AZUR TECH *Winter*

2 DÉCEMBRE 2025 | 10H00

ATELIER : Moins de faux positifs, plus de sécurité : l'IA au service de la chasse aux secrets

Mickael BRIDARD
Jean-Philippe CARLENS
Nicolas DE TOFFOLI

AMADEUS

telecom-valley.fr/azurtechwinter

Telecom Valley | Animateur Azuréen Numérique amadeus

SIFT

SMART INTELLIGENT FINDING TRIAGING

Nicolas DE TOFFOLI
Jean-Philippe CARLENS
Mickael BRIDARD

02DEC25

Telecom Valley | Animateur Azuréen Numérique

AZUR TECH Winter

2 DÉCEMBRE 2025 | SOPHIA ANTIPOLIS

LES TECHNOLOGIES DU NUMÉRIQUE

We're a Team!

Nicolas DE TOFFOLI

Senior Manager,
Secure Development
Lifecycle

Triathlete

Jean-Philippe CARLENS

Software Developer,
Security Scanner
Team

Triathlete & gamer

Mickael BRIDARD

Technical Information
Security Officer,
Engineering Toolchain

Gamer

Amadeus. It’s how travel works.

Serving customers in

190+ countries

6th consecutive year included in the Financial Times list of

Diversity leaders

Global team of

20,000+ professionals

One of the largest R&D investors in the software industry in Europe.
Gross investment in 2024

€1,365 million

Bookings in 2024

470+ million

Passengers boarded in 2024

2.2+ billion

Payments processed

$120+ billion

Revenue 2024

€6,141.7 million

Source: Global Report 2024

AGENDA

Swimming

AppSec is critical

Biking

Scanning our code for secrets

Running

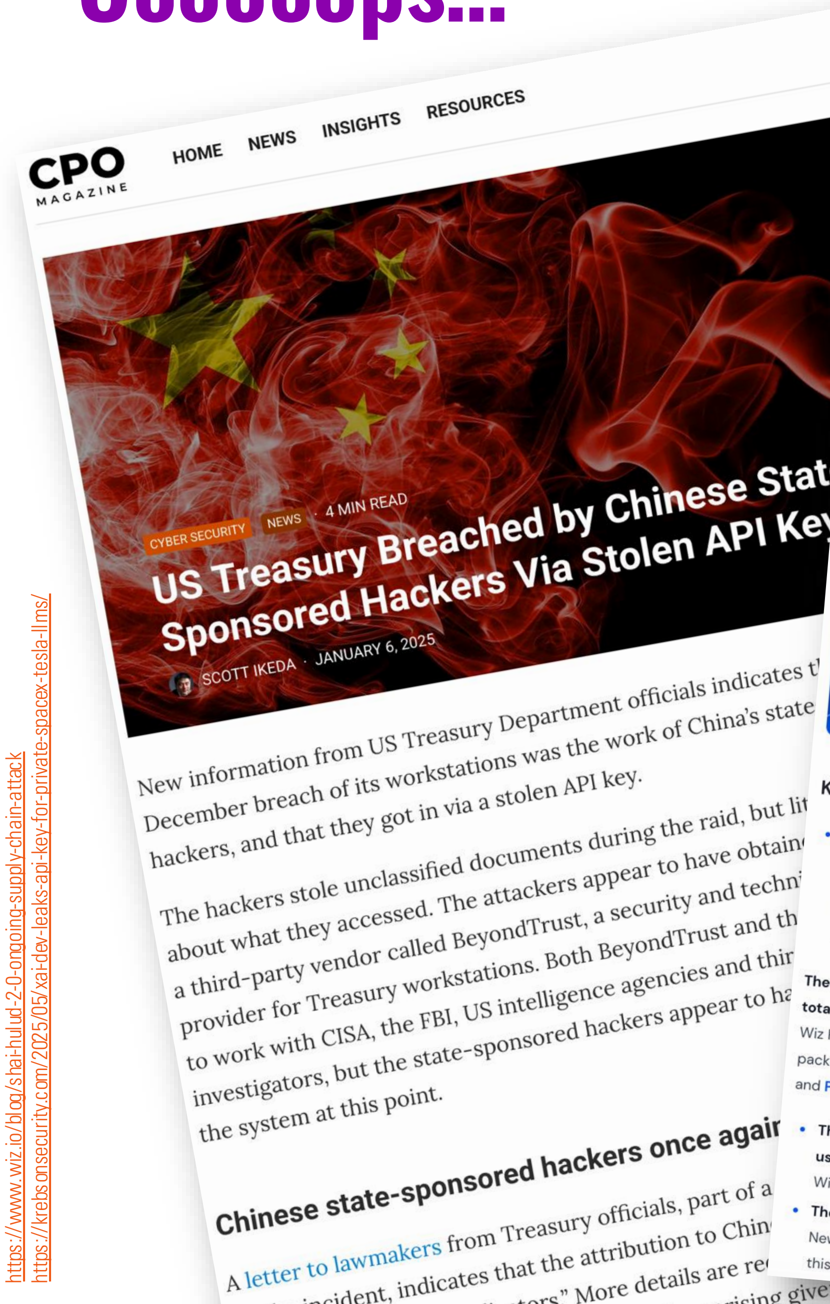
SIFT for smart triaging of findings

Celebration!

SWIMMING

APPSEC IS CRITICAL

Ooooooops...



CPO MAGAZINE

HOME NEWS INSIGHTS RESOURCES

CYBER SECURITY · NEWS · 4 MIN READ

US Treasury Breached by Chinese State Sponsored Hackers Via Stolen API Key

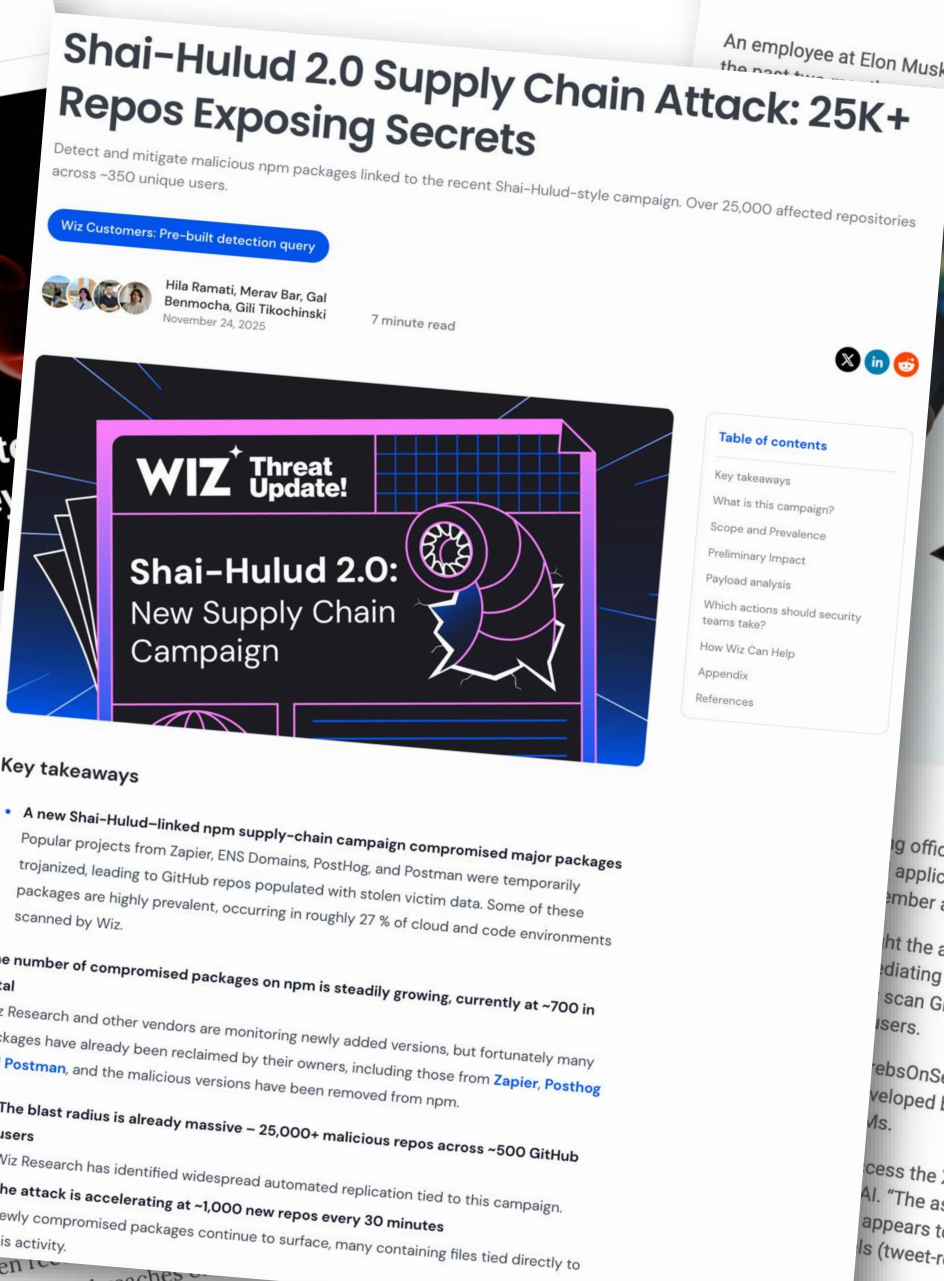
SCOTT IKEDA · JANUARY 6, 2025

New information from US Treasury Department officials indicates t December breach of its workstations was the work of China's state hackers, and that they got in via a stolen API key.

The hackers stole unclassified documents during the raid, but lit about what they accessed. The attackers appear to have obtain a third-party vendor called BeyondTrust, a security and techn provider for Treasury workstations. Both BeyondTrust and th to work with CISA, the FBI, US intelligence agencies and thir investigators, but the state-sponsored hackers appear to ha the system at this point.

Chinese state-sponsored hackers once agai

A letter to lawmakers from Treasury officials, part of a incident, indicates that the attribution to Chin tors." More details are re rising given rec



Shai-Hulud 2.0 Supply Chain Attack: 25K+ Repos Exposing Secrets

Detect and mitigate malicious npm packages linked to the recent Shai-Hulud-style campaign. Over 25,000 affected repositories across ~350 unique users.

Wiz Customers: Pre-built detection query

Hila Ramati, Merav Bar, Gal Benmocha, Gili Tikochinski
November 24, 2025
7 minute read

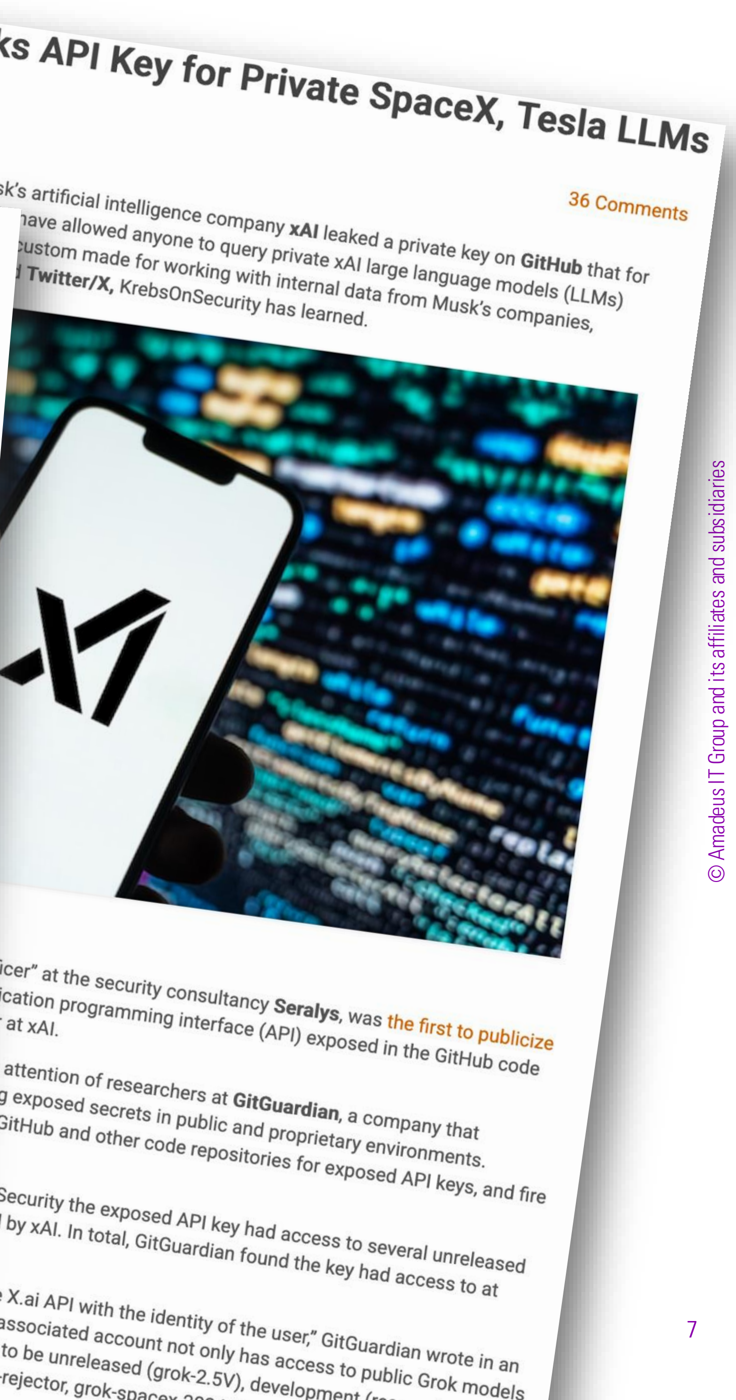
Table of contents
- Key takeaways
- What is this campaign?
- Scope and Prevalence
- Preliminary Impact
- Payload analysis
- Which actions should security teams take?
- How Wiz Can Help
- Appendix
- References

Key takeaways

- **A new Shai-Hulud-linked npm supply-chain campaign compromised major packages**
  Popular projects from Zapier, ENS Domains, PostHog, and Postman were temporarily trojanized, leading to GitHub repos populated with stolen victim data. Some of these packages are highly prevalent, occurring in roughly 27 % of cloud and code environments scanned by Wiz.

The number of compromised packages on npm is steadily growing, currently at ~700 in total
Wiz Research and other vendors are monitoring newly added versions, but fortunately many packages have already been reclaimed by their owners, including those from Zapier, Posthog and Postman, and the malicious versions have been removed from npm.

- **The blast radius is already massive – 25,000+ malicious repos across ~500 GitHub users**
  Wiz Research has identified widespread automated replication tied to this campaign.
- **The attack is accelerating at ~1,000 new repos every 30 minutes**
  Newly compromised packages continue to surface, many containing files tied directly to this activity.



xAI Dev Leaks API Key for Private SpaceX, Tesla LLMs

May 1, 2025

36 Comments

An employee at Elon Musk's artificial intelligence company **xAI** leaked a private key on **GitHub** that for the past two m have allowed anyone to query private xAI large language models (LLMs) custom made for working with internal data from Musk's companies, d **Twitter/X**, KrebsOnSecurity has learned.

g officer" at the security consultancy **Seralys**, was the first to publicize application programming interface (API) exposed in the GitHub code ember at xAI.

ht the attention of researchers at **GitGuardian**, a company that ediating exposed secrets in public and proprietary environments. scan GitHub and other code repositories for exposed API keys, and fire users.

rebsOnSecurity the exposed API key had access to several unreleased veloped by xAI. In total, GitGuardian found the key had access to at Ms.

cess the X.ai API with the identity of the user," GitGuardian wrote in an AI. "The associated account not only has access to public Grok models appears to be unreleased (grok-2.5V), development ls (tweet-rejector, grok-space

Sources
https://www.cpomagazine.com/cyber-security/us-treasury-breached-by-chinese-state-sponsored-hackers-via-stolen-api-key/
https://www.wiz.io/blog/shai-hulud-2-0-ongoing-supply-chain-attack
https://krebsonsecurity.com/2025/05/xai-dev-leaks-api-key-for-private-spacex-tesla-llms/

© Amadeus IT Group and its affiliates and subsidiaries

Open door for hackers

- What is a secret?
  - User password
  - API token
  - SSH key

- In 2024, 23M+ new hardcoded secrets added to public GitHub:
  - +25% vs. 2023
  - Source: GitGuardian

- Cost can be huge:
  - Business
  - Reputation
  - Fines
  - …up to billions of $

Shifting security left

A s,

we out

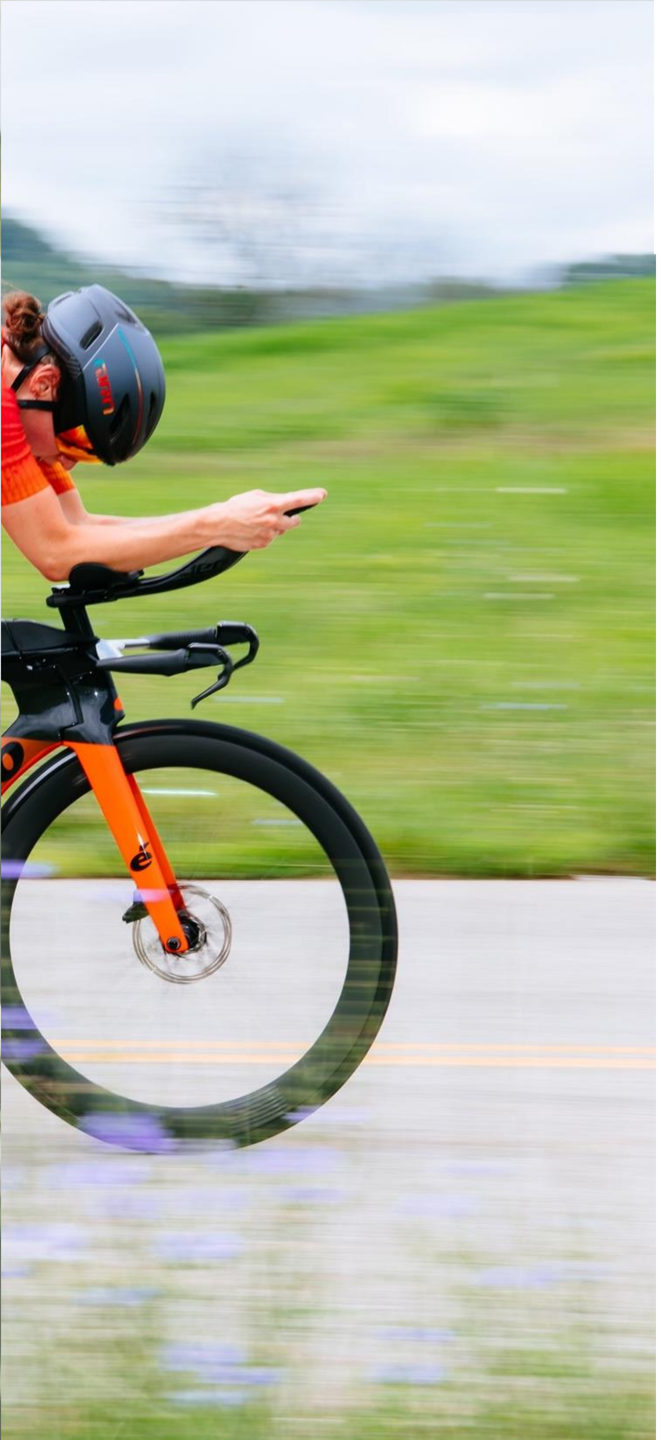
SECRETS!

Secret scan is part of Amadeus Secure Development Lifecycle

Any company is concerned...

BIKING

SCANNING OUR CODE FOR SECRETS



The bike leg

The longest and most strategic part...

Where keeping a steady pace is everything!

Which means scanning everything, regularly, everywhere!

We need a ~~bike~~ scanner!

STATIC
APPLICATION SECURITY TESTING

DYNAMIC
APPLICATION SECURITY TESTING

SOFTWARE COMPOSITION
ANALYSIS

Multiple categories… like bikes

Software Composition Analysis

Detect third-party dependencies

Report their known vulnerabilities

Maybe the MTB bike?

docker

Maven

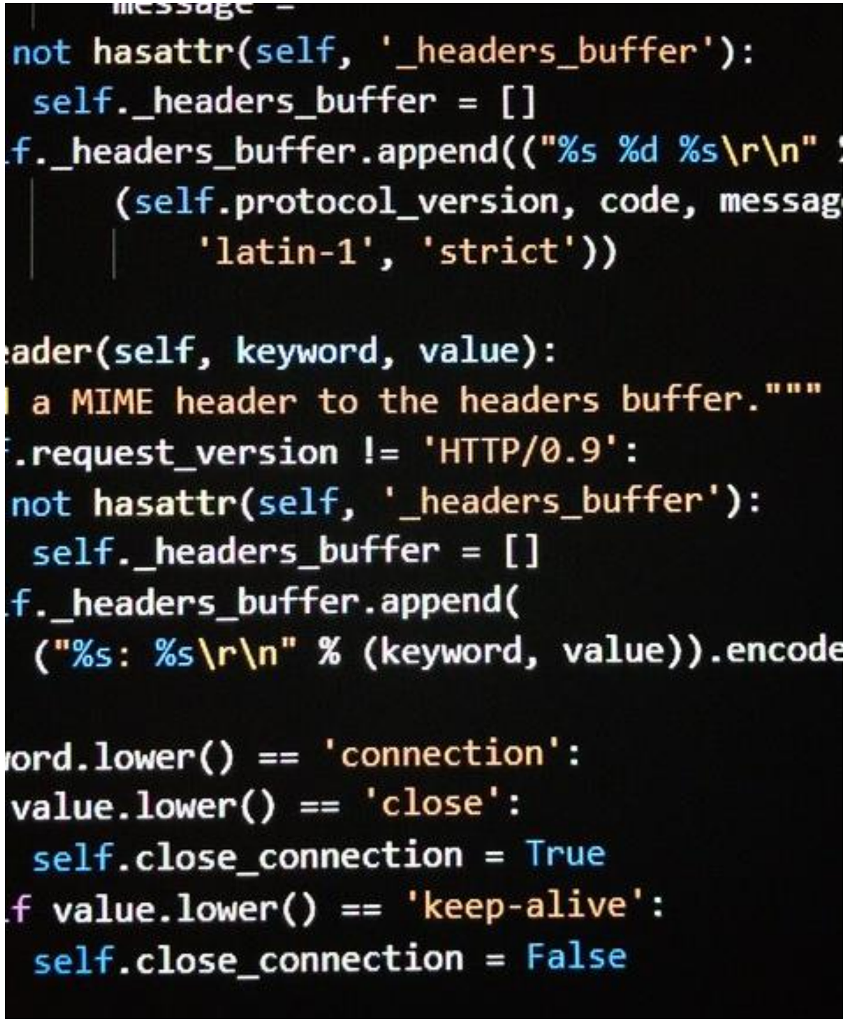
pypi

nuget

Dynamic Application Security Testing

- Tests while it's running
- Runtime and misconfigurations issues
- No source-code access

Static Application Security Testing



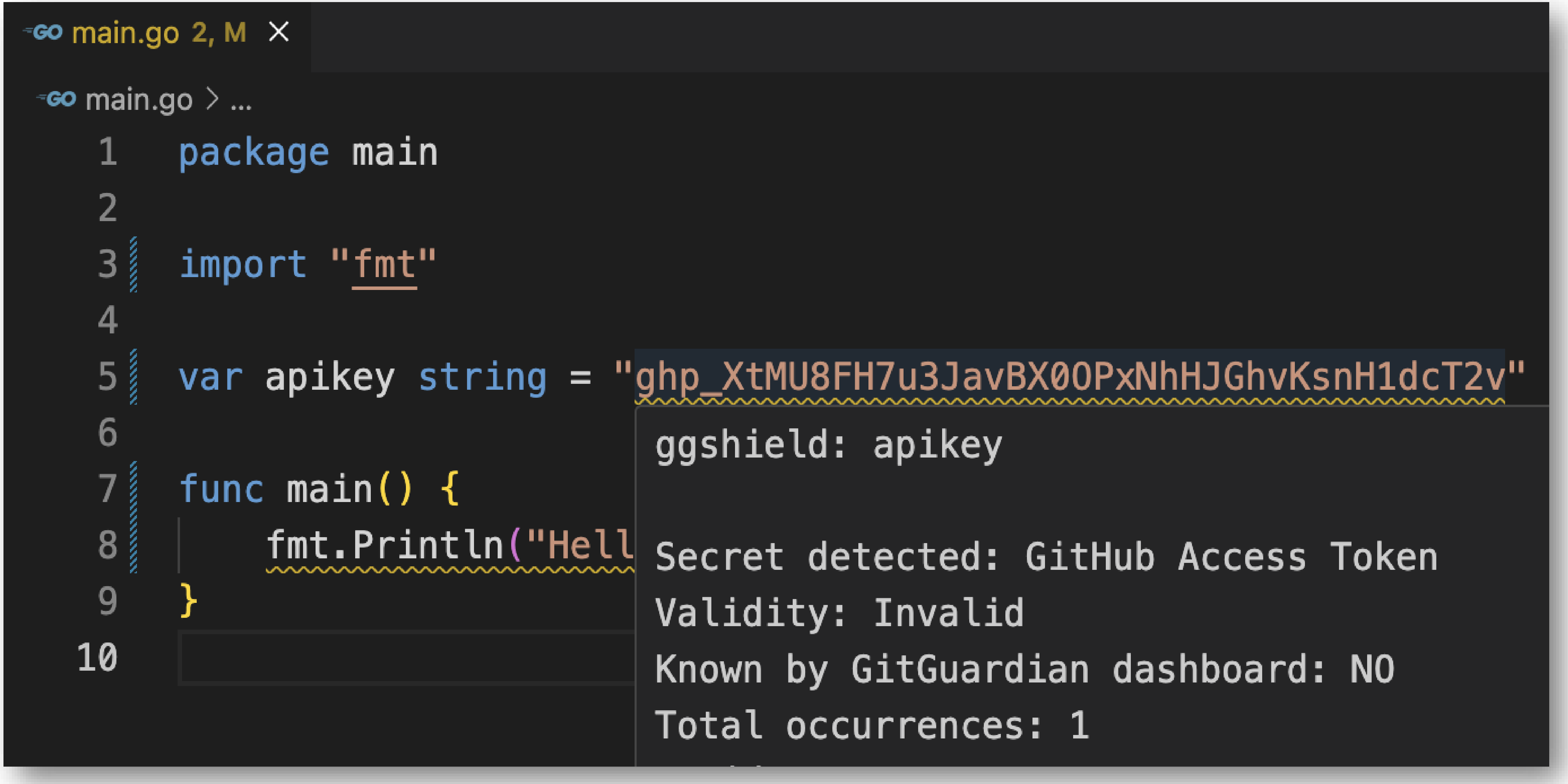
```
    message =
not hasattr(self, '_headers_buffer'):
 self._headers_buffer = []
f._headers_buffer.append(("%s %d %s\r\n"
    (self.protocol_version, code, messag
        'latin-1', 'strict'))

ader(self, keyword, value):
 a MIME header to the headers buffer."""
.request_version != 'HTTP/0.9':
not hasattr(self, '_headers_buffer'):
 self._headers_buffer = []
f._headers_buffer.append(
 ("%s: %s\r\n" % (keyword, value)).encode

ord.lower() == 'connection':
value.lower() == 'close':
 self.close_connection = True
f value.lower() == 'keep-alive':
 self.close_connection = False
```

Analysis of source-code

Detect insecure code

Shift-left practice

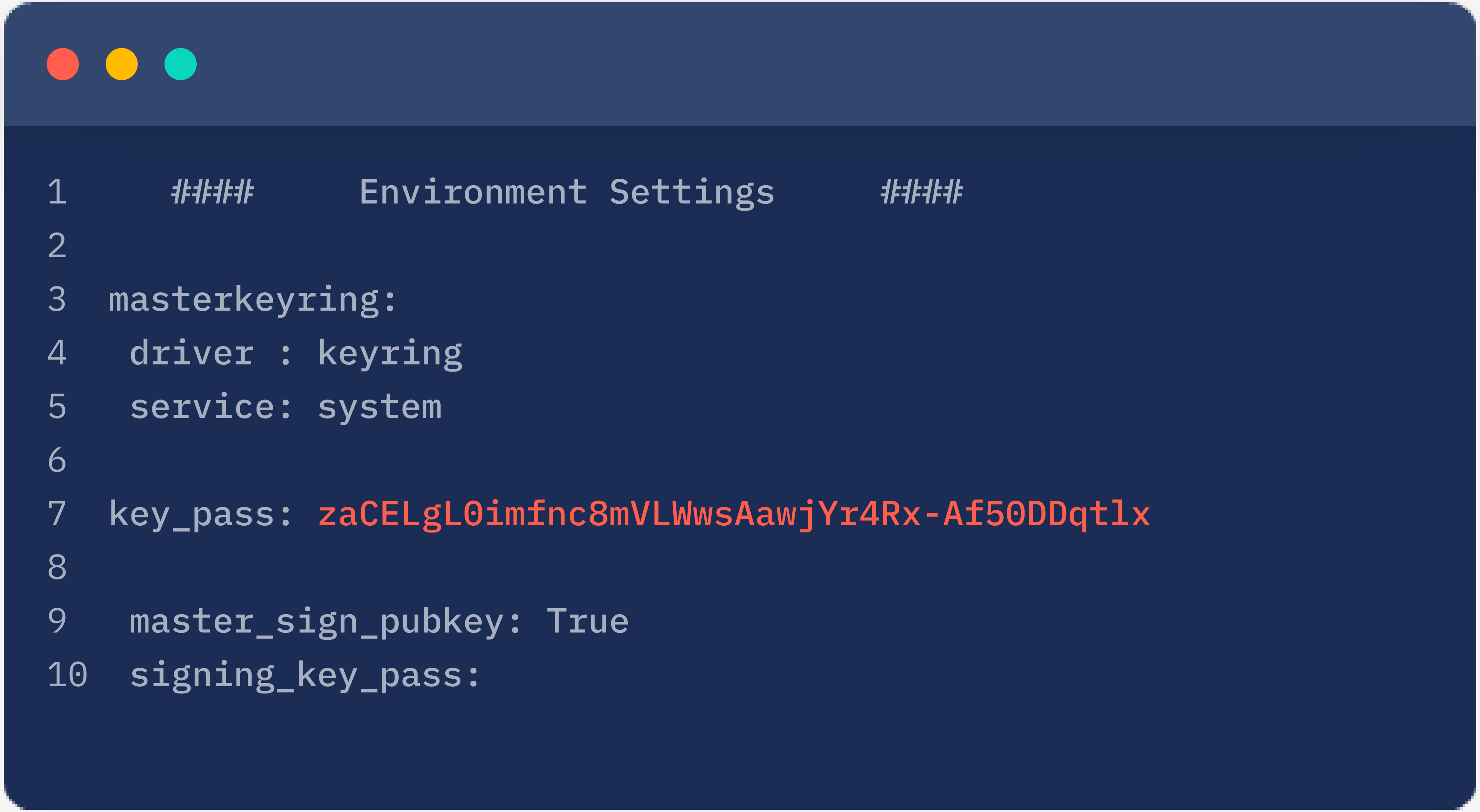
String values



```go
package main

import "fmt"

var apikey string = "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"

func main() {
	fmt.Println("Hell
}

```

```
ggshield: apikey

Secret detected: GitHub Access Token
Validity: Invalid
Known by GitGuardian dashboard: NO
Total occurrences: 1
```

Configuration files



```
   #####     Environment Settings     #####

masterkeyring:
 driver : keyring
 service: system

key_pass: zaCELgL0imfnc8mVLWwsAawjYr4Rx-Af50DDqtlx

 master_sign_pubkey: True
 signing_key_pass:
```

And more!

Cryptographic key files

Log files

Documentations

Which model for our bike?

Proprietary solutions:

- Managed service maintained by an external company
- Integrated
- Advanced detection models (built-in ML/AI, vendor-maintained rules)
- Faster onboarding at large scale

Directly a premium one?

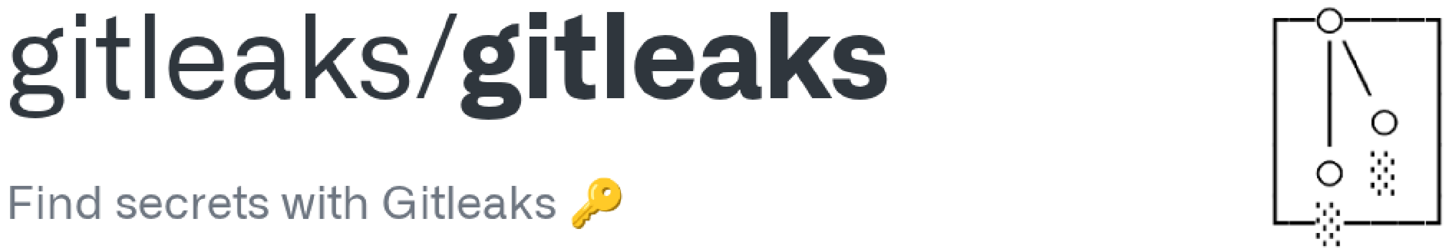
Which model for our bike?

Free, open-source solutions:

- Flexible, can run everywhere: laptop, CI/CD, automations, Gitlab, GitHub…
- Cost-effective: no license, simple to scale org-wide
- Vendor-neutral: no lock-in, no external data exfiltration concerns

We can win the race with a cheap one?

Our choice:



gitleaks/**gitleaks**

Find secrets with Gitleaks 🔑

- Widely used in the industry
- Portable
- Configurable
- Simple

How does it work?

A set of named regular expressions, rules, describing how a secret looks like

AWS access key

AKIA[0-9A-Z]{16}

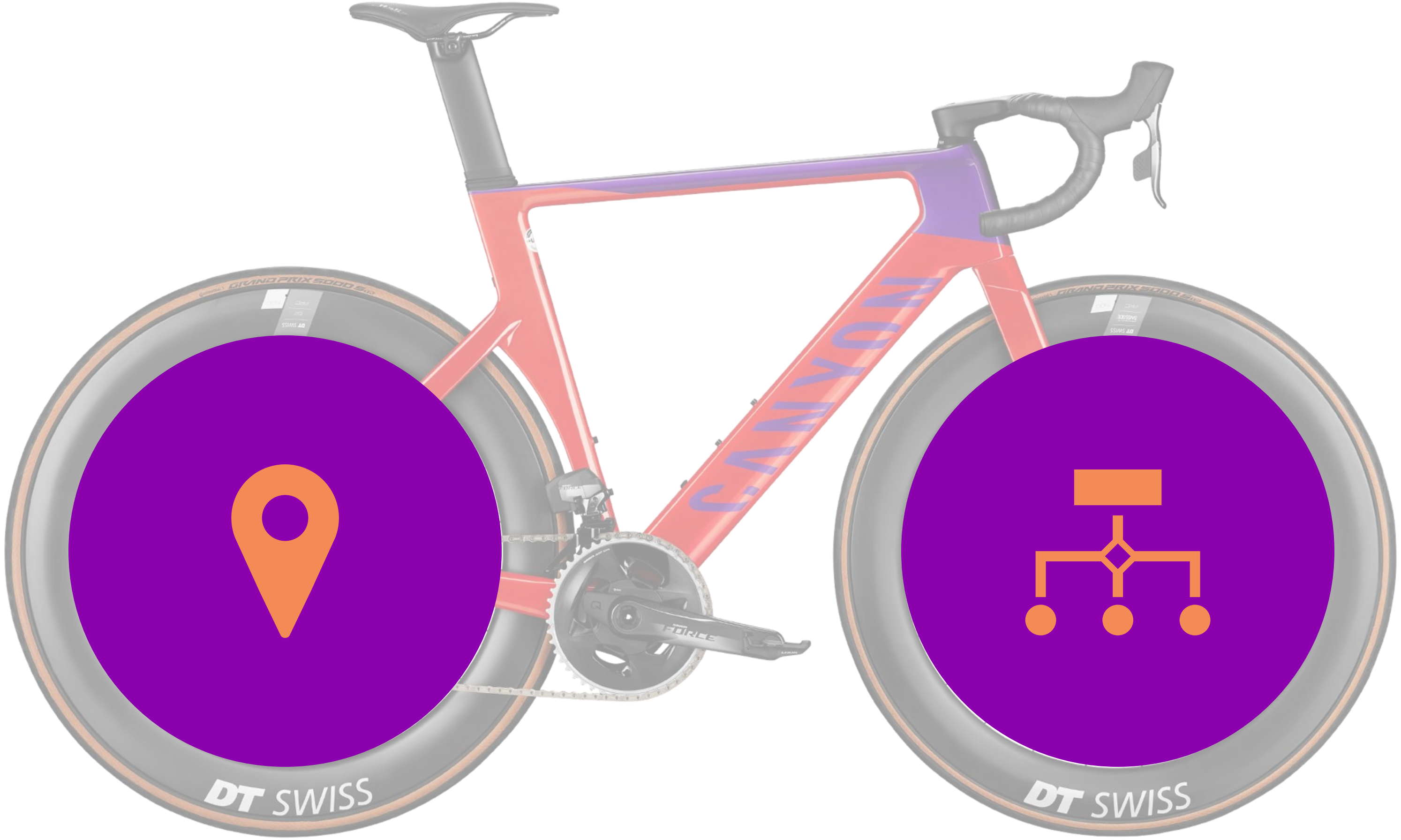
GitHub token

ghp_[0-9A-Za-z]{36}

Private key

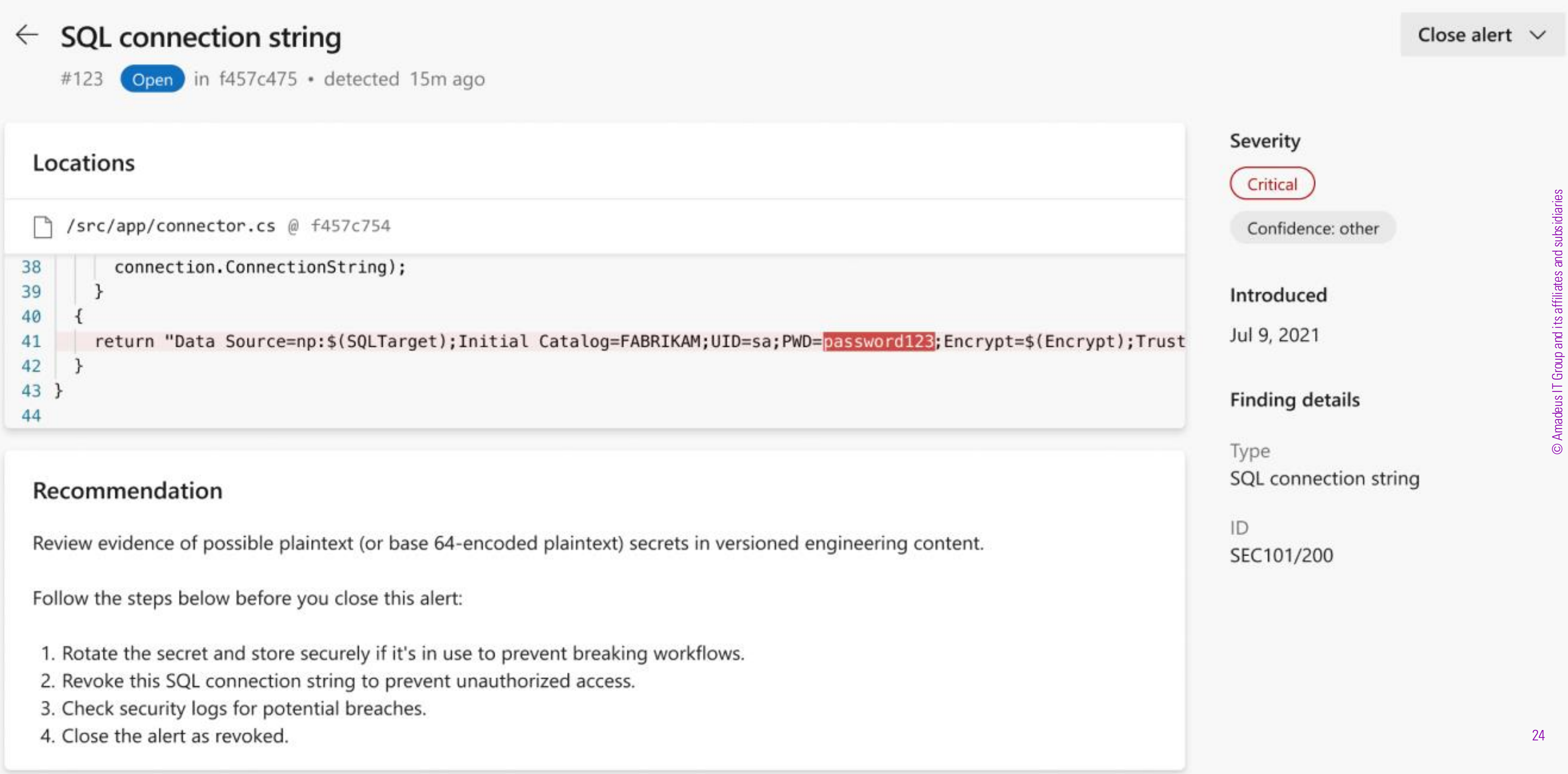
-----BEGIN (RSA|EC) PRIVATE KEY-----

Two types of wheels for the bike



ENTIRE GIT HISTORY **PULL-REQUESTS**

Report findings with context



SQL connection string

#123 Open in f457c475 • detected 15m ago

Locations

/src/app/connector.cs @ f457c754

```
            connection.ConnectionString);
        }
    {
        return "Data Source=np:$(SQLTarget);Initial Catalog=FABRIKAM;UID=sa;PWD=password123;Encrypt=$(Encrypt);Trust
    }
}
```

Recommendation

Review evidence of possible plaintext (or base 64-encoded plaintext) secrets in versioned engineering content.

Follow the steps below before you close this alert:

1. Rotate the secret and store securely if it's in use to prevent breaking workflows.
2. Revoke this SQL connection string to prevent unauthorized access.
3. Check security logs for potential breaches.
4. Close the alert as revoked.

Severity

Critical

Confidence: other

Introduced

Jul 9, 2021

Finding details

Type
SQL connection string

ID
SEC101/200

We deployed at large scale

Prevent new leaks

- Scan triggered on pull-requests
- Changes only

Remediate existing leaks

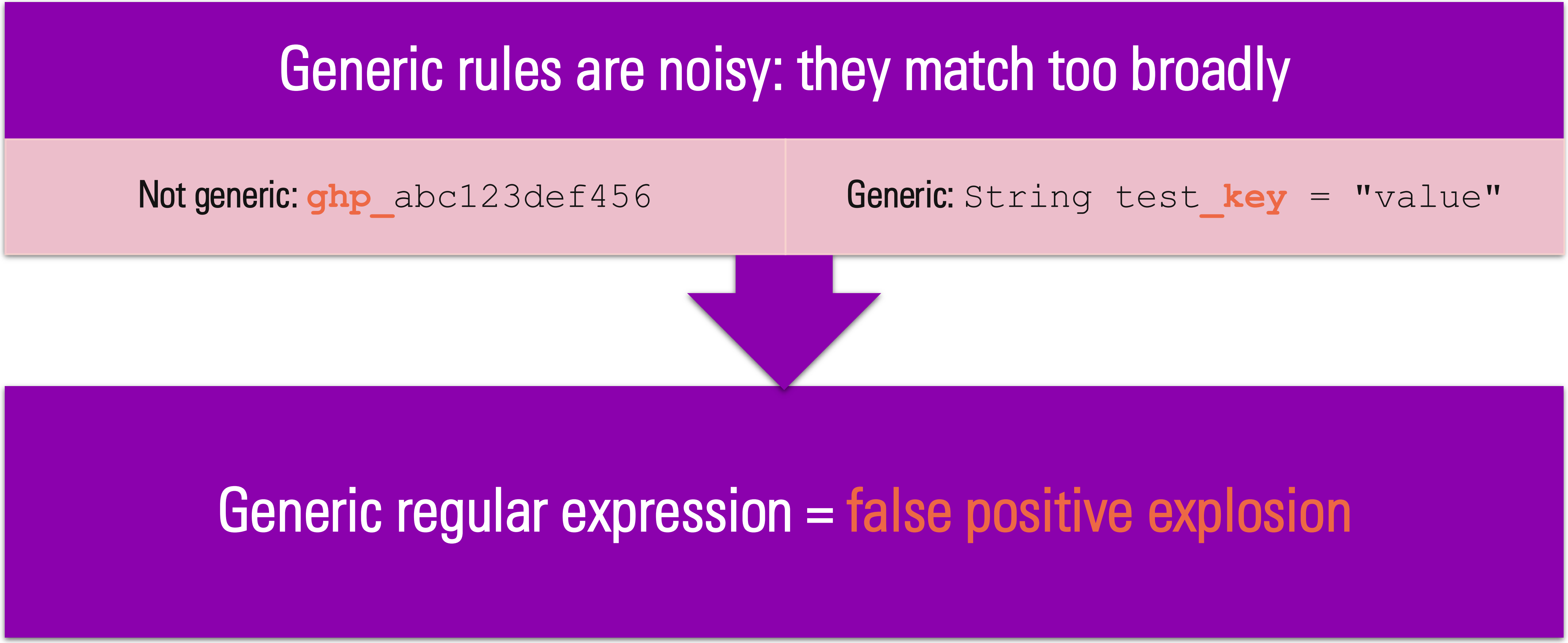
- Recurrent scan of all repositories
- Full history

Great news!

Many findings

… including a lot of false positives

Regular expressions easily generate false positives



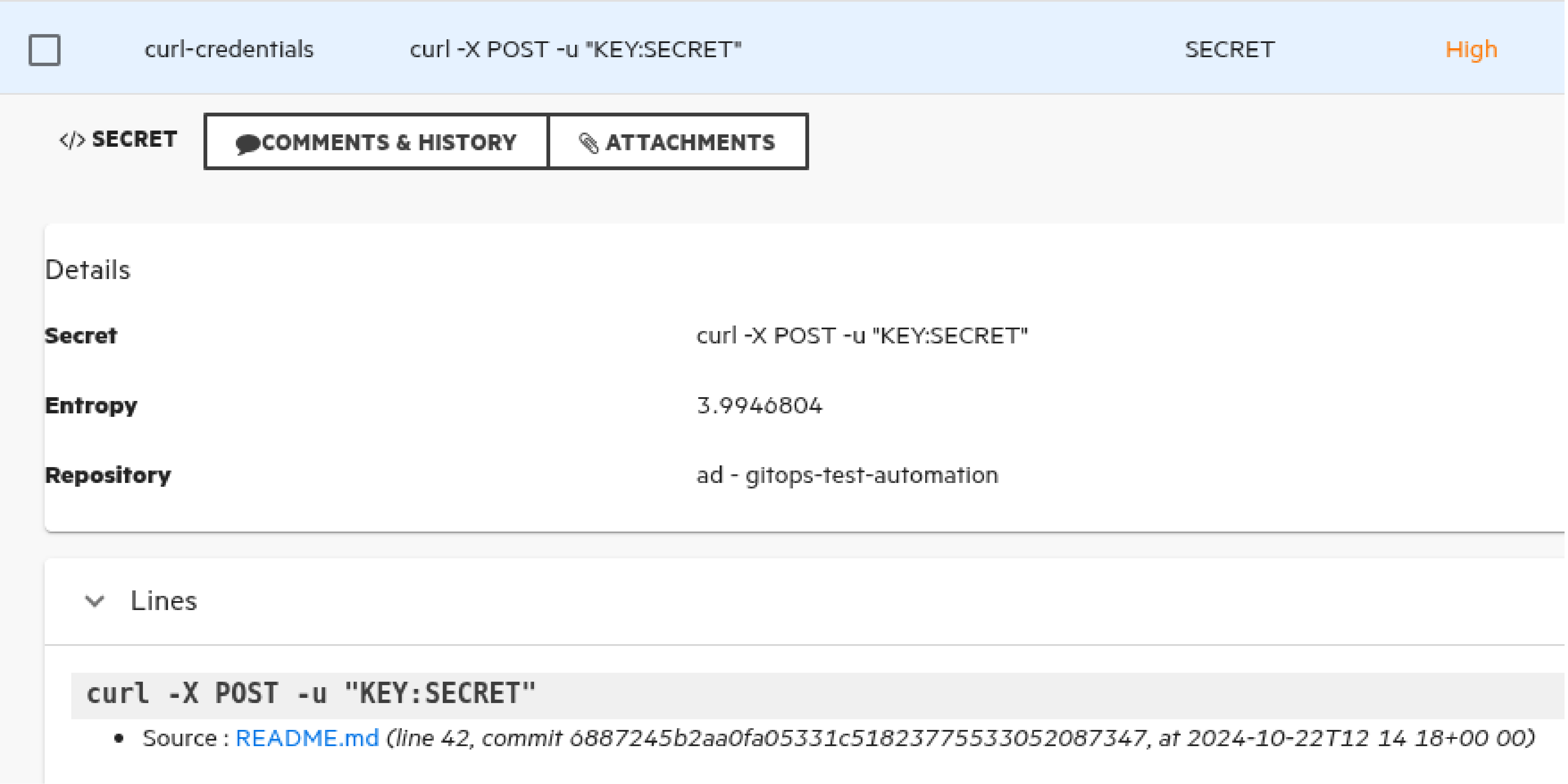
Generic rules are noisy: they match too broadly

Not generic: `ghp_abc123def456`

Generic: `String test_key = "value"`

Generic regular expression = false positive explosion

Annoying…



□	curl-credentials	curl -X POST -u "KEY:SECRET"	SECRET	High

Details

Secret	curl -X POST -u "KEY:SECRET"
Entropy	3.9946804
Repository	ad - gitops-test-automation

Lines

```
curl -X POST -u "KEY:SECRET"
```

- Source : README.md *(line 42, commit 6887245b2aa0fa05331c51823775533052087347, at 2024-10-22T12 14 18+00 00)*

RUNNING

SIFT FOR SMART TRIAGING OF FINDINGS

Triaging findings is time consuming

Can you tell which of these are real secrets?

```
Filename: conf/system/prod/server.conf.old
Line 5:    Password="v9$Tq7#pLm2@Xc8!"
Rule: generic-key
```

✓

```
Filename: etc/apps/graphql/schema/plugins.py
Line 12:   key="cvssV3TemporalVector"
Rule: generic-key
```

✗

```
Filename: api/.npmrc
Line 2:    auth = T2hObzpUaGlzSXNBRmFrZQ==
Rule: generic-api-key
```

✓

How did we reason as a human?

EXTRACT

```
Filename: api/.npmrc
Line 2:    auth = T2hObzpUaGlzSXNBRmFrZQ==
Rule: generic-api-key
```

REASON

- api/.npmrc → a valid NPM configuration file
- The key name is auth → self-explanatory ☺
- Base64 encoded string → looks like a valid credential
- ⚠ But once decoded: OhNo:ThisIsAFake → obviously not used in Amadeus

CONCLUDE

✗

True Positive
False Positive

Let’s ask a LLM to perform the analysis!

EXTRACT

REASON

CONCLUDE

```
Filename: api/.npmrc
Line 2:   auth = T2hObzpUaGlzSXNBRmFrZQ==
Rule: generic-api-key
```

>>> Tell me if it is a false positive or a potential secret and why

LLM Model 1

The presence of an auth directive inside an .npmrc file strongly indicates a credential configured for npm registry access. The value looks like an encoded token, which matches typical secret patterns. Given the file path and the semantic meaning of auth in .npmrc, I would classify this as a potential secret. ✓

LLM Model 2

An .npmrc file carrying an auth = <value> line is typically used to store registry access tokens. The final decision should verify the actual auth value (e.g., by decoding or validating whether it is a placeholder). I would classify this as potential secret. ✓

LLM Model 3

While .npmrc files often contain registry authentication, this specific entry looks like a non-sensitive placeholder. The value is Base64-encoded and decodes to OhNo:ThisIsAFake, which is clearly not a production credential. I would classify this as a false positive. ✗

How would different LLMs perform on a real project?

<u>Human triage</u>

- 📋 1 project
- ✓ 10 true positives
- ✕ 44 false positives
- ⌛ ~45 min triage

A FEW HOURS LATER...

Results of an early assessment with several LLMs

Human triage

- 📋 1 project
- ✓ 10 true positives
- ✕ 44 false positives
- ⌛ ~45 min triage

Models	True Positives Identified	False Positives Identified
codellama:13b	60%	20%
mistral-nemo	100%	27%
phi4	100%	27%
qwen2.5-coder:14b	70%	75%
mistral-small	100%	88%

Analysis performed in March 2025

↑

Promising results… in a few minutes!

Turning a generic LLM into a finding triage assistant

ROLE

```
You are an expert in software and infrastructure security
You like simplicity and pragmatism
```

OBJECTIVE

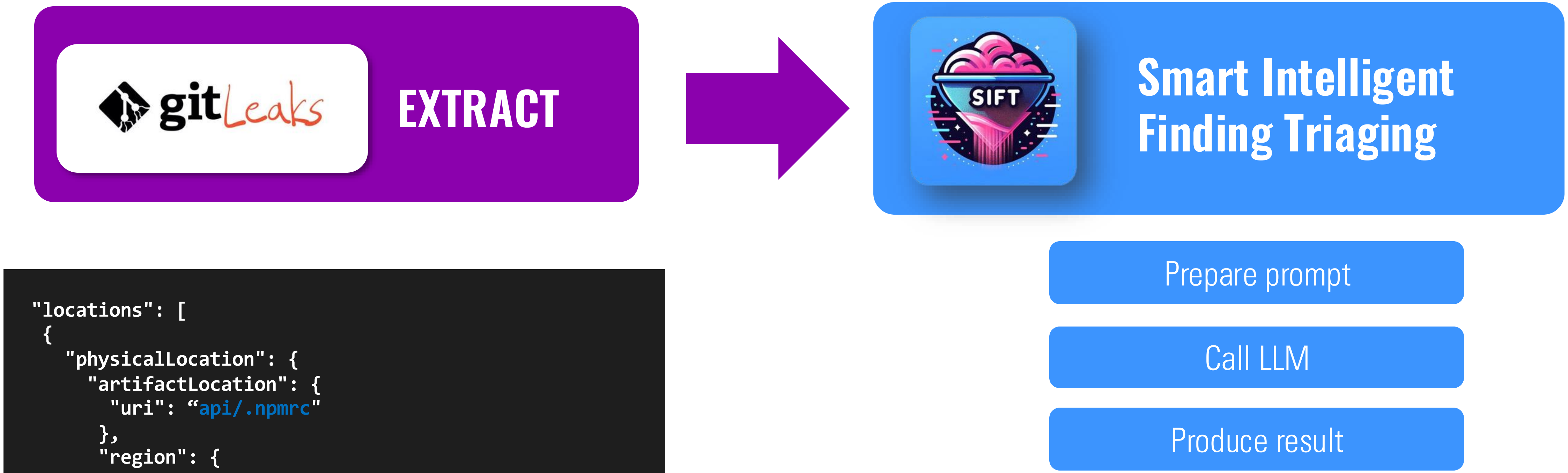
```
You analyze security alerts and decide if you consider them as
potential secrets or false positive
You rely only on the alert inside the <SecurityScannerData> tags
Your answer must be binary: TRUE POSITIVE or FALSE POSITIVE
```

FORMAT

```
Your answer must be a machine-readable JSON object containing:
-  "result" : true or false
-  "reasons": a short explanation supporting your conclusion
```

LLM's brain configuration

Turning our reasoning into automation: SIFT!



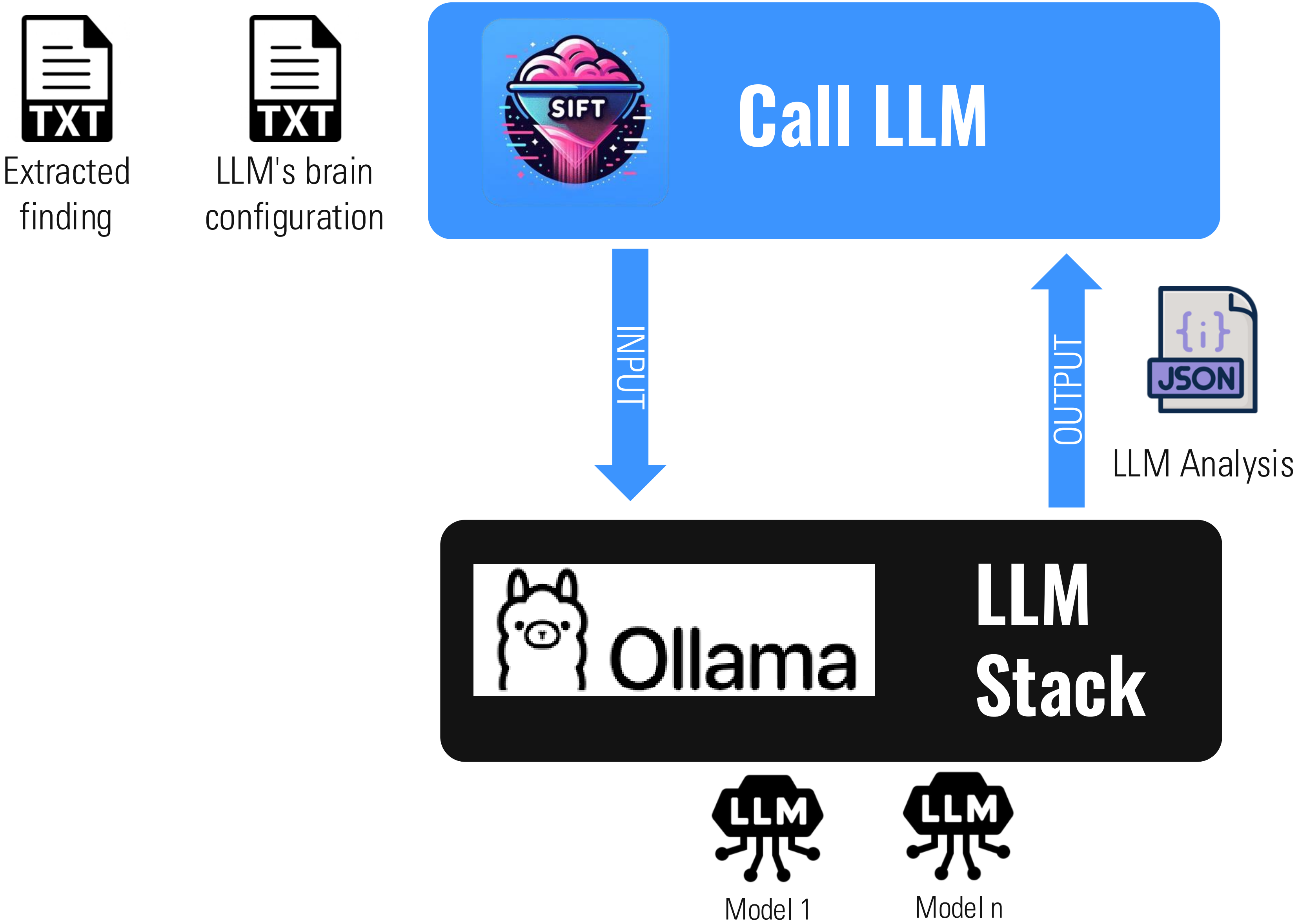
```
"locations": [
 {
   "physicalLocation": {
     "artifactLocation": {
       "uri": “api/.npmrc"
      },
      "region": {
        "startLine": 2,
        "snippet": {
          "text": “auth = T2hObzpUaGlzSXNBRmFrZQ=="
        }
       }
     }
 }
],
```

GitLeaks result (raw finding)

```
"analysis": {
    "result": "false",
    "reasons": "The detected string is encoded in base64. The string
decodes to OhNo:ThisIsAFake which suggests it might be a placeholder
or example text. Despite it is present in an api/.npmrc file, this
is likely a false positive."
  }
```

SIFT result (human-like reasoning, at scale)

What’s behind the scene “Call LLM”?



Extracted finding

LLM's brain configuration

Call LLM

INPUT

OUTPUT

LLM Analysis

LLM Stack

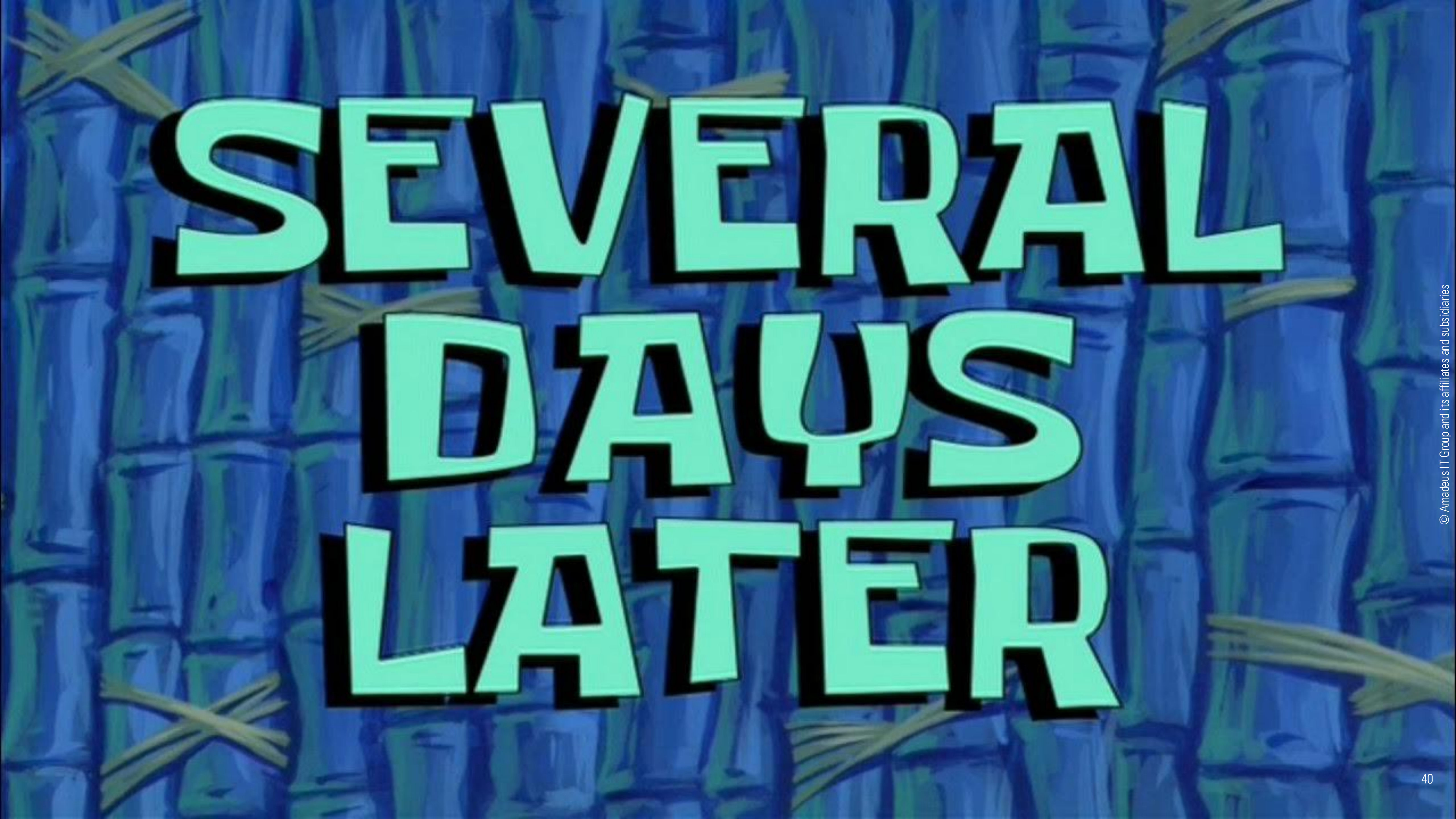
Model 1

Model n

Let’s run SIFT on all already triaged findings!

<u>Human triage</u>

- ☰ 15,000 findings
- ✓ 1,500 true positives
- ✕ 13,500 false positives
- ⌛ ~59 estimated days


SEVERAL
DAYS
LATER

How did SIFT compete with humans on 15.000 findings?

1.500
10% of True Positives

13.500
90% of False Positives

475
Hours of triage
(~59 working days)

Figures shown are representative and do not reflect actual company

97%
True Positive Agreement

82%
False Positive Agreement

83
Hours of triage
(~3 days - not sleeping)

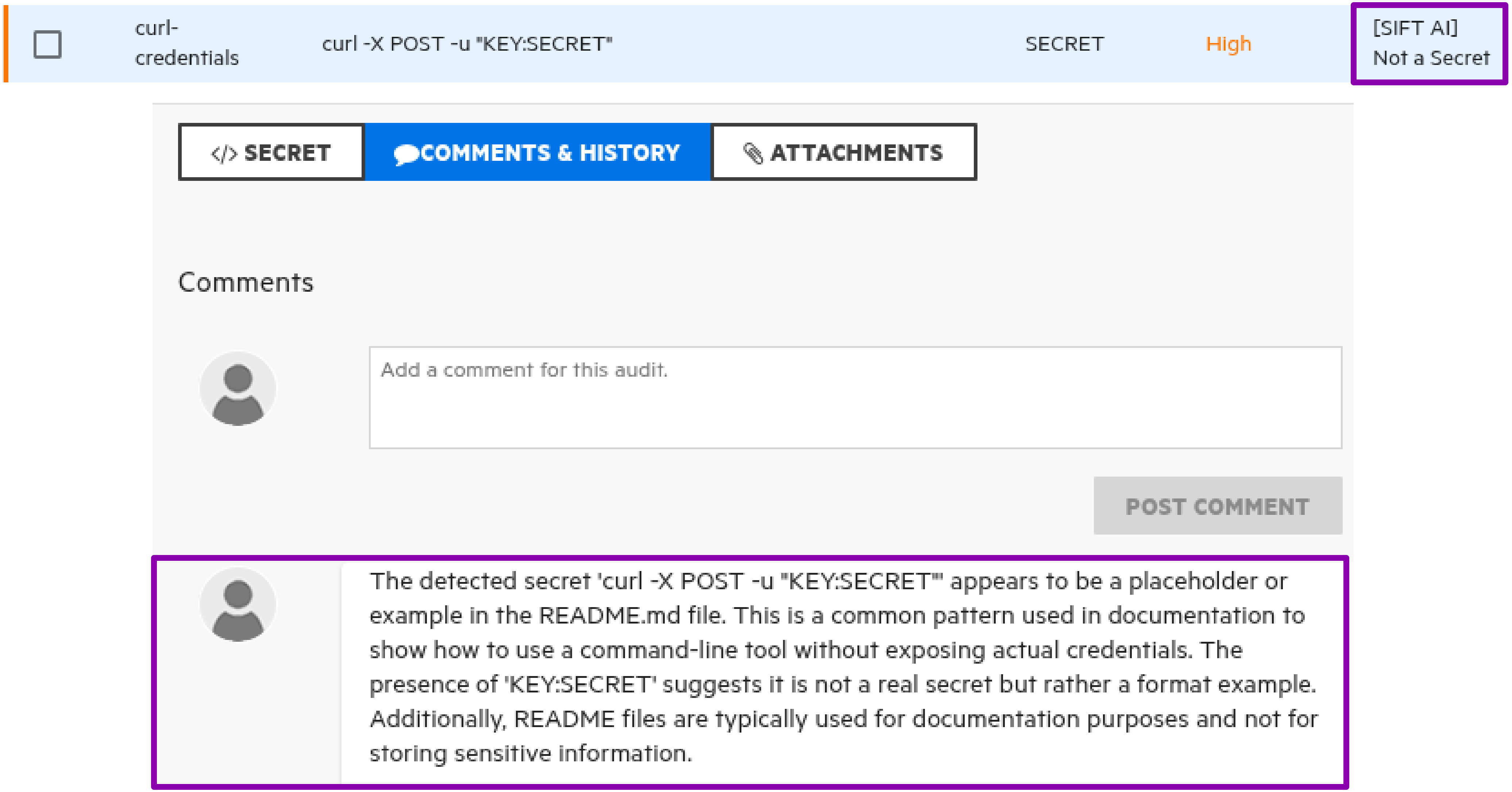
We have a new product that will reduce needed effort to win the race against secrets findings.

SIFT: *Smart Intelligent Finding Triaging*

And it is 100% legal, secured and open-sourced!

SEVERAL
WEEKS
LATER

Bye false positives!



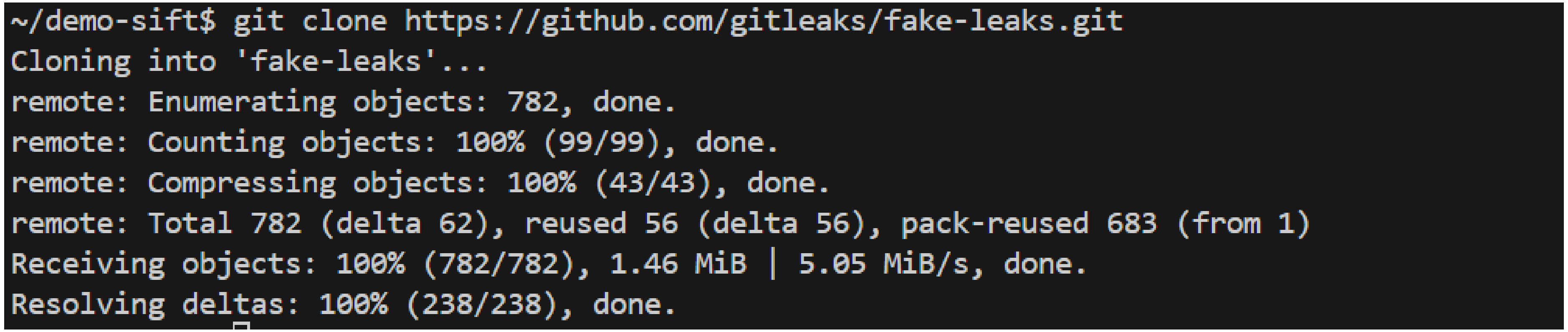
curl-credentials | curl -X POST -u "KEY:SECRET" | SECRET | High | [SIFT AI] Not a Secret

The detected secret 'curl -X POST -u "KEY:SECRET"' appears to be a placeholder or example in the README.md file. This is a common pattern used in documentation to show how to use a command-line tool without exposing actual credentials. The presence of 'KEY:SECRET' suggests it is not a real secret but rather a format example. Additionally, README files are typically used for documentation purposes and not for storing sensitive information.

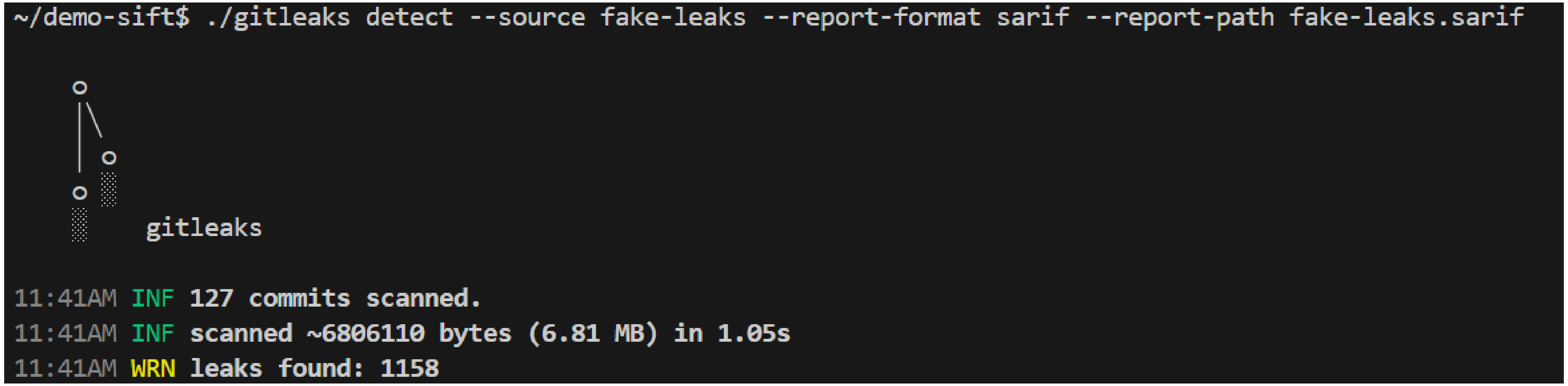
DEMO TIME

Step 1 – Run gitleaks on a project

We scan gitleaks/fake-leaks repository



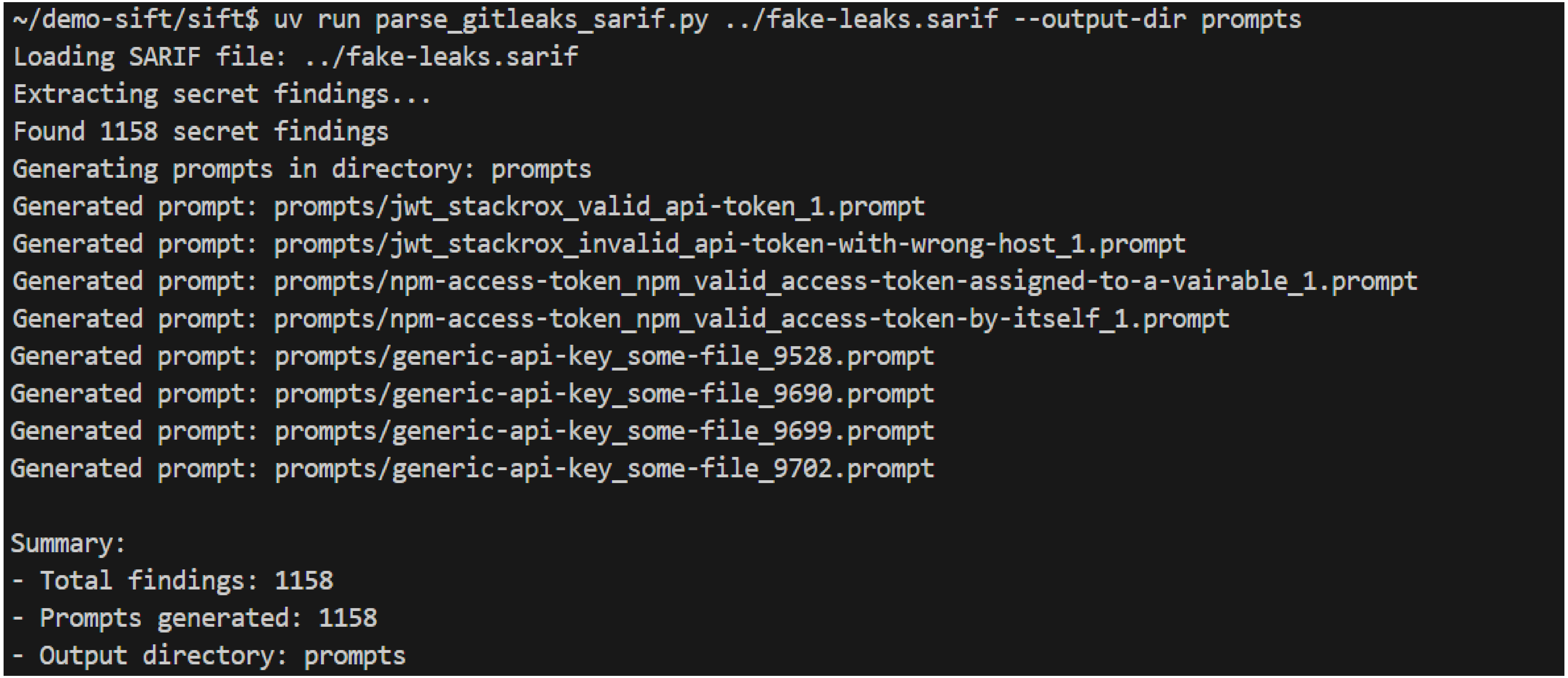
```
~/demo-sift$ git clone https://github.com/gitleaks/fake-leaks.git
Cloning into 'fake-leaks'...
remote: Enumerating objects: 782, done.
remote: Counting objects: 100% (99/99), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 782 (delta 62), reused 56 (delta 56), pack-reused 683 (from 1)
Receiving objects: 100% (782/782), 1.46 MiB | 5.05 MiB/s, done.
Resolving deltas: 100% (238/238), done.
```



```
~/demo-sift$ ./gitleaks detect --source fake-leaks --report-format sarif --report-path fake-leaks.sarif

    ○
    │╲
    │ ○
    ○ ░
    ░    gitleaks

11:41AM INF 127 commits scanned.
11:41AM INF scanned ~6806110 bytes (6.81 MB) in 1.05s
11:41AM WRN leaks found: 1158
```

Step 2 – Transform gitleaks results into prompts

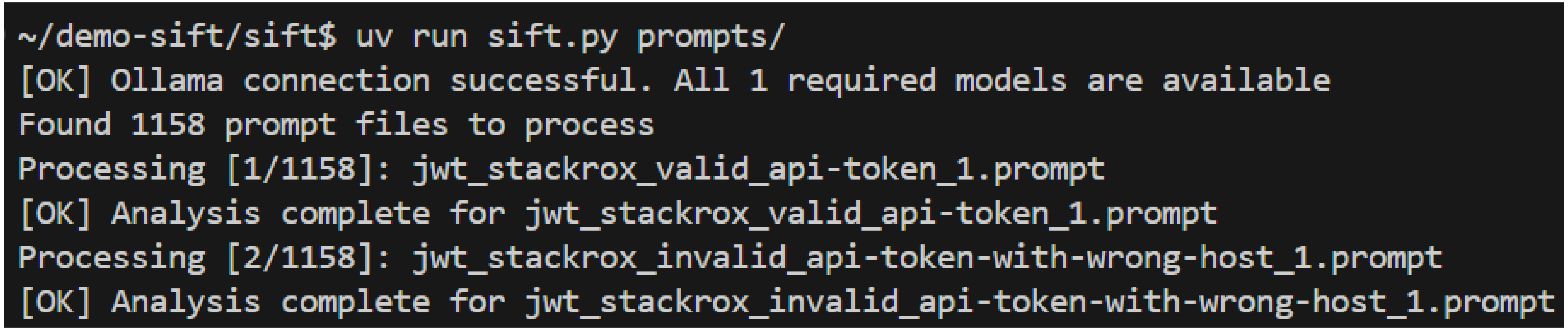
We use sift/parse_gitleaks_sarif.py



```
~/demo-sift/sift$ uv run parse_gitleaks_sarif.py ../fake-leaks.sarif --output-dir prompts
Loading SARIF file: ../fake-leaks.sarif
Extracting secret findings...
Found 1158 secret findings
Generating prompts in directory: prompts
Generated prompt: prompts/jwt_stackrox_valid_api-token_1.prompt
Generated prompt: prompts/jwt_stackrox_invalid_api-token-with-wrong-host_1.prompt
Generated prompt: prompts/npm-access-token_npm_valid_access-token-assigned-to-a-vairable_1.prompt
Generated prompt: prompts/npm-access-token_npm_valid_access-token-by-itself_1.prompt
Generated prompt: prompts/generic-api-key_some-file_9528.prompt
Generated prompt: prompts/generic-api-key_some-file_9690.prompt
Generated prompt: prompts/generic-api-key_some-file_9699.prompt
Generated prompt: prompts/generic-api-key_some-file_9702.prompt

Summary:
- Total findings: 1158
- Prompts generated: 1158
- Output directory: prompts
```

Step 3 – SIFT them all!

We use sift/sift.py

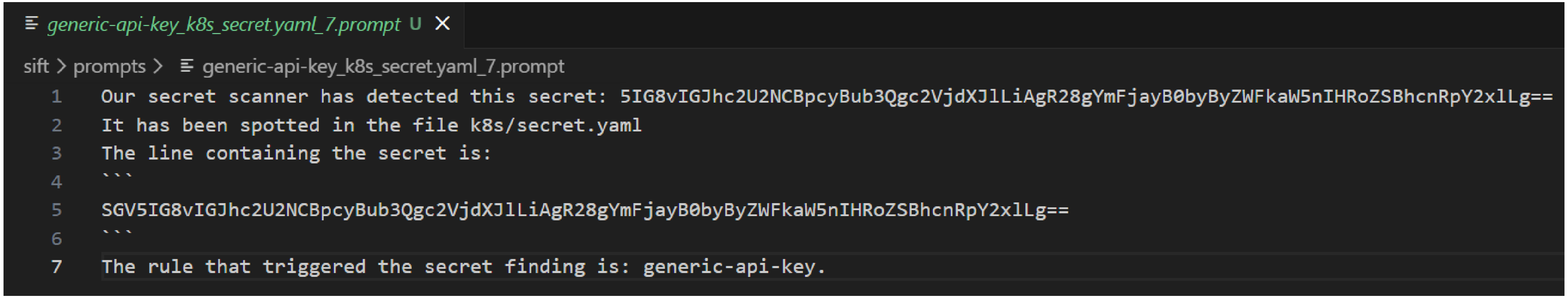


```
~/demo-sift/sift$ uv run sift.py prompts/
[OK] Ollama connection successful. All 1 required models are available
Found 1158 prompt files to process
Processing [1/1158]: jwt_stackrox_valid_api-token_1.prompt
[OK] Analysis complete for jwt_stackrox_valid_api-token_1.prompt
Processing [2/1158]: jwt_stackrox_invalid_api-token-with-wrong-host_1.prompt
[OK] Analysis complete for jwt_stackrox_invalid_api-token-with-wrong-host_1.prompt
```

Step 4 – Check results

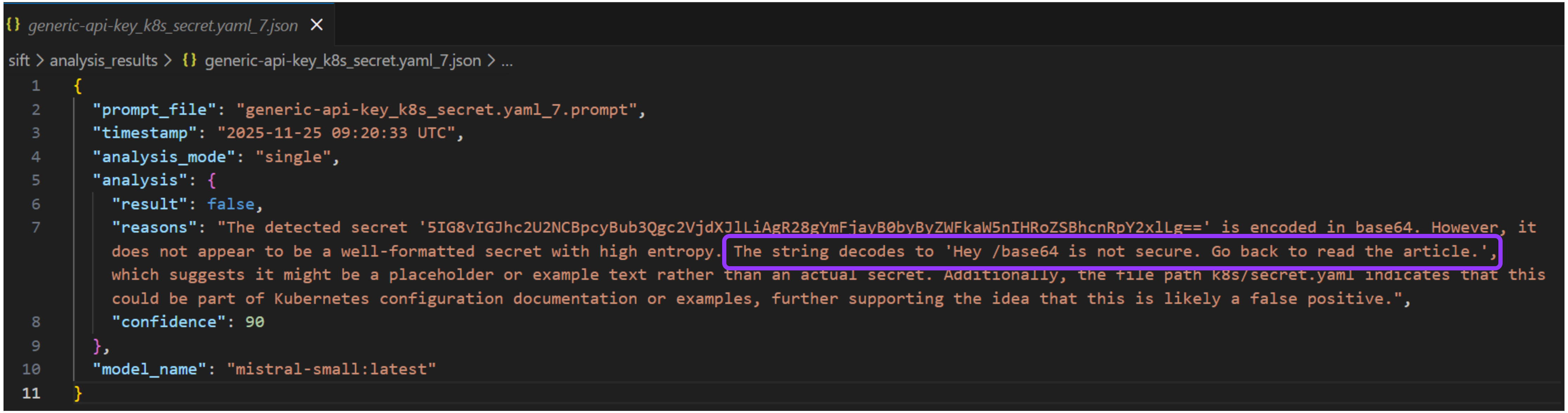
You use your eyes

Extracted finding


generic-api-key_k8s_secret.yaml_7.prompt U ×
sift > prompts > generic-api-key_k8s_secret.yaml_7.prompt

```
Our secret scanner has detected this secret: 5IG8vIGJhc2U2NCBpcyBub3Qgc2VjdXJlLiAgR28gYmFjayB0byByZWFkaW5nIHRoZSBhcnRpY2xlLg==
It has been spotted in the file k8s/secret.yaml
The line containing the secret is:
```
SGV5IG8vIGJhc2U2NCBpcyBub3Qgc2VjdXJlLiAgR28gYmFjayB0byByZWFkaW5nIHRoZSBhcnRpY2xlLg==
```
The rule that triggered the secret finding is: generic-api-key.
```

LLM Analysis


generic-api-key_k8s_secret.yaml_7.json ×
sift > analysis_results > {} generic-api-key_k8s_secret.yaml_7.json > ...

```
{
  "prompt_file": "generic-api-key_k8s_secret.yaml_7.prompt",
  "timestamp": "2025-11-25 09:20:33 UTC",
  "analysis_mode": "single",
  "analysis": {
    "result": false,
    "reasons": "The detected secret '5IG8vIGJhc2U2NCBpcyBub3Qgc2VjdXJlLiAgR28gYmFjayB0byByZWFkaW5nIHRoZSBhcnRpY2xlLg==' is encoded in base64. However, it does not appear to be a well-formatted secret with high entropy. The string decodes to 'Hey /base64 is not secure. Go back to read the article.', which suggests it might be a placeholder or example text rather than an actual secret. Additionally, the file path k8s/secret.yaml indicates that this could be part of Kubernetes configuration documentation or examples, further supporting the idea that this is likely a false positive.",
    "confidence": 90
  },
  "model_name": "mistral-small:latest"
}
```

CELEBRATION!

Wrap-up

- At Amadeus, we do have some ideas…

  and now we have SIFT! 😄

- Significant improvements for:
  - Triage accuracy
  - User eXperience

- Open Source initiative: https://github.com/AmadeusITGroup/sift

THANK YOU (SIFT)

Telecom Valley | Animateur Azuréen Numérique

AZUR TECH *Winter*

2 DÉCEMBRE 2025 | SOPHIA ANTIPOLIS

LES TECHNOLOGIES DU NUMÉRIQUE