GitGuardian

5th Edition

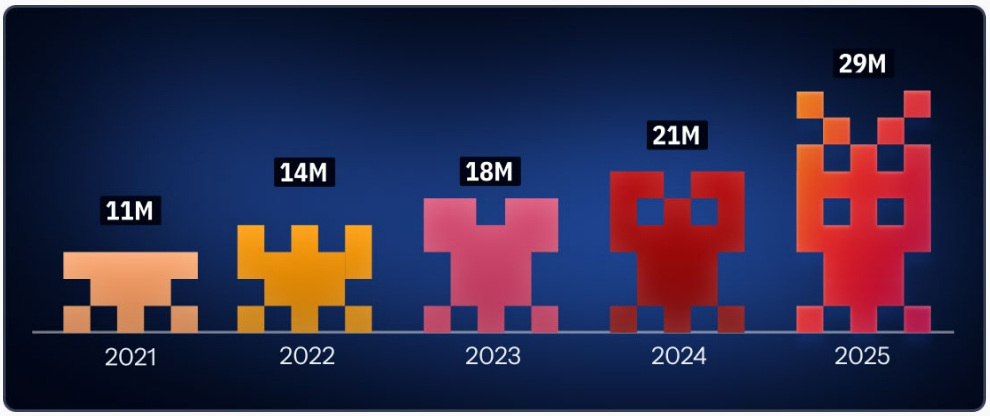
THE STATE OF Secrets Sprawl 2026

The Year Software Changed Forever

01. Executive summary

AI scaled software delivery, and secrets sprawl is accelerating faster than remediation

Secrets detected on public GitHub (2021-2025)



For the software industry, 2025 marked a turning point. The rapid rise of AI-assisted coding transformed "vibe coding" from leading-edge technology to mainstream practice in under twelve months, collapsing the barriers to software creation. We've entered an era where coding is no longer the domain of engineers alone.

Every year, GitGuardian monitors billions of public commits across GitHub, the world's largest code hosting platform, for hardcoded secrets, such as API keys, passwords, and certificates. In 2025, we found 28.65 million new hardcoded secrets in new public GitHub commits. This is not cumulative. That was just the number of secrets added in 2025. This marks a 34% increase from our previous report, which covered 2024, marking the largest single-year jump we have ever recorded.

28.65 million ↑ 34%

New hardcoded secrets in public GitHub commits

Far from being a new trend, this is an accelerating one. GitGuardian has tracked secrets sprawl since 2021, watching the numbers climb from 11 million to over 28 million in just five years. What feels new, aside from an accelerated pace of leaks, is what kinds of secrets we are seeing leaked. The tech stack has shifted, signaling a move to a world where everyone is using AI to produce code and build new kinds of applications, many of which are Agentic AI.

This year, the data led us to three major findings: **AI has reshaped the threat landscape, internal systems are a dangerous blind spot**, and **remediation remains a significant challenge for most teams.**

AI is fueling a new generation of leaks

AI-assisted development has moved from experiment to default, and credentials are leaking at every layer of the stack. Eight of the ten types of leaked secrets showing the sharpest increase year over year are tied to AI services. LLM infrastructure (orchestration, retrieval augmented generation (RAG), vector storage) is leaking **5x faster** than core model providers. Despite AI guardrails, developers who rely on Claude Code to produce code and co-author commits leak secrets at 2x the baseline rate, demonstrating that the human factor remains critical. And MCP servers exposed **24,000+ secrets** in their first full year of adoption.

Internal systems are 6x more prone to leak

Public repositories tell only half the story. Internal repos are **6x more likely** to contain hardcoded secrets than public ones. Secrets found in self-hosted, private GitLab instances and Docker registries were 3 to 4 times more likely to be valid. **28% of incidents** originate entirely outside repositories—in Slack, Jira, Confluence, and similar tools. These leaks outside of the codebase are **13% more likely to be categorized as critical** than secrets discovered inside the code.

64% of secrets leaked in 2022 remain valid and vulnerable today

Every year, we analyze new leak data, and we also re-check previous findings to find patterns over time. In the 2025 State of Secrets Sprawl report, we showed that nearly 70% of credentials confirmed as valid in 2022 were still valid as of January 2025.
When we retested the same dataset in January 2026, the validity rate remained over 64%. **This means those secrets have been exploitable by anyone who finds them for 4 years.**

This confirms the overall trend we first saw last year: once a real credential leaks into a public commit, it often remains usable for far longer than any team would claim acceptable. Detection is only the first step. Until revocation and rotation become routine, owned, and automated, a leaked secret is not a short-lived mistake. It is a durable access path that can sit in plain sight for years.

The path forward

We're in the middle of a revolution transforming how software is created, distributed, and protected. Unless we act, the rate of secrets sprawl will only keep accelerating. The industry must shift from reactive detection to risk-intelligent NHI governance.

Organizations need to move from the position of only asking "Where are my leaked secrets?" to one where they can answer the questions of "What non human identities exist in my environment?" "Who owns them?" and "What data or resources can they access?" This is especially true for any organization that plans to embrace agentic AI at scale.

Key numbers at a glance

Metric	2024	2025	YoY
Total secrets detected	21,391,792	28,649,024	+33.9%
AI services secrets	702,613	1,275,105	+81.5%
Public commits	1.36 billion	1.94 billion	+42.7%
Active developers	17,108,640	22,790,156	+33.2%
Repositories with secrets	2,867,503	4,012,054	+39.9%
Secrets per repository	~0.31	~0.32	Stable
Secrets per 1K commits	15.7	14.8	-6%

02. How leaky was 2025?

Record commit volume, record leakage:
exposed credentials remain a persistent breach vector

Four forces reshaping software security

1. More people are building apps with AI, not just developers

Public commits YoY growth rate

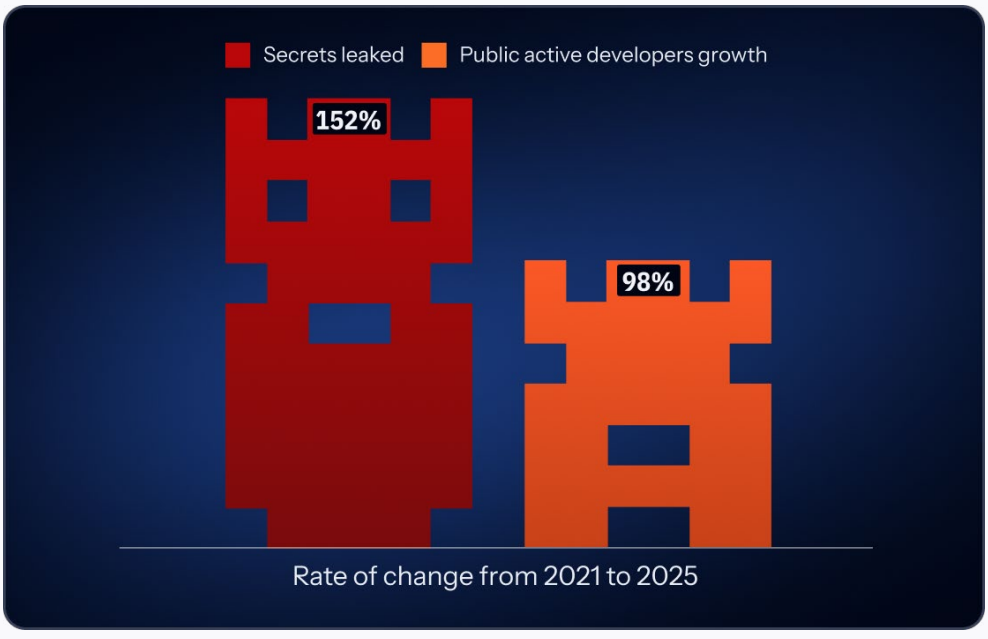
Active developers (2019-2025)

GitHub's public user base grew by 33.2%, from 17.1 million in 2024, to nearly **22.8 million active developers by the end of 2025.**

Since 2019, the developer population has tripled, up from **7.6 million**. Meanwhile, public commits surged **42.7% to 1.94 billion**, nearly quadruple the 514 million commits in 2019.

Since 2021, leaked secrets have grown substantially faster than the public developer population, 152% vs 98%.

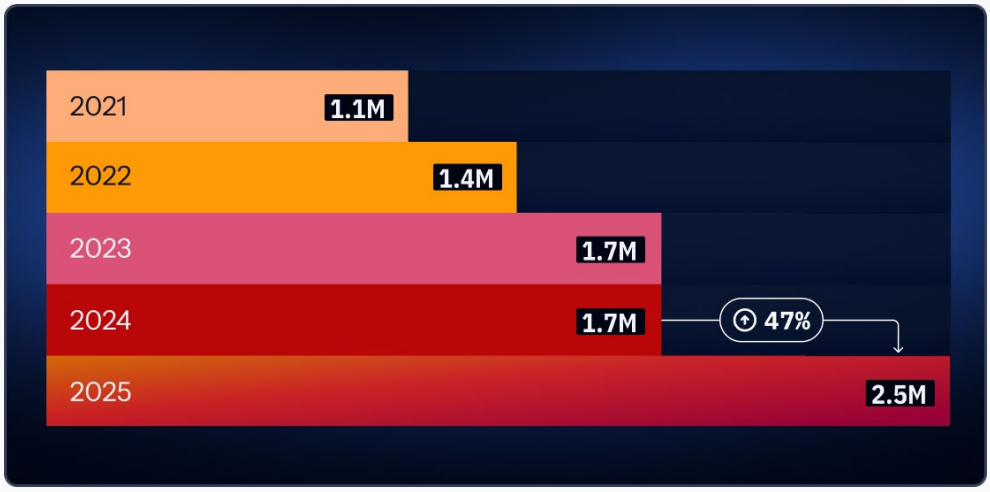
Rate of secrets leak is faster than new developers joining



A record-breaking year for Good Samaritan alerts

2025 was a record-breaking year for GitGuardian Pro Bono alerting, the free automated system that notifies the commit author immediately when we detect a secret in a public commit. Our platform sent **2.5 million email alerts** in 2025, a **47%** increase from 2024.

Pro Bono alert emails sent (2021-2025)



On the other hand, the number of secrets per repository has remained remarkably stable over the past years. Part of this stability likely reflects GitHub Push Protection doing its job of catching some of the most common credentials before they can become public.
But even with this safety net in place, the explosion in total secrets is **directly correlated with the unprecedented volume of code reaching GitHub**.
Push Protection holds the line on density, but can't stop the tide of volume.
More developers, more code, more secrets.

2. The AI infrastructure boom

The most striking trend in 2025 data is the explosion of AI-related credentials.

1,275,105 81%
AI services secrets detected in 2025

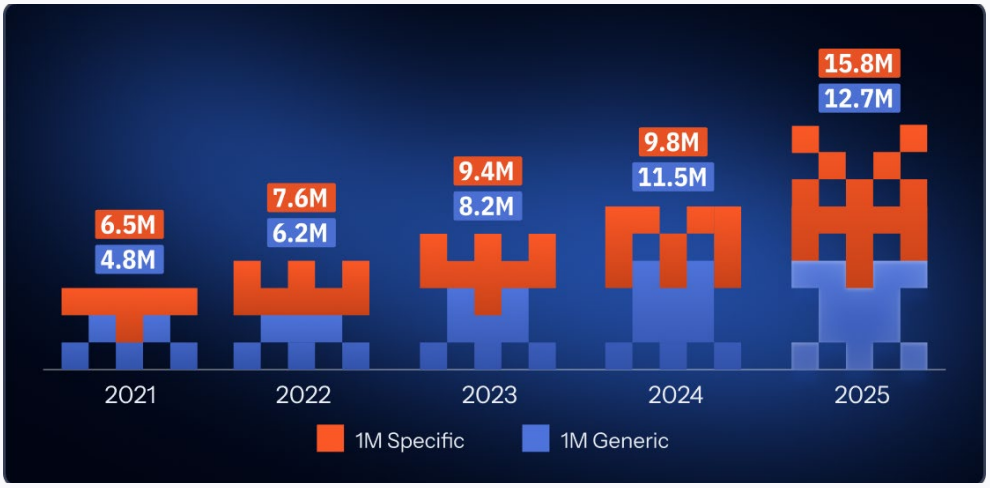
We found 1,275,105 leaked secrets for AI services in 2025. This is an 81% increase over 2024. **API keys from up-and-coming vendors are way more likely to slip through the cracks of GitHub's Push Protection** and create a security incident.

As new AI providers emerge, there's an inevitable lag before protection catches up.
A good example of this is the 113,000 new DeepSeek API keys we found in 2025.

3. Specific secrets surge

This year, for the first time, the majority of leaked secrets found were tied to a specific service or provider. **55.4% of all sprawled secrets were found using our "Specific Detectors,"** up from just 46.3% in 2024.

Specific vs generic secrets (2021-2025)



Several factors explain this shift.
First, modern applications integrate more third-party services than ever, each adding specific, pattern-identifiable credentials to the codebase: payment processors, cloud storage, authentication providers, etc.

Second, GitGuardian continues to expand its specific detector coverage. We added 105 detectors in 2025. Among them were a lot of AI-specific detectors (80% more than in 2024). In other words, secrets that might previously have been flagged as generic, based on usage and context, are now correctly identified as belonging to a specific provider.

The AI infrastructure boom is directly inflating the specific detectors category as well. **AI's share of secrets found with specific detectors grew 26% year over year, up from 2.7% in 2024 to 3.4% in 2025.**

The fastest growing specific detectors (YoY)

* For detectors >3k findings

4. Secrets at the source: the developer workstation

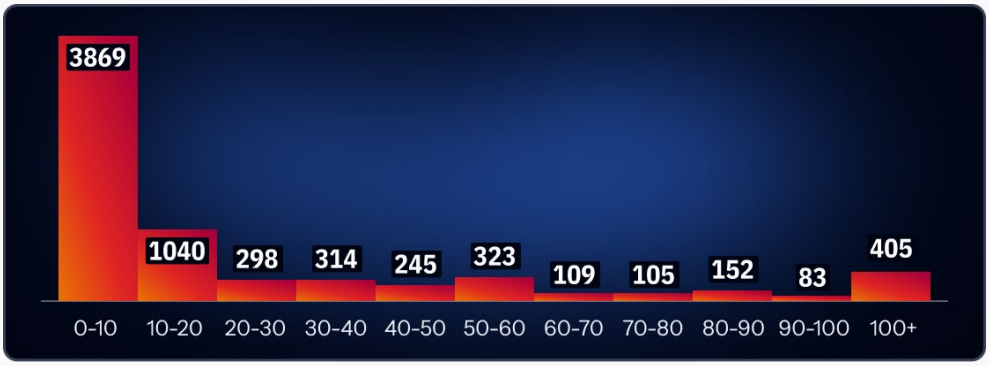
This year, for the first time, GitGuardian has direct empirical data on what secrets actually look like from a developer perspective at scale. Developers have always been part of the attack surface, but we have seen an acceleration of credential harvesting attacks in the past 6 months.

The Shai-Hulud 2 supply chain attack gave us a unique window to answer this question: **How many secrets live in a typical development environment?**
By compromising npm packages and executing at install time, the malware systematically harvested environment files and ran structured local secret scans across thousands of real machines.
GitGuardian analyzed data from this recent attack, and re-scanned them with our detection engine, treating the attack dataset as a proxy for the true state of secrets on development endpoints.

The results are striking. Across **6,943 compromised machines**, we identified **294,842 secret occurrences**, corresponding to **33,185 unique secrets**. At least **3,760 were still valid** at the time of analysis. On average, each live secret appeared in **roughly eight different locations** on the same machine. This means the same credential is replicated across dotfiles, shell profiles, build outputs, IDE configs, and tool caches. Each copy is an independent vector for theft.

The distribution of secrets per machine tells its own story.

Shai-Hulud 2 - Count of secrets per machine



Most machines fall in the 0–10 range, but the long tail is significant.

~30% of compromised machines held more than 10 secrets, and 5% of machines carried over 100

The types of credentials exposed reinforce the real stakes.
GitHub tokens dominated the validated set:

- → **581 personal access tokens**
- → **386 OAuth tokens**
- → **104 Fine-Grained PATs**
- → **101 GitLab tokens**

Each one enables repository access, workflow manipulation, or potential lateral movement across the software supply chain.
And because 59% of compromised machines were CI/CD runners rather than personal workstations, this exposure extends well beyond the individual developer into shared build infrastructure.

We feel these numbers are very conservative. The attacks only ran where the malicious package was installed; it did not reach agent memory folders, IDE caches, or the growing surface area of AI-generated artifacts that now routinely include credentials.
The real density of secrets on developer machines is almost certainly higher.

03. How AI is fueling a new generation of leaked secrets

AI multiplies services and tokens, expanding attack surface across the stack

2025 was the year the "AI stack" became part of the normal tech stack, as evidenced by the kinds of secrets we found, including model keys, retrieval tools, fast backends, and AI monitoring tools. In larger terms, it shows AI adoption is expanding the number of services, truly non-human identities that connect to make up a modern project. That expanse of NHIs drives the number of credentials that can leak.

> **"80% of new developers on GitHub use Copilot in their first week."**
>
> GitHub's Octoverse 2025 report

4 key insights:

1. **8 of the top 10 fastest-growing types of leaked secrets YoY are tied to AI services**
2. LLM infrastructure (RAG, orchestration: Brave Search, Firecrawl, Dify) is growing **5x faster** than core AI labs (OpenAI, Anthropic, Groq...).
3. Developers entering via AI-assisted coding favor plug-and-play managed services like Supabase (+992% leaked secrets YoY), Fastly (+577%) over custom infrastructure.
4. Secret leak rates in AI-assisted code were, on average across the year, roughly double the GitHub-wide baseline, with a clear spike mid year, before the models caught up.

OpenRouter's position at the top of the fastest-growing detectors leaderboard (not limited to AI) clearly shows developers' preference for model-agnostic platforms over single-vendor lock-in. Like Groq (+211%), NVIDIA (+184%) or Hugging Face (stable), these platforms serve as inference gateways, letting developers access and swap among multiple LLMs via a single API, rather than committing to one provider's proprietary model.

Top 20 growing specific AI detectors

The expanding AI attack surface

Developers are building full products around LLM services, such as orchestration, monitoring, vector storage, and retrieval services, with each layer adding credential exposure risks.

Supabase (on the graph in the previous section) is a good example of this, as it has now arrived in the top 20 most leaked secrets, representing 1.27% of the total, with over 248,600 occurrences. Supabase is widely loved because it makes it easy to stand up a database-backed application quickly, and it has become a common default for modern AI projects. Last year, we reported 97% year-over-year growth in leaks. **This year, that growth rate jumps to 992%.**

Other support services follow the same curve:

- Retrieval APIs feeding context to models: **Brave Search** (+1,255% year-over-year increase in leaked secrets), **Firecrawl** (+796%), **Perplexity** (+657%)
- Orchestration for multi-step AI workflows: **LangChain** (+108%)
- Experiment tracking and evaluation: **Weights & Biases** (+114%)
- Embeddings and search: **Jina** (+334%)

Beyond infrastructure, we're seeing leaks from platforms that use AI to deliver services, particularly in media generation and agent workflows.

Clipdrop is an image generation and editing tool, and **Vapi** is a real-time voice AI for customer-facing agents.

Dify and **Coze** are tools for assembling agentic systems, chaining prompts, and deploying AI features without writing everything from scratch. Coze personal access tokens (PAT) are particularly concerning, because these tokens often carry broad account-level permissions and frequently appear in local config files and automation scripts.

The pattern is clear: a single model integration quickly becomes a network of services and credentials, which explains why we are seeing LLM infrastructure leaks growing 5x faster than core LLM providers.

GitHub's fastest-growing project had a secret problem

In January 2026, OpenClaw (formerly Moltbot) became the fastest-growing project in GitHub history: **17,830 stars in a single day**, crossing 200,000 at the time of writing.

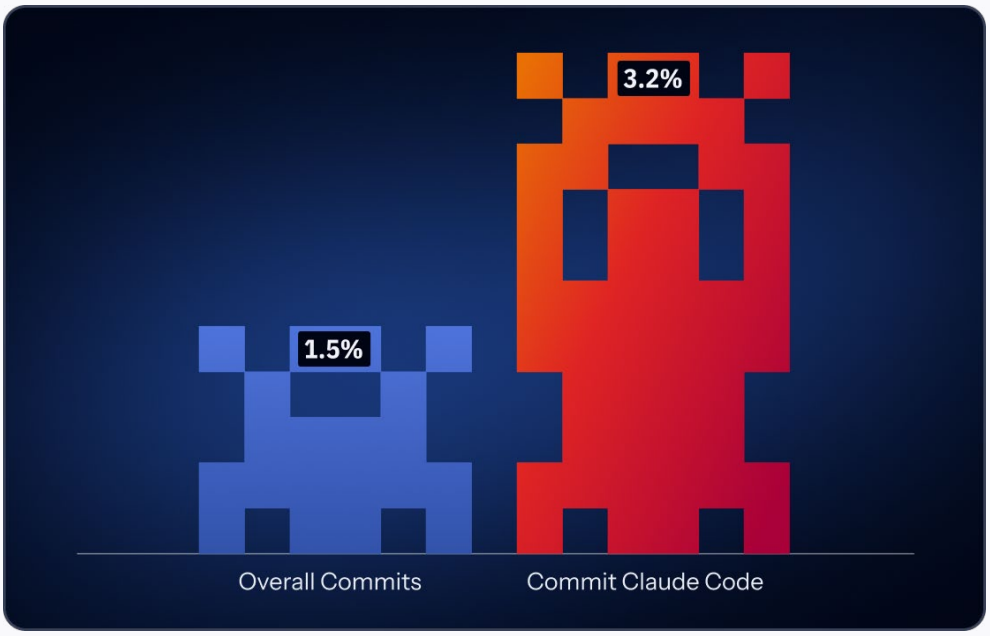
This overnight success unfortunately came with drawbacks: a few days later, GitGuardian detected 180+ leaked secrets from public Moltbot workspaces, including a Notion token exposing a healthcare company's entire corporate documentation and Kubernetes credentials granting full cluster access to a fintech startup.

GitGuardian offered a response with a ggshield skill for OpenClaw that lets users ask their AI assistant "Is this safe to push?", hoping to claw back some security for the community.

Claude Code co-authored commits leaked secrets twice as much across 2025

Claude Code co-authored commits leaked secrets twice as much across 2025. Across 1.94 billion public commits in 2025, the global average was 1.5% containing an exposed credential. Commits co-authored by Claude Code leaked a key 3.2% of the time.

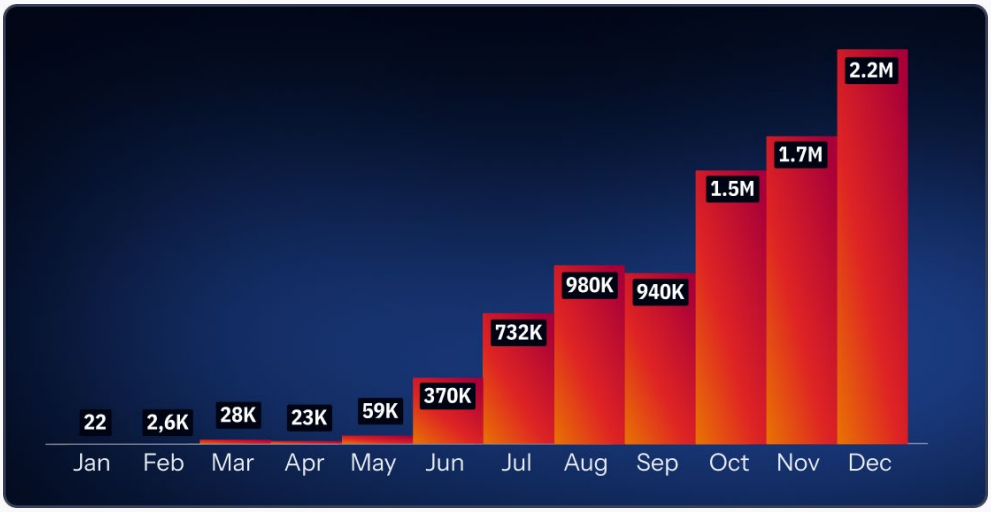
Secrets for all commits vs commits by Claude Code



Claude Code's rapid adoption vs. leak curve

Last year, we reported that repos using AI coding assistants leaked at a 40% higher rate. This year, we dug deeper into one agent specifically: Anthropic's Claude Code. First introduced at the beginning of 2025, it saw massive adoption in the second half of the year. Monthly Claude Code co-authored commits went from just 22 in January to **2.16 million in December.** Across 2025, these commits represented just 0.4% of everything we publicly scanned, but accounted for **0.9% of all leaks.**

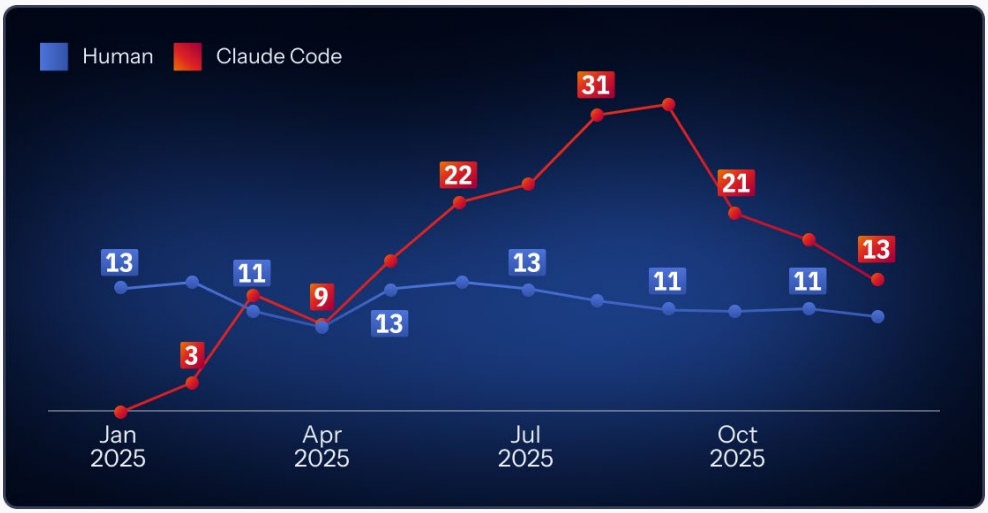
Count of commits co-authored by Claude Code



Across public GitHub repositories in 2025, Claude Code-assisted commits showed a clear, measurable elevation in secret leakage early in the year, paired with a distinctive commit profile. Our analysis compared commits identified via Claude Code "Co-Authored" commits against human-only commits. As Claude Code usage ramped, the average Claude Code-assisted commit carried more change per commit and leaked secrets more often per commit than the baseline.

The first half of 2025 showed the most pronounced spike in leak rate. Claude Code assisted commits climbed from near zero in January to a peak of 31 secrets per 1,000 commits in August, roughly 2.4x the human baseline at the high point.

Secrets leaked per 1k commits



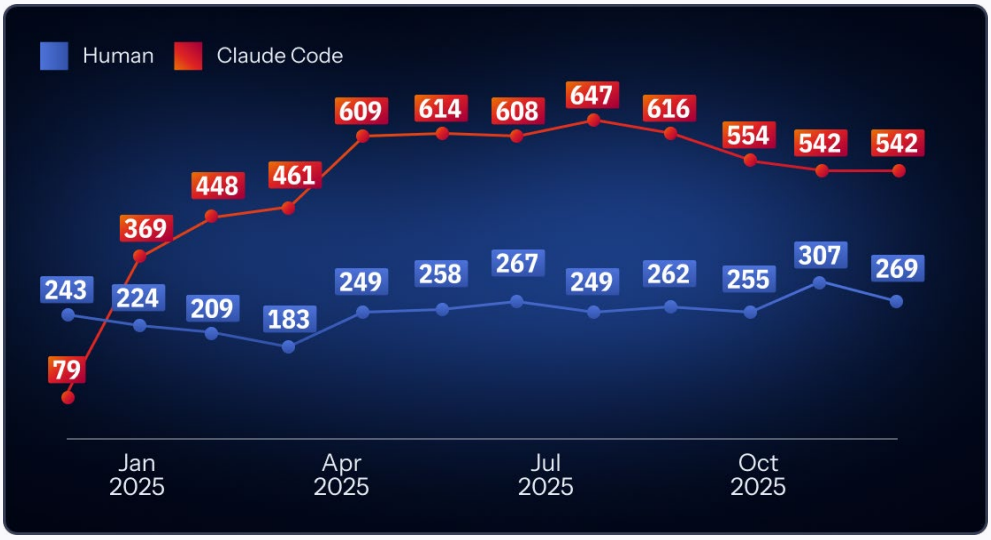
After peaking in July and August, the Claude Code leaked secret rate declined steadily, reaching 13 secrets per 1,000 commits by December, effectively matching the human baseline. This drop aligns with the late September 2025 release of Claude Sonnet 4.5 and suggests meaningful improvements in model behavior or in the surrounding workflow that produces Claude Code-assisted commits.

It is important to note that this trend shows a convergence in leakage rate, not a return to pre-AI conditions. Claude Code-assisted commits remain consistently larger, which raises the operational importance of secret detection even more.
As AI-assisted coding scales, the most reliable path is to treat larger AI-generated change sets as higher-impact review units, maintain automated scanning in the developer workflow, and keep remediation tight enough that a leaked secret does not remain valid long enough to be exploited.

More AI use means more lines of code

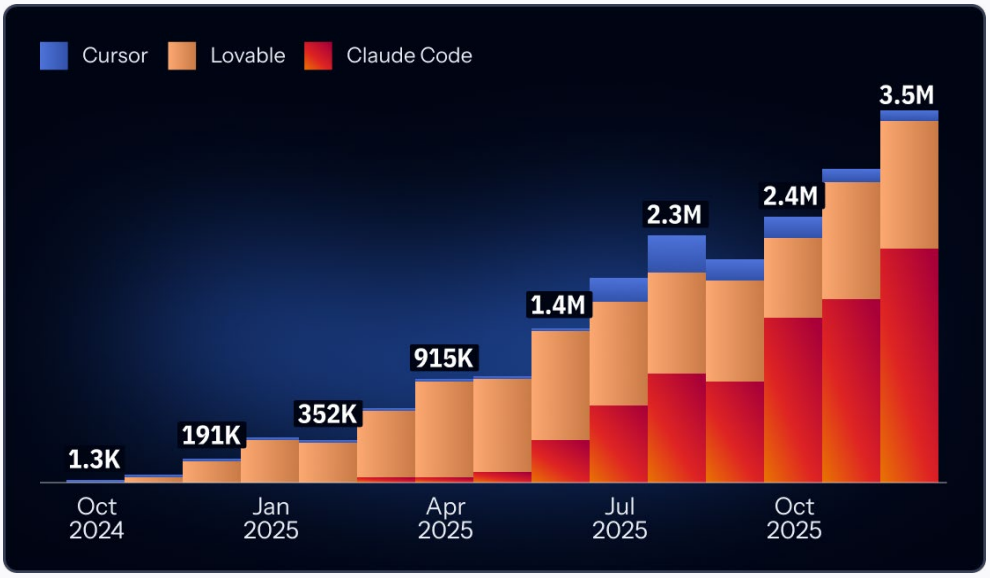
Commit size is another sign of how Claude Code assistance changes development. From April onward, Claude Code commits averaged about 2x the number of lines per commit. Larger commits mean more opportunity for credential exposure in a single review and a single merge. By the end of the year, though, the number of lines per human vs AI-assisted commits evened out, showing improvements in the model's results.

Mean of lines of code per commit



And Claude Code is one of many AI code assistants in use (see the Methodology section for how we classify AI-assisted commits, including bot-authored vs co-authored signals). AI-assisted code production ramped sharply through 2025, with multiple assistants contributing to the overall volume.

Count of commits assisted by AI



While our findings show an elevated leak rate in Claude Code-assisted commits, it's essential to recognize that AI models are tools. The developer remains firmly in control of the commit. Even as AI coding assistants like Claude Code continuously improve their security guardrails with monthly updates, developers retain the ability to override suggestions, ignore warnings, or even explicitly prompt the AI to include sensitive information. This human agency means that leaked secrets ultimately reflect human decisions, whether through oversight, time pressure, or intentional choices to bypass security recommendations.

As AI models evolve to become more security-aware, the corresponding evolution in developer security practices and organizational policies becomes equally critical. The solution isn't simply better AI, it's better human-AI collaboration with robust security awareness at every step.

Generic AI secrets

This year, for the first time, our machine-learning classification includes an AI category for generic secrets. 4.1% of all detected secrets fall into this AI/ML bucket with high confidence.

Our generic detection also includes a large "other" category (~42%) we can't map to a specific purpose, and we believe the true AI-related fraction is likely significantly higher. In other words, that 4.1% is almost certainly a floor, not a ceiling. And this already makes it the 4th biggest bucket behind Data Storage (29%), Messaging Systems (8.7%) and Social Networks (6.1%).

Generic secrets by category (%)

New services, same NHI issues, increased speed

AI is clearly changing which services show up in our ecosystems, but our breached NHI policy signals show it is not changing the underlying failure modes. Our research, for the first time, analyzed data from customers using our NHI Governance platform, tracking the state of each secret found. The policies we put in place are derived from the OWASP Top 10 for NHI Risks, which addresses the machine-to-machine authentication challenges that enterprises face.

The dominant issue is lifecycle negligence: 60.4% of flagged issues are long-lived secrets that persist well past their expiration date, when they should have been rotated or replaced.

Distribution of NHI policy breaches

From there, the “speed + reuse” pattern becomes obvious: 17% of issues are secrets leaked internally, and 15.6% are duplicated secrets, which is what you expect when credentials get copied into new workloads, new repos, and new pipelines faster than governance can keep up.

Public leakage (5.2%) is serious, but most risk accumulates quietly inside the organization, in the form of internal leaks (17%) before anything reaches the public internet.

If AI is empowering teams to ship faster while these same trends remain so prominent, we can extrapolate something uncomfortable about AI-driven code and the state of NHI lifecycles: **creation velocity is outpacing identity maturity.**

AI accelerates this problem. It makes it easier to scaffold projects and connect services, but also easier to **reproduce insecure patterns at scale** when the default move is “just add a key.” Duplication and internal leakage together account for nearly a third of all issues (33%), pointing to a world where identities are created quickly, cloned freely, and rarely retired.

The conclusion is simple: creation velocity is outpacing identity maturity. New services will keep arriving. Without automated, enforced NHI lifecycle management, AI will keep amplifying the same loop of long-lived credentials, unclear accountability, and accelerating sprawl.

From secrets sprawl to NHI governance

AI-assisted development has moved from experiment to default. With it comes an explosion of non-human identities: service accounts, API keys, agent tokens, MCP configurations—all authenticating, all acting, all capable of leaking.

Our data shows the gap is widening:

Claude Code co-authored commits leak at
2x the baseline

LLM infrastructure leaks are growing
5x faster than core providers

MCP configs alone exposed
24,000+ secrets in their first year

This is no longer a detection problem. It's a governance problem. And governance starts with three questions every organization must answer:

1. **What NHIs exist in your environment?**
2. **Who owns them?**
3. **What can they access?**

If you can't answer all three, your AI adoption is outpacing your security posture.

Leaks in MCP configuration files

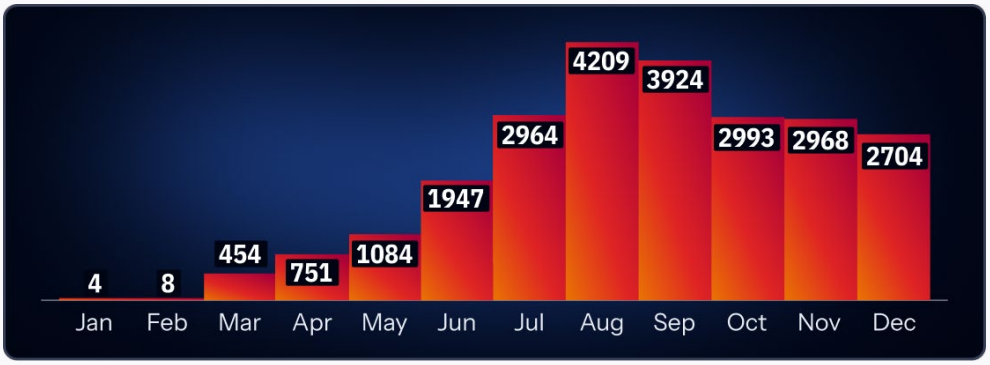
In early 2025, the Model Context Protocol (MCP) emerged as the standard for connecting LLMs to external tools and data sources.

As third-parties rushed to expose their services through this new channel, MCP configuration files quickly became a high-concentration point for hardcoded secrets.

We identified 24,008 unique secrets exposed in MCP-related configuration files across public GitHub in 2025.

In November 2024, Model Context Protocol (MCP) was first introduced. By early 2025, it had become the standard and reached a clear peak in August, but the story doesn't end there: the curve stays elevated through the rest of the year.

Unique secrets count per month - MCP configuration



The official guides are part of the problem

MCP setup guides often normalize putting credentials directly into MCP configuration. Popular MCP servers' quickstarts show API keys passed as command-line arguments inside the MCP server config (e.g., --figma-api-key=YOUR-KEY) or stored in env inside the same JSON file that gets committed to version control. Postgres MCP examples include full postgresql://username:password@... connection strings under DATABASE_URI. When official documentation treats hardcoding as a default, sprawl follows.

Best practices for MCP credential management:

- **Never store secrets in MCP config files.** Use environment variables managed by a dedicated secrets manager, not inline values in JSON or CLI args.
- **Clients, not servers, should own the secrets.** MCP servers should request credentials from clients at query time rather than embedding them in server-side configuration.
- **Exclude config files from version control.** Add MCP configuration directories to .gitignore and treat them as sensitive artifacts.
- **Only use MCP servers over secure channels.** Ensure remote servers are accessed via TLS.
- **Scan before you push**. Tools like ggshield can detect secrets in MCP config files before they reach version control.
- **Enforce human-in-the-loop for sensitive actions.** Require manual approval before any MCP action touching production systems, databases, or deployment pipelines.

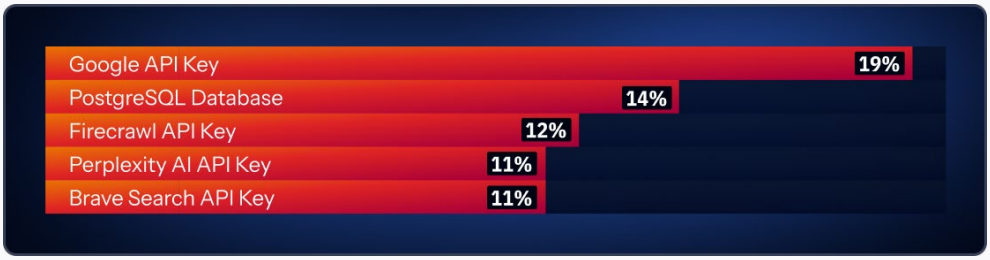
When it comes to risk, however, volume alone isn't enough. Not every detected secret is exploitable. Valid ones definitely are. We found 2,117 unique, valid MCP-related credentials, representing 8.8% of our findings.

2,117 8.8%

unique credentials were verified as working at the time of detection.

The top types of found MCP-related secrets map directly to common API platforms and web-search tooling (e.g., Google, Perplexity, Brave), data access layers (e.g., PostgreSQL and MongoDB connection strings), and developer productivity services (e.g., Figma, Notion, Netlify, Sentry).

TOP 5 valid unique secrets in MCP configuration



What risks do these MCP configuration secrets present?

1. **Direct data exposure**
   database secrets can enable read/write access to production datasets, often with overly broad permissions.

2. **Platform and API abuse**
   leaked API keys can be monetized through usage fraud, quota exhaustion, and account compromise.

3. **Workflow compromise**
   tokens for build/deploy and monitoring services can enable persistence, tampering, or covert exfiltration in the development pipeline itself.

As teams integrate more systems to enable AI-assisted workflows, the number of secrets grows, and without robust guardrails they sprawl into places that are easy to copy, share, and inadvertently publish.

Smithery

AI Infrastructure: A new and vulnerable ecosystem

GitGuardian's security research team disclosed a critical vulnerability in Smithery.ai one of the most popular MCP server registries, demonstrating how a single path traversal bug in the platform's Docker build process exposed an overprivileged token which granted **arbitrary code execution on all 3,000+ hosted MCP servers** and access to the API keys and secrets of thousands of customers across hundreds of services.

It's a stark reminder that as MCP registries and hosting platforms mature, they become high-value targets: compromise one centralized layer, and you can cascade across many integrations.

04. Internal leaks: code & other data sources

Internal repos 6x more likely to contain a leak. Secrets leak beyond code: repos, tickets, chat, docs, and automation

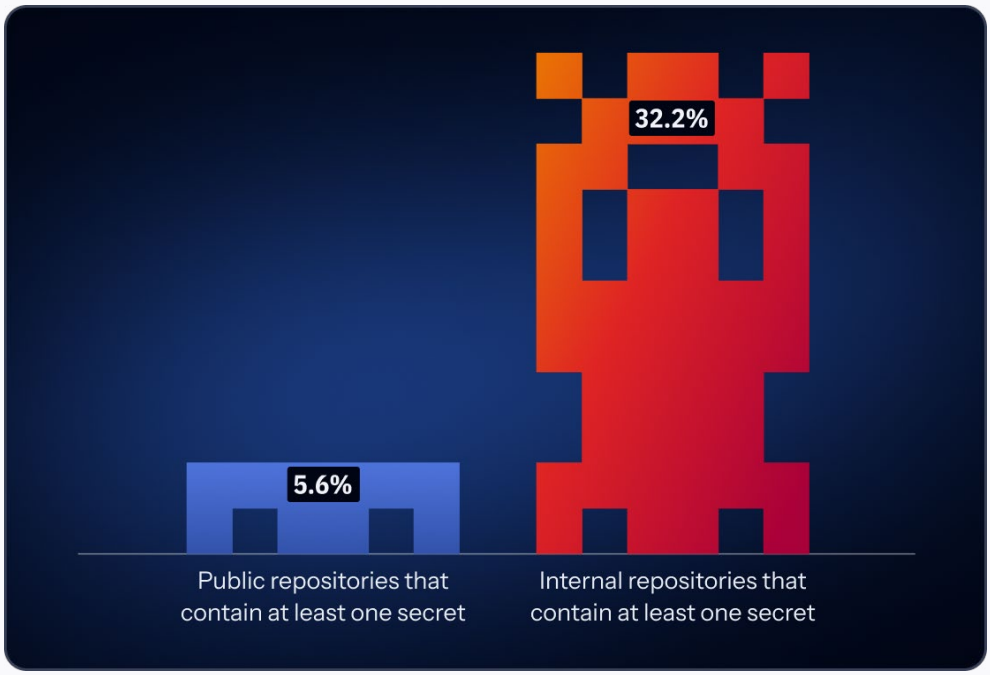
Internal repos 6x more likely to contain a leak

Publicly exposed secrets only tell half the story.

According to our data, internal repositories are 6× more likely to contain hardcoded secrets than a public one.

While 5.6% of public repositories we monitored in 2025 exposed at least one hardcoded secret, the number was 32.2% of internal repositories.

Public and internal repositories that contain at least one secret



The reason for this gap can be tied to the antipattern of "security through obscurity." Whether intentional or not, development teams tend to be less cautious within a closed perimeter, such as an internal or private code repository. They assume that exposing a secret there is less harmful because it isn't subject to public scrutiny. Yet this mindset means there is a silent buildup of hardcoded secrets that are slated to be removed "later."

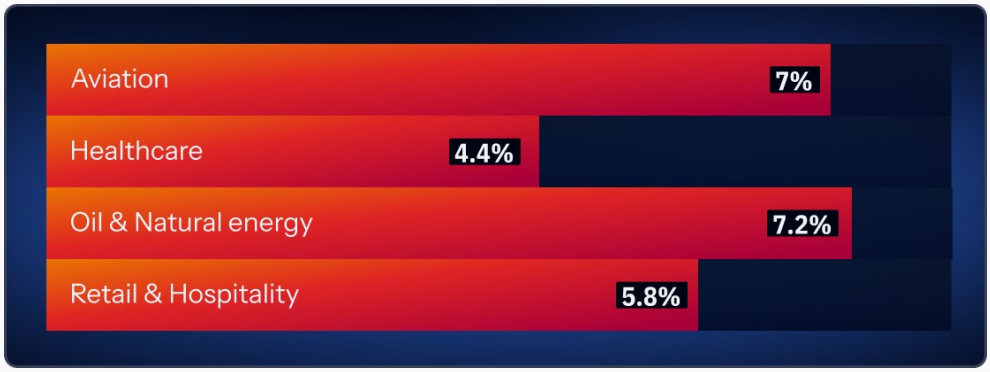
A single exposed secret can become a fast path to lateral movement. Focusing controls solely on catching leaks in public code misses the highest-risk part of an organization's leakage problem. Internal repositories tend to have a build up of the most mission critical credentials, used for CI/CD integrations, cloud resource management, database access, and internal tooling tokens. This is exactly what attackers are counting on finding once they find a foothold.

The right strategy is to treat internal repos as first-class leak sources. Prevent secrets at commit time when possible, detect continuously throughout the whole software development, and close the loop with fast rotation and remediation workflows rather than relying on repository privacy as a safety net.

Industry snapshots: How enterprise secrets get exposed on public GitHub

Public GitHub exposure is often framed as an "individual developer mistake," but this data shows a more consequential reality: a non-trivial portion of secrets found in public repositories are enterprise-owned or enterprise-relevant. That matters because enterprise credentials typically grant access to production systems, data, CI/CD, or third-party integrations. These are exactly the assets attackers want.

Public repositories active in 2025 that contain at least a secret by industry



These leaks don't only come from official company repositories. Developers often use personal accounts, or repos outside their GitHub organization, to prototype, collaborate, or deliver work connected to a company project. Too often, real credentials are used for these POC or modeling efforts, and those secrets can and do end up exposed in public. The outcome is the same as if they pushed an organization's private repository public: enterprise secrets become discoverable on the open internet.

Consulting firms: where secrets sprawl becomes a third-party incident

While the risk of internal developers leaking an organization's secrets is obvious, consulting firms and contractors bring just as much, if not a greater risk to the business. They regularly operate across multiple client environments, tools, and codebases while holding credentials, tokens, and configuration knowledge for all of those clients. This actually amplifies the danger while making it much harder to immediately tell whose secrets were actually leaked.

Looking at consulting firms appearing in the dataset, we found developers associated with 13 firms **leaked 1,834 secrets which were classified as critical or highly sensitive incidents.** We know the problem of third-party leaks extends well beyond just these firms.

141
sensitive incidents per firm

1,203
potentially impacted customer companies

Top 5 firms =
72%
of incidents

The Red Hat consulting breach

In October 2025, the cybercrime group "Crimson Collective" exfiltrated **570GB of data from 28,000 repositories** on Red Hat's internal consulting GitLab instance, affecting approximately **800 organizations** worldwide.

The leaked Customer Engagement Reports contained exactly what attackers prize: API keys, database credentials, authentication tokens, VPN configurations, and infrastructure details. Affected organizations included Bank of America, JPMorgan Chase, IBM, Cisco, the U.S. Navy, and NSA.

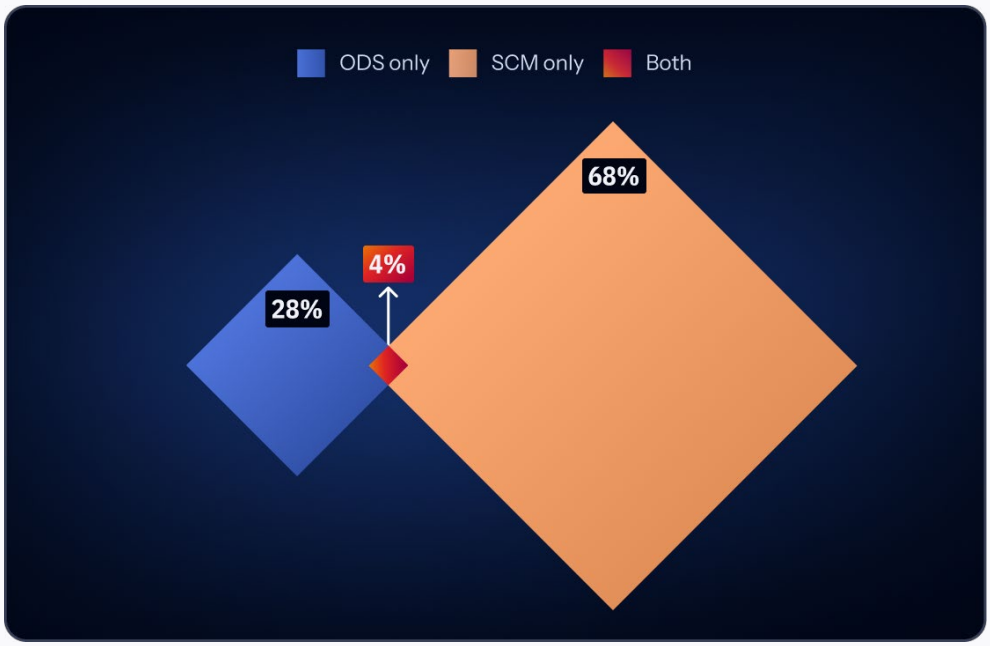
The attackers claimed these harvested credentials enabled them to pivot directly into customer infrastructure. The Centre for Cybersecurity Belgium (CCB) issued a high-risk advisory warning of potential supply chain impact.

The lesson: consulting firms—and any internal repository aggregating credentials across multiple clients—are high-value targets.

One in four internal leaks didn't come from code

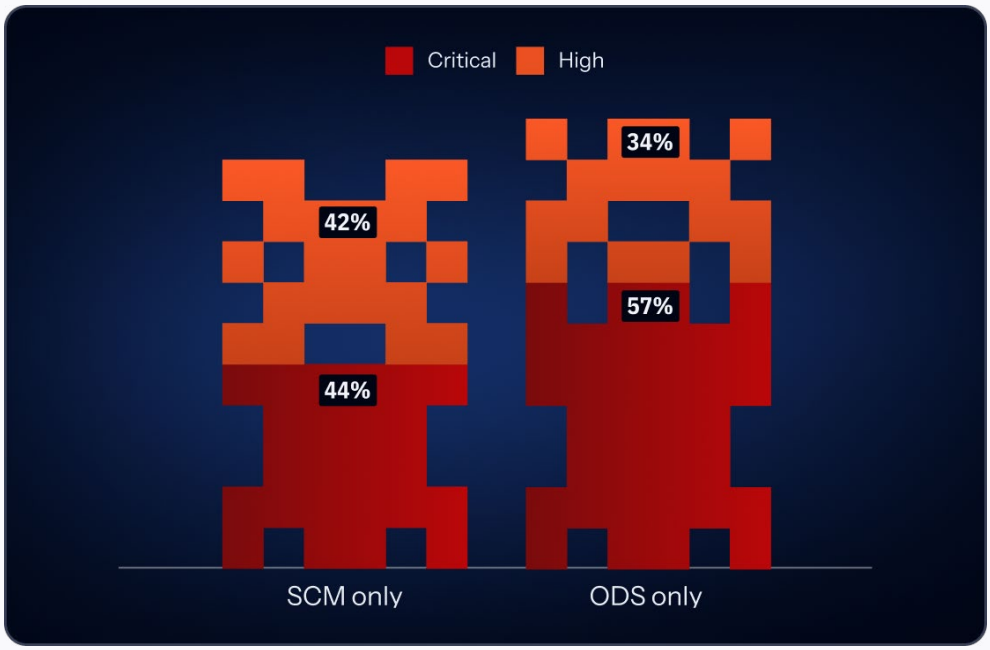
Secrets leak from places other than source code. Plaintext credentials also show up in collaboration and productivity tools like Slack, Jira, or Confluence. These platforms make it easy for people to paste access keys, often to speed up troubleshooting, incident response, or just day-to-day coordination. At GitGuardian, we refer to this large bucket of services outside of the code as "Other Data Sources" (ODS). These tools typically lack the built-in guardrails you see in code workflows, which makes them an increasingly important part of secrets sprawl.

Secrets in SCM and productivity & collaboration tools (ODS)



In 2025, most incidents, 68%, still came from code using source control management (SCM), such as Git. But a full 28% of leaks happened entirely outside code, in ODS. The overlap of secrets found in both SCM and ODS is surprisingly small: only 4% appear in both. This aligns with the broader pattern we highlighted in previous reports that secrets found in SCM and collaboration tools tend to be largely non-overlapping. We believe this means that often companies are using secrets managers, vaults, to keep secrets from being exposed in code, but those same secrets are being insecurely shared and exposed through these collaboration channels. Scanning only the code will miss a meaningful portion of leaks.

Incidents created in 2025 by severity



Secrets shared through Slack/Jira/Confluence and similar tools are more often rated as critical or high severity vs secrets found only in code.

The data from 2025 shows this split clearly. For SCM-only incidents, 43.7% are rated as critical. For ODS-only incidents, 56.7% were judged as critical, 13 percentage points higher than SCM-only leaks.

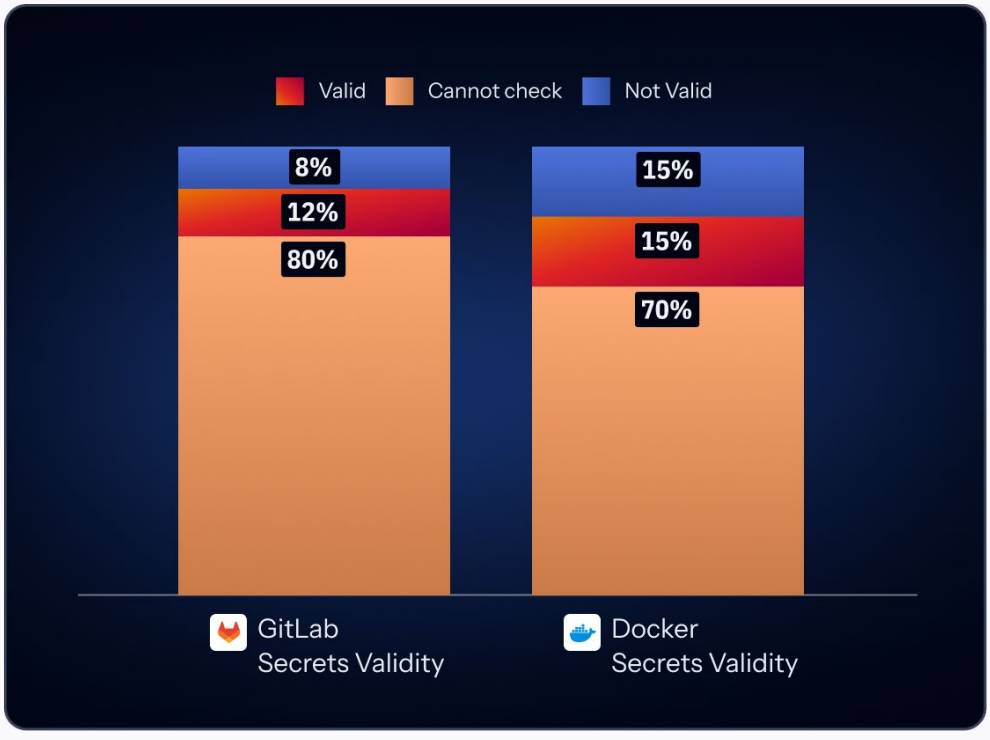
Secrets leaked from ODS are often shared during urgent troubleshooting or incident response work. On the other hand, in most organizations nearly all source code undergoes structured and multi-layer reviews, including Git workflows and CI checks, so many critical leaks are caught earlier. This means ODS coverage is essential to reducing risk across the organization.

Unintentionally exposed: 80K secrets leaked on self-hosted GitLab and Docker registries

Our research across 2025 revealed that thousands of self-hosted GitLab instances and Docker registries were left accessible on the web without proper authentication. These unintentionally publicly exposed resources were analyzed for secrets, leading to the discovery of 80,000 credentials. A full 10,000 of these were found to be valid, meaning immediately usable by attackers.

GitLab repositories contained a higher total volume of secrets, at 57,000 of the total, but a smaller percentage, only 12%, were valid. In contrast, Docker images contained fewer total secrets, 23,000, but 15% of these were found to be usable. The Docker findings are especially troubling, since unlike GitLab instances, Docker images are meant to be and are commonly shared across many clusters and nodes, amplifying the danger.

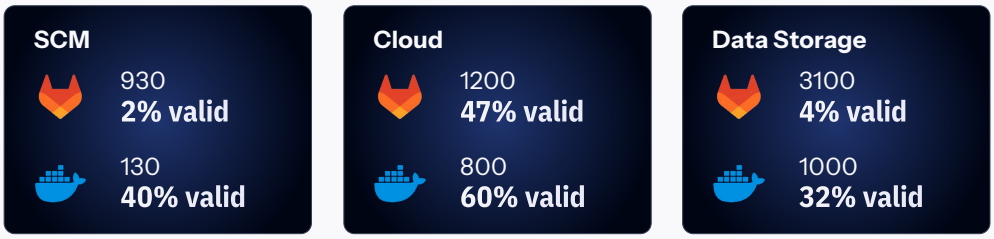
Secrets validity - GitLab vs Docker self-hosted instances



Total secrets: 57,000
Specific: **20,000**
Generic: **37,000**

Total secrets: 23,000
Specific: **9,000**
Generic: **14,000**

We also found that **the closer an asset is to production, the higher the likelihood of finding a valid credential**. Take cloud credentials as an example: 60% of those found in Docker containers were valid, compared with only 47% in GitLab. The gap is even wider for SCM secrets: 40% valid in Docker containers versus just 2% in GitLab—and for Data Storage credentials, 32% versus 4% respectively.



When combined with our other findings, it shows the rate of leaks from Docker and self hosted GitLab is 3-4x higher compared to public GitHub repositories. We believe this means hardcoded secrets in private resources are an overlooked security risk for too many teams.

Secrets exposures

We found secrets in 12% of scanned GitLab repositories and 18% of scanned Docker images. These percentages reflect asset-level exposure inside already discovered self-hosted GitLab instances and Docker registries.

Private Repos, Public Secrets

This reinforces a key insight from secrets sprawl: even if a secret originates from a private resource, it can end up in Docker images that could be publicly exposed. This creates a catastrophic nested, 'Russian dolls' effect: publicly exposed leaks contain valid secrets that grant access to private infrastructure, thereby exposing more secrets and compounding the initial breach.

An attacker's ideal attack path typically sneaks in at the boundary between an organization's internal private code, production configurations, compilation artifacts, and the public internet. Attackers work to exploit an organization's "private" secrets just the same as they would "public" ones. To an adversary, where a secret is found in plaintext is the least interesting part of finding one.

Beyond secrets, the publicly exposed data also carried strong indicators of private resources never meant to be public, such as:

- → References to internal database hosts and non-public infrastructure patterns
- → Confidentiality and privacy notices inside code
- → Significant Personally Identifiable Information (PII) including 300,000+ email addresses, of which 2,000 had .gov domains.

Roughly 18% of Docker registries became inaccessible just weeks after they were discovered by our scan. Yet, in several instances, credentials that had been exposed

remained valid even after public access was revoked. **Merely removing secrets from being exposed does not make you safe, only properly revoking them can.**

Privacy is not a security control

Internal repositories are
6x more likely to leak

Our research shows that internal repositories are **6x more likely to leak** than public ones. Private Collaboration tools account for **27% of incidents**, with higher severity than code-based leaks. Self-hosted infrastructure exposes secrets at **3-4x the rate** of public GitHub. And as the Red Hat breach demonstrated, a single compromised internal system can cascade into hundreds of downstream organizations.

The "security through obscurity" mindset has failed us. Attackers aren't waiting for secrets to appear on public GitHub. They're targeting internal GitLab instances, Docker registries, and collaboration tools directly. The Shai-Hulud campaign, the Red Hat breach, and our analysis of 80,000 exposed secrets all point to the same conclusion: the perimeter between "private" and "public" is porous, and secrets flow through it constantly.

Three principles for 2026:

1. **Treat internal repos as first-class leak sources.**
   The highest-value credentials, including CI/CD tokens, cloud access keys, and database credentials, live in private systems. Treat remediating these with the same rigor as you would any findings in public code.
2. **Extend detection coverage beyond just the code.**
   Critical secrets live in Slack, Jira, Confluence, and other collaboration tools, as they are often added there when teams are working through incidents or troubleshooting an issue. If you're only scanning repositories, you're likely missing a quarter of your potential exposure.
3. **Eliminate hardcoded secrets entirely.**
   The solution isn't choosing between public and private, or self-hosted versus cloud. It's removing the root cause: long-lived, static credentials that live in code, configs, and chat logs instead of secrets management systems. This is a path that is constantly becoming more viable as technology evolves.

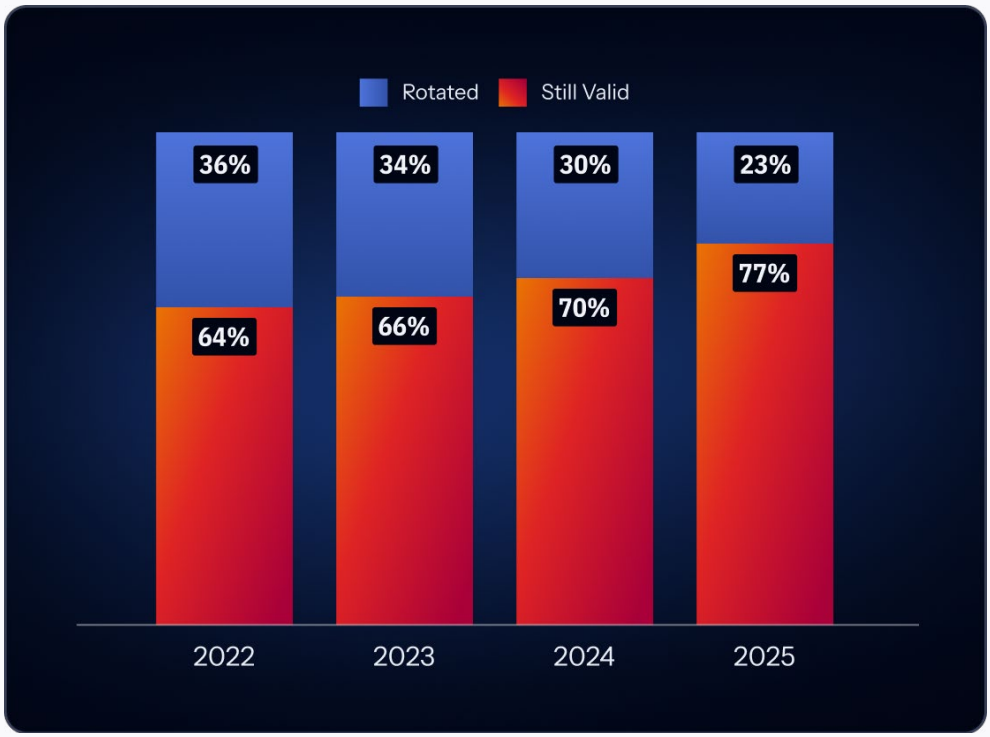
05. 64% of valid secrets from 2022 are still active and exploitable

Secrets persist for years, extending exploitability far beyond expectations

Every year, GitGuardian analyzes new leaks, but we also go back and retest some of our findings to see what has changed over time. In the 2025 report, we showed that nearly 70% of credentials confirmed as valid in 2022 were still valid as of January 2025, meaning they have not been rotated or otherwise remediated.

When we retested the same dataset in January 2026, the validity rate remained over 64%. That persistence is not a rounding error. It is an operational signal that remediation, not detection, is still the industry's limiting factor.

Percentage of secrets still valid after exposure



This is evidence that many organizations lack the governance needed to achieve a viable, repeatable remediation path for any leaked secret. These teams are still relying on long-lived secrets, ones never designed with lifecycle management in mind.

Rotation is almost never a single button click. Credentials are embedded in build systems, copied into multiple repos, baked into container images, referenced in CI variables, and shared across teams and vendors. The short-term choice many teams make isn't the safest one, but the one that means not breaking anything: "do nothing."

For these organizations, leaked secrets are durable access paths, sitting in public long after they were detected, because the organization cannot reliably replace them without breaking what they power.

46% of critical secrets are missed by validation-only prioritization

In 2025, a dangerous assumption persists across the security industry: **If a secret can't be validated, deprioritize remediating it.** While it is important to fix these valid secrets, relying on a validation step for prioritization can cause significant blind spots for your organization, leading teams to deprioritize remediation for some of your riskiest leaks.

Just because a secret "cannot be validated" does not mean there is no danger.

This is the security impact we are observing:

46%
of critical secrets are ignored

83%
of high-and-above risk secrets never get addressed

Precision hovers around
50%

meaning teams waste resources on secrets with minimal security impact

Three fatal flaws of "valid-only" remediation

1. Critical secrets live in the "long tail"

Secrets detection is a lagging security mechanism. There is always a gap between what can be detected and validated and what credentials actually exist.
The long tail is growing, not shrinking.

Building and maintaining validation checkers for the landscape of SaaS and PaaS providers is always a catch-up game. Across over 1,270+ providers with authentication keys, we observed throughout code bases:

- → Each provider requires dedicated infrastructure
- → APIs change without notice
- → New services launch constantly
- → Regional and industry-specific platforms proliferate

Among the **1.3 million secrets** that cannot be validated today:

17,000
are classified as critical

80,000
more are high risk

This has driven our work to optimize our generic detectors, which are able to catch critical secrets before they can cause harm. Do not ignore them by default just because they can't be validated.

2. “Generic secrets” drive half of incidents

We are using the term “generic secrets” as a shorthand to describe all the private keys, custom API tokens, and access mechanisms detected with our “generic detectors,” which use entropy checks and context to determine if a string is a secret. These generic findings are too often deprioritized by teams because they can’t be validated automatically. Many security teams treat them as low priority by default.

The data tells a different story:

35%
of critical incidents are connected to generic secrets

51%
of high or critical incidents can be traced back to generic credentials

Secrets detection is a very noisy process overall, making it hard to understand where to put remediation efforts. Validation-only approaches have no way to separate signal from noise here, so they just ignore the entire category. And it takes only one usable secret to compromise a system.

Google

In 2025, in a joint research with Google, GitGuardian analyzed one of the most difficult types of generic secrets to categorize: cryptographic private keys. Combining a dataset of 1 million leaked private keys with the historical data from the Certificate Transparency logs, the research found that 4.5% of leaked keys can be mapped to trusted X.509 certificates. While only 2,600 of those certificates were valid as of September 2025, a retrospective analysis showed that **half the private keys were associated with a valid certificate when they leaked, creating an exploitation opportunity for attackers.** This represents more than 4,000 HTTPS certificates getting compromised because of a leaked key per year.

Generic secrets can represent severe incidents, even for less obvious ones, like cryptographic private keys. Even more concerning, affected organizations included multiple Fortune 500 companies and a trusted Certificate Authority. All those victims could have had their web assets intercepted, representing a severe threat to their security.

As a result of this research, we have sent more than **4,000 alert emails to 600 organizations.** Disappointingly, the response rate for this disclosure campaign was under 10%. To us, this illustrates how most companies still fail at understanding the severity of the secret sprawl issue.

All keys and certificates were finally revoked after a coordinated disclosure to the Certificate Authorities.

3. “Valid” doesn’t always mean “dangerous”

Not all working credentials create the same level of risk:

→ Sandbox and test environment tokens often are not connected to critical assets.

→ Low-privilege service accounts with only access to trivial data or services.

Our risk models show **~10% of valid secrets** are inherently low-impact. Without performing the steps of proper contextual risk scoring, teams mostly rotate based on detection order or validation status, not on actual threat or risk level.

Security teams operating under a validation-only prioritization approach waste a lot of time rotating credentials that pose little danger, while high-risk secrets without checkers accumulate unnoticed.

The cost of getting prioritization wrong

Validation-only approach	Risk-based approach
Works only where checkers exist	Works across all secrets
Misses generic and unknown secrets	Surfaces the critical 1% of generics
Treats all valid secrets equally	Focuses on blast radius and privilege
High noise, low signal	High signal, manageable volume
46% of critical secrets were missed	**Full coverage with intelligent triage**

Why companies must establish a new standard for secrets security

A credential that validates successfully is not necessarily dangerous, and a secret with no validation checker is not necessarily safe. Understanding the risks a leak brings requires context beyond just validity.
Effective secrets security requires four capabilities working together:

1. **Enrichment** – Understanding what each secret unlocks
2. **Context** – Assessing privilege, scope, and exposure
3. **Risk scoring** – Prioritizing based on actual business impact
4. **Full coverage** – Addressing the long tail, not just the validated minority

> **"The difference between success and failure isn't finding more secrets, it's knowing which ones to fix first."**

The organizations that understand this distinction are building secrets security programs that scale. Those still clinging to validation-only approaches are systematically exposed to exactly the threats that matter most. This approach is also burning out their security teams with unmanageable alert volumes and eroding developer trust with false priorities.

The tools and approaches need to catch up with modern-day reality. That means moving past the false confidence in validation-only mentality and embracing risk-intelligent prioritization across the entire secrets landscape.

06. From reactive detection to NHI Governance

Conclusion

Secret detection will always be reactive because it observes something that already exists. The goal should not be to just have “fewer secret leak findings,” but to move toward a world of **complete visibility into the state of secrets**, especially for non-human identities (NHIs). This is at the heart of any NHI Governance strategy.

This path starts by placing all discovered secrets in centralized vault platforms and making those vaults **the source of truth.** When teams can reliably retrieve secrets from a single source, they stop inventing their own fragmented secrets-storage strategy, which is a major driver of secrets sprawl.

Next, once the secrets are safely stored in a vault, teams need to focus on **automating rotation.** This is critical. If a secret has to exist, it should not live forever. Regular replacement of valid secrets shortens the window an attacker can exploit them, and it forces teams to treat credentials as something with a lifecycle, not as a one-time setup task. Rotation is how you turn “we found it” into “it cannot be used.”

Fixing developer workflows so the secure path becomes the default path is essential.

This is made a lot simpler once a vaulting strategy is in place. Developers will keep shipping code with secrets because it gets the code to work and the feature shipped.

The **only sustainable approach is to make it easier to create, store, and call valid secrets,** and making it more attractive than just hardcoding those keys. Remove the need for shared .env files and copied tokens with better alternatives. Another part of improving developer workflows is to **empower them to meaningfully shift scanning earlier** so incidents never happen. Many developers dislike security tools because they have historically been noisy, have produced false positives, and have provided no remediation guidance. This has shifted, though, at least for GitGuardian, as we help teams stop a secret leak before it lands anywhere permanent. That is why ggshield belongs on the workstation and why the GitGuardian VS Code extension matters.

Finally, start the transition to identity-based authentication for NHIs as fast as your environment allows. Frameworks like SPIFFE, implemented by open source SPIRE, replace “shared string” authentication with strongly attested workload identity, where each workload is issued a verifiable proof of identity. In this model, each workload can obtain just-in-time, short-lived authentication to other services. This **turns “credential handling” into “identity management”** for service accounts, CI jobs, Kubernetes workloads, and AI agents. It is also an approach many commercial IAM companies are embracing. It aligns directly with mature NHI governance because ownership, scope, and policy become enforceable at the identity layer instead of being implied by scattered secrets.

The long-term exit from long-lived secrets is reducing static credentials and adopting short-lived, identity-driven access wherever possible.

That is the bridge from secrets management programs to NHI governance. The objective should not be to only find leaked strings, but to continuously be able to prove what non-human identities exist, who owns them, and what they can access.

Methodology (State of Secrets Sprawl 2026)

This report measures secrets exposure trends on public GitHub in a way that is consistent over time and engineered to minimize false positives, especially for "generic" secrets that do not follow stable provider-specific patterns.

Study perimeter and data source

Unless otherwise specified, metrics in this report are derived from **GitGuardian's continuous monitoring of public GitHub activity.** We analyze commits and repositories for exposed authentication material ("secrets") using GitGuardian's detection engine, and we aggregate results
on a calendar-year basis for year-over-year comparisons.

To keep the time series comparable, we apply the same counting logic across years and update historical figures when improvements to detection/false-positive handling materially change measurement quality (see "Rescans and false-positive reduction" below).

What we count as a "secret"

In this report, a "secret" refers to digital authentication material that can grant access to systems, services, or data (for example: API keys, tokens, passwords, private keys, credentials embedded in connection strings). We distinguish between:

- → Specific secrets (linked to known services and patterns, e.g., provider API keys with recognizable formats)
- → Generic secrets (unstructured credentials such as passwords, private keys, or custom tokens that don't match a standardized format and historically have higher false-positive risk)

Detection approach & counting rules

GitGuardian uses a combination of deterministic detectors (patterns, checksums, prefixes) and ML-assisted classification for generic secrets.

To ensure the numbers reflect real risk rather than pattern noise, we apply filtering and validation logic per detector:

- → For detectors known to be more susceptible to generic false positives, we apply an additional check and only count a secret when its **first occurrence** is determined to be valid.
- → For detectors with inherently high precision (for example, those relying on strong prefixes/structures), we include detections directly without that extra filtering step.

This approach follows the same principle used in the prior report to raise precision while limiting the impact on recall for detectors that intentionally cast a wide net.

False-positive reduction (FP Remover) & why 2026 numbers are more exact

A major methodology upgrade in the 2026 report is how we control false positives.
For the 2026 edition, we strengthened accuracy by using our upgraded False Positive Remover to reduce generic false positives at scale and improve consistency across the full historical window.
To ensure the 2026 time series is consistent and benefits from the improved FP handling, we reprocessed historical public GitHub data back to 2021 using the updated pipeline so that comparisons are not distorted by "old vs. new" false-positive behavior.

False-positive reduction (FP Remover) & why 2026 numbers are more exact

A major methodology upgrade in the 2026 report is how we control false positives.
For the 2026 edition, we strengthened accuracy by using our upgraded False Positive Remover to reduce generic false positives at scale and improve consistency across the full historical window.
To ensure the 2026 time series is consistent and benefits from the improved FP handling, we reprocessed historical public GitHub data back to 2021 using the updated pipeline so that comparisons are not distorted by "old vs. new" false-positive behavior.

Our False-Positive Remover is an ML module designed to identify benign strings that look like secrets in isolation but are not exploitable credentials in context. This is cutting false positives substantially while improving the actionability of detection at scale.

Operationally, this means deliberately accepting a small risk of filtering out marginal/ambiguous cases (including some "negotiable" items like test/dummy credentials) in exchange for higher confidence and better prioritization of the exposures most likely to matter. Internal evaluation materials indicate very high precision (≈99%+) and low residual error rates in practice.
The same ML stack is also used to help rank incidents by risk using contextual metadata (location, file type, branch, secret type, age, occurrences).

Outlier handling

To avoid distorted metrics, we exclude extreme outliers where repositories exhibit abnormal leak rates (e.g., behavior consistent with automated generation or repetitive committing that can skew aggregate trends). This mirrors the perimeter controls applied in the 2025 methodology to preserve the integrity of comparative metrics.

Self-Hosted GitLab and Docker Registry

Our research methodology consisted of three phases: Internet-scale discovery, targeted content retrieval, and automated secrets scanning. We first identified potentially exposed self-hosted GitLab instances and container registries using a mix of Certificate Transparency Logs hostnames extraction and Internet scanning. This included Shodan product queries and HTTP fingerprinting using GitLab-specific headers. We then confirmed unauthenticated access by probing standard endpoints.
For GitLab instances, we tested exploration and API paths, and for Docker registries, we checked /v2/_catalog. For registries, especially those reachable only over plain HTTP, where common tooling like docker pull often fails, we interacted directly with the Docker Registry HTTP API to enumerate repositories and tags, fetch manifests, and download blobs/layers. Finally, we scanned the retrieved repositories and image contents with automated detectors. Where possible, we performed validity checks against known service endpoints to identify credentials that were still usable.

Definition of "AI-assisted commits"

A public GitHub commit is classified as AI-assisted when it can be fully or partially attributed to an AI coding tool using signals in commit metadata:

- Bot: the commit was authored or pushed by an AI agent operating under its own GitHub App identity (matched against known AI bot accounts). In this case, the AI tool is the direct author.
- Co-authored: the AI-assisted commits were made by a human but include a Co-Authored-By trailer referencing a known AI tool email (e.g., noreply@anthropic.com for Claude). Here, the AI tool is a collaborator and a human ultimately committed the change.
- If both are present, Bot takes precedence, since it indicates direct control over the commit.

These definitions are used to quantify AI-assisted activity and to compare leak incidence rates for AI-assisted vs baseline commits.

Remediation

Dataset: This analysis covers all open secrets incidents from business workspaces using the GitGuardian platform in the US tenant.

Risk score: Each secret is scored using GitGuardian's Risk Score, a 0-to-100 scale that combines contextual signals and metadata. Secrets scoring 85 or above are classified as critical, and those scoring 65 or above are high risk.

5th Edition

The State of Secrets Sprawl 2026

Data analysis by **GitGuardian**

Learn more at gitguardian.com

GitGuardian