Stop Committing Your Secrets - Git Hooks To The Rescue

@ChiBrrCon
Dwayne McDaniel
@mcdwayne
GitKraken
Developer Evangelist
Living in Chicago, IL
GitKraken
Legendary Git Tools

GitKraken Client for Desktop
GitLens for VS Code
Git Integration for Jira
What is a Git talk doing at ChiBrrCon II?

- Devs all use Git on the regular to manage projects, most of which end up on services like Azure, AWS, or GCP.
- Almost all repos end up on a Git repo hosting service like GitHub, GitLab, or BitBucket.
- All of those services require secrets (passwords, API keys, credentials, certificates, etc).
What is a Git talk doing at ChiBrrCon II?

- Over two million secrets were detected on public GitHub alone in 2020 and it is growing ~ 20% Year-Over-Year -- GitGuardian State of Secrets Sprawl on GitHub Report.

- 85% of those secrets are found in developers’ personal repositories - outside of corporate control. – same GitGuardian report
Master Locksmiths Association (MLA) @MLA_lock... · Jan 17, 2015
Good security fail spotted by one of our members :-) 

Code for the door is 

C2597X
package main

import "fmt"
"os"

func main() {
    var dbName string
    var dbPass string

    fmt.Println("Please enter database name:")
    fmt.Scanf("%s", &dbName)

    fmt.Println("Please enter database password:")
    fmt.Scanf("%s", &dbPass)

    if dbName == databaseName && dbPass == secretKey {
        fmt.Println("Welcome to the database!")
        fmt.Println("Your secret phrase is: ", secretPhrase)
        os.Exit(0)
    } else {
        fmt.Println("Sorry, wrong database name or password")
    }
}
Over two million secrets were detected on public GitHub alone in 2020 and it is growing ~ 20% Year-Over-Year.

Git takes snapshots of your work
Git stacks those snapshots on top of one another, into a graph.
Git also creates multiple dimensions and controls reality
Git is the central mechanism for moving code around and enables DevOps and CI/CD.
Git Is Awesome!!!
But Git does not make code *more or less* secure by itself
Git does give us a way to exclude entire types of files or directories...

```
*.gem
*.rbc
.\.config
/coverage/
/InstalledFiles
/pkg/
/spec/reports/
/spec/examples.txt
/test/tmp/
/test/version_tmp/
/tmp/

# Used by dotenv library to load environment variables.
# .env

# Ignore Byebug command history file.
.bybebug_history

## Specific to RubyMotion:
.dat
.repl_history
.build/
+.bridgesupport
.build-iPhone05/
.build-iPhoneSimulator/
```
Using a `.gitignore` file can make sure you do not commit a `secrets.json`, an `.aws` directory or wherever you store API keys, usernames and passwords, or security certificates locally**
In a perfect world, this would be the end of the talk, but again...

```go
package main

import {
  "fmt"
  "os"
}

func main() {
  
  var dbName string
  var dbPass string

  fmt.Println("Please enter database name:")
  fmt.Scan("%s", &dbName)

  fmt.Println("Please enter database password:")
  fmt.Scan("%s", &dbPass)

  if dbName == "53CR3T4T48453" && dbPass == "SUP3R5CR3T" {
    fmt.Println("Welcome to the database!")
    fmt.Println("Your secret phrase is:", secretPhrase)
    os.Exit(0)
  } else {
    fmt.Println("Sorry, wrong database name or password")
  }
}
```
The issue is NOT that a dev tested a secret.

The issue is the dev forgot to remove it from your code before you committed and pushed your code.
If you commit a secret and push it to the repo, then...

You're gonna have a bad time
There are tools like **SonarLint**, **DeepSource**, and **GitHub Advanced Security** that can detect secrets in your repo.

You *can* (in theory) remove secrets from a repo, but it is not easy.

In fact it’s downright painful.
In addition to the hassle of secrets removal from a codebase, you have now destroyed a lot of trust

- Keys now need rotated

- Teams need to take time away from their day to day work to 'deal' with it

Management can't be happy...
What we need is some sort of automation that stops us from committing our secrets...
Git provides a path....

Git Hooks
You do a thing, Git triggers a thing.
There are 17 available hooks

“Every Git repository has a .git/hooks folder with a script for each hook you can bind to. You're free to change or update these scripts as necessary, and Git will execute them when those events occur.” – Matthew Hudson

https://GitHooks.com

These 3 hooks trigger before a commit happens

- applypatch-msg
- pre-applypatch
- post-applypatch
- pre-commit
- prepare-commit-msg
- commit-msg
- post-commit
- pre-rebase
- post-checkout
- post-merge
- pre-receive
- update
- post-receive
- post-update
- pre-auto-gc
- post-rewrite
- pre-push
Git comes with a bunch of sample hooks to get you started.

```bash
#!/bin/sh

# An example hook script to verify what is about to be committed.
# Called by "git commit" with no arguments. The hook should
# exit with non-zero status after issuing an appropriate message if
# it wants to stop the commit.
#
# To enable this hook, rename this file to "pre-commit".

if git rev-parse --verify HEAD >/dev/null 2>&1 then
    against=HEAD
else
    # Initial commit: diff against an empty tree object
    against=$(git hash-object -t tree /dev/null)
fi
```
Git hooks are just scripts though…

The built-in scripts are mostly shell (sh) scripts, but you can use ANY scripting language (JavaScript, PHP, Python, Ruby, Perl, Bash, etc), so long as it can be run as an executable.
An ideal solution would look like:

Before I commit,
Something should **check** my code for any hardcoded:

- **Usernames & Passwords**
- **API Keys**
- **Security Certificates**
- **Any other credentials**

If any of those are detected,
**then** throw an error and **do not** make the commit.
You *can* build this yourself, using good ol’ fashion scripting know-how!

```
#!/usr/bin/env bash
curl https://icanhazdadjoke.com

if git grep -E "password = *" | git grep -E "password=*" ; then
    echo "NO HARDCODE PASSW"D"
    exit 2
fi

if git grep -E "^[A-Z0-9]{20}" ; then
    echo "NO HARDCODE KEYS"
    exit 2
fi
```
You *can* build this *anything* yourself, using good ol’ fashion scripting know-how!

```python
5    password = exAmplepassWORD
6    password =
7    password=******
8    More test...
9
10   Key1 = AKIAIOSFODNN7HELLOWEKEY
11   Key2 = wJalrXUt>eKFI/K7MENG/bPxrF7ICYHELLOWEKEY
```

Documents/git-venture $ git add .
Documents/git-venture $ git commit -m 'readme spacing update'
What did the left eye say to the right eye? Between us, something smells!
```plaintext
README:password=exAmplepassWORD
README:password =
README:password=******
NO HARDCODE PASSWORD
```
Documents/git-venture $
The issues then become...

- We have to manually build and maintain this.
- You have to get team buy in
- You need to have the DevSecOps team vet it
- Need to account for ‘$API_ENV_VAR’ is OK to use in code.
- Account for example passwords (like ‘th1si5@g00Denuff’).
- Tell the difference between long strings and an access_key_id or secret_access_key.
- Ensure you do not accidentally hard code keys in this scheme in case it gets compromised.
- Provide a way to add more rules and patterns in a sane way.
- Keep track of APIs updates.
- Get your team to adopt your crazy hand rolled scripts.
- Make this more scalable.
- And a bunch of other things you would discover along the way.
- Etc, etc, etc.
- Why are you still reading this? Go to the next slide already.
Open Source To The Rescue!!!

https://github.com/awslabs/git-secrets
git-secrets

Prevents you from committing passwords and other sensitive information to a git repository.

```
git secrets --scan [-r|--recursive] [--cached] [--no-index] [--untracked] [<files>...]
git secrets --scan-history
git secrets --install [-f|--force] [<target-directory>]
git secrets --list [--global]
git secrets --add [-a|--allowed] [-l|--literal] [--global] <pattern>
git secrets --add-provider [--global] <command> [arguments...]
git secrets --register-aws [--global]
git secrets --aws-provider [<credentials-file>]
```
How to do it for AWS!

1. Install the tool on your machine (powershell script, make, or brew).
2. Install git-secrets per repo or tell your machine to auto-install it on all repos.
3. Run `git secrets --register-aws`
4. Run `git secrets --add` to add prohibited or allowed patterns.
5. Git-secrets will trigger upon every new commit for installed repos.
`git secrets --register-aws`:
Adds common AWS patterns to the git config and ensures that keys present in ~/.aws/credentials are not found in any commit. The following checks are added:

i. AWS Access Key IDs via
   (A3T[A-Z0-9]|AKIA|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{16}
ii. AWS Secret Access Key assignments via ":" or ":=
iii. AWS account ID assignments via ":" or ":=" surrounded by optional quotes
iv. **Allowed** patterns for example AWS keys (AKIAIOSFODNN7EXAMPLE and wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY)
v. **Known** credentials from ~/.aws/credentials
What does this add to your files?

1. Adds a line to those three Git hooks

   ```bash
   Documents/git-venture $ git secrets --install -f
   Installed commit-msg hook to .git/hooks/commit-msg
   Installed pre-commit hook to .git/hooks/pre-commit
   Installed prepare-commit-msg hook to .git/hooks/prepare-commit-msg
   ```

   ```bash
   git-venture > .git > hooks > $ commit-msg
   1  #!/usr/bin/env bash
   2  git secrets --commit_msg_hook -- "$@"
   ```

2. Adds info to your `.git/config` file

   ```bash
   git-venture > .git > config
   [secrets]
   patterns = password\s*=\s+*\n   providers = git secrets --aws-provider
   patterns = ("\"\\"|)?(AWS|aws|Aws)?_?(SECRET|secret|Secret)?_?(ACCESS|access|Access)?_?(KEY|key|Key)?("\"\\"|
   patterns = ("\"\\"|)?(AWS|aws|Aws)?_?(ACCOUNT|account|Account)?_?(ID|id|Id)?("\"\\"|
   allowed = AKIAIOSFODNN7EXAMPLE
   allowed = wJalrXUttnFEMI/K7MDENG/bPxBficyexampleKEY
   ```
What does this look like in action?

Documents/git-venture $ git add .
Documents/git-venture $ git commit -m 'readme spacing update'
READEME:3:Key1= AKIAIOSFODNN7HELLOWEKEY
READEME:4:Key2 = wJalrXUtFEMI/K7MDENG/bPXRfiCYHELLOWEKEY

[ERROR] Matched one or more prohibited patterns

Possible mitigations:
- Mark false positives as allowed using: git config --add secrets.allowed ...
- Mark false positives as allowed by adding regular expressions to .gitallowed at repository's root directory
- List your configured patterns: git config --get-all secrets.patterns
- List your configured allowed patterns: git config --get-all secrets.allowed
- List your configured allowed patterns in .gitallowed at repository's root directory
- Use --no-verify if this is a one-time false positive

Documents/git-venture $

Threw an error and does not make a commit!

Note: Those are obviously fake keys. Don't ever share real keys.
That's all well and good for AWS, but I use X...?
Again, OSS to the rescue!

GCP - https://github.com/deshpandetanmay/git-secrets

Azure - https://github.com/msalemcode/git-secrets

Add Your Own

```bash
git secrets --scan [-r|--recursive] [--cached] [--no-index] [--untracked] [<files>...]
git secrets --scan-history
git secrets --install [-f|--force] [<target-directory>]
git secrets --list [--global]
git secrets --add [-a|--allowed] [-l|--literal] [--global] <pattern>
git secrets --add-provider [--global] <command> [arguments...]
git secrets --register-aws [--global]
git secrets --register-gcp [--global]
git secrets --aws-provider [<credentials-file>]
```
In conclusion:

● Do NOT hardcode secrets.
● Do NOT commit secrets.
● Use automation.
● Leverage Open Source tools that already exist!
THANK YOU!

Dwayne McDaniel
dwayne.mcdaniel@gitkraken.com
www.gitkraken.com
Twitter: @mcdwayne

Questions???