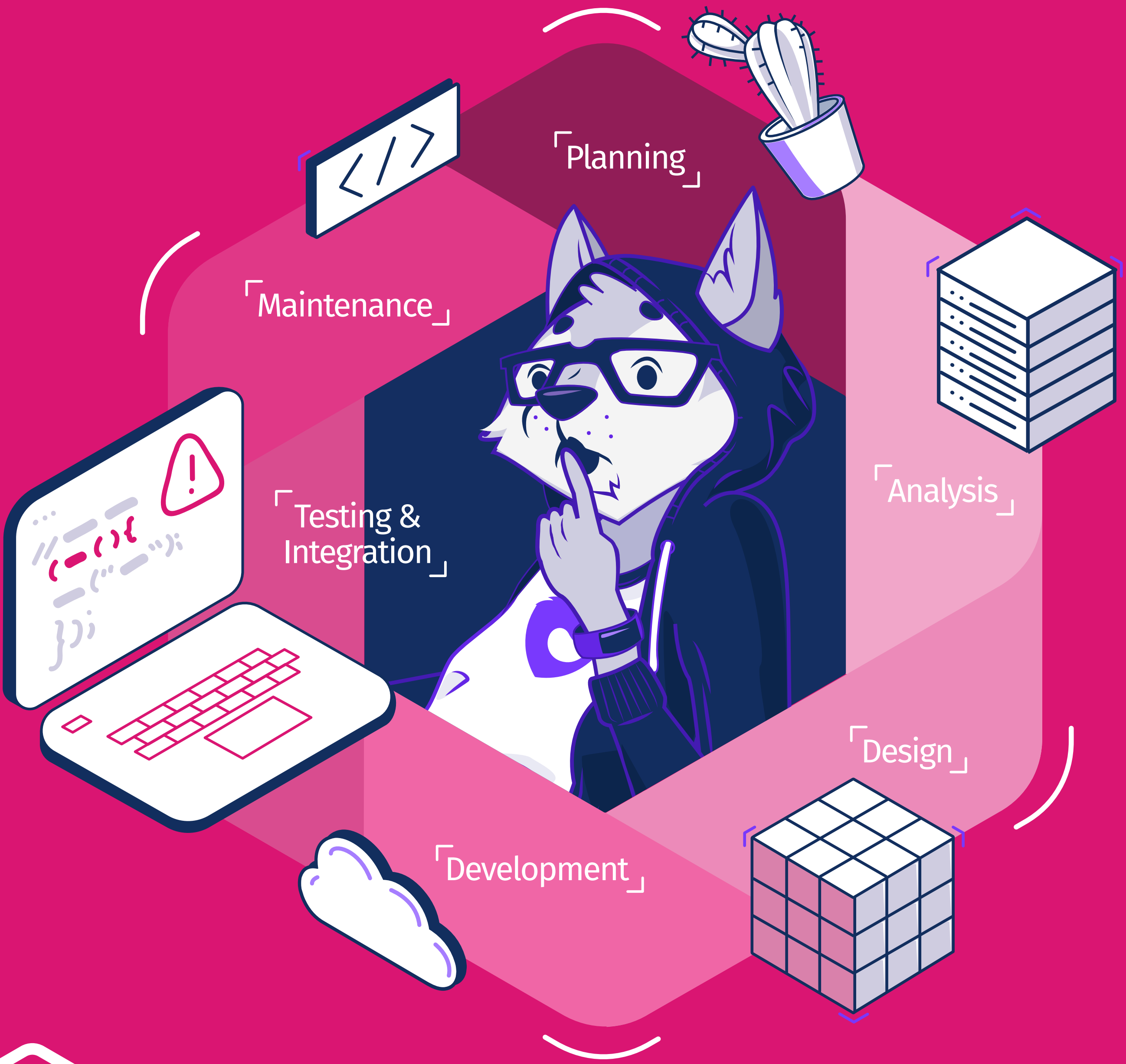
CHECK POINT™

Protecting Secrets Throughout the SDLC with CloudGuard Spectral



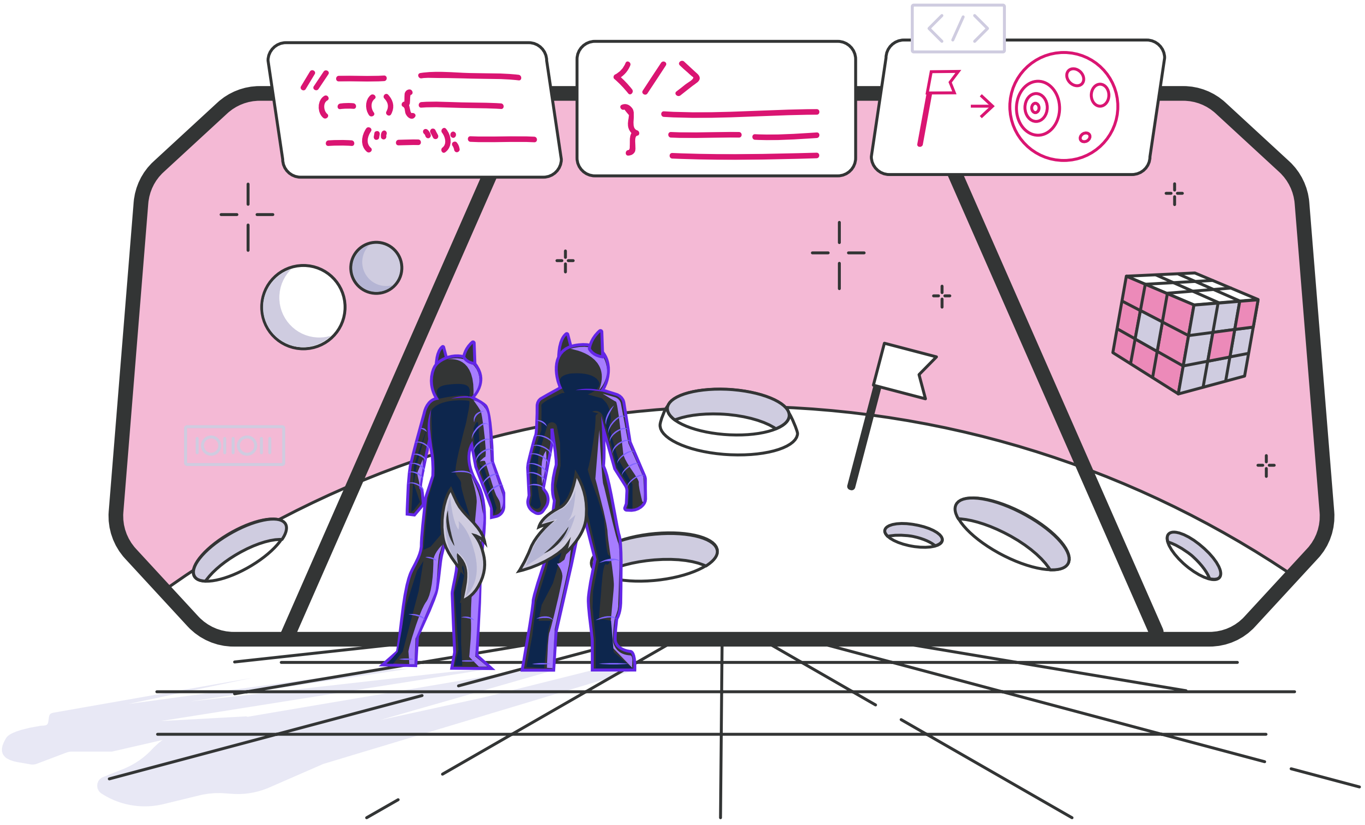
Contents

Introduction

For developers, secret and credential leakage is a problem as old as public-facing repositories. Unfortunately, in 2021 it is officially a significant risk. One that is easy to ignore until it is too late.

In a rush to deliver, developers will often hard-code credentials in code or neglect to review code for exposed secrets. The results can be embarrassing, at best - but devastatingly costly in other cases.

This whitepaper will review the dangers of secret leakage, the challenges in protecting secrets in the SDLC, and strategies for secret leakage mitigation.



Understanding secret leakage

A MailChimp API key can be used to retrieve private user information, while a YouTube API key in the wrong hands can be abused to circumvent metered access fees.

The 2005 release of Git marked the first mass-scale source-control management (SCM) software freely available to every developer. Soon after, source code repository websites based on Git's technology began offering free public code repository services to every developer and more feature-rich private repositories.

Due to their ease of use and collaborative features, repository sites such as GitHub, SourceForge, Bitbucket, and GitLab quickly rose to prominence. As of late 2020, GitHub had become a significant entity in the developer community. It supports 56 million registered developers hosting a total of 190 million code repositories, of which 28 million are public. GitHub's success solidified the place of repositories as an essential part of modern software development.

Over time, options to manage source code in public and private repositories have expanded, and access to robust third-party services has proliferated. As a result, secret access keys used to communicate and interact with third-party services and internal company services became a security requirement.

While code repositories may be the most significant vector for secret leakage, they are not the only factor. Secrets are often spread through other outlets. These include company messaging services, log entries, memory dumps, filesystem artifacts, temporary files, shell history, ticketing systems, dropbox accounts, or even on a Wiki page.

With so many secret credentials required to execute even simple projects, the opportunities for secrets to leak has skyrocketed. According to a 2019 study by North Carolina State University researchers, the most common reasons for secret leakage are human error and misconfiguration. These human errors do not appear correlated to developer experience or repository activity; but rather to inattention to details, information overload, and lacking awareness of security practices extending across the entire SDLC.

The anatomy of a secret leak:
How secrets become compromised

The type of secrets that may be left publicly exposed vary in nature. The secrets can be API keys used to access 3rd party services that host sensitive information or incur a rated access fee. Other common exposures are authentication credentials, which range from private keys used to encrypt communication (such as RSS, EC, and PGP) to passwords, tokens, and keys used to authenticate a login on Virtual Machines, Databases, Server Control panels, and more.

Due to the vast number of interconnected services using secrets to authenticate communication and access data, there are a wide variety of possible abuses that may result from compromised secrets.

1 Scenario 1:

Leaked access key leads to compromises intellectual property and blackmail

```
[
function initiliaizeAWSCredentials {

  const AWS_key = “AKIAJSIE00XXMHXXXXXX”;
  const AWS_Secret = “xxXXx00000xxXXxxXX/x0xx0xxxXxxXx+xXxXX00”;

  connectToAWS(const AWS_key, const AWS_Secret);

}
]
```

A developer working on a project utilizing Amazon Web Services accidentally commits a source file containing an AWS access key ID to their public repository instead of the company's private repository.

Quickly realizing a mistake was made, the developer erases the file from the public repository. They think the issue has been resolved. However, the developer doesn't realize that the AWS key remains in historical records kept by the public repository due to lacking security training practices. It may even be accessible after sanitizing the commit history.

A hacker scanning project activity discovers the AWS access key and uses it to retrieve sensitive company intellectual property. The hacker blackmails the company and threatens public release of the intellectual property unless their demands are met.

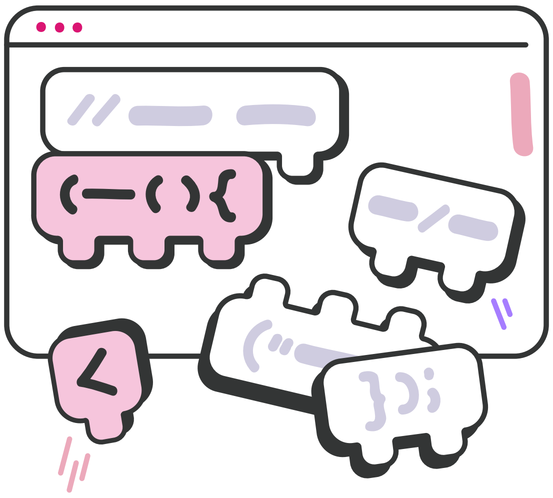
2 Scenario 2:
Hard-coded database password leads to malware injection

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <LogzioAppender name="Logzio">
      <logzioToken>DJabj8j83ha07a6a6ca8hehbaryoxBpkypBANN</logzioToken>
      <logzioType>logz</logzioType>
      <logzioUrl>https://listener.logz.io:8071</logzioUrl>
    </LogzioAppender>
  </Appenders>
  <Loggers>
    <Root level="all">
      <AppenderRef ref="Logzio"/>
    </Root>
  </Loggers>
</Configuration>
```

In this scenario, a web developer is working on a website utilizing a database to store web page content. The developer hard-codes the database login credentials into the source code. This makes it easier to debug database access while constructing the website.

Years later, the developer decides to open-source the website's backend. They post the entire source code to an open repository - but without remembering that login credentials were previously hard-coded within the source code for extra convenience.

A malicious entity scanning through the project's activity log identifies the database login credentials. With full access to the website's database, the malicious entity modifies the database to inject malware code into the website's pages. The malicious code infects anyone visiting the website with malware.



The results of secret leakage: How bad can it be?

The following real-world examples demonstrate the depth of problems caused by secret leakage.

Stack Overflow’s full complete infiltration

On April 30, 2019, a hacker began probing Stack Overflow's network. It wasn't until May 12 of that month that Stack Overflow became aware of the hack and started taking action. By then, it was too late - and their entire codebase was pilfered.

> **The Stack Overflow 2019 hack was guided by advice from none other than...Stack Overflow.**
>
> **- The register**

The Stack Overflow incident is an interesting case of multi-stage privilege escalation. During the incident, the hacker used Stack Overflow's website and product Q&A section as resources. This information allowed the hacker to learn about the Stack Overflow products and how they could be abused. The hacker was initially able to bypass access controls limiting logins to those users with an access key.

Then, step by step, the hacker acquired secrets spread throughout the system, which provided access to additional resources. The resources included credentials to a TeamCity service account and later on an SSH certificate that handed full access to internal repositories housing Stack Overflow's codebase. Eventually, the hack culminated in the theft of Stack Overflow's codebase. The hacker also gained moderator and developer-level access across the entire Stack Exchange Network, which eventually exposed the hacker's activities as they became public.

Stack Overflow suffered undisclosed monetary damages, and several of its services had to be blocked for days to assess the level of intrusion and mitigate the hacker's actions. Stack Overflow initiated an additional security audit by a 3rd party to complement the internal incident analysis.

The results of the hack led to significant changes:

- Security keys were rotated.
- Passwords were reset.
- Account creation policies were changed.
- The loss of publicly identifying information was disclosed.

Because Stack Overflow professionally handled the incident, they only slightly tarnished their brand reputation. However, with their entire codebase exposed, the possibility of new vulnerabilities emerging still clouds Stack Overflow's future to this day.

Amazon’s leak intercepted in minutes

On the morning of January 13, 2020, an AWS DevOps Cloud Engineer committed nearly a gigabyte's worth of data to a public GitHub repository. They were caught in roughly 30 minutes by an automated detection engine.

> **Amazon Engineer Leaked Private Encryption Keys.**
> **Outside Analysts Discovered Them in Minutes**
>
> **- Gizmodo**

The Amazon incident was a simple case of a careless developer accidentally committing internal company information on a public repository. Luckily, a security analyst had an automatic detection engine monitoring the repository and quickly notified Amazon of the issue.

Amazon took action, and within 4 hours, they removed the repository. Amongst the published data, Amazon AWS and RSA keys marked "admin", "cloud", and "rootkey" were discovered. But due to legal restrictions, they did not test the keys to verify if they were still functional.

It's possible the Amazon data was intercepted by malicious agents monitoring the repository who used the same security analyst methods to detect the accidental code commit. AWS was forced to immediately reset, invalidate, or rotate every secret exposed by this public release to prevent future intrusion.

Private medical records exposed everywhere on GitHub

In July 2020, security researcher Jelle Ursem tried notifying medical institutions of security leaks he found on publicly accessible GitHub repositories. However, he found the process challenging because there was no one to talk to about the leak.

> **I have found admin credentials to some super sensitive medical billing processing system and get nothing but silence on all available contact channels and no action for months.**
>
> **- DataBreaches.net**

During regular security sweeps of public code repositories, security researcher Jelle Ursem found nine medical institutions (Xybion, MedPro Billing, Texas Physician House Calls, VirMedica, MaineCare, Waystar, Shields Health Care Group, AccQData) with secret leaks spread across multiple GitHub public repositories. The leaked secrets included URLs to administration control panels with no authentication required; and login credentials to Databases, SFTP servers, Microsoft Office 365 Suites, and Google's G Suites.

Overall, between 150,000 to 200,000 private medical records were exposed by the leaks. However, trying to contact the medical institutions proved difficult. Some institutions did not have anyone in charge of source code security, so there was simply no one to talk to about the issue. Other institutions had staff that was not properly trained to handle security alerts and misinterpreted the alert as a phishing attempt. They instead chose to ignore the issue altogether, extending the public exposure for months.

It is hard to estimate such negligence's monetary cost, as it may lead to future lawsuits and regulatory actions that have yet to be exhausted. Reputationally, public exposure of private medical information paints an ugly picture of lax security practices and insufficient training within these institutions.

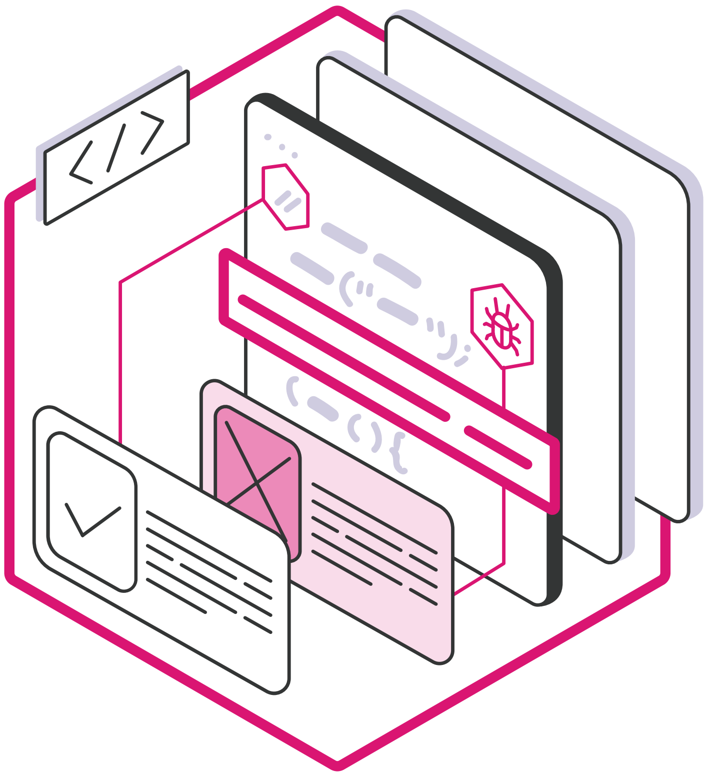
Swiss developer harvests secrets

On July 26th, 2020, Swiss developer Tillie Kottman posted a massive data dump from over 50 high profile companies (including Disney, Microsoft, Lenovo, and others) to a public GitHub repository.

> **This list will be viewed by cyber criminals far and wide looking for vulnerabilities as well as confidential information in a scarily short space of time.**
>
> **- TomsGuide**

Tille Kottman, a Swiss software developer & security activist, has publicly aggregated source code from over 50 high-profile companies leaking secrets through publicly accessible or misconfigured repositories. The aggregated code now resides in a single and easy-to-navigate GitHub repository. Tille attempted to strip secrets from the code on a "best-effort" basis and respected the companies' takedown notices. Unfortunately, a lot of the code remains publicly accessible, leaving the door open to future exploits.

The reputational and monetary loss in this incident is incalculable. It does, however, demonstrate that secret-keeping and secret security practices are still lacking in high-profile companies.



The challenge of protecting secrets in software development

Software developer independence and reliance upon open source repositories have introduced new risks and cultural changes. These risks and changes must be accounted for to protect secrets throughout the SDLC.

The human factor

Developing complex software today requires keeping secrets. Whether it's login credentials to a database or an API key to a mapping service, the information must be stored somewhere secure. When trying to deliver on a deadline, working under stress, or simply thinking of their convenience, developers will often take shortcuts or make silly - but expensive - mistakes.

While there are many examples of human error, these are the most common:

Hardcoded credentials

Let's face it, the most convenient place for a developer to store authentication credentials is right in the code, possibly even within the function that uses the credentials. While very convenient, this practice is inherently insecure, especially in a corporate environment where multiple developers might share the code. The Internet is littered with horror stories demonstrating what happens when convenience precedes security.

Insecure credential storage

Code is not the only place where secrets leak. Developers may not understand how to use a repository's ".gitignore" feature correctly. They may falsely assume credentials stored in a separate file are not being committed to the repository. Other developers may find it convenient to send colleges authentication credentials as plain text over internal or external messaging applications without being aware their conversations may be tracked. With so many secret storage options, it becomes ever more challenging to detect and remediate such cases.

Accidental commit to a public repository

Many developers use multiple code repositories, usually a mix of personal and work-related projects. This mix makes it easy for a tired or distracted developer to accidentally commit private, work-related data to their personal and very public repository. To make the issue worse, the developer may assume they can erase the problematic files from their public repository. Unfortunately, they do so without being aware that repositories keep historical records of their actions - records easily accessible by bad actors.

Linking to control infrastructure

Let's face it, the most convenient place for a developer to store authentication credentials is right in the code, possibly even within the function that uses the credentials. While very convenient, this practice is inherently insecure, especially in a corporate environment where multiple developers might share the code. The Internet is littered with horror stories demonstrating what happens when convenience precedes security.

Keep in mind that humans are flawed and will always find new and ingenious ways to mess things up. As they say, "To err is human." Providing security training accompanied by real-time monitoring is a vital step toward mitigating the human factor.

Homegrown solutions & code reviews are not enough

Building a homegrown solution or using free, open-source tools to prevent source leakage is problematic. The investment required to create a detection solution and maintain it over time as technology evolves is not financially viable.

Currently, open-source tools such as truffleHog are very rudimentary in their scope of action. They produce many false-positive results while missing many more secrets when compared to commercial tools. Beyond detection issues, free tools do not provide true integration into the CI/CD pipeline. This makes them unreliable and cumbersome to use in production work.

Code review suffers from a different set of issues. Reviewers are mostly concerned with new code being introduced into the codebase. They may not even track the project's entire history or developer activities that may lead to exposed secrets.

For example, a developer may accidentally commit a secret to a repository and later, realizing their mistake, try to correct the error by deleting the secret. The net-zero effect means the secret will no longer be part of the reviewed code but will instead lurk within the project's historical records.

Remediation demands a quick response

Unfortunately, public repositories do not provide logging features to track if a file containing a secret was accessed and by whom.

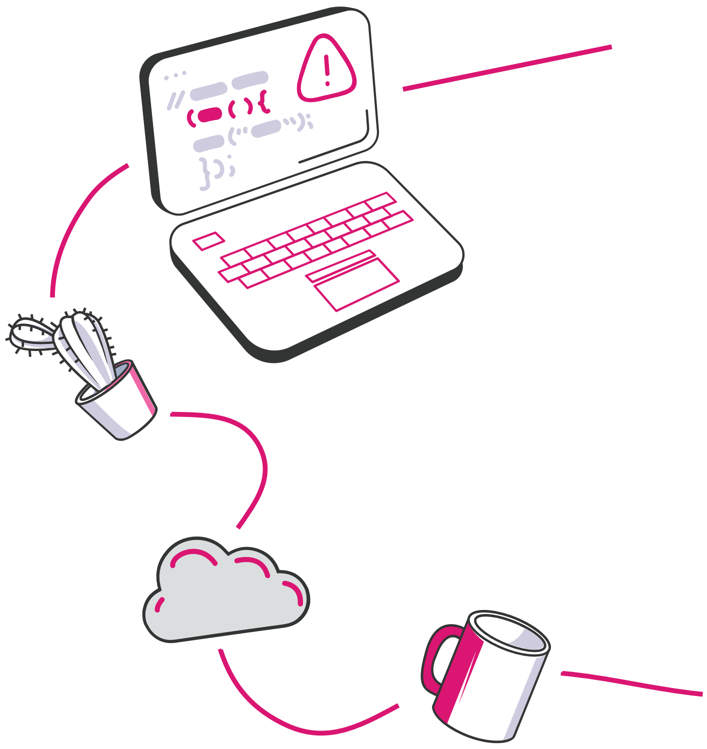
In practice, this means that as soon as a secret is publicly exposed, the assumption must be that it has already been compromised. Both security researchers and malicious entities have developed automated software that scans developer repository actions in real-time. Even if a secret was only exposed for a few minutes, it might be a few minutes too late.

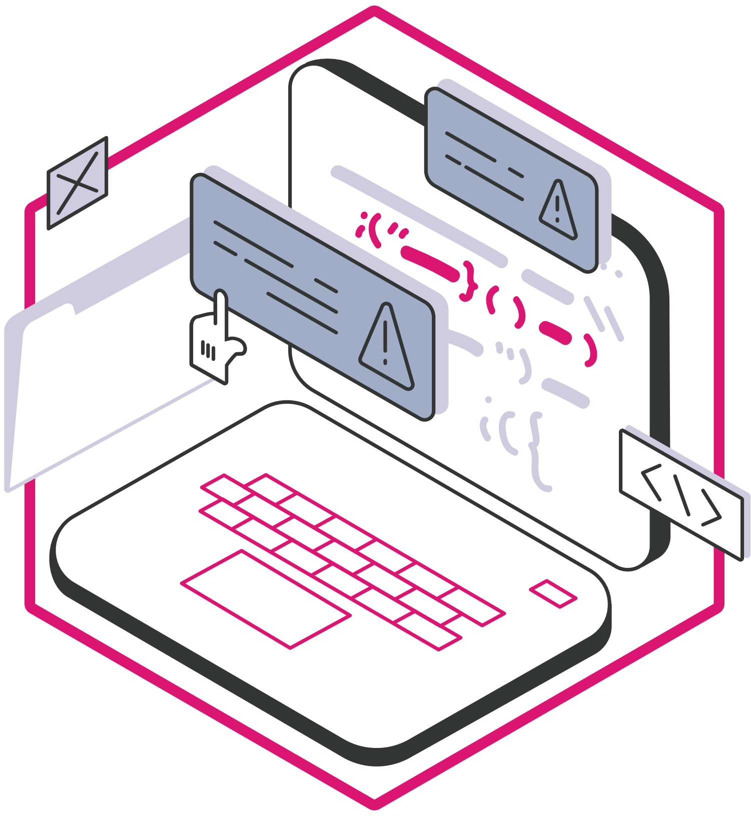
If the secret was not discovered instantly, only committing an update that excludes the secret does not remove the secret from the repository. It may still be present in the project's history pages. It may also be a direct link to a resource that the repository may not have scrubbed but is still indexed by external search engines such as Google.

Without quick remediation at the root of the leak, every exposed secret would require a hard-reset. Depending on the secret's scope-of-use, this may require:

- Resetting API keys across the entire SDLC
- Updating certificates on multiple servers to restore secure communication while incurring downtimes that could easily lead to financial and reputational damage

Such a calamity can often be triggered by a single thoughtless developer, when they could have easily mitigated it at the source with the right tools and security practices.





What to consider in a security solution that protects from secret leakage

Any security solution that guards against secrets leaking must be able to protect employees from their own mistakes by design. Training alone is not enough. Technological solutions must be put in place to safeguard against human error.

Here are some of the top features to consider in a security solution:

Smart Detection

Detect secret credentials based on smart filtering and machine learning analysis.

Real-time commit verification

Intercept secrets before they are committed to an insecure code repository.

Sanitize historical records

Scan a project's historical records to ensure no secrets are still lurking inside.

Clear results

Limit the number of false positives using user-feedback and machine learning algorithms.

Post-incident analysis

Provide all the information management. Auditors may require demonstrating transparency and compliance when reacting to leaked secrets.

Mitigating secret leakage with automated secret detection by CloudGuard Spectral

Secret leakage prevention doesn't have to be a painstaking and repetitive process. With CloudGuard Spectral platform, you can continuously scan and monitor known and unknown assets to stop leaks at the source.

How it works

1

Connect your repository or CI/CD

CloudGuard Spectral integrates with all leading CI systems with built-in support for Jenkins, Azure and others.

2

CloudGuard Spectral continuously scans your repos for secrets for code vulnerabilities and misconfigurations

Leveraging our combination of hundreds of custom detectors and proprietary machine learning models to detect issues in near real-time.

3

Receive alerts for your repos for secrets code vulnerabilities

Customize alerts, build your own detectors and policies to meet target KPIs, and put an end to endless infosec meetings.

What our users think

Perion

CloudGuard Spectral has automatically identified and surfaced security flaws that our company was not aware of, it helped us be more secure and avoid operational risks.

Maayan Yosef
Cloud & DevOps Architect

SimilarWeb

CloudGuard Spectral reduces cost by shifting left our security efforts while observing more than 300+ repos, which enables us a safe open-source transformation.

Elad Kaplan
Serving Infrastructure Team Leader

KRYON

Securing code is a key piece of our security puzzle. Check Point Spectral understands the challenge that companies like ours face and has built a unique platform to help us protect our data assets. We're excited to work with them and this important solution.

Udi-Yehuda Tamar
Cloud & DevOps Architect

About Check Point CloudGuard

CloudGuard Spectral can help secure your digital assets by equipping your teams with an automated security-first tool and platform. We know that DevOps live in the command line — so we've designed our tools to be compatible with your current workflows.

All you have to do is add CloudGuard Spectral to the CI as a build step. We then help scan your codebase using our array of detectors for risks such as sensitive data and credentials. If there is a problem with security, CloudGuard Spectral can automate a failed build and alert you to the issues. Our suite of services includes full reporting and tracing source files and positions for risk assessment and mitigation.

When your codebase is secured by design, it reduces the potential modes of malicious entrances. Our tools and platform is built in Rust — a language that excels at performance, safety, and safe concurrency. Secure your codebase today and ensure that your code is clear from potential security breaches and leaks. For more information, spectralops.io

CHECK POINT™

Worldwide Headquarters
5 Ha'Solelim Street, Tel Aviv 67897, Israel | Tel: 972-3-753-4555 | Fax: 972-3-624-1100 | Email: info@checkpoint.com

U.S. Headquarters
959 Skyway Road, Suite 300, San Carlos, CA 94070 | Tel: 800-429-4391; 650-628-2000 | Fax: 650-654-4233

www.checkpoint.com