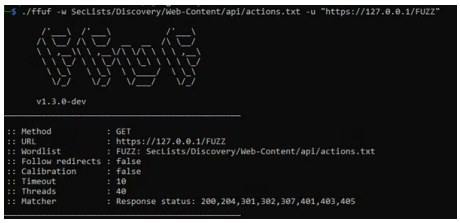
Hacker tools: FuFF (Fuzz Faster u Fool)

BY ANNA HAMMOND · APRIL 20, 2021 · LAST UPDATED ON MARCH 6, 2025

Time is money, and certainly when it comes to bug bounty! Good tools can help you find bugs before others do – but only if you know how to properly use them.
We will be reviewing some of our favourite open-source tools and providing you with some tips and tricks on how to use them. The first tool we're covering is FuFF by Finnish hacker @joohoi.

As the name describes, FFuF is a fast web fuzzing tool created in GO.
To understand the program we first need to understand what fuzzing is.
Fuzzing is the automated process of sending random data to an application to find misconfigurations, unexpected behavior, or hidden parameters. FFuF is the fuzzer of choice for lots of researchers these days.

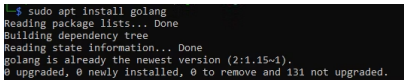


The installation

Installing GoLang:

FFuF is written in GO, a very fast and fairly new language. First, this needs to be installed on your system. This can easily be done with a packet manager. We will use "apt" to install golang. If you want to install it from the website, check out https://golang.org

```
sudo apt install golang
```



Installing FFuF:

FFuF is actively been developed and updated onto their Github page. Go check it out at https://github.com/ffuf/ffuf. To install we will use GO to get the latest version. If you want to install Go from the source, follow the instructions on the git-page.

```
go get -u github.com/fuff/fuff
```

To verify it is installed correct you can check the version with *ffuf -V*

That's all we need to do to install the fuzzing tool.
The binary is installed on *<user>/go/bin/fuff*, If you want it to be available on every directory, you need to add *go/bin* to your **PATH** variable.

Downloading wordlists:

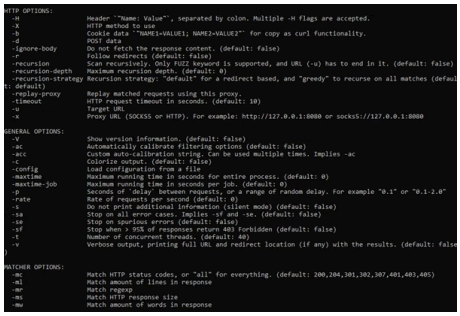
FFuF is a fuzzer, and fuzzers need wordlists or input lists. If you don't have any wordlists we recommend "SecLists" (https://github.com/danielmiessler/SecLists). This is a collection of common wordlist for various purposes.

```
Sudo git clone https://github.com/danielmiessler/SecLists
```

The Basics

Now that we have prepared everything, we can start exploring FFuF. When we run the program without any arguments, we get an overview of all its parameters and a couple of examples. We will explain a few to get you going.

The basics come down to providing a wordlist (**-w**), an URL (**-u**), then put the tag FUZZ where you want the fuzzing to be done.



Directory or File Discovery:

With FFuF we can do directory discovery in a very fast way. There are lots of tools out there that do the same, but FFuF gives some extra benefits that we will discuss later in the article. The most basic command to start directory discovery is the following.

```
fuff -w wordlist -u http://site.com/FUZZ
```

This will replace FUZZ with every word in your wordlist, and give the response.

If we want to go a bit further, we can add extensions to our wordlist. With this, we can discover hidden files as well. Note the **(,)** separated list with the **(.)** in front. Whatever you put in the extension list, will be added to the words in your wordlist. For example, if the wordlist contains “admin” end you put .php in the extension list, this will result in *admin.php* and *admin* to be checked.

```
fuff -w wordlist -u http://site.com/FUZZ -e .php,.zip,.txt
```

An important part of automated fuzzing is the ability to search directories recursively. FFuF has an option when a directory is discovered, it will also search that directory for hidden paths. Important is we can specify recursion depth with **-recursion-depth**. Keep in mind that this can take a lot longer because when a directory is found, FFuF will check every word in the wordlist against the new directory.

```
fuff -w wordlist -u http://site.com/FUZZ -recursion -recursion-depth 2 -e .php,.zip,.txt
```

With the above command, you may get lots of output you don’t want to see. To filter your results, we can add some extra flags to our FFuF command. This lets us only filter HTTP 200 or 302 codes for example with the **-mc 200,302** flag. There are more filters to check out, run ffuf without parameters to check them out.

Authenticated testing:

There is also a possibility to do authenticated testing. Here comes the power of FFuF where other tools may fail. Sometimes the application that needs to be tested is behind a login. For this, we can use the **-b** (cookie) flag.

```
fuff -w wordlist -u http://site.com/FUZZ -b “PHPSESSION=xxxxxxx”
```

Other Interesting parameters

Switch from GET to POST fuzzing:

For now, we have done only fuzzing with GET requests, but it’s also possible to fuzz with POST requests. This comes in handy when we want to fuzz for hidden parameters for example. For this, we need to set the request type with **-X** and the POST data with **-d.**

```
fuff -w wordlist -u http://site.com/FUZZ -X POST -d “search=FUZZ”
```

Delaying between requests:

It is possible a target only allows X number of requests per minute. Often you read in the scope details that the client is only allowing X requests, we can configure this with the **p** parameter. With the **-p** (in seconds) flag we can control the number of requests per second.

```
fuff -w wordlist -u http://site.com/FUZZ -p 2
```

Threading:

Threading is important and also controlling our threads is. The default FFuF thread pool handles 40 threads at the same time. Sometimes we need fewer or more threads. This depends on the target or our own resources. The threading we control with the **-t** flag.

```
fuff -w wordlist -u http://site.com/FUZZ -t 20
```

Conclusion

FFuF is a handy tool for web application testers. You can put in words from a wordlist everywhere you put the word FFUF. We discussed the most common flags to get you going, but there are lots to more discover. The project is actively being developed and has updates regularly. FFuF is an easy-to-understand tool with lots of options. Hope this was useful for you and happy hunting.