puppet

Secret Management in the world of Infrastructure as Code

DevOpsDays Amsterdam 2017

Peter Souter

Technical Account Manager | Puppet
@petersouter

Who am I?

@petersouter

petems
IRC/Slack/GitHub

Technical Account Manager

6 years using Puppet

2 years @ Puppet Inc

Work with customers on their holistic Puppet Program

Help customers get the best use of Puppet

Evangelise and work with the community

puppet

I’m super excited to be here

My first ever talk slot at a DevOpsDays!

puppet

I’m slowly hitting all my Tech Talk Ambitions

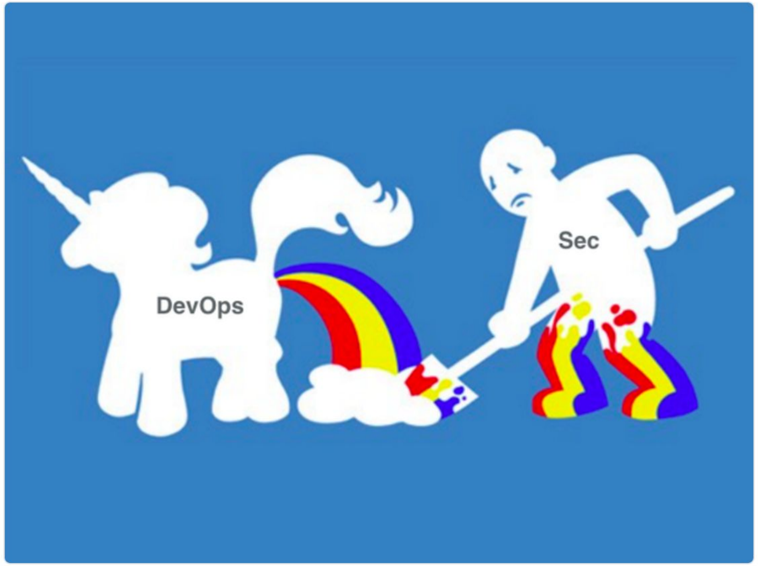
- **Speak at FOSDEM - Done! 2016**
- **Speak at Config Management Camp - Done! 2016**
- **Speak at PuppetConf - Done! 2016**
- **Speak at a DevOpsDays - Done! 2017**
- **Speak at LISA - WIP**
- **Speak at a VelocityConf - WIP**

So what are we here to talk about?

We’re in the Security Slot right? So let's talk security!

puppet

Every time someone uses this picture, Pete Cheslock gets his wings!



https://twitter.com/petecheslock/status/595617204273618944

What are we going to cover?

- What are the risks of leaking secrets in your infrastructure?
- How can prevent leaks from your Infrastructure as code?
- What parts of the DevOps toolchain can help you?
- How do you detect leaks and what can you do when they happen?

https://flic.kr/p/7LcF2W

So what are secrets in IaC?

It’s always good to define something if you’re discussing it

What are secrets in IT?

Small

A few kb at most

Radioactive

Consequences are dire from a leak

Required

The infrastructure won't work without them!

Examples

Passwords, API Keys, SSH Keys, SSL Certs...

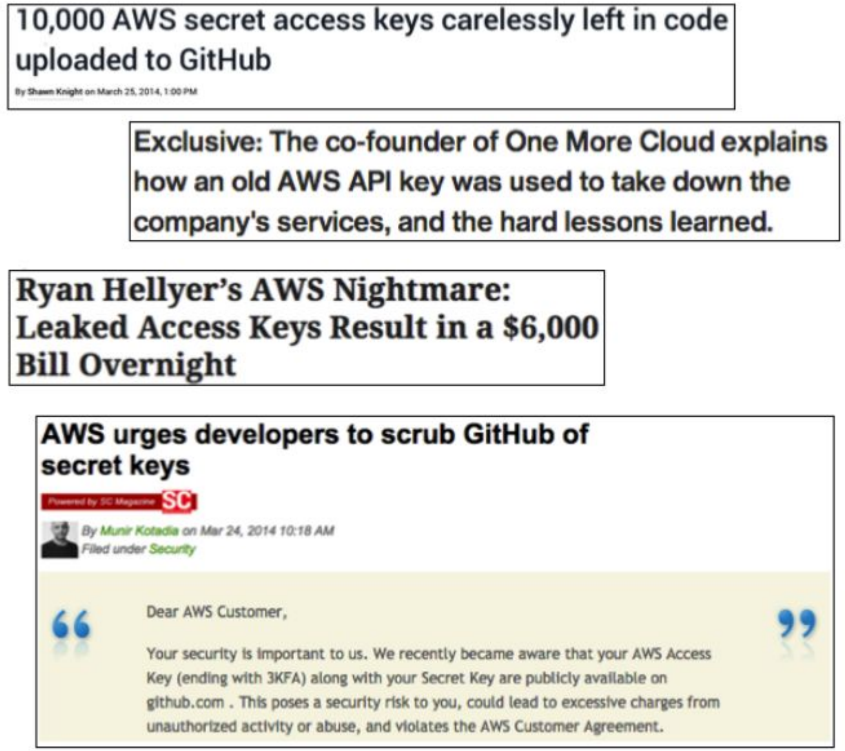
https://flic.kr/p/dHrwpb

The Risks

How bad could it be?

puppet

We’ve all seen things like this...



10,000 AWS secret access keys carelessly left in code uploaded to GitHub

By Shawn Knight on March 25, 2014, 1:00 PM

Exclusive: The co-founder of One More Cloud explains how an old AWS API key was used to take down the company's services, and the hard lessons learned.

Ryan Hellyer’s AWS Nightmare: Leaked Access Keys Result in a $6,000 Bill Overnight

AWS urges developers to scrub GitHub of secret keys

Powered by SC Magazine SC

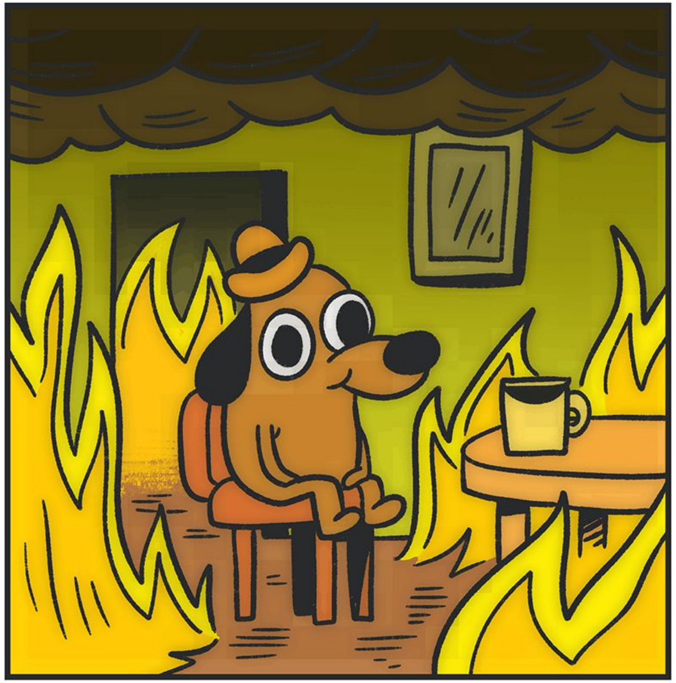
By Munir Kotadia on Mar 24, 2014 10:18 AM
Filed under Security

Dear AWS Customer,

Your security is important to us. We recently became aware that your AWS Access Key (ending with 3KFA) along with your Secret Key are publicly available on github.com . This poses a security risk to you, could lead to excessive charges from unauthorized activity or abuse, and violates the AWS Customer Agreement.

puppet

Worst Case Scenario: Organisational Catastrophe



- Ransom
- Data theft
- Loss of Customers
- Legal and PR fires

Preventing Leaks

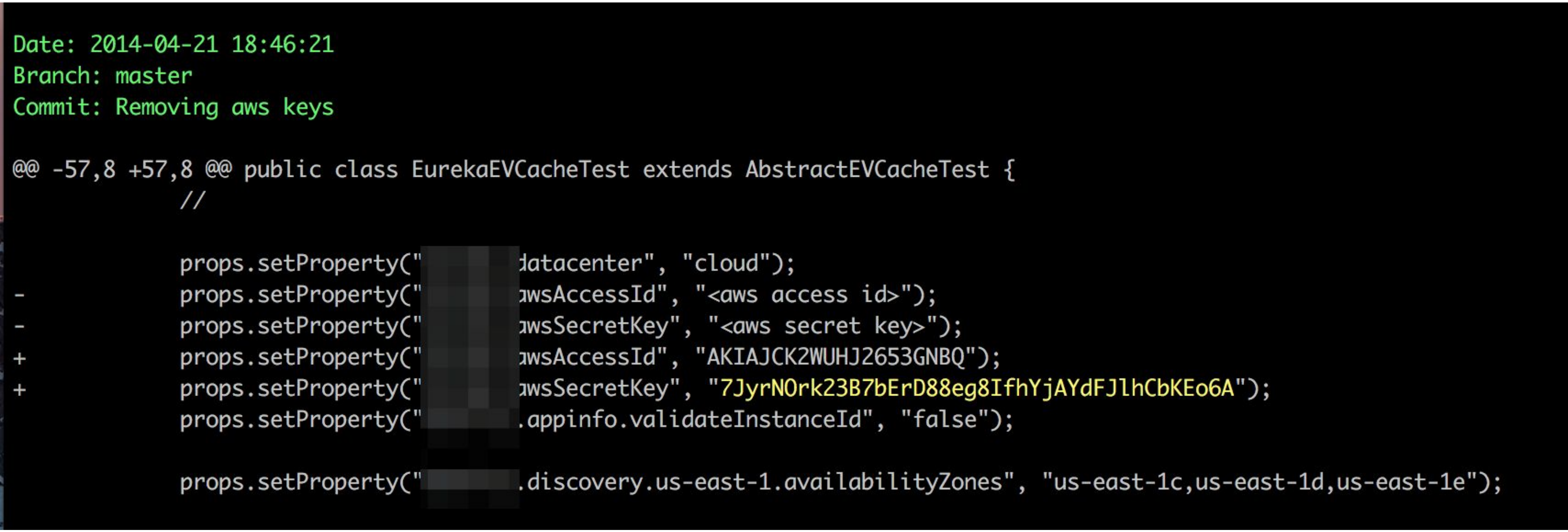
Plugging the holes

puppet

First things first: Remove existing plaintext secrets

Clean up the current codebase and keep it clean

Trufflehog



```
Date: 2014-04-21 18:46:21
Branch: master
Commit: Removing aws keys

@@ -57,8 +57,8 @@ public class EurekaEVCacheTest extends AbstractEVCacheTest {
             //

             props.setProperty("[redacted]datacenter", "cloud");
-            props.setProperty("[redacted]awsAccessId", "<aws access id>");
-            props.setProperty("[redacted]awsSecretKey", "<aws secret key>");
+            props.setProperty("[redacted]awsAccessId", "AKIAJCK2WUHJ2653GNBQ");
+            props.setProperty("[redacted]awsSecretKey", "7JyrNOrk23B7bErD88eg8IfhYjAYdFJlhCbKEo6A");
             props.setProperty("[redacted].appinfo.validateInstanceId", "false");

             props.setProperty("[redacted].discovery.us-east-1.availabilityZones", "us-east-1c,us-east-1d,us-east-1e");
```

puppet

https://github.com/dxa4481/truffleHog

Gittyleaks

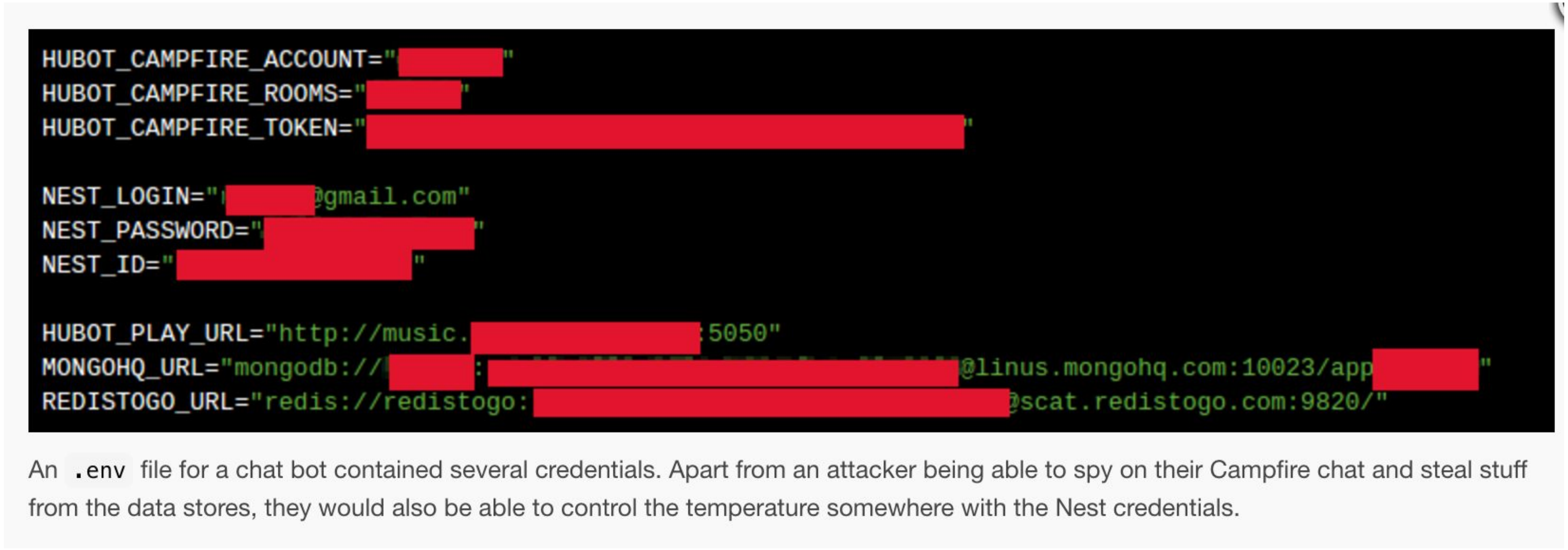
```
--------------------------------------------------------------------
gittyleaks' Bot Detective at work ...
--------------------------------------------------------------------

file: site/profiles/templates/rhn/RHN-ORG-TRUSTED-SSL-CERT.erb
what: Key
value: (2048
match:
    Public-Key: (2048 bit)
num_of_revisions: 59
```

puppet

https://github.com/kootenpv/gittyleaks

GitRob



```
HUBOT_CAMPFIRE_ACCOUNT="[redacted]"
HUBOT_CAMPFIRE_ROOMS="[redacted]"
HUBOT_CAMPFIRE_TOKEN="[redacted]"

NEST_LOGIN="[redacted]@gmail.com"
NEST_PASSWORD="[redacted]"
NEST_ID="[redacted]"

HUBOT_PLAY_URL="http://music.[redacted]:5050"
MONGOHQ_URL="mongodb://[redacted]:[redacted]@linus.mongohq.com:10023/app[redacted]"
REDISTOGO_URL="redis://redistogo:[redacted]@scat.redistogo.com:9820/"
```

An `.env` file for a chat bot contained several credentials. Apart from an attacker being able to spy on their Campfire chat and steal stuff from the data stores, they would also be able to control the temperature somewhere with the Nest credentials.

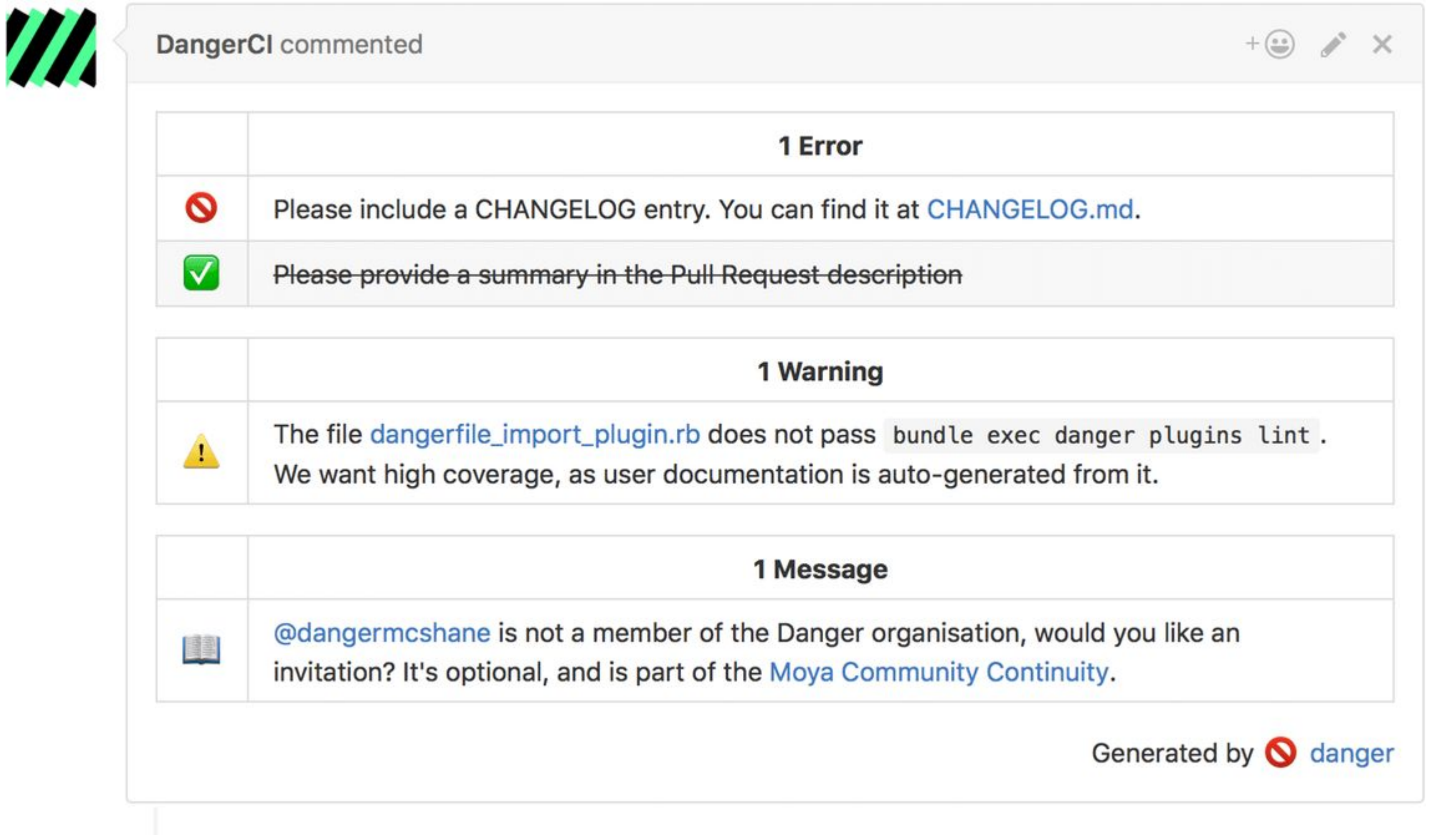
Manual Grepping

```
$ git grep -i -e
"(api\\|key\\|username\\|user\\|pw\\|password\\|pass\\|email\\|mail
)" -- `git ls-files | grep -v .html` | cat
```

puppet

Build pipelines are super useful for preventing the re-introduction of leaks

Danger.systems


DangerCI commented

	1 Error
🚫	Please include a CHANGELOG entry. You can find it at CHANGELOG.md.
✅	~~Please provide a summary in the Pull Request description~~

	1 Warning
⚠️	The file dangerfile_import_plugin.rb does not pass `bundle exec danger plugins lint`. We want high coverage, as user documentation is auto-generated from it.

	1 Message
📖	@dangermcshane is not a member of the Danger organisation, would you like an invitation? It's optional, and is part of the Moya Community Continuity.

Generated by 🚫 danger

https://github.com/Netflix/Scumblr

Danger.systems

```
# set the patterns to watch and warn about if they need security review
@S_SECURITY_FILE_PATTERN ||= /Dangerfile|(auth|login|permission|email|twofactor|sudo).*\.py/

...

warn("Changes require @getsentry/security sign-off")
message = "### Security concerns found\n\n"
securityMatches.to_set.each do |m|
    message << "- #{m}\n"
end
markdown(message)
```

puppet

https://github.com/getsentry/sentry/blob/ac8fe045fb161e67140d5d2959381b74f0738dc8/Dangerfile

Then figure out how to protect those secrets

Encryption, architectural changes or moving to a secret service

Most Infrastructure as Code tools have a separate data layer

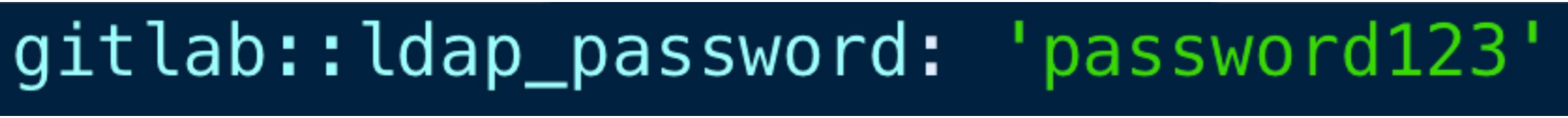
Puppet uses Hiera as a data layer

```
gitlab::gitlab_rails_config:
  ldap_enabled: true
  ldap_servers:
    acmeldapserver:
      label: 'acme LDAP'
      host: 'ldap.acme.net'
      port: 389
      uid: 'uid'
      method: 'plain'
      bind_dn: 'UID=puppetmaster,OU=System,OU=Accounts,DC=acme,DC=net'
      password: 'puppetmaster'
      active_directory: false
      allow_username_or_email_login: false
      block_auto_created_users: false
      base: 'OU=People,OU=Accounts,DC=acme,DC=net'
      user_filter: '(|(description=Systems Administrator)(description=Systems Developer)(description=Manager))'
```

puppet

Bad!

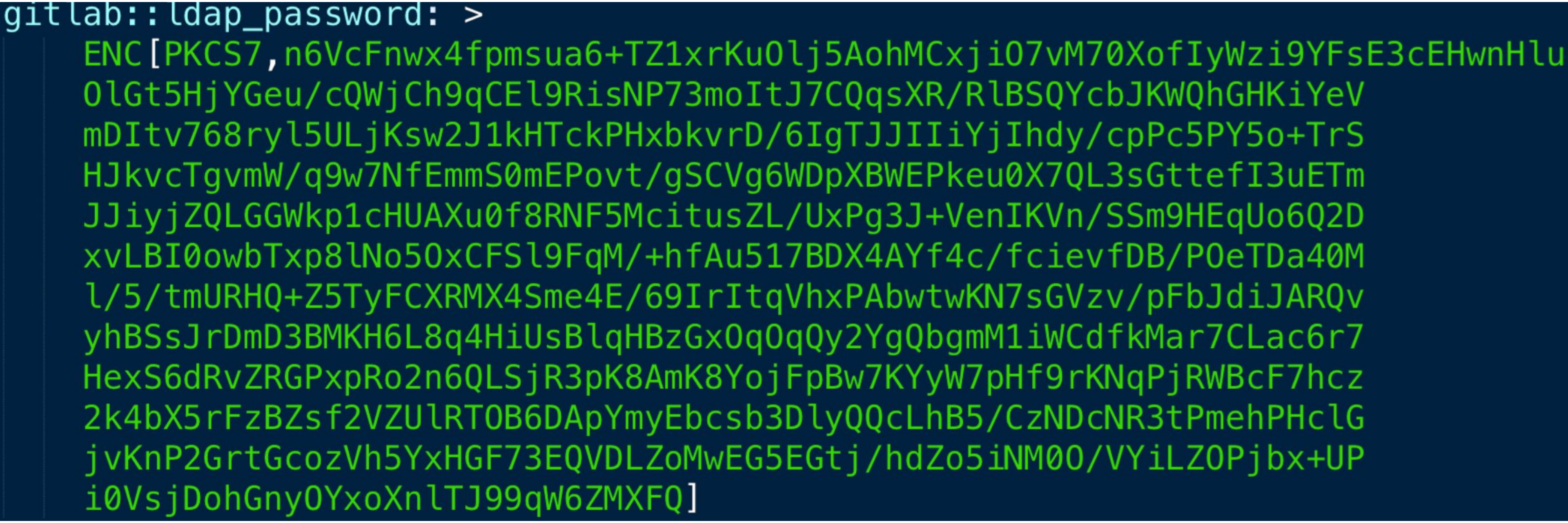
Plaintext :(



```
gitlab::ldap_password: 'password123'
```

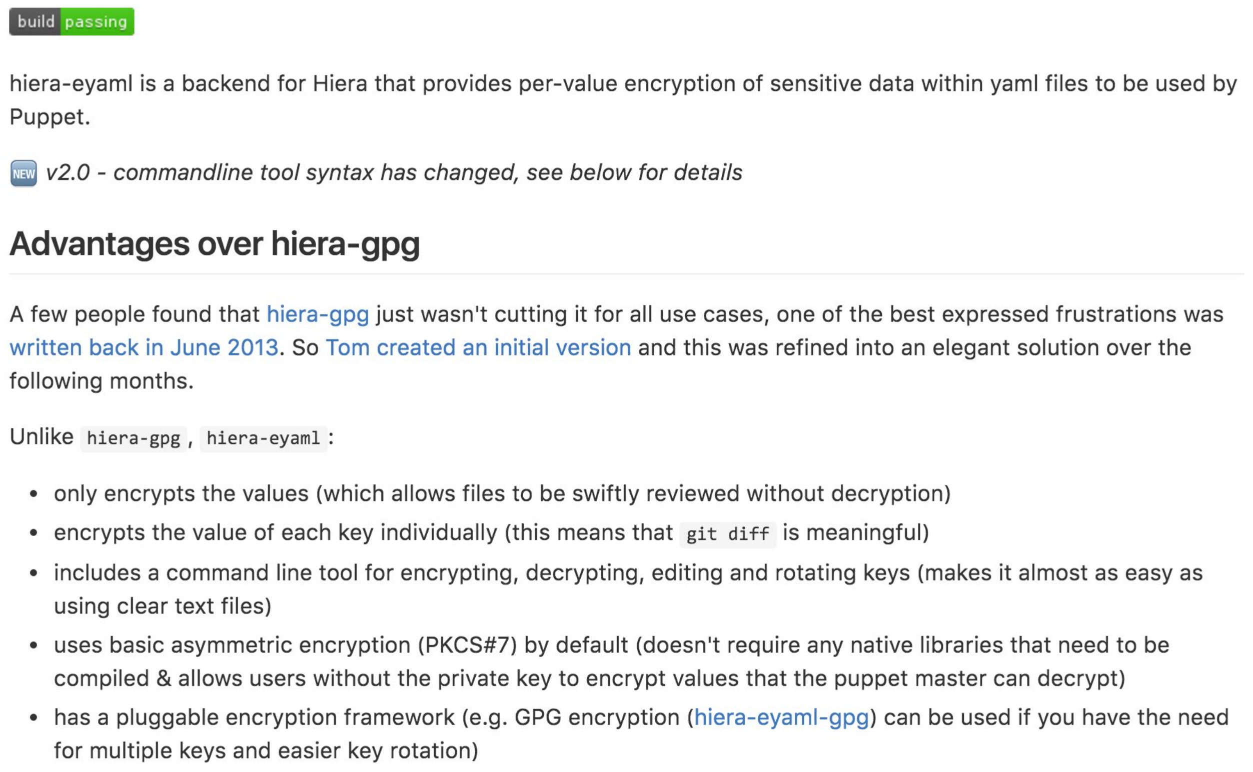
Good!

Encrypted :D



```
gitlab::ldap_password: >
    ENC[PKCS7,n6VcFnwx4fpmsua6+TZ1xrKuOlj5AohMCxjiO7vM70XofIyWzi9YFsE3cEHwnHlu
    OlGt5HjYGeu/cQWjCh9qCEl9RisNP73moItJ7CQqsXR/RlBSQYcbJKWQhGHKiYeV
    mDItv768ryl5ULjKsw2J1kHTckPHxbkvrD/6IgTJJIIiYjIhdy/cpPc5PY5o+TrS
    HJkvcTgvmW/q9w7NfEmmS0mEPovt/gSCVg6WDpXBWEPkeu0X7QL3sGttefI3uETm
    JJiyjZQLGGWkp1cHUAXu0f8RNF5McitusZL/UxPg3J+VenIKVn/SSm9HEqUo6Q2D
    xvLBI0owbTxp8lNo5OxCFSl9FqM/+hfAu517BDX4AYf4c/fcievfDB/POeTDa40M
    l/5/tmURHQ+Z5TyFCXRMX4Sme4E/69IrItqVhxPAbwtwKN7sGVzv/pFbJdiJARQv
    yhBSsJrDmD3BMKH6L8q4HiUsBlqHBzGxOqOqQy2YgQbgmM1iWCdfkMar7CLac6r7
    HexS6dRvZRGPxpRo2n6QLSjR3pK8AmK8YojFpBw7KYyW7pHf9rKNqPjRWBcF7hcz
    2k4bX5rFzBZsf2VZUlRTOB6DApYmyEbcsb3DlyQQcLhB5/CzNDcNR3tPmehPHclG
    jvKnP2GrtGcozVh5YxHGF73EQVDLZoMwEG5EGtj/hdZo5iNM0O/VYiLZOPjbx+UP
    i0VsjDohGnyOYxoXnlTJ99qW6ZMXFQ]
```

hiera-eyaml



hiera-eyaml is a backend for Hiera that provides per-value encryption of sensitive data within yaml files to be used by Puppet.

🆕 *v2.0 - commandline tool syntax has changed, see below for details*

Advantages over hiera-gpg

A few people found that hiera-gpg just wasn't cutting it for all use cases, one of the best expressed frustrations was written back in June 2013. So Tom created an initial version and this was refined into an elegant solution over the following months.

Unlike `hiera-gpg`, `hiera-eyaml`:

- only encrypts the values (which allows files to be swiftly reviewed without decryption)
- encrypts the value of each key individually (this means that `git diff` is meaningful)
- includes a command line tool for encrypting, decrypting, editing and rotating keys (makes it almost as easy as using clear text files)
- uses basic asymmetric encryption (PKCS#7) by default (doesn't require any native libraries that need to be compiled & allows users without the private key to encrypt values that the puppet master can decrypt)
- has a pluggable encryption framework (e.g. GPG encryption (hiera-eyaml-gpg) can be used if you have the need for multiple keys and easier key rotation)

Theoretically, you should be able to release the of the code you write publically without any sort of security issues

This is actually a tenet of 12 Factor Apps...

Apps sometimes store config as constants in the code. This is a violation of twelve-factor, which requires strict separation of config from code. Config varies substantially across deploys, code does not.

A litmus test for whether an app has all config correctly factored out of the code is whether the codebase could be made open source at any moment, without compromising any credentials.

Note that this definition of “config” does not include internal application config, such as config/routes.rb in Rails, or how code modules are connected in Spring. This type of config does not vary between deploys, and so is best done in the code.

Example: GDS

Government Digital Service, UK

Meeting the Digital Service Standard

To meet point 8 (understand security and privacy issues) you must:

- Make all new source code open and reusable
- Publish code under an appropriate licence
- Explain your reasoning for any code you haven’t made open

You’ll have to explain how you did this at your service assessments.

Meeting the Digital Service Standard

When GOV.UK was first set up we were unable to publish our Puppet repository because our code and secrets were tied together. This goes against patterns like the 12-factor app which “requires strict separation of config from code”

> “A litmus test for whether an app has all config correctly factored out of the code is whether the codebase could be made open source at any moment, without compromising any credentials.”

This wasn’t true for our Puppet repository, but we gradually moved our credentials into a separate repository (rotating them as we did so).

Check code for unique strings that look secret-y

```
$ strings modules/**/*.pp | tr ' '
  '\n' | sort -n | uniq | view -
```

Note: Requires zsh for the strings function!

It’s not just the code!

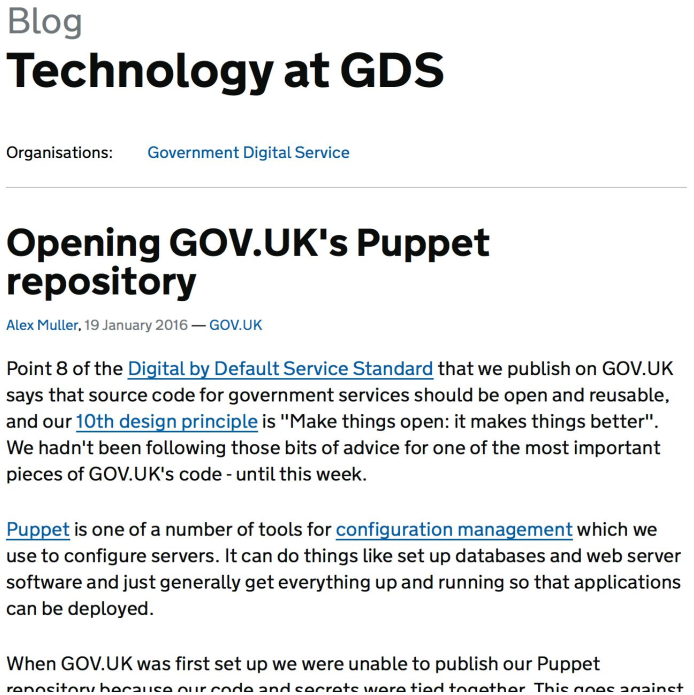
Git commits can contain sensitive data

```
$ git commit -a -m "Changed the
password to password1"
```

Manually searching through git commits for sensitive information...

```
$ while read line; do echo $line;
git --no-pager log -p -S $line; done
< puppet_search
```

Want to know more?



Blog
Technology at GDS

Organisations: Government Digital Service

Opening GOV.UK's Puppet repository

Alex Muller, 19 January 2016 — GOV.UK

Point 8 of the Digital by Default Service Standard that we publish on GOV.UK says that source code for government services should be open and reusable, and our 10th design principle is "Make things open: it makes things better". We hadn't been following those bits of advice for one of the most important pieces of GOV.UK's code - until this week.

Puppet is one of a number of tools for configuration management which we use to configure servers. It can do things like set up databases and web server software and just generally get everything up and running so that applications can be deployed.

When GOV.UK was first set up we were unable to publish our Puppet repository because our code and secrets were tied together. This goes against

Opening GOV.UK's Puppet Repository
https://gdstechnology.blog.gov.uk/2016/01/19/opening-gov-uks-puppet-repository/

Git Repo https://github.com/alphagov/govuk-puppet

The Toolchain

What existing tooling can be used to help?

Command Line Encryption

- **Can be operationally difficult, not always designed with config management in mind**
- **Key rotation is still a PITA**
- **Big trend right now for cool companies to write encryption and secret handling apps in Go: YMMV on this...**

Examples: GPG, mozzila/sops, Shopify/ejson

Secret Servers: Why?

- Dynamic secrets
- ACL (Access control policies)
- Leasing and renewal
- Revocation
- Encryption
- Auditing
- Supportability

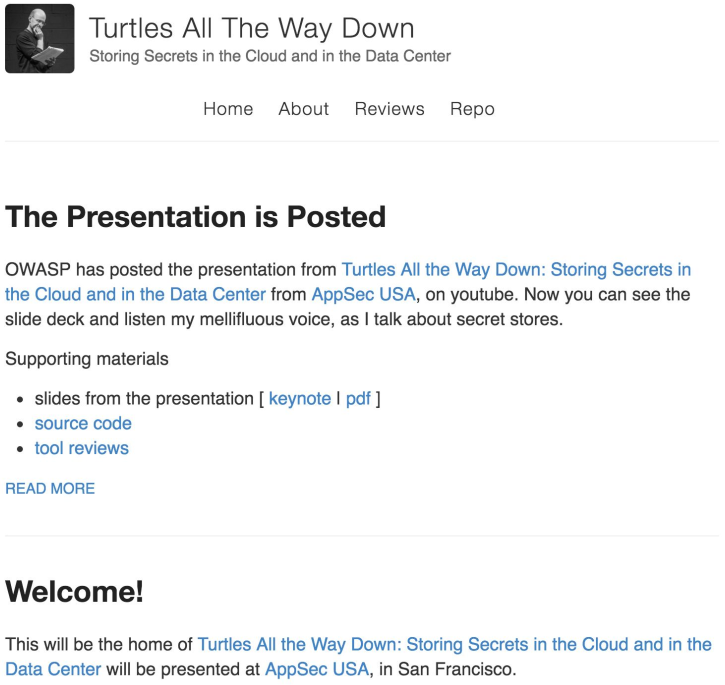
Examples: Vault, Conjur, Keywhiz, Confidant, CyberArk

Cloud Native Secret Services

- **AWS: KMS**
- **GCE: KMS**
- **Azure: Key Vault**
- **Openstack: Barbican**

VCS based encryption

- Transcrypt
  Git-Crypt
  Blackbox
- High operational overhead
- Encrypting files, not data
- Good Summary: Turtles All The Way Down: Storing Secrets in the Cloud and in the Data Center



Turtles All The Way Down
Storing Secrets in the Cloud and in the Data Center

Home About Reviews Repo

The Presentation is Posted

OWASP has posted the presentation from Turtles All the Way Down: Storing Secrets in the Cloud and in the Data Center from AppSec USA, on youtube. Now you can see the slide deck and listen my mellifluous voice, as I talk about secret stores.

Supporting materials

- slides from the presentation [keynote I pdf]
- source code
- tool reviews

READ MORE

Welcome!

This will be the home of Turtles All the Way Down: Storing Secrets in the Cloud and in the Data Center will be presented at AppSec USA, in San Francisco.

http://danielsomerfield.github.io/turtles

https://www.youtube.com/watch?v=OUSvv2maMYI

Detecting leaks and reacting

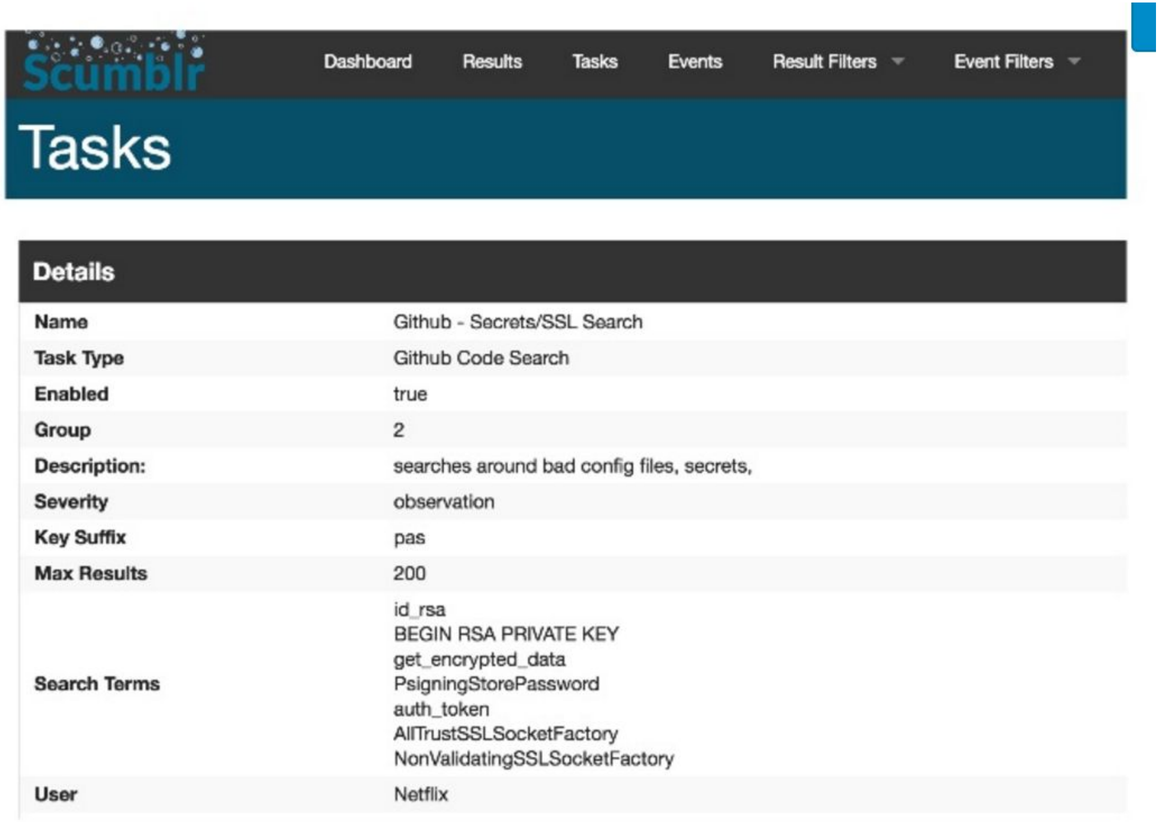
How to keep your head when everyone's losing theirs

puppet

Generic procedure upon the detection of leaked credentials

- Roll new keys and reset passwords
- Monitor systems for intrusive behaviour
- Recreate machines from base
- Keep track of actions for post-mortem

Scumblr



Tasks

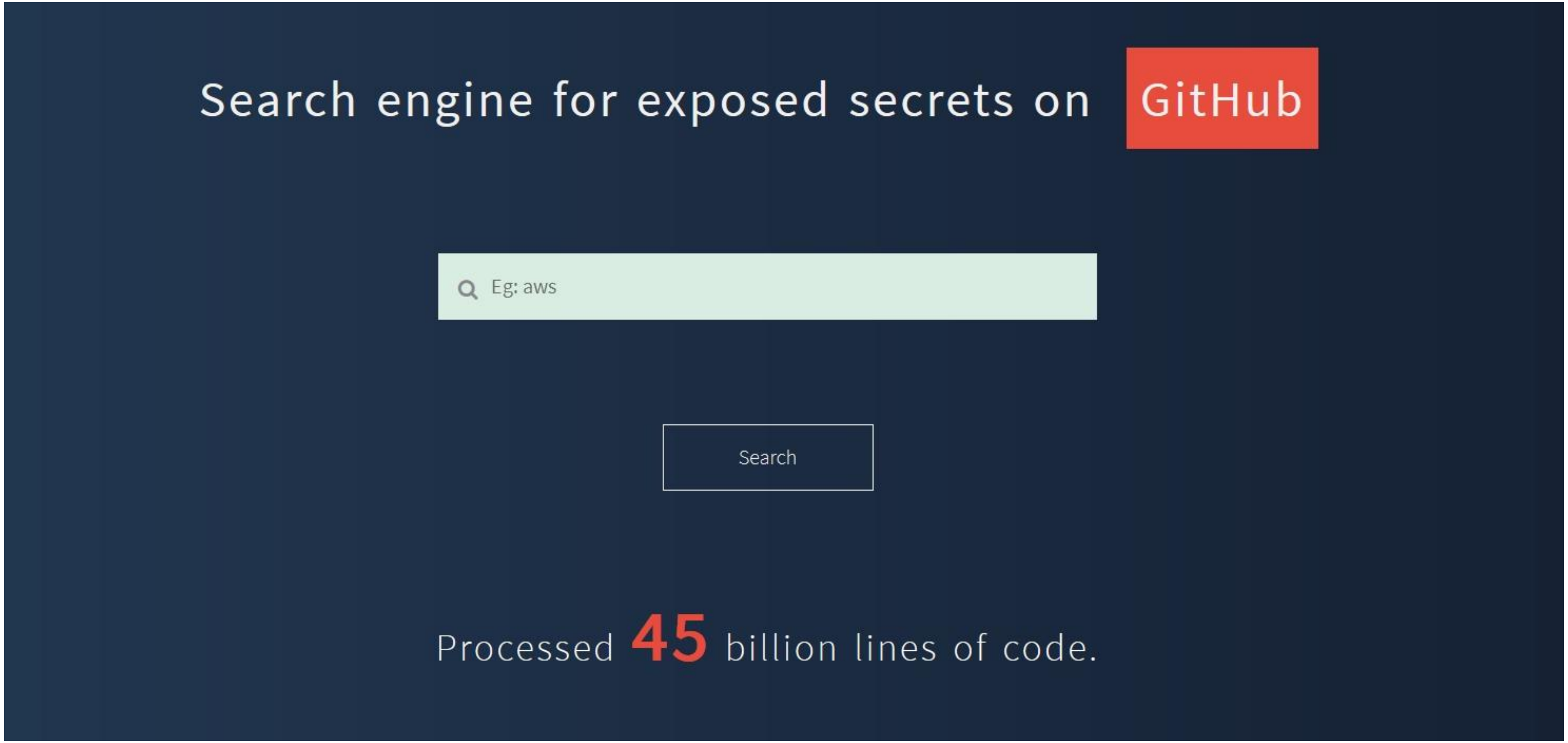
Details

Name	Github - Secrets/SSL Search
Task Type	Github Code Search
Enabled	true
Group	2
Description:	searches around bad config files, secrets,
Severity	observation
Key Suffix	pas
Max Results	200
Search Terms	id_rsa BEGIN RSA PRIVATE KEY get_encrypted_data PsigningStorePassword auth_token AllTrustSSLSocketFactory NonValidatingSSLSocketFactory
User	Netflix

puppet

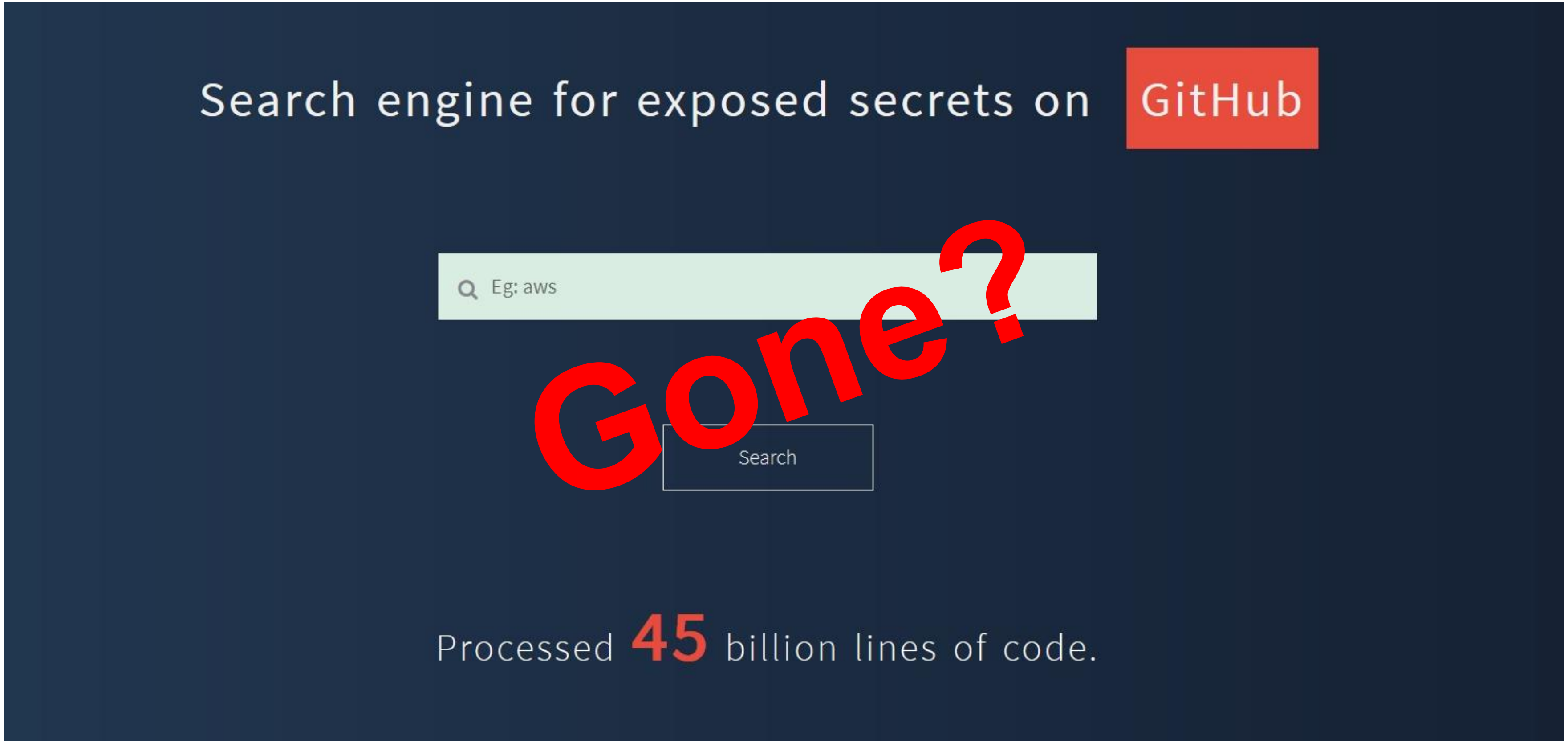
https://github.com/Netflix/Scumblr

Gitleaks.com


Search engine for exposed secrets on GitHub

Processed 45 billion lines of code.

Gitleaks.com


Search engine for exposed secrets on GitHub

Processed 45 billion lines of code.

Unfortunately, there’s no silver bullet to detect leaked secrets

A lot of it is about monitoring and metrics, gating and reviews

Outliers and anomalies are what to look for

It’s largely a people and process problem

Who here has a HIDS system operating?

Credential gets leaked → Unusual activity logged and alerted → Blue team goes out and fixes things

Making sure security is part of your workflow, rather than an afterthought

“Shift security left”

Shifting left

“security must **“shift left,”** earlier into design and coding and into the automated test cycles, instead of waiting until the system is designed and built and then trying to fit some security checks just before release”

- **DevOpsSec: Delivering Secure Software Through Continuous Delivery, Jim Bird**

How do we pro-actively guard against secrets being leaked?

- **Game days and internal evil attempt teams**
- **Continuous security integration (CI tests/code-review)**
- **Dedicated security stories for sprints**
  - Evil users or (mis)use cases
  - https://www.owasp.org/index.php/Application_Threat_Modeling
- **Embedded security team members**
- **Pentests - internal and external**

Game Day example: Agent spoofing

Let's say someone gets access to an agent machine.

What’s the worst they can do?

What information can they fetch?

What passwords do they have locally?

What can they detect remotely?

Game Day example: Laptop theft

Give someone a standard workstation

Are your workstation FDE?

What credentials are on the average machine?

How much damage can they do?

How long does it take to be detected?

Summary

What have we learnt?

Leaking things is bad

Consequences are dire

Start by removing plaintext secrets

Make sure the code is clean enough to be released

puppet

Make sure the data is kept secret

With tooling that fits with your workflows and architecture

Ensure that those secrets are kept secret

People, processes and automated testing

puppet

Know what to do when things go wrong

Runbooks, workflows, game day trainings and such

puppet

Move security left

Make it a part of your process, rather than an afterthought

Want to know more?

- **Behind Closed Doors - Managing Passwords in a Dangerous World -** Noah Kantrowitz
  https://coderanger.net/talks/secrets/
- **Turtles All The Way Down Storing Secrets in the Cloud and in the Data Center -** Daniel Somerfield
  http://danielsomerfield.github.io/turtles
- **Secrets and LIE-abilities: The State of Modern Secret Management -** Jeff Nickoloff
  https://medium.com/on-docker/secrets-and-lie-abilities-the-state-of-modern-secret-management-2017-c82ec9136a3d
- **Detecting and Mitigating Secret-Key Leaks in Source Code Repositories -**
  https://people.eecs.berkeley.edu/~rohanpadhye/files/key_leaks-msr15.pdf
- **Infrastructure Secret Management Software Overview**
  https://gist.github.com/maxvt/bb49a6c7243163b8120625fc8ae3f3cd

Q&A