version control systems, package repositories must be considered part of the extended attack surface of application security.

A promising initiative to improve the security of the open-source ecosystem was launched by the Cybersecurity and Infrastructure Security Agency (CISA) in partnership with the Open Source Security Foundation (OpenSSF) last year. Together they published the Principles for Package Repository Security. The document recommends integrating package API tokens "into common third-party secret scanning programs" and enabling automatic token revocation with a dedicated endpoint.

Building on its prior research, GitGuardian has expanded its investigation into the pervasiveness of leaked secrets within PyPI packages by incorporating new data. This enhanced analysis aims to provide a deeper understanding of secrets sprawl within the Python package index, highlighting the ongoing challenges and risks associated with secure code management in open-source libraries.

THE STATE OF SECRETS SPRAWL IN PYPI

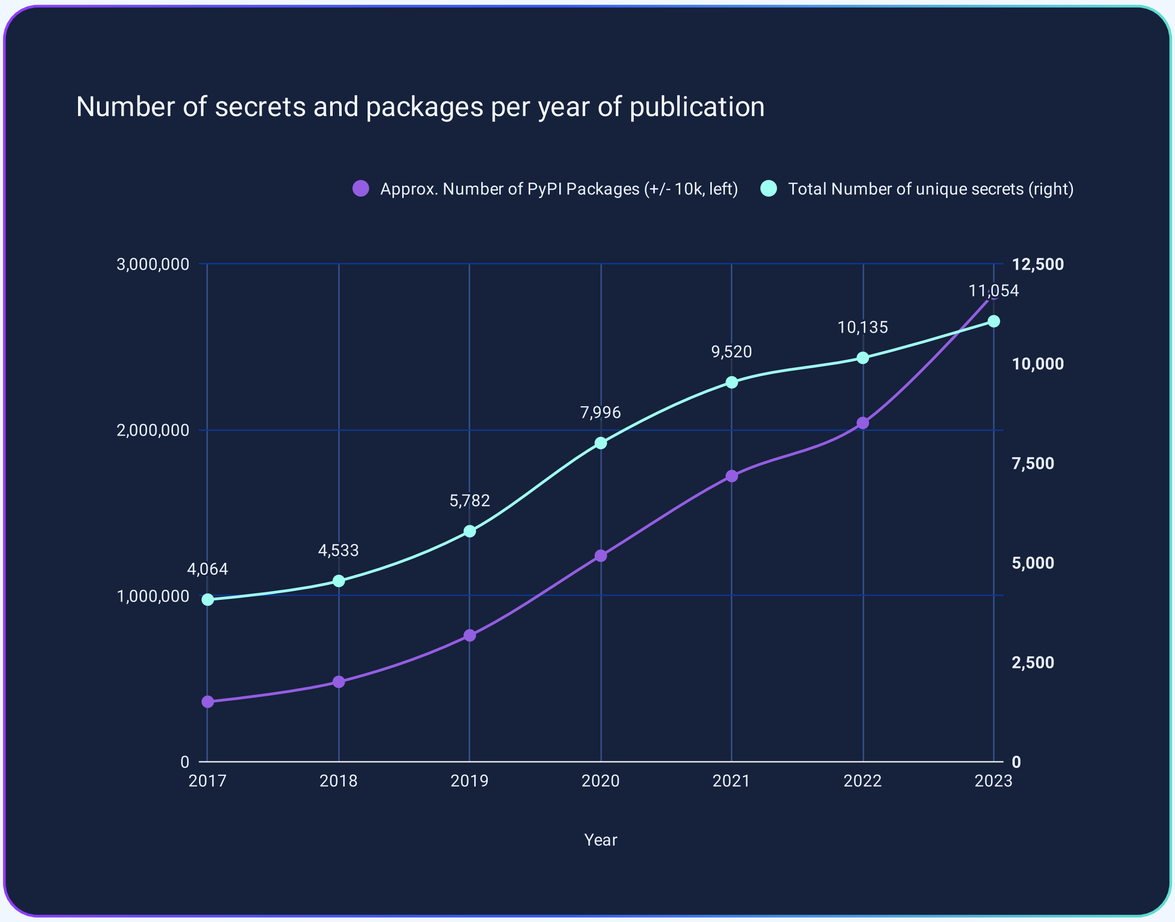
The Python Package Index, better known as PyPI, is the official 3rd party package management system for the Python community. The central repository boasts over 500K hosted projects, 10M files, and over 31 billion monthly downloads.

GitGuardian analyzed the number of unique secrets leaked in PyPI packages: **11,054 unique secrets were exposed in package releases in 2023.** Approximately 10,000 of those secrets had been there since before 2023, and over 1,000 had been introduced that year.

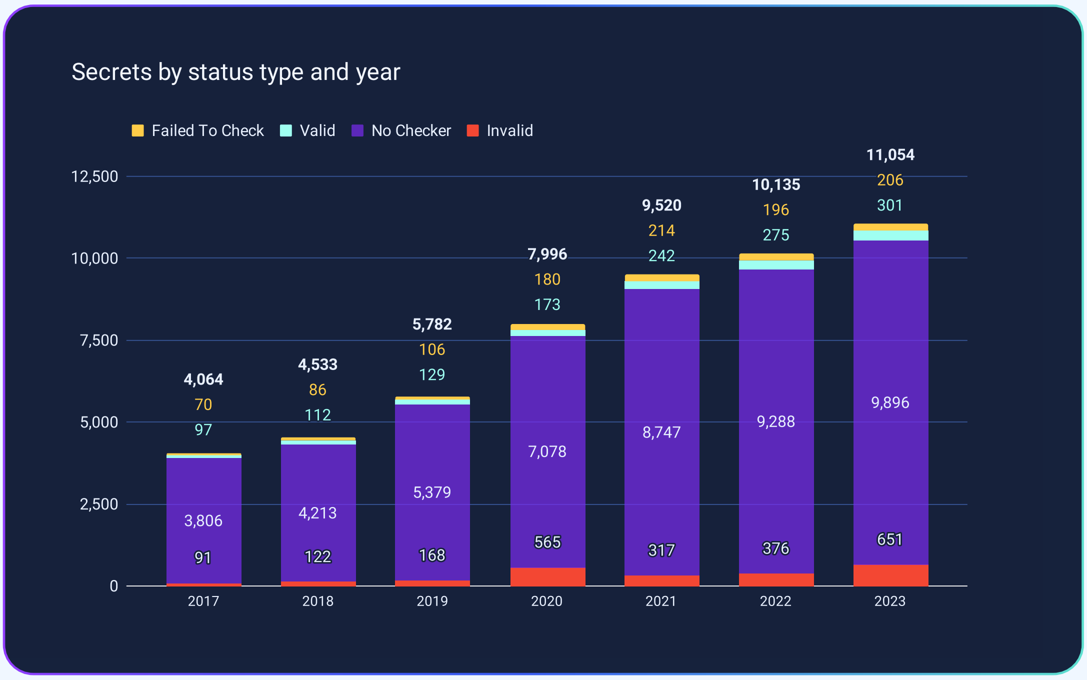
Adding up all the secrets shared across all the releases (5 million), we found 56,866 occurrences of secrets, indicating the same secret is often found in multiple releases of the same package. The reason behind this is simple: many package maintainers often don't realize a secret is shipped with the code library at every release!

> 💡 A "release" on PyPI is a specific project version. For example, the requests project has many releases, like "requests 2.10" and "requests 1.2.1". A release may consist of one or multiple files.

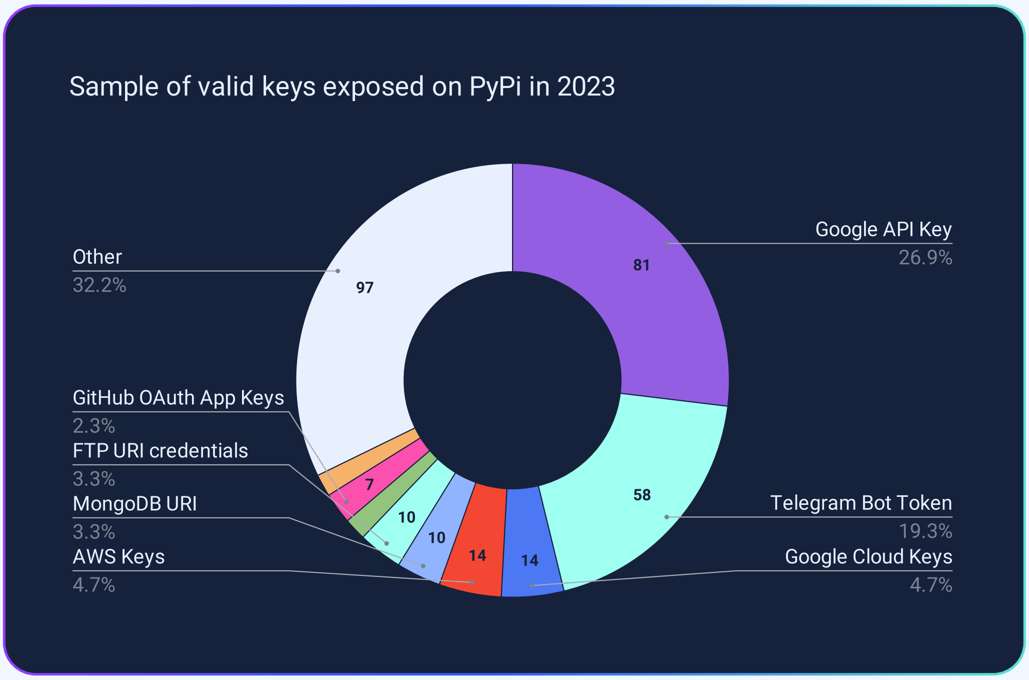
Leaked secrets closely follow the trend of new packages being added to the index repository, as illustrated in this chart:



The pervasiveness of secrets across releases explains how **97 secrets detected in packages dating from 2017 were still valid in late 2023 at the time of the study:**



As highlighted in the above chart, a large share of secrets couldn't be validated because no checker exists. Focusing on valid secrets only, here is a sample of some of the most frequent secrets we discovered:



WHY ARE WE SEEING LEAKED SECRETS IN PYPI PACKAGES?

Of course, secrets hard-coded in source code are at risk of being packaged and exposed in a central repository such as PyPI. However, several scenarios could account for the presence of a secret in a PyPI package without it being visible in the source code:

- **Private Repositories:** the source code may be hosted in a private repository, shielding it from public scrutiny but not from being included in publicly distributed packages.
- **Local Packaging:** the package could be compiled/packaged on a local machine, allowing secrets to be inadvertently incorporated during the packaging process without being exposed in any public code repository.
- **Build-Time Insertion:** the secret might not be embedded directly in the source but introduced during the build or packaging process. This can happen through environment variables, build scripts, or other mechanisms that insert secrets into the final package, bypassing the source code stored in version control systems.

> “In the course of outreach for this project, we discovered at least 15 incidents where the publisher was unaware they had made their project public. Without naming any names, we did want to mention some of these were from very large companies that have robust security teams. Accidents can happen to anyone.”
>
> **Tom Forbes**, Staff Engineer at **GitGuardian**

Welcome to GitHub.com, the Largest Attack Surface in The World

Exposed secrets are not the only security threats lurking in the vastness of GitHub. In 2023, several studies unmasked malicious operations targeting or using the platform as a vector for supply chain attacks. As GitHub's popularity soars, it increasingly attracts malevolent actors, positioning it as a central hub for cyber threats:

- PyTorch critical supply chain compromise: Researchers exploited a critical CI/CD vulnerability in PyTorch, a major ML framework used at Google, Meta, and Boeing. Key to the attack was stealing GitHub Personal Access Tokens (PATs) and AWS keys to move laterally across the supply chain. The vulnerability enabled unauthorized upload of malicious PyTorch releases and backdooring dependencies. The exploit involved self-hosted runners, often targeted due to their insecure default settings and ability to run arbitrary code from fork pull requests.
- Spoofing committers to make malicious repos look reliable: GitHub's ability to spoof and forge commits' metadata allows malicious actors to mislead developers into using repositories hosting malicious code. Timestamps and committer identities on GitHub commits can be easily forged.
- Using GitHub as malicious infrastructure: Researchers explained the frequent abuse of GitHub's services by cybercriminals and advanced persistent threats (APTs) for various malicious infrastructure schemes. These include payload delivery, dead drop resolving (DDR), full command-and-control (C2), and exfiltration. GitHub's popularity among threat actors lies in its ability to allow them to blend in with legitimate network traffic, making detection and attribution challenging for defenders.

> "Overall, DarkOwl detected 20,921,173 mentions of GitHub on the darknet, of which 5,723,002 are from last year alone. Across authenticated sites, which are the most high-value forums and marketplaces, 90,255 mentions of GitHub were tracked."
>
> **Erin Brown**, Director of Intelligence and Collections at **DarkOwl**

- Dependabot impersonation campaign: In 2023, malicious commits were detected on hundreds of GitHub repositories, appearing to be contributed by Dependabot but actually carrying a payload designed to exfiltrate GitHub secrets and steal web-based password forms. The attackers utilized stolen GitHub personal access tokens from victims to make these malicious commits, even compromising repositories within private GitHub organizations due to the comprehensive access provided by the victims' tokens.
- 3% of all GitHub repositories are potentially vulnerable to RepoJacking: Repo-jacking is a specific software supply chain attack type allowing malicious actors to gain control over a GitHub namespace by registering a username made newly available by a name change. For example, XYZ changes their GitHub organization name from "xyz" to "xyzcorp," making it possible for someone to register as "xyz." According to this study, millions of repositories hosted under a different organization name in the past are vulnerable to this kind of attack, including ones owned by Lyft and Google. What makes them valuable is all the legacy links around the web that never got updated to reflect the name change.

Solving Secrets Sprawl

The risk companies face from the rapid sprawl of API keys, configuration variables, and secrets within engineering teams cannot be overlooked. Secrets serve as the keys to a company's most valuable assets, making their management and protection a critical aspect of overall security strategy.

Moreover, the threat posed by secrets sprawl extends beyond individual companies, impacting supply chains and critical infrastructure. This concern is echoed by national organizations like the Cybersecurity and Infrastructure Security Agency (CISA) and the National Institute of Standards and Technology (NIST).

Secret scanning is a specific requirement of the new "Strategies for the Integration of Software Supply Chain

Security in DevSecOps CI/CD pipelines" publication from NIST, along with SAST, SCA, and DAST. Meanwhile, in its recent Cybersecurity Advisory, CISA underscores the risks associated with plaintext credentials, warning against lateral movements and privilege escalation.

Despite the recognized importance of managing secrets sprawl, the widespread adoption of emerging best practices remains limited. While secrets management tools are a valuable part of the solution, they alone are not sufficient to address this complex issue. So, what can effectively solve secrets sprawl?

Fortunately, through our work with numerous organizations since 2017, we have gained valuable insights into how companies with the strongest security postures successfully tackle this challenge. By learning from these successful implementations, we can identify effective strategies and best practices to manage secrets sprawl and enhance overall security.

Awareness & Training

Cultivating secure coding practices is an ongoing process. Awareness and training are two interconnected aspects that play a crucial role in mitigating the issue of secrets sprawl within an organization.

One of the significant challenges organizations face is the division between teams that create secrets, primarily developers, and those responsible for securing them. This division often leads to siloed operations and potential adversarial relationships. Despite these differing approaches, both teams share the common goal of secure code and adherence to best practices.

To bridge this gap, informal "lunch and learns" are highly effective. These sessions can raise awareness and foster a shared understanding of security issues without creating overwhelming pressure. Such initiatives form the cornerstone of successful security champion programs, promoting collaborative efforts towards common security goals.

“Security training” often brings to mind lengthy, tedious tutorials on basic security practices. However, training can be a much more engaging and productive experience. By focusing on practical applications, such as using playbooks or participating in a Capture The Flag (CTF) challenge, training is made more relevant and appealing.

Training sessions also provide an excellent opportunity to address new threats and answer questions, ensuring that processes stay current and effective. The ultimate goal of training is to enhance communication within the team, making issue resolution a routine collaboration rather than a point of conflict.

Contact us to set up a Lunch & Learn or a CTF session

Effective Processes

Consistent results require consistent processes. In cybersecurity, clear and communicated procedures are essential to ensure success. Instead of merely handing over security findings and expecting optimal outcomes, it’s crucial to establish and convey effective processes for tasks such as creating credentials or rotating secrets within a DevOps environment.

As previously mentioned, having clearly established guidelines to address the question “What happens after secret leaks?” is vital in tackling the issue of secrets sprawl. Automating processes should also be a priority, particularly in critical supply chain steps such as secure code reviews. However, this is not always the case, as highlighted in last year’s survey:

> “27% of respondents revealed that they rely on manual code reviews, which are inefficient, to protect themselves from secret leaks.”
> Voice of Practitioners: The State of Secrets in AppSec 2023

Designing these processes can begin with a simple step, such as drafting a flowchart on a whiteboard. The key to success lies in effective communication, which includes thorough documentation, comprehensive training, and convenient tools.

Fast and (mostly) automated feedback loops empower developers to take ownership of the remediation process. This approach ultimately contributes to fostering a blameless culture within the organization, where mistakes are seen as opportunities for learning and improvement, rather than as blameworthy events.

Take the 5-min Secrets Management Maturity Questionnaire to measure your team's maturity.

Combining Secrets Detection & Management

Today, security teams are waking up to two realities: the foundational importance of securing secrets in application security programs, and the inevitability of secrets sprawl due to human error.

Secrets management tools are essential for maintaining good credential hygiene, providing a secure, central location for storing, distributing, and rotating secrets. However, these tools cannot guarantee their use and may be bypassed. Even in organizations with mature security practices, secrets can sprawl in unexpected ways, creating a security loophole that evades even the most stringent surveillance.

To combat secrets sprawl, organizations should use discovery tools and effective controls. Secrets detection and remediation platforms assist in this by enabling continuous evaluation of secrets' security, enforcing consistent policies throughout the software development process, and speeding up incident resolution. These platforms foster collaboration between security and development teams, enhancing the security of the SDLC over time.

While evaluating tools to address the issue of secrets sprawl, particularly leaks into public GitHub repositories, might be a new process for you, GitGuardian has helped hundreds of enterprises find the right solution in less time than they initially anticipated. We understand the common hurdles anyone adopting a new solution faces, including how to communicate the value and urgency of solving the issue as you build a case. We are more than happy to help you navigate these challenges.

Enterprise Buyer's Guide For Secrets Detection: 10 key considerations for security decision-makers

After years of honing our expertise in secrets detection and remediation, GitGuardian has partnered with CyberArk, a leader in identity security and secrets management. Together, we are building the industry's **first end-to-end secrets security solution**.

The first use case involves detecting secrets that have been leaked on public GitHub repositories. Our integration identifies these leaks, notifies users, and initiates remediation steps. The second use case focuses on detecting secrets within a company's internal perimeter. This includes:

- Identifying secrets in the source code that aren't securely stored in CyberArk's Conjur Cloud (rogue secrets). Our aim is to inform users and encourage the adoption of good secrets security practices.
- When a detected secret in the user's source code is already vaulted, our integration alerts the user and initiates a secret rotation flow for timely remediation.

> "94% of surveyed respondents plan to improve their secrets practices in the next 12-18 months."
> Voice of Practitioners: The State of Secrets in AppSec 2023

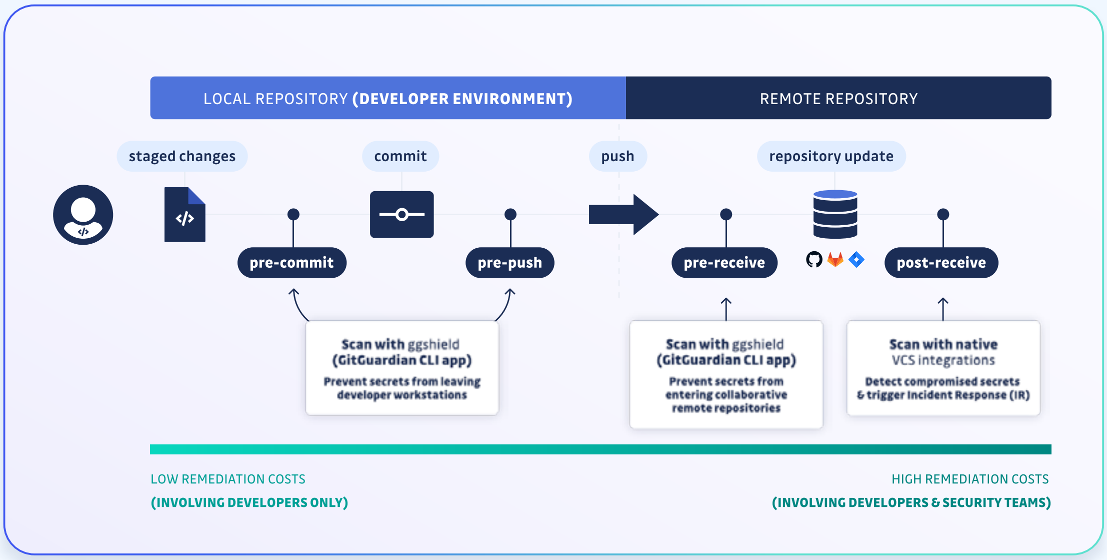
Preventing Leaks & Breaches

A multi-layered detection strategy is crucial for preventing hard-coded secrets and ensuring robust security.

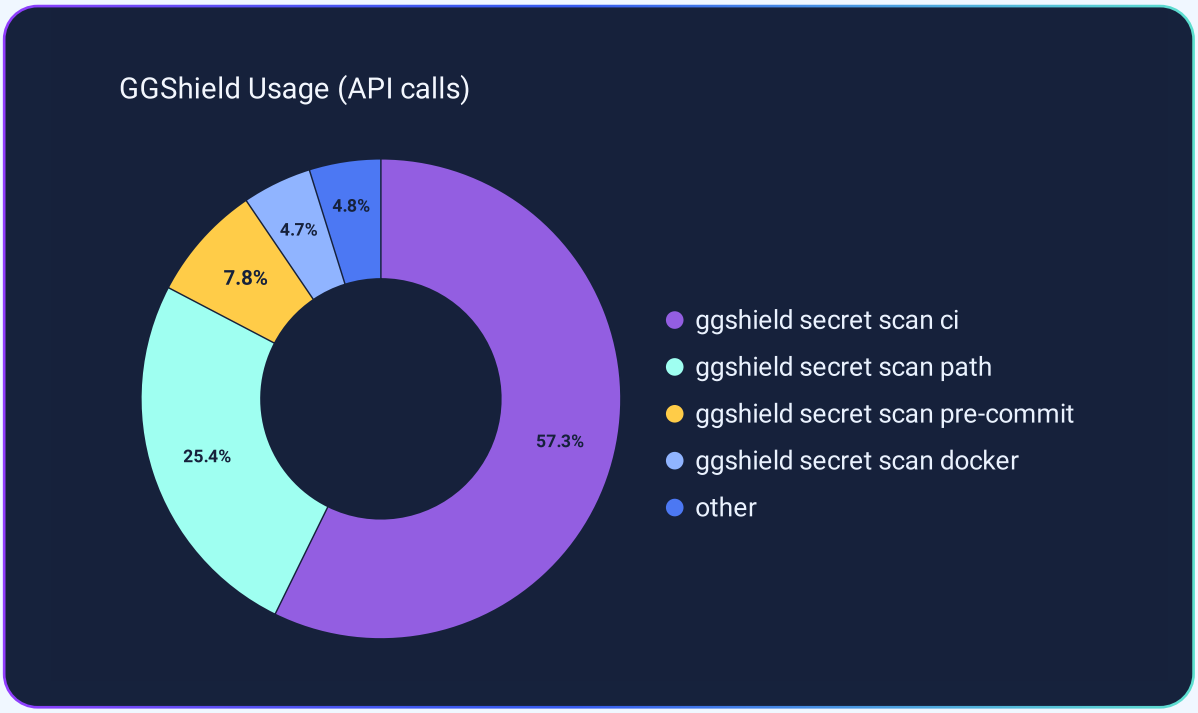
First, real-time monitoring of repositories provides full visibility, strengthening your Version Control System (VCS) against exposed secrets.

Second, employ a shift-left approach to address security concerns earlier in the development lifecycle. Ideally, secret leaks should be identified before they are committed to the repository. This is best achieved by integrating security tools into the development environment, enabling developers to detect and remediate security issues as they code.

GGshield, the GitGuardian command-line interface (CLI), was designed with this use in mind. It detects hard-coded secrets in **pre-commit hooks**, reducing the risk of a leak and promoting a culture of security awareness among developers. For organizations hosting their VCS, ggshield can also be used as a **pre-receive hook** to prevent leaky commits from entering repositories organization-wide.


LOCAL REPOSITORY **(DEVELOPER ENVIRONMENT)**
REMOTE REPOSITORY
staged changes
commit
push
repository update
pre-commit
pre-push
pre-receive
post-receive
Scan with ggshield **(GitGuardian CLI app)**
Prevent secrets from leaving developer workstations
Scan with ggshield **(GitGuardian CLI app)**
Prevent secrets from entering collaborative remote repositories
Scan with native VCS integrations
Detect compromised secrets & trigger Incident Response (IR)
LOW REMEDIATION COSTS
(INVOLVING DEVELOPERS ONLY)
HIGH REMEDIATION COSTS
(INVOLVING DEVELOPERS & SECURITY TEAMS)

GGShield is easily integrated into CI pipelines and other continuous integration tests. In 2023, this accounted for 57.3% of the nearly 8.5 million ggshield API calls made by GitGuardian customers.



In 2023, ggshield prevented 417K policy breaks per month!
Finally, ggshield is not just for hard-coded secrets. The tool can scan for infrastructure-as-code misconfigurations and create honeytokens.

IMPLEMENTING HONEYTOKENS FOR ENHANCED SECURITY

Managing secrets sprawl on a large scale can be daunting, particularly in large organizations where the scope of the secrets security tech debt makes it difficult to prioritize remediation according to the risks.

🍯 **What is a Honeytoken?**
Honeytokens are decoy credentials acting as tripwires, revealing attacker information (e.g., IP Address, User Agent, Location, etc.) without granting access to real customer resources. When a hacker triggers these decoys during a secret scan, it alerts the security team to a potential security incident.

As you begin the process of cleaning up historical secrets, a task that could take months or even years, honeytokens can provide reassurance by alerting you in case of a breach. This allows you to respond immediately and mitigate potential damage.

When dealing with a significant secrets debt, involving hundreds or thousands of incidents, honeytokens are invaluable. They guide the security team in prioritizing and rotating critical secrets, ensuring a proactive and efficient remediation process.

Honeytokens are your tool to maintain a vigilant defense against potential security threats while steadily addressing the issue of secrets sprawl. This approach combines immediate threat detection with long-term security enhancement.

About GitGuardian

GitGuardian is the security platform for the DevOps generation. GitGuardian enables security teams to define and enforce secure coding practices consistently and globally at every step of the software development process.

Centered on collaboration between security and development teams, GitGuardian also helps organizations enhance their security posture by decentralizing and accelerating the remediation of hard-coded secrets vulnerabilities and misconfigurations in infrastructure-as-code and open-source dependencies.

Widely adopted by developer communities, GitGuardian is the #1 security application on GitHub Marketplace and is used by over 300 thousand developers and leading companies, including Snowflake, Orange, Iress, Mirantis, Maven Wave, Payfit, and Bouygues Telecom.

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 post-validation 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 400 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 detector will only detect AWS 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.

Software supply chain security:

A software supply chain is a logistical pathway that covers anything required to build a software artifact. It is the set of assembled components, building tools, and processes from source code to production deployment. Supply chain security is about securing each link in the chain by ensuring that components supplied by third parties have not been compromised and comply with security requirements.

Methodology

GITHUB METRICS

All the GitHub metrics are extracted from the State of the Octoverse 2023. Some metrics, such as the number of new developers and repositories per month, have been computed based on the newly released GitHub Innovation Graph, an open dataset offering quarterly data on public activity that dates back to 2020.

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 2023.

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 did 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.

FILE EXTENSION RISK SCORE

For any given file, we have two random variables:

1. **T the file type**
2. **S the presence of a secret**

Let us note:

» **Ω the set of all files**

» **Ωs the set of files with secrets**

» **Ωx the set of files of type x**

Now, for each file type x, we have:

» **|Ωx| / |Ω| = P(x) the probability of the file type x**

- **|Ωs,x| / |Ωs| = P(x|s) the probability of file type x conditionally on the file exposing a secret**

Now, per Bayes theorem, we have P(x|s) / P(x) = P(s|x) / P(s)

- **P(s|x) is the probability of the file having a secret knowing that the file is of type x**
- **P(s) is the global probability of a file having a secret.**

Since P(s) is a normalization constant (it doesn't depend on the file type), the risk score is defined by P(s|x), the probability of the file having a secret knowing that the file is of type x. This probabilistic score is between 0 and 1.

After running a statistical test of independence (χ2), we found a probability of 0 that we would observe this data if the file extension and the presence of a secret were independent variables. Therefore, these results are significant.

PASSWORD STRENGTH COMPUTATION

The strength of a password is determined by its entropy, measured in bits. Entropy quantifies the unpredictability or randomness of a password, which in turn affects its security level. To calculate this entropy, we used a standard computation assuming a uniform distribution of characters. This formula provides a measure of the password's complexity and, by extension, its strength against brute-force attacks:

Bits of entropy	Password strength
0-35	Very weak
36-59	Weak
60-119	Strong
120 and above	Very strong

CHATGPT FOR SECRETS DETECTION

Prompt:

You are an expert in secret detection in source code.

You receive source code snippets and your role is to detect secrets.

The first line is the file name and the following lines are the file content.

Answer using the following format when a secret is present:

Name: [name of the secret]

Secret: [copy of the secret]

Description: [short description of the secret]

Repeat multiple times if multiple secrets are present.

If no secret is present, answer only "ALL CLEAR"

Three examples of false positives raised by ChatGPT when used for secrets detection:

```
Code snippet

print('Modified this file')
print('Third Modification')
print('changes in second branch')
+print('123456 new change in second branch')

-------

Name: Numeric Secret
Secret: 123456
Description: This code snippet contains a numeric

secret, specifically the number 123456.
```

```
Code snippet

@@ -168,7 +168,7 @@ class Server_run():
     def run(db):
         from http.server import HTTPServer
         handler = handlerwithdb(db)
-        server = HTTPServer(("192.168.0.178", 9000), handler)
+        server = HTTPServer(("192.168.51.201", 9000), handler)
         print("Starting server, use <Ctrl-C> to stop")
         server.serve_forever()
 #Server_run.run()

------

Name: IP address
Secret: 192.168.0.178
Description: The IP address 192.168.0.178 is hard-coded in the code.

Name: IP address
Secret: 192.168.51.201
Description: The IP address 192.168.51.201 is hard-coded in the code.
```

```
Code snippet

@@ -96,6 +96,11 @@ def webserver(request):
     docker_id_db = start_postgres()
     wait_for_container(docker_id_db)
     db_connection_string = f”postgres://postgres:notsecretpassword@
{get_ip_from_id(docker_id_db)}:5432”
+    db_initializer = subprocess.check_output(
+        [‘docker’, ‘run’, ‘--rm’,
+         ‘--name’, ‘initdb’,
+         ‘-e’, f”AIRFLOW__CORE__SQL_ALCHEMY_
CONN={db_connection_string}”,
+         get_image_name(), ‘airflow’, ‘initdb’]).decode().strip()
     docker_id = subprocess.check_output(
         [‘docker’, ‘run’, ‘--rm’,
          ‘--name’, ‘webserver’,
------
Name: db_connection_string
Secret: postgres://postgres:notsecretpassword@
{get_ip_from_id(docker_id_db)}:5432
Description: Connection string for PostgreSQL database.

Name: AIRFLOW__CORE__SQL_ALCHEMY_CONN
Secret: {db_connection_string}
Description: SQLAlchemy connection string for Airflow.

Name: notsecretpassword
Secret: notsecretpassword
Description: Password for the PostgreSQL user.
```

UNVEILING SECRET EXPOSURES WITH HASMYSECRETLEAKED

We conducted a study on a set of 403,571 leaked secrets and found the following:

- **94.74% of the secrets were not publicly exposed.**
- **2.16% of the secrets were publicly exposed more than 10 times. However, these were not considered in our analysis due to the likelihood of being false positives.**
- **3.11% of the secrets were publicly exposed less than 10 times, with an average of 3.48 occurrences.**

Among the secrets that were publicly exposed, 17.27% had exactly one occurrence exposed on GitHub.com.

GitGuardian

The State of **Secrets Sprawl 2024**

DATA ANALYSIS BY GITGUARDIAN

Learn more at **www.gitguardian.com**

© 2024 GitGuardian. All Rights Reserved.