Secrets leakage detection & prevention

Meethack (Torino, 2024-06-18)

Agenda

- Houston, we have a *problem*
- *Detection* is important…
- … but *Prevention* is better!
- *Paved roads*, the cultural change
- Let’s wrap it up!
- Questions?

https://en.wikipedia.org/wiki/Smokey_Bear

Houston, we have a *problem*

Leaked secrets could lead to data breaches

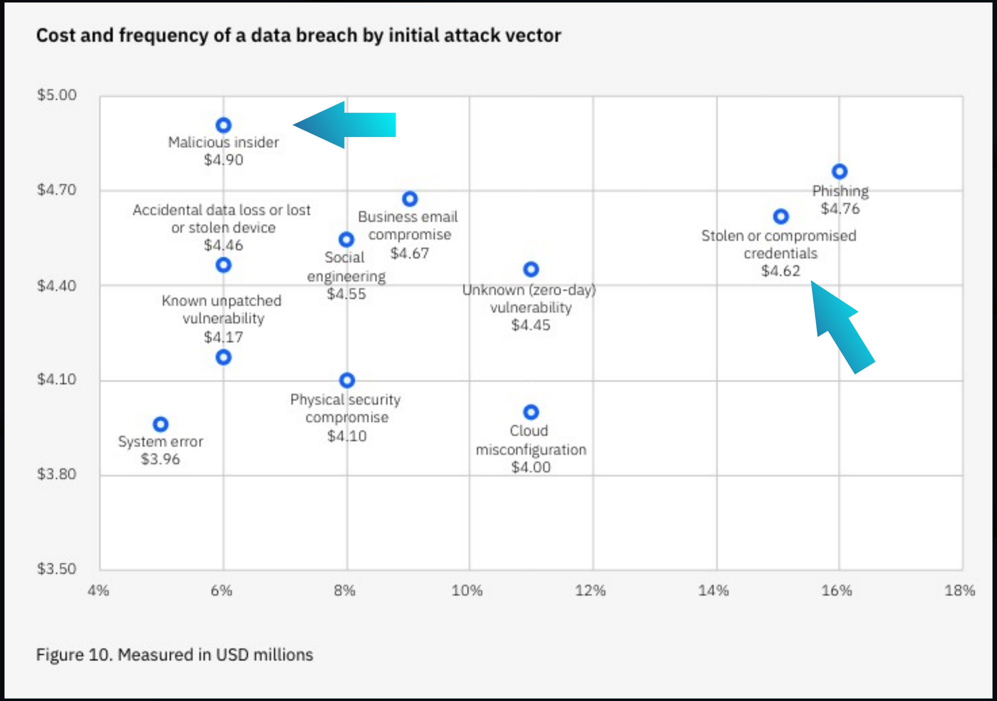


Figure 10. Measured in USD millions

- The usage of **stolen or compromised credentials** is the second common initial vector, by frequency, for a data breach.
  - With a frequency of 15% and a cost of 4.62M USD.
- The **malicious insider** is the highest initial vector, in terms of cost, for a data breach.
  - With a frequency of 6% and a cost of 4.90M USD.
- *“Assume breach”*

“Cost of a Data Breach Report 2023”, Ponemon Institute

They are called *secrets* for a reason

Secrets encompass confidential information, such as: passwords, encryption keys, API tokens, digital certificates, etc.

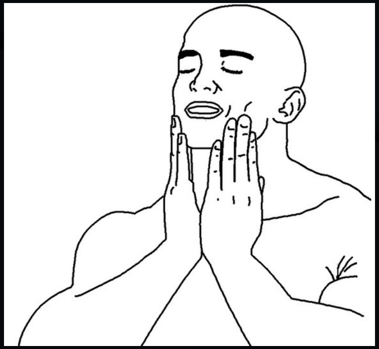
Secrets are pivotal for authenticating and authorizing access to secured resources and systems.

Detection is important...

Detection lets you know when there is a problem

- Secrets detection is part of *Static Application Security Testing* (*SAST*).
- There are several tools, commercial or not, able to perform this kind of checks:
  - *gitleaks* - https://github.com/gitleaks/gitleaks
  - *trufflehog* - https://github.com/trufflesecurity/trufflehog
  - *ggshield* - https://github.com/GitGuardian/ggshield
  - *detect-secrets* - https://github.com/Yelp/detect-secrets
  - *git-secrets* - https://github.com/awslabs/git-secrets
  - *Semgrep Secrets* - https://semgrep.dev/products/semgrep-secrets
  - ...
- In this talk Gitleaks will be used, but the **concepts are the same**!

Detection has its own limitations



Sometimes detection is easier…

```
aws_secret="AKIAIMNOJVGFDXXXE4OA"
```

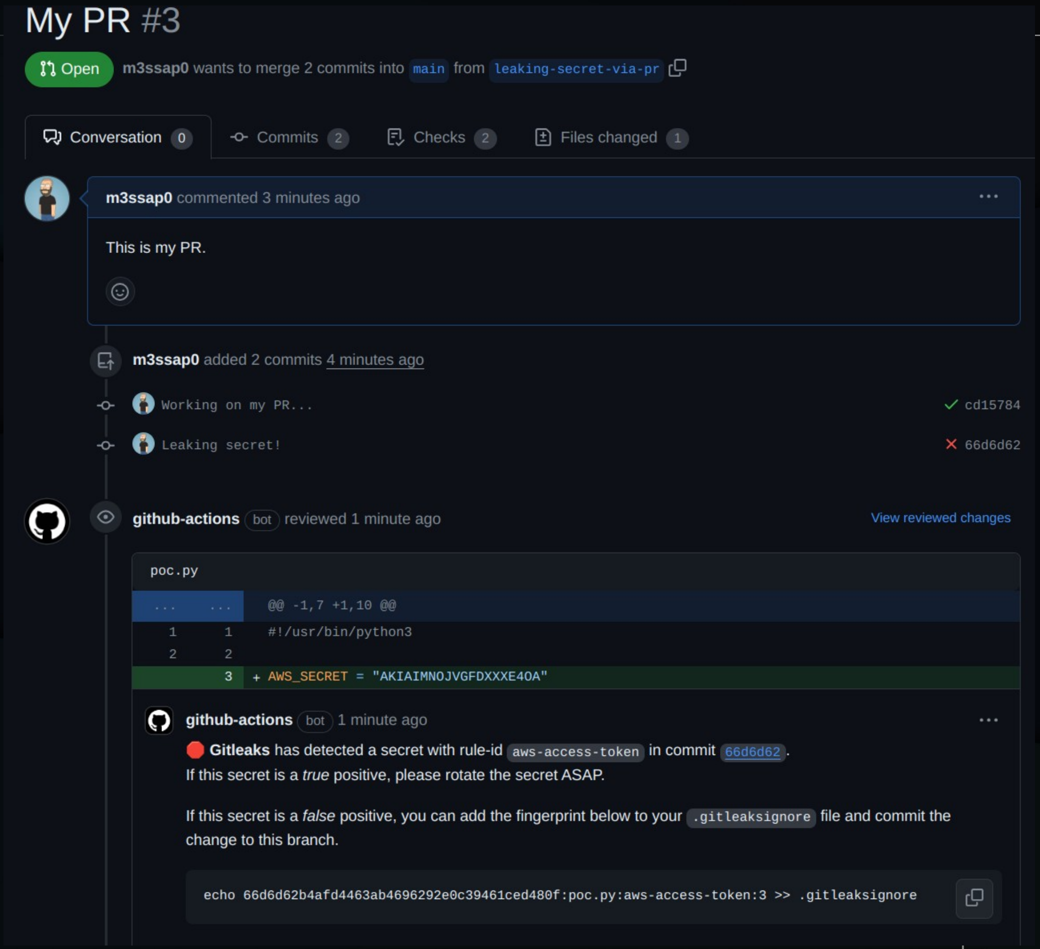
Sometimes detection is harder…

```
password_field_label="password-fld-lbl-1"
```

```
my_password="$up3rP4ssw0rd!"
```

Centralize detection in CI/CD to spot problems

- It’s unrealistic to scale Application Security activities without leveraging on automation.
- Look for plugins for your CI/CD ecosystem.
  - Gitleaks has an official GitHub Action.



My PR #3

Open m3ssap0 wants to merge 2 commits into `main` from `leaking-secret-via-pr`

Conversation 0 | Commits 2 | Checks 2 | Files changed 1

m3ssap0 commented 3 minutes ago

This is my PR.

m3ssap0 added 2 commits 4 minutes ago

```
Working on my PR...     ✓ cd15784
Leaking secret!         ✗ 66d6d62
```

github-actions bot reviewed 1 minute ago — View reviewed changes

```
poc.py
...   ...   @@ -1,7 +1,10 @@
1     1     #!/usr/bin/python3
2     2
      3     + AWS_SECRET = "AKIAIMNOJVGFDXXXE4OA"
```

github-actions bot 1 minute ago

🛑 **Gitleaks** has detected a secret with rule-id `aws-access-token` in commit `66d6d62`.
If this secret is a *true* positive, please rotate the secret ASAP.

If this secret is a *false* positive, you can add the fingerprint below to your `.gitleaksignore` file and commit the change to this branch.

```
echo 66d6d62b4afd4463ab4696292e0c39461ced480f:poc.py:aws-access-token:3 >> .gitleaksignore
```

https://github.com/gitleaks/gitleaks-action

Example of a GitHub workflow

```yaml
name: gitleaks
on: [pull_request, push, workflow_dispatch]
permissions:
  # Allow access to commit list
  contents: read
  # Allow access to adding comments
  discussions: write
  pull-requests: write
jobs:
  scan:
    name: gitleaks
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
        with:
          fetch-depth: 0
      - uses: gitleaks/gitleaks-action@v2
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

https://github.com/gitleaks/gitleaks-action

Customize the solution based on your needs

- ~166 standard rules provided by Gitleaks.
- Rules are based on regexes.
- You can create your custom rules via TOML files and use them
  - with the `-c` param of the executable
  - or the `GITLEAKS_CONFIG` environment variable of the GHA.

```toml
# Your custom Gitleaks configuration file.
title = "Your custom Gitleaks rules"


# Extending default rules.
[extend]
useDefault = true


[[rules]]
# Put your custom rules here.
```

https://github.com/gitleaks/gitleaks/blob/master/config/gitleaks.toml

Example of a Gitleaks rule

https://github.com/gitleaks/gitleaks/blob/79cac73f7267f4a48f4bc73db11e105a6098a836/config/gitleaks.toml#L124

```
[[rules]]

id = "aws-access-token"

description = "Identified a pattern that may indicate AWS
credentials, risking unauthorized cloud resource access and data
breaches on AWS platforms."

regex = '''(?:A3T[A-Z0-9]|AKIA|ASIA|ABIA|ACCA)[A-Z0-9]{16}'''

keywords = [

    "akia","asia","abia","acca",

]
```

Keywords are used for **pre-regex check filtering**.

Rules that contain keywords will perform a quick string compare check to make sure the keyword(s) are in the content being scanned.

https://github.com/gitleaks/gitleaks?tab=readme-ov-file#configuration

… but *Prevention* is better!

Pre-commit hooks can prevent leaks

- A leaked secret – even if detected – is still a leaked secret.
- *Pre-commit* hooks can be configured in your workstation to perform scan locally, blocking dangerous commits and preventing leaks from happening.

- Install Gitleaks (it requires Go).
- Create a folder to store global hooks, for example:
  - `/home/<your_user>/gitconfig/hooks`
- In that folder, create a file named **exactly**:
  - `pre-commit`
- In that file, write the script to perform the check (Python example in the next slide).
- Make the file executable.
- Edit global git config file, usually `.gitconfig` in your home, to add the following lines.

```
[core]
    hooksPath = /home/<your_user>/gitconfig/hooks
[hooks]
    gitleaks = true
```

Example of *pre-commit* hook in Python

```
def gitleaksEnabled():
    out = subprocess.getoutput('git config --bool hooks.gitleaks')
    if out == "false":
        return False
    return True

if gitleaksEnabled():
    exitCode = os.WEXITSTATUS(os.system('gitleaks protect -v --staged --redact'))
    if exitCode == 1:
        print('Warning: gitleaks has detected sensitive information in your changes.')
        sys.exit(1)
else:
    print('gitleaks precommit disabled (enable with `git config hooks.gitleaks true`)')
```

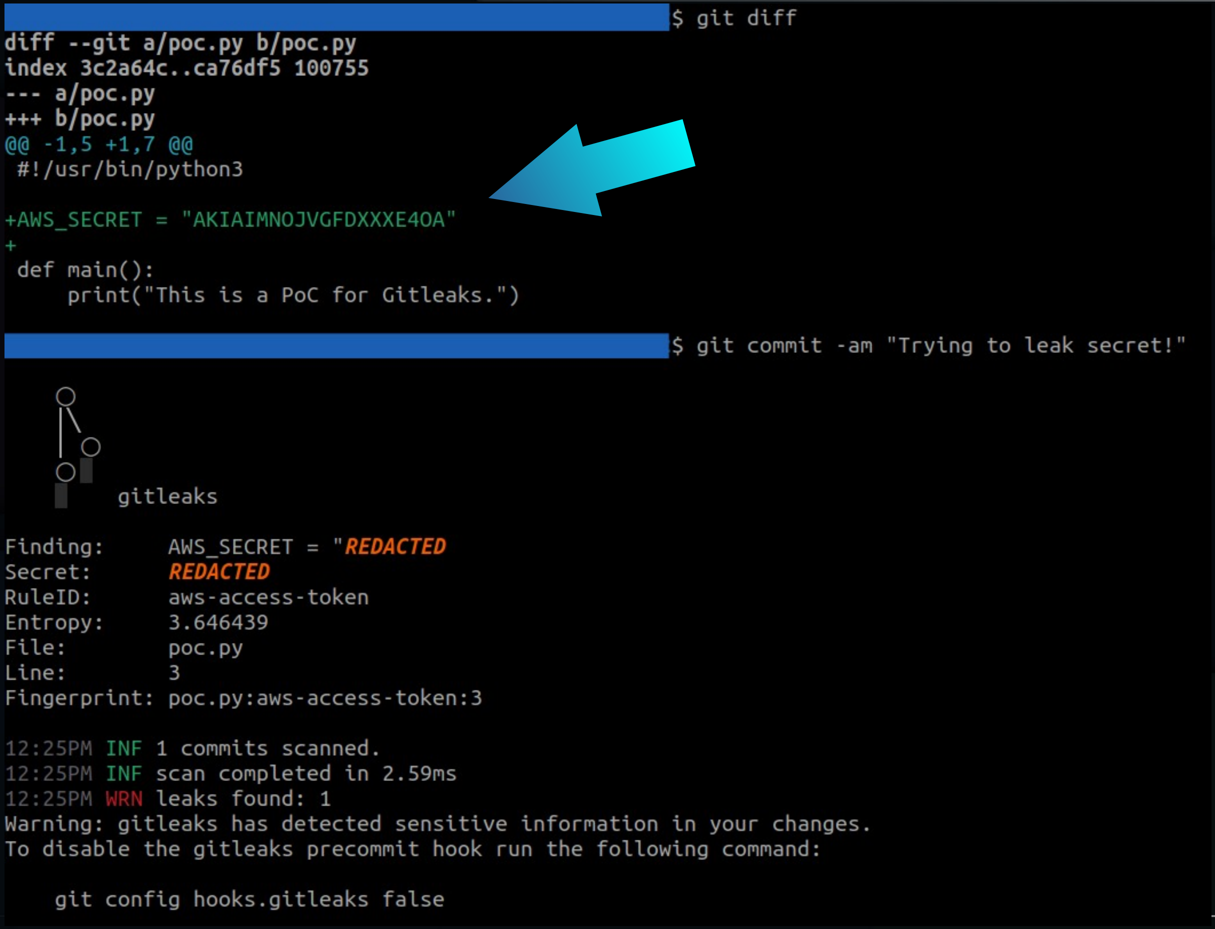
Used to scan uncommitted changes in a git repo. This command should be used on developer machines.

To check for changes in commits that have been **`git add`**ed.

Redact secrets from logs and stdout.

https://github.com/gitleaks/gitleaks/blob/master/scripts/pre-commit.py

Commit blocked on the development workstation



```
$ git diff
diff --git a/poc.py b/poc.py
index 3c2a64c..ca76df5 100755
--- a/poc.py
+++ b/poc.py
@@ -1,5 +1,7 @@
 #!/usr/bin/python3

+AWS_SECRET = "AKIAIMNOJVGFDXXXE4OA"
+
 def main():
     print("This is a PoC for Gitleaks.")

$ git commit -am "Trying to leak secret!"

    ○
    │╲
    │ ○
    ○ ▌
    ▌   gitleaks

Finding:     AWS_SECRET = "REDACTED
Secret:      REDACTED
RuleID:      aws-access-token
Entropy:     3.646439
File:        poc.py
Line:        3
Fingerprint: poc.py:aws-access-token:3

12:25PM INF 1 commits scanned.
12:25PM INF scan completed in 2.59ms
12:25PM WRN leaks found: 1
Warning: gitleaks has detected sensitive information in your changes.
To disable the gitleaks precommit hook run the following command:

    git config hooks.gitleaks false
```

Paved roads, the cultural change

Make the wrong road also the hard one

- *Paved roads* aka *secure defaults*, *golden paths*, ...
- Give to software engineers solutions, not just problems to solve.
- Invest in the adoption of secrets management tools:
  - *HashiCorp Vault*
  - *Google Cloud Secret Manager*
  - *AWS Secrets Manager*
  - *Azure Key Vault*
  - ...
- Software engineers will have a concrete solution to their problem and you will effectively manage the secrets ecosystem.

Let’s wrap it up!

A problem, but complementary ways to solve it

- Secrets leaked in source code can be used by malicious actors to compromise other platforms in your ecosystem.
- Automatic tools exist to perform checks.
  - Centralize the scan to scale.
  - Customize the solution with your own rules.
  - Prevent at development workstations.
- Invest in the culture and provide solutions via usable secure defaults.

Thank you!
Questions?

https://m3ssap0.github.io/assets/resources/meethack/Secrets%20leakage%20detection%20&%20prevention.pdf