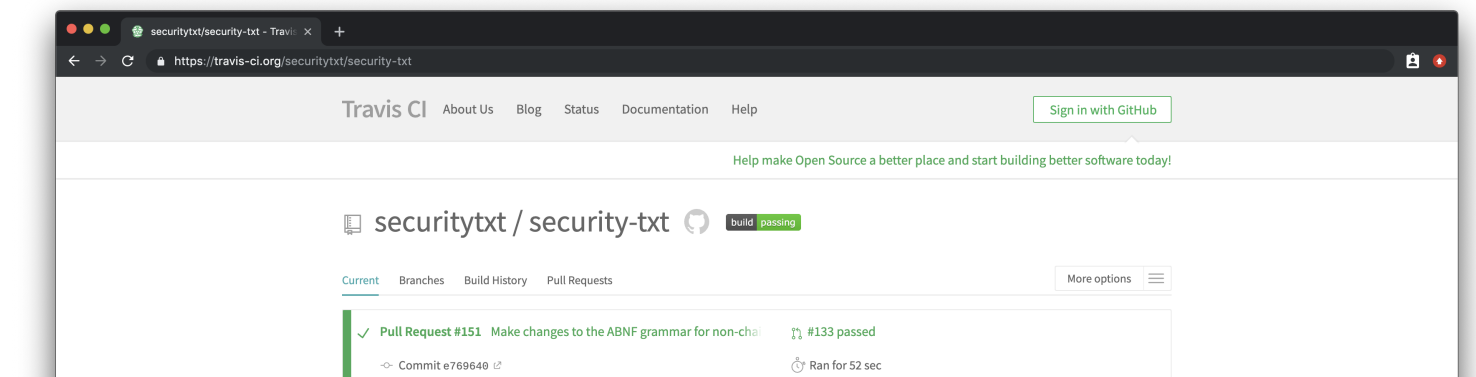


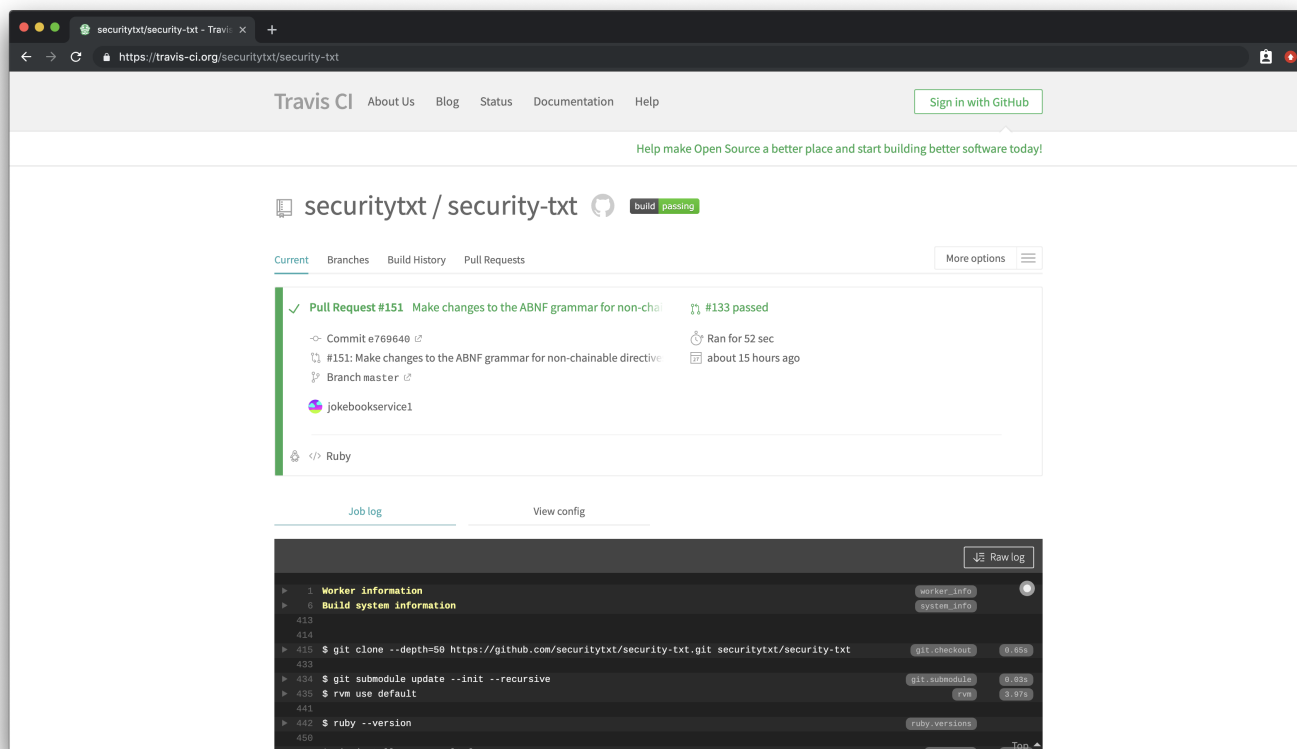
[Home](#)

# "CI Knew There Would Be Bugs Here" — Exploring Continuous Integration Services as a Bug Bounty Hunter

Apr 26, 2019



When it comes to bug bounty hunting and finding exciting areas to explore, it is vital to familiarise yourself with the technologies that vendors and companies rely on. One particularly interesting environment that caught our eye was popular integrations used by various open-source projects, primarily as part of their development life cycle. Some prime examples of continuous-integration services ("CI services") including [Travis CI](#), [Circle CI](#), and [GitLab CI](#) turned out to be extremely rewarding for us as bug bounty hunters.



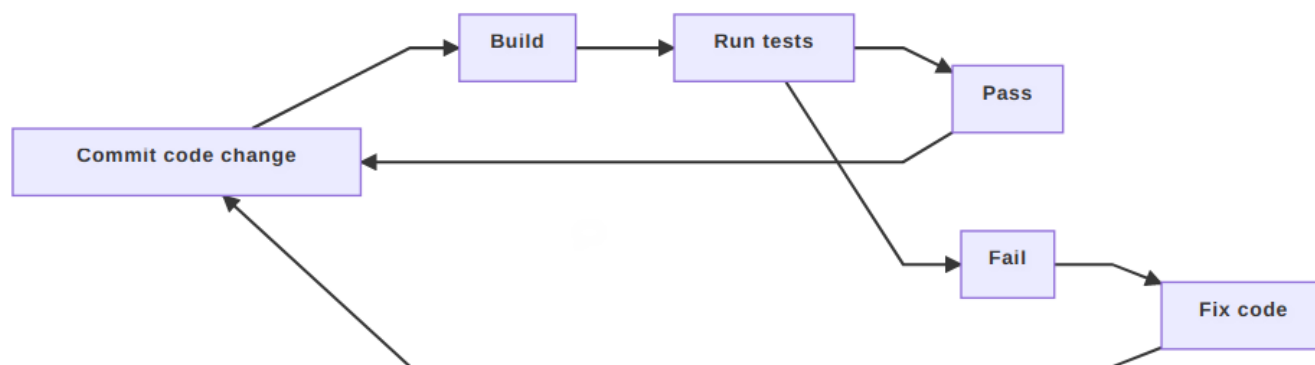
We set out to automate fetching and searching large data sets from these CI services. This technical write-up will touch on the numerous challenges we faced, how we reduced the number of false-positives in our searches, notable findings, and finally, list some tricks we picked up along the way.

The team that worked on this research includes Justin Gardner (@Rhynorater), Corben Leo (@hacker\_), and EdOverflow (@EdOverflow) — with some testing and helpful suggestions by Karim Rahal (@KarimPwnz), @streaak, @d0nutptr, and BBAC.

## Introduction to continuous-integration services

Continuous integration (CI) is the practice of committing code-changes and automatically building and testing every change. Nowadays, it is rare not to stumble across an open-source project that does not use a continuous-integration service at some point in the development cycle. The various

services out there offer straightforward set-up configuration steps and beautiful interfaces for quickly testing and building code continuously.



*/r/dataisugly/ candidate right here.*

The [security.txt project](#), for instance, uses Travis CI to build new draft documents whenever a commit is pushed. This allows the team to quickly determine if any further changes to the specification could potentially break the Internet Draft when compiled using [kramdown-rfc2629](#) — a tool that enables one to write everything in Markdown and then convert it to XML2RFC XML markup.

## What went through our heads

Something that readers have requested in the past is to have a designated section on how the authors came up with research topics such as the type of work we are presenting here. This section will hopefully illustrate where the idea stems from initially.

All the authors of this write-up have lots of experience working on open-source projects on GitHub and have, over the years, learned techniques that simplify the development process for open-source project maintainers. GitHub offers lots of integrations at <https://github.com/marketplace>, where one

particular category stands out: [“Continuous integration”](#). This is where we discovered that due to the way lots of open-source teams strive for complete transparency and openness in the development process, projects were hesitant to hide build log data on continuous-integration platforms. Admittedly, integrations such as Travis CI do offer private profiles as a premium feature on travis-ci.com, but during our research, the vast majority of projects appeared to only use the public instance travis-ci.org — note the “.org” top-level domain.

It is worth noting that continuous-integration services have already been targeted in the past for sensitive information by bug bounty hunters and third-parties as seen in [“A HackerOne employee’s GitHub personal access token exposed in Travis CI build logs”](#) and the [“API under attack”](#) Travis CI incident report:

“We are currently undergoing a distributed attack on our public API that we believe is aimed at revealing GitHub authentication tokens. Countermeasures are holding, and we will update accordingly.”

— Travis CI (Sep. 2015)

So much so that platforms such as Travis CI introduced built-in secrets detection to prevent accidental exposure of sensitive information as seen below. [\[1\]](#)

```
146 Setting environment variables from repository settings
147 $ export SAUCE_USERNAME=[secure]
148 $ export SAUCE_ACCESS_KEY=[secure]
149
```

Travis CI replaces potentially sensitive information with the `[secure]` keyword at runtime.

To prevent leaks made by these components, we automatically filter secure environment variables and tokens that are longer than three characters at runtime, effectively removing them from the build log, displaying the string `[secure]` instead.

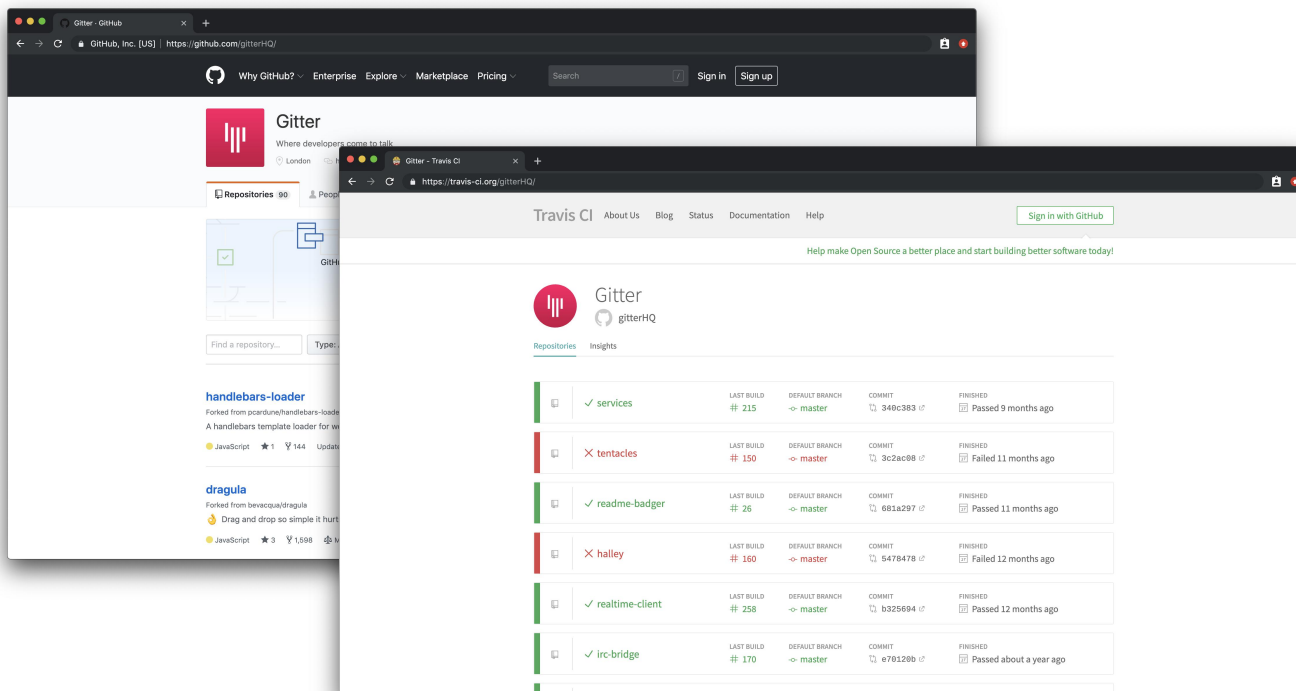
## Automating the boring tasks

Approaching a large attack surface manually, such as the one Travis CI presents, would be an incredibly tedious task. Therefore, we had to work with the available API documentation these CI vendors provide and develop tooling to automate fetching build logs rapidly.

To better illustrate the process of going from a bug bounty program to an extensive data set for further investigation, we will use [Travis CI's API documentation](#) as an example.

The initial phase of our process was to fetch bug bounty programs' GitHub organisations. There were multiple ways of going about doing this, but for the best results, a simple Google search for "company name" and "GitHub" would do the job. Next, we had to check if the GitHub handle was on Travis CI. To make this process smoother, we used a browser bookmarklet that would redirect us from GitHub to Travis CI using the GitHub handle.

```
javascript:window.location="https://travis-ci.org"+window.location.pathname;
```



*The bookmarklet would redirect from the GitHub page to Travis CI.*

If the target was present on GitHub, we hit the projects API endpoint on Travis CI and retrieve a list of all projects.

```
https://api.travis-ci.org/owner/%s/repos?limit=100&offset=%d
```

Travis CI's API is case sensitive; fortunately, the bookmarklet ensures that you are using the correct handle when issuing API requests. To gather the contents of build logs, we need to hit the build IDs API endpoint and then `/log.txt`.

```
https://api.travis-ci.org/repo/%s/builds?limit=100&offset=%d
https://api.travis-ci.org/job/%d/log.txt
```

Now that the contents of all the build logs belonging to the target are stored locally, we can start grepping. Due to the size of the data we were analysing, we resorted to `ripgrep` when sieving through the logs locally.

```
$ rg -ia "$1" -j 12 --no-filename --no-line-number --pretty
```

Besides bug bounty program's GitHub accounts, the authors also gathered build logs belonging to all members of the GitHub organisation. It turns out that some members were running builds on their account not realising that their secrets were being exposed in the build logs.

```
#!/bin/bash

users=$(curl -s -H "application/vnd.github.hellcat-preview+json" -H "Authorization: token $1" https://api.github.com/repos/$2/builds)

while read -r hehe; do
    secretz -t "$hehe";
done <<< "$users"
```

The team behind this project has decided to refrain from publishing any of the tools built to fetch build logs since we do not want to be directly responsible for any disruptions to CI platforms' performance. This write-up will inevitably draw more attention to the large attack surface some CI platforms present; therefore, the authors would like to remind the reader, when using the platform's API, to please take care in not inundating every endpoint with lots of requests at once. We were very cautious not to take down any services during the entire process and would advise others to follow suit.

## Results and notable findings

Overall, the most impactful findings were predominately GitHub access token leaks. In this section, we will cover four notable reports that our team submitted.

While grepping through the Travis CI build logs of a employee's account on a public program, we discovered a GitHub access token with read and write

access to the GitHub organisation. This token would have allowed us to push code to any of the repositories listed under the GitHub organisation. The program awarded us their highest payout to date according to their HackerOne *"Program Statistics"*.

To expand our scope, we considered multiple platforms. In a private Bugcrowd program's Travis CI build log from 2013, we found a GitHub access token and were awarded a P1-severity payout — the highest possible severity score on Bugcrowd. [2]

The most surprising response from all the vendors that we reported findings to was Discourse's bug bounty program. Karim Rahal discovered an employee's GitHub access token with read and write access to all public repositories under the Discourse GitHub organisation. To demonstrate the potential impact of this issue, we pushed a harmless file to one of the organisation's least active repositories to not draw too much attention. The file was subsequently removed from the repository.

discourse / errorpages

Watch 16 Star 1 Fork 1

Code Pull requests 0 Insights

delete frog.txt

master

Browse files

majakomel committed on Feb 19 Verified 1 parent 6cdd0fc commit 09dc71dc6e20b192a54d9ede0e3ff2c08aa3e400

Showing 1 changed file with 0 additions and 1 deletion. Unified Split

1 frog.txt View file

```
... .. @@ -1 +0,0 @@
1 - Frogs find bugs.
... ..
```

0 comments on commit 09dc71d

Discourse awarded Karim the lowest possible bounty of \$128. We requested further clarification as to how the team determined the bounty amount, but we



have yet to hear back from the Discourse team — that is 60 days without a response. [3]

Another critical bug was discovered on a public cryptocurrency program on HackerOne. This program was using secret variables within Travis CI to create an SSH key. The details for this configuration can be found [here](#) and [here](#). After digging through thousands of logs, the following line was detected by a tool Justin Gardner wrote:

```
-----BEGIN RSA PRIVATE KEY-----
```

A developer had added `cat deploy_key` in their CI configuration file which outputted the SSH key in the log. With the SSH key, an attacker could have logged into several deployment servers in the program's infrastructure. A bounty of \$1000 was awarded because the servers were non-production.

## Tips and tricks

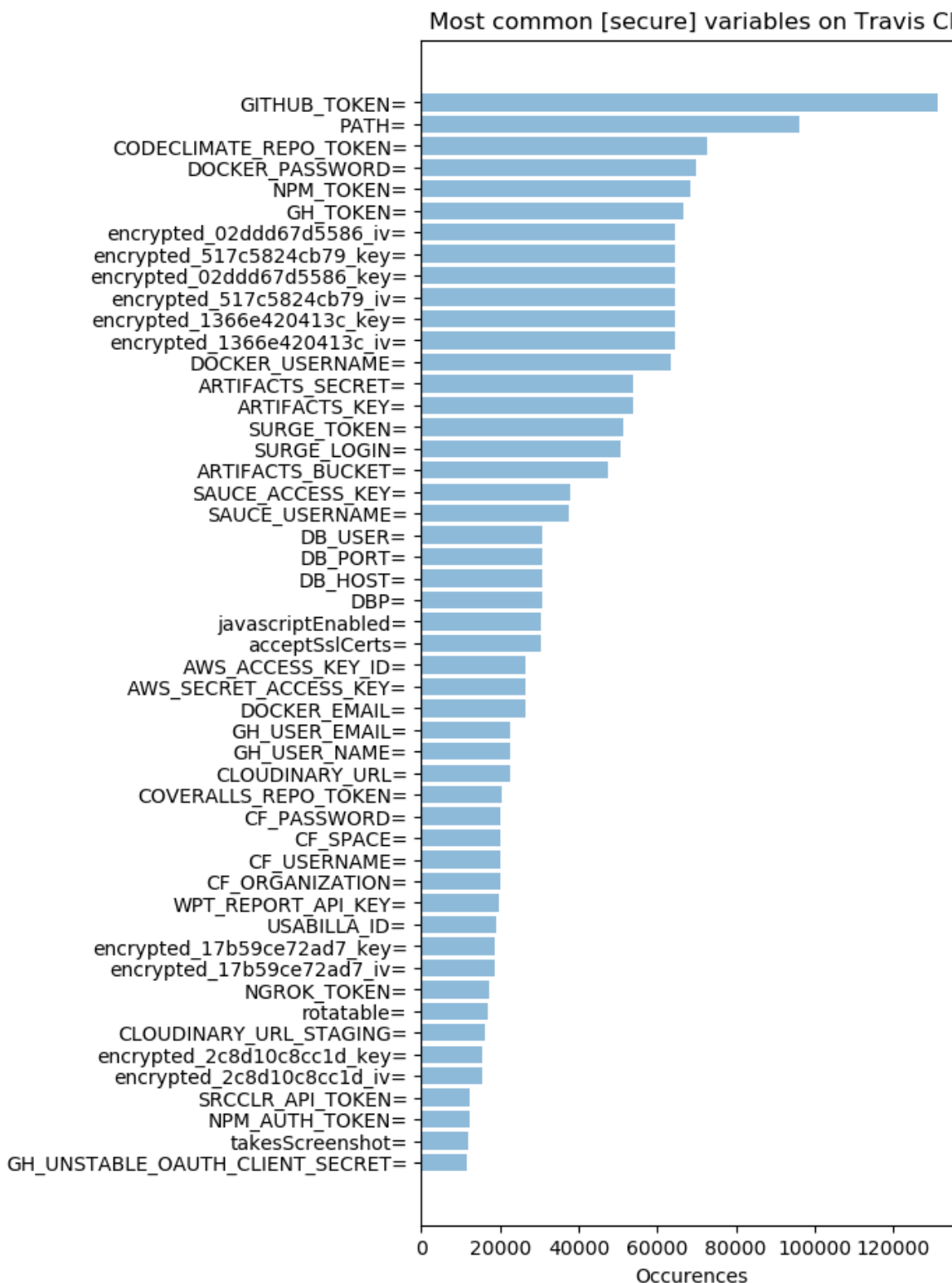
Usually, a simple grep for `export` statements in the build logs would be a good starting point. The `export` command is used for setting environment variables in the log prompt and therefore can expose sensitive information.

```
$ rg -ia "export " -j 12 --no-filename --no-line-number --pretty
```

Of course one should not restrict themselves solely to variables set using the `export` command; refining search terms to “token”, “key”, “password”, and “secret” can help uncover specific leaks. To reduce the number of false positives, we recommend appending `=` and `:` to your search terms.

We encourage readers to create a list of all variables with the `[secure]` keyword and then search using those variable names in all projects. This will

help you find unsecured instances of sensitive data using common variable naming conventions. Karim Rahal gathered `[secure]` variables from 5,302,677 build logs, the 50 most common of which can be seen below.



Full list can be found [here](#).

In addition, setting up continuous monitoring of your favourite bug bounty program's CI builds, and running your tooling every time the team pushes a new commit to GitHub is a great way to catch exposed secrets in real time before the team has time to act.

Do not restrict yourself to keys and tokens; CI platforms are a great source of information for reconnaissance too. Sieve through the logs to find hidden endpoints and URLs belonging to the target.

Check for CI config files on GitHub to determine what CI integration your target is using. There may be other CI platforms out there that were not covered in this write-up where secrets are exposed.

A fun task that we included in our grep process was to find strings and errors messages commonly associated with missing or broken dependencies. With missing npm packages, this can sometimes lead to code execution by claiming the package name on a remote registry as demonstrated in <https://hackerone.com/reports/399166>. Some example error messages include:

- "is not in the npm registry." ([npm](#))
- "No matching distribution" ([PyPI](#))
- "Could not find a valid gem" ([RubyGems](#))

## Conclusion and further research

This research has helped us get a better understanding of the large attack surface that continuous-integration services present — almost hidden in plain sight — and has turned out to be extremely fruitful when bug bounty hunting.

Since a reasonably limited amount of platforms were included in this research, future studies and projects could consider covering further CI platforms and

integrations.

We applaud platforms such as Travis CI that allow users to hide sensitive environment variables in their logs. In our view, this is a step in the right direction to preventing the type of security leaks that we encountered.

Not only has this work provided us with lots of successful bug bounty stories and valid findings, but it has also shown that collaboration, as seen here with this project, can go a long way while bug bounty hunting. ■



## Support my work

If you enjoy reading my write-ups and would like to support my work, please check out my "Buy me a coffee" page.

 Buy me a coffee

---

Subscribe to my newsletter 

## Get updates on new blog posts and regular bug bounty tricks & tips.

Email Address

Subscribe