GitHub dorking for beginners: How to find more vulnerabilities using GitHub search

BY BLACKBIRD-EU · JULY 13, 2025 · LAST UPDATED ON JULY 28, 2025

Bug bounty hunters who spend time in content discovery and reconnaissance are always rewarded well for their efforts, as they often come across untested and hidden assets or endpoints. GitHub dorking is another way to leverage public search engines to discover hidden assets, endpoints and even secrets to increase your chances of finding vulnerabilities. This article is a guide specifically for beginners with no prior knowledge about using GitHub to find vulnerabilities.

Let's dive in!

What is GitHub dorking?

Most organisations today deploy custom code to launch applications, services and other types of software products to keep the company up and running. The standardised way to deploy code is through a platform that keeps track of file changes and provides a distributed version control functionality.

GitHub is one of those many available platforms that provides this out of the box, including public code repositories. Whenever code is pushed to these public repositories, it becomes available for anyone to view. Often, you'll find that unnecessary data and even secrets are accidentally pushed to these public repositories.

In this article, we want to focus on finding these secrets while also fetching any other valuable information that could benefit us during our reconnaissance phase.

Basic search operators

Before we dive into the complex search queries that can easily yield vulnerabilities, let's grab the basics first and take a look at the most essential operators that GitHub search supports. You can always navigate to the official full list of code search syntax on GitHub.

Keyword search

The most common search query you'll want to perform is by narrowing down your search query to a specific organisation, a member of an organisation or a keyword (such as your target domain).

For instance, let's suppose we only want to list code that is linked to a specific organisation (our bug bounty target). In that case, we'd want to make use of the `org` search operator:

```
org:"E Corp"
```

This search filter will only include results that are linked to your specified organisation.

We can do the same in case we have a username of a possible employee. The following search filter will only include results that are linked to a specific GitHub user:

```
user:"hunter2"
```

Usually, what we want to achieve is chaining multiple operators to construct our search query. This will help us narrow down to exactly the type of results we want to be examining while filtering out all the rest.

To do so, we can make use of boolean operators. The two main boolean operators are `AND`, and `OR`. Let's construct a simple search query where we want to return results from a specific organisation that contains our target domain:

```
org:"intigriti" AND "example.com"
```

You can further narrow down your searches by grouping queries. Suppose you're only interested in code that includes your target domain plus is part of a specific organisation, and you want to filter by specific extensions:

```
(org:"intigriti" AND "example.com") AND (extension:json OR extension:yaml)
```

Excluding specific results

It is quite common to have unwanted data that clutters your results page. To exclude these unwanted results, we can use the `NOT` keyword. Here's an example that excludes a specific keyword that we're not interested in:

```
org:"example" extension:json NOT "dev.example.com" NOT "engineering.example.com"
```

Advanced search operators

Let's take a look at some more advanced search operators that, when used correctly, can help us yield more accurate results.

- `filename:` - Search for specific filenames (useful for filtering for configuration and build files).
- `language:` - Filter by programming language.
- `path:` - Search within specific directory paths.
- `sort:` - Helps sort based on interactions, reactions, comments, created date, relevance, author date, committer date, or recency of the updated items.
- `created:` - Filter by creation date of a file.
- `pushed:` - Filter by recency of a file.
- `archived:` - Include/exclude archived repositories.

More interesting use cases

We've covered all the search syntax. Let's take a look at realistic search operations you might want to try out to find hard-coded secrets and other vulnerabilities using GitHub search. These are only a few examples. We recommend you take the principles behind the outlined examples below to help construct search queries that better fit your target.

Third-party service keys

Many applications integrate with third-party services for payments, email delivery, and other functionality. These services require API keys for authentication, which developers sometimes hard-code directly into their source code. Let's take a look at a few examples.

Finding hard-coded Stripe secret keys

Stripe is a commonly integrated third-party service that handles payments. Developers need to supply both the private and the public key whenever communicating with Stripe's public API. We can narrow down our searches to look for cases where they've been hard-coded:

```
org:"example" ("sk_live_" OR "pk_live_")
```

Finding hard-coded AWS access & secret key

AWS is a well-known cloud-based service provider that provides API access to account for all types of developer use-cases, such as S3 buckets. Authentication to this API is often established with an AWS Access Key and an Access Secret Key. With this information, we can construct the following search query that will pull all code results that match our filters:

```
org:"example" (AWS_ACCESS_KEY_ID OR AWS_ACCESS_SECRET_KEY)
```

Searching for hard-coded OpenAI API key

Most companies today have integrated AI into their core products and services. OpenAI is often selected as the go-to provider for common AI-powered tasks as it provides a wide variety of AI-based services. Authentication to this API is done via an API key. You can use the search syntax below to filter results that have this API key hard-coded in a file:

```
org:"example" /"sk-[a-zA-Z0-9]{20,50}"/
```

Configuration and build files

Sensitive configuration and build files often contain hard-coded secrets. If they're ever deployed with other code, you'll be able to browse these files using the following search query:

```
org:"example" (filename:.env OR filename:.env.local OR filename:travis.yml OR filename:Dockerfile or filename:docker-compose.yaml OR filename:package.json OR filename:web.config OR filename:settings.py)
```

Hard-coded links

GitHub code search can also be used to perform content discovery, from finding untouched subdomains and other assets to finding new input parameters possibly subject to injection attacks.

Here's an example of a simple search query that'd return all hard-coded URLs:

```
org:"example" /http(s)?:\/\//
```

Database connection strings

Database connection strings contain all the necessary information to connect to a database, including usernames, passwords, and hostnames. MongoDB and MySQL are two of the most commonly used databases in web applications, and developers often accidentally commit these connection strings to public repositories.

```
org:"example" ("mongodb://" OR "mongodb+srv://" OR "mysql://")
```

Authentication & security tokens

JWT (JSON Web Token) secrets are used to sign and verify tokens in authentication systems. If these secrets are exposed, attackers can forge valid tokens and bypass authentication entirely. These secrets are often stored in environment variables or configuration files. However, sometimes they end up hard-coded in source code.

```
org:"example" ("jwt_secret" OR "JWT_SECRET" OR "jwtSecret")
```

More common search queries

Here's the complete list of common search queries you could perform to find possible secrets, files and other hard-coded references that could lead to vulnerabilities:

```
org:"example" (AWS_ACCESS_KEY_ID OR AWS_ACCESS_SECRET_KEY)                  # Hard-coded AWS access & secret key
org:"example" ("sk_live_" OR "pk_live_")                                    # Hard-coded Stripe secret keys
org:"example" (SENDGRID_API_KEY OR sendgrid_api_key)                        # SendGrid API keys
org:"example" /"sk-[a-zA-Z0-9]{20,50}"/                                     # Hard-coded OpenAI API key
org:"example" (ANTHROPIC_API_KEY OR anthropic_api_key)                      # Anthropic API keys
org:"example" (PAYPAL_CLIENT_SECRET OR paypal_client_secret)                # PayPal credentials
org:"example" (SQUARE_ACCESS_TOKEN OR square_access_token)                  # Square payment tokens
org:"example" (AZURE_CLIENT_SECRET OR AZURE_CLIENT_ID)                      # Azure credentials
org:"example" (CLOUDFLARE_API_TOKEN OR CF_API_TOKEN)                        # Cloudflare tokens
org:"example" (filename:.env OR filename:.env.local OR filename:travis.yml) # Configuration and build files
org:"example" /http(s)?:\/\//                                               # Hard-coded links
org:"example" ("mongodb://" OR "mongodb+srv://" OR "mysql://")              # Database connection strings
org:"example" ("jwt_secret" OR "JWT_SECRET" OR "jwtSecret")                 # Authentication & security tokens
org:"example" (extension:pem OR extension:key OR extension:p12 OR extension:pfx)  # Certificate files
org:"example" (SLACK_BOT_TOKEN OR SLACK_WEBHOOK_URL)                        # Slack integration tokens
org:"example" (GITHUB_TOKEN OR GITHUB_PAT OR GH_TOKEN)                      # GitHub personal access tokens
org:"example" /\/\/(.*\.)?amazonaws\.com/                                   # AWS endpoints
org:"example" /\/\/(.*\.)?firebaseio\.com/                                  # Firebase endpoints
```

Conclusion

GitHub dorking can provide you with an edge and help you discover possible secrets while also helping you expand your initial attack surface. However, you must know what you're looking for and narrow down your searches to avoid browsing through code that's not in our interest.

So, you've just learned something new about using GitHub to find more vulnerabilities... Right now, it's time to put your skills to the test! You can start by practising on vulnerable labs and CTFs or... browse through our 70+ public bug bounty programs on Intigriti and who knows, maybe earn a bounty on your next submission!

START HACKING ON INTIGRITI TODAY