TIPS TOOLS PRIVACY

SECURITYTRAILS BLOG · JUN 04 · SECURITYTRAILS TEAM

Top GitHub Dorks and Tools Used to Scan GitHub Repositories for Sensitive Data

Reading time: 12 minutes

For years, using Github as your main repository for application development version control has been the industry standard. Even in the presence of many competitors, Github still stands as the number one option.

Millions of developers push code changes to GitHub several times in a single day, and those changes can be overwhelming when you're working simultaneously with distributed dev teams 'round the clock. Troubles arise when developers accidentally include password credentials, usernames or cloud-based keys in their public and private repos.

TABLE OF CONTENTS

Latest academic studies

Top 4 ways to scan GitHub repos for credentials

- GitHub dorks
- Gitrob
- Repo security scanner

TruffleHog
How can I prevent my sensitive information from being exposed in GitHub?
Git-secrets
Git-Hound
GitHub security best practices
Final thoughts

Unfortunately, remote attackers are aware of this. They've identified GitHub as an easy place to find exposed sensitive information. We've seen this happen to many companies — notably in the Uber GitHub data-leak case, when AWS notified customers to review their repos for exposed data, as well as in the Slack tokens exposure incident.

In the same way that Google dorks can be used to scan websites for sensitive data, we will now explore the top ways to scan GitHub for critical data such as usernames and passwords, database credentials, API data, cryptographic keys, etc., so you can detect these security issues before the bad guys do.

Latest academic studies

Earlier this year, researchers Michael Meli, Matthew R. McNiece and Bradley Reaves from North Carolina State University released a white-paper containing the results of their massive, full GitHub scan of the critical files contained in around 100k GitHub accounts.

After scanning millions of GitHub accounts in a six-month period, they began analyzing the results — and found a lot more exposed data than they ever imagined.

Text strings containing usernames, passwords, API tokens, configuration files, database snapshots and cryptographic keys were publicly accessible over GitHub.

Fig. 01 courtesy of Michael Meli, Matthew R. McNiece, and Bradley Reaves. North Carolina State University.

The results are stunning and include a full in-depth investigation, which is a must-read for all who want to protect any access data in pushed code.

Now that we've read their study, we're going to reproduce a small part of their reconnaissance work, to show our readers just how easy it is for anyone to access information seemingly concealed in GitHub.

Top 4 ways to scan GitHub repos for credentials

To protect your critical credentials, you first need to understand how remote attackers can access your data. Let's look at the most popular methods of scanning public GitHub repositories for credentials, tokens and keys.

GitHub dorks

This is the oldest and most traditional way to access sensitive data from public repositories, and because it's a part of public sources, it can be included in any OSINT research.

What's needed? Only GitHub account and some basic knowledge about programming variables; in other words, the things you'll be looking for, such as database user and password, secret access keys, tokens, etc.

Here at SecurityTrails, we wanted to see how much time an average user takes to locate sensitive data from public repositories, so we tested a few random sensitive keywords such as:

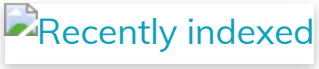
- password
- dbpassword
- dbuser
- access_key
- secret_access_key
- bucket_password
- redis_password
- root_password

It took less than 60 seconds to find the first results of GitHub repositories leaking data about database connections — including usernames and passwords — as you can see below:

There's an even more effective approach to take, by searching for specific credentials strings in configuration files, such as this example: `filename:sftp-config.json password`.

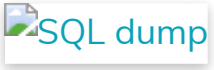
As shown, we were able to obtain the IP address, root password and SSH port credentials.

Another tip that may increase your success in finding leaked data is ordering the results, by setting the GitHub filter to 'Recently indexed', as you see below:



What about smtp login credentials? It's easy by searching for strings like: `filename:.env MAIL_HOST=smtp.gmail.com`. Output example:

How about SQL dumps? Simply use `extension:sql mysql dump`, and the results will appear before your very eyes:



There are numerous GitHub dorks that can be used to scan GitHub repositories, including:

- `filename:.npmrc _auth`
- `filename:.dockercfg auth`
- `extension:pem private`
- `extension:ppk private`
- `filename:id_rsa or filename:id_dsa`
- `extension:sql mysql dump`
- `extension:sql mysql dump password`
- `filename:credentials aws_access_key_id`
- `filename:.s3cfg`
- `filename:wp-config.php`
- `filename:.htpasswd`
- `filename:.env DB_USERNAME NOT homestead`
- `filename:.env MAIL_HOST=smtp.gmail.com`
- `filename:.git-credentials`

And the list goes on. A Google search will show you even more results.

As we are not legally allowed to connect to these servers, we didn't tried to access any of these remote systems or services. However, we certainly believe most of the critical information seen in the results to be accurate, as the information was recently indexed by GitHub. We believe that if a real attacker can access this, he can definitely connect to most of these systems.

Is there any way to automate the GitHub dork exploration process? There are tools available that may help you automate Github dorks queries, such as github-dork.py, a basic python tool that can search through your entire

GitHub repo and test against dorks specified in a text file.

Gitrob

Gitrob is very useful for collecting information during a security audit or reconnaissance task. This OSINT tool works by scanning the public repos hosted at GitHub. Once you've configured your GitHub API (which requires activation of a valid account), it can be used to get information such as user names, passwords, keys and other critical data from an organization.

It works by cloning any user repo, exploring the commit history, and reporting files with certain known signatures that may match with potentially sensitive content.

Installing Gitrob

Installing Gitrob isn't that complex if you follow the official documentation.

First, make sure you have Go 1.8 or newer installed, and that you have created your own GitHub personal token.

Then, grab the source, and install all its dependencies and the core by running:

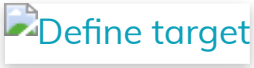
```
go get github.com/michenriksen/gitrob
```

Once that's done, export your GitHub access token by running:

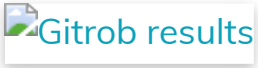
```
export GITROB_ACCESS_TOKEN=your.token.here
```

Alternatively, you can download the pre-compiled binaries that work really well on most scenarios. Once you extract the .zip or .tar.gz file, merely execute the binary file and you are ready to play!

Define your target, as in this example:



This might take a few seconds, depending on how much information is found. Once it's finished, you'll be able to browse the results from the web interface:



Just as when you run a manual GitHub dork query, it will yield some false positives. Our best advice< grab a coffee and start exploring the results, one by one.

There are several other Gitrob options to explore, by using the `--help` option:

Repo security scanner

Repo security scanner is a command line-based tool that was written with a single goal: to help you discover GitHub secrets that developers accidentally made by pushing sensitive data. And like the others, it will help you find passwords, private keys, usernames, tokens and more.

To test it, you'll need to download the latest release from here. Extract the compressed file, and move the binary to any bin directory inside your OS, such as /usr/bin or /usr/local/bin.

Run it against any repo's history and wait for the results.

The syntax is simple:

```
git log -p | scanrepo
```

This tool's only requirement is that you need to download any repos you want to investigate first. Once that's done, it will scan your repos easily and within seconds, as you'll see in the following video:

0:00

TruffleHog

TruffleHog searches through GitHub repositories and digs through the commit history and branches, looking for accidentally committed secrets.

Installation is fairly easy. Its only requirement is to have pip previously installed. Then, you can run:

```
pip install truffleHog
```

Expected output:

```
[20:41]root@securitytrails(2):sectrails[0]# pip install truffleHog
Collecting truffleHog
Using cached https://files.pythonhosted.org/packages/a3/6d/a251a89aae727260c32491a52ff01fcb2002141f455e9e9fd9c794824b5c/truffleHog-2.0.99-py2.py3-none-any.whl
Requirement already satisfied: truffleHogRegexes==0.0.7 in /usr/lib/python2.7/site-packages (from truffleHog) (0.0.7)
```

```
Requirement already satisfied: GitPython==2.1.1 in /usr/lib/python2.7/site-packages (from truffleHog) (2.1.
1)
Requirement already satisfied: gitdb2>=2.0.0 in /usr/lib/python2.7/site-packages (from Git Python==2.1.1->t
ruffleHog) (2.0.5)
Requirement already satisfied: smmap2>=2.0.0 in /usr/lib/python2.7/site-packages (from gitdb2>=2.0.0->GitPy
thon==2.1.1->truffleHog) (2.0.5)
Installing collected packages: truffleHog
Successfully installed truffleHog-2.0.99
[20:41]root@securitytrails(2):sectrails[0]#
```

Once that's completed, you can start playing with this tool by using this syntax:

```
trufflehog --regex --entropy=False [https://github.com/user/repo][18]
```

As an example, we'll now explore trufflehog default repo, which already contains "secrets" that have been exposed for testing purposes:

```
truffleHog --regex --entropy=False https://github.com/dxa4481/truffleHog.git
```

In the following test, we ran this tool against a real repository. This time it was EVCache, a memcached system for AWS EC2 services:

```
trufflehog --regex --entropy=False https://github.com/Netflix/EVCache
```

Now you'll see some interesting stuff, like 'Generic Secrets', as well as items related to AWS keys:

How can I prevent my sensitive information from being exposed in GitHub?

Any Cloud services (AWS, Google Cloud, Microsoft Azure, DigitalOcean, etc) that access keys and password credentials are like treasure troves for the bad guys.

Once they get hold of your access data, they can use it to access private information about your company, clients and app source code; they can shut down part of your infrastructure and use your cloud resources. As you can see, it's the worst-case scenario.

Is there any way to prevent this from happening? While there's no way to control 100% of the human errors made by development teams (such as pushing code with hardcoded passwords or keys), there are manual methods (GitHub dorks) and tools to help you detect these types of accidents.

There are also tools created specifically to help developers. Let's take a look at a few of them.

Git-secrets

Git-secrets is a handy tool written by the AWS team to prevent developers from sharing confidential keys publicly. This utility will help you approve or deny changes before pushing them to your repo.

Installing Git-secrets is different with each operating system. You'll always do best to follow the official documentation.

Once you have Git-secrets installed, you'll need to add the AWS rules for auditing secret keys:

```
git secrets --register-aws
```

Move to the git-secrets directory and run scan:

```
git secrets --scan
```

This will help you identify any private keys exposed in your repo.

Git-Hound

Git-Hound is a great alternative to Git-secrets. This cli-based utility is basically a Git plugin written in Go, one that helps developers prevent secrets from being committed into their repositories. For this, Git-Hound uses sniffing technology against certain commits using regular expressions.

This plugin will try to find matches for regular expressions specified in a separate file called .githound.yml. Once it does find something, it will show a warning before allowing the commit, or execute a fail command and stop the commit right away.

Git-Hound can detect sensitive information such as user credentials, access tokens, and even configuration files and system file names.

Installing Git-Hound

The easiest way to install Git-Hound is by using a pre-compiled binary that can be downloaded from here and requires no dependencies at all.

Next, the typical way to run Git-Hound is by sniffing changes since the last commit, and passing to git commit once you're sure it's clean.

```
git hound commit
```

Alternatively, you can also sniff your entire repo history by running:

```
git log -p | git hound sniff
```

You can run git-hound in many other ways, take a few minutes to explore other options:

```
[10:44]root@securitytrails(2):Downloads[0]# ./git-hound_linux_amd64 --help
Usage of ./git-hound_linux_amd64:
-bin string
Executable binary to use for git command (default "git")
-config string
Hound config file (default ".githound.yml")
-no-color
Disable color output
-v Show version
```

All the tools we've explored can work for both sides; for blue and read teams, to prevent or to attack targets by discovering secrets. So if you're on a blue team, keep that in mind and start using these tools to scan your own repos

from a defensive point of view.

Apart from using manual and automated scans against your repos, the best prevention comes from following recommended security practices.

GitHub security best practices

A few key practices can definitely help you avoid security issues, or if it's too late for that, they can quickly resolve issues and prevent them in the future. Take note and be prepared:

- Review your code, always: this will help you identify bad security practices by any of your employees.
- Clear your GitHub history, to safeguard you most sensitive information.
- Use ENV variables to store critical information in CI/CD. Tools like Vault are one of the top suggestions for these cases.
- If you're certain you've already exposed data, make sure to invalidate tokens and passwords.
- Configure a 2FA for all your GitHub accounts, as this type of data-leak can not only affect public repositories, but private ones too, if attackers gain access by brute forcing or through other aggressive methods.
- Write and publish a disclosure policy in your SECURITY.md file.
- Never allow your company devs to share GitHub credentials with anyone.
- As soon as employees no longer work for your company, be sure to revoke all their accesses.

You should also take a look at GitHub's advice about removing sensitive data from a repository, and check out this interesting research by IBM: Detecting and Mitigating Secret-Key Leaks in Source Code Repositories.

Final thoughts

While GitHub has been working hard to prevent data leaks from their repositories (lately including the capability to detect tokens from Cloud providers as well as SSH credentials, and more), after reading this post it's clear that there's a lot more to be done to offer real protection against human factors.

We really hope this information gives you a better idea about how easy it is to extract sensitive data from GitHub repositories, while encouraging DevOps teams to follow GitHub best security practices. It takes a concerted effort to avoid exposing sensitive information that may lead to massive data leaks or remote intrusions into databases or systems.

The reality is you can't blame GitHub if any of your employees — or yourself — expose critical information to the public. It's your organization's responsibility to make sure your DevOps and engineering teams take the necessary steps to make sure sensitive data remains protected while working with any version control systems.

Are you concerned about exposing sensitive data from your online apps?

There is much more to explore by using SurfaceBrowser™, our all-in-one enterprise-security product designed to help you discover unseen sensitive exposed data about your domain names, SSL certificates, IP addresses, DNS records and open ports in your server infrastructure. Book a demo with our Sales team today!

Sign up for our newsletter!

Email

name@company.com

Subscribe

‹ PREVIOUS NEXT ›

Related Posts

DNS Hijacking: How to Identify ar

Learn about DNS Hijacking, different ty
ways to identify and protect yourself fr

Cybersecurity Red Team Versus Blue Team — Main Differences Explained

We've previously explored the Top 20 OSINT Tools available, and today we'll go through the list of top-used Kali Linux software.

Become Yours: Cybersecurity

ata, and the way it's stored and
uccessful business.

PRODUCTS

DNS History
API
API Pricing
API Documentation
Feeds

COMPANY

Blog
Our Story
Careers
Contact us
Product Manifesto

RESOURCES

Domain Stats
Data Bounty Program
Integrations
Fortune 500 Domains
Developer Hub
Service Status

LATEST FROM OUR BLOG

· NEW DNS Enumeration: Top DNS Recon Tools and Techniques
· 5 Subdomain Takeover #ProTips
· CMS Detector: What CMS a Website is Using and the Best Tools to Find ...
· ASN Lookup Tools, Strategies and Techniques

· What is Reverse DNS? Top Tools for Performing a Reverse DNS Lookup