ffuf / **ffuf** Public

Code | Issues 163

31 Branches | 27 Tags | Go to file | Add file | Code

joohoi Fixed setting unlimited rat...

de9ac86 · 8 months ago

.github	Fix linter workfl...	2 years ago
_img	More mascot st...	2 years ago
pkg	Fixed setting un...	8 months ago
.gitignore	Fixed behavior ...	4 years ago
.goreleas...	Fix the v2 taggi...	last year
CHANGE...	Fixed setting un...	8 months ago
CONTRIB...	fix csv output fil...	9 months ago
LICENSE	Prepare release ...	3 years ago
README....	Prepare for v2.1...	9 months ago
ffufrc.ex...	Add -raw cli flag...	9 months ago
go.mod	Automatic brotli...	9 months ago
go.sum	Automatic brotli...	9 months ago
help.go	Add -raw cli flag...	9 months ago
main.go	Fix autocalibrati...	9 months ago

About

Fast web fuzzer written in Go

#web #infosec #pentesting #fuzzer

Readme

MIT license

Activity

Custom properties

11.8k stars

158 watching

1.2k forks

Report repository

Releases 25

v2.1.0 Latest
on Sep 16, 2023

+ 24 releases

Sponsor this project

joohoi Joona Hoikkala

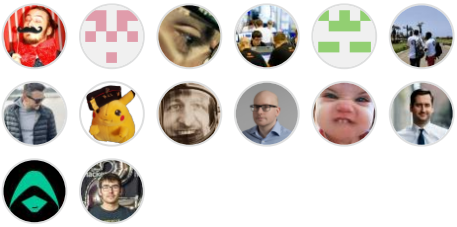
Sponsor

Learn more about GitHub Sponsors

Packages

No packages published

Contributors 51



+ 37 contributors

ffuf - Fuzz Faster U Fool

A fast web fuzzer written in Go.

Installation

- Download a prebuilt binary from releases page, unpack and run!

  or

- If you are on macOS with homebrew, ffuf can be installed with: `brew install ffuf`

  or

- If you have recent go compiler installed: `go install github.com/ffuf/ffuf/v2@latest` (the same command works for updating)

  or

- `git clone https://github.com/ffuf/ffuf ; cd ffuf ; go get ; go build`

Ffuf depends on Go 1.16 or greater.

Example usage

The usage examples below show just the simplest tasks you can accomplish using `ffuf`.

More elaborate documentation that goes through many features with a lot of examples is available in the ffuf wiki at https://github.com/ffuf/ffuf/wiki

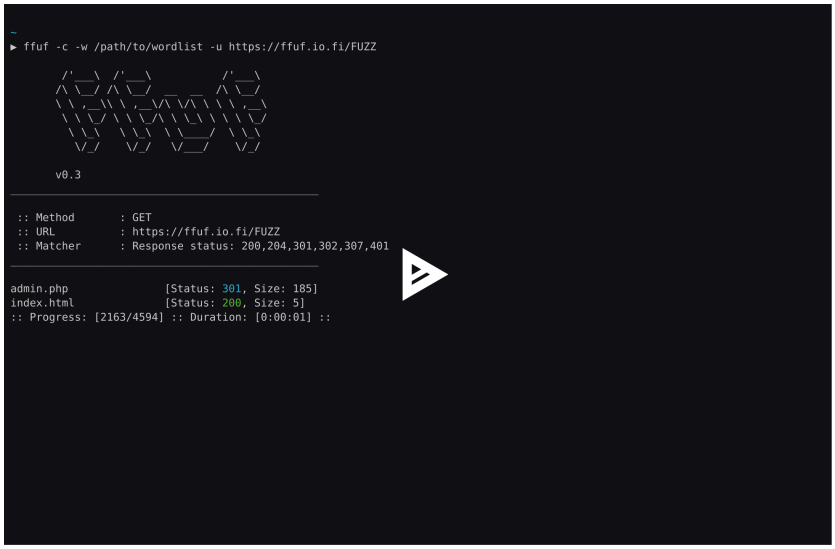
Languages

- **Go** 100.0%

For more extensive documentation, with real life usage examples and tips, be sure to check out the awesome guide: "Everything you need to know about FFUF" by Michael Skelton (@codingo).

You can also practise your ffuf scans against a live host with different lessons and use cases either locally by using the docker container https://github.com/adamtlangley/ffufme or against the live hosted version at http://ffuf.me created by Adam Langley @adamtlangley.

Typical directory discovery



By using the FUZZ keyword at the end of URL (`-u`):

```
ffuf -w /path/to/wordlist -u
https://target/FUZZ
```

Virtual host discovery (without DNS records)

Assuming that the default virtualhost response size is 4242 bytes, we can filter out all the responses of that size (`-fs 4242`)while fuzzing the Host - header:

```
ffuf -w /path/to/vhost/wordlist -u
https://target -H "Host: FUZZ" -fs 4242
```

GET parameter fuzzing

GET parameter name fuzzing is very similar to directory discovery, and works by defining the `FUZZ` keyword as a part of the URL. This also assumes a response size of 4242 bytes for invalid GET parameter name.

```
ffuf -w /path/to/paramnames.txt -u
https://target/script.php?FUZZ=test_value -
fs 4242
```

If the parameter name is known, the values can be fuzzed the same way. This example assumes a wrong parameter value returning HTTP response code 401.

```
ffuf -w /path/to/values.txt -u
https://target/script.php?valid_name=FUZZ -
fc 401
```

POST data fuzzing

This is a very straightforward operation, again by using the `FUZZ` keyword. This example is fuzzing only part of the POST request. We're again filtering out the 401 responses.

```
ffuf -w /path/to/postdata.txt -X POST -d
"username=admin\&password=FUZZ" -u
https://target/login.php -fc 401
```

Maximum execution time

If you don't want ffuf to run indefinitely, you can use the `-maxtime`. This stops **the entire** process after a given time (in seconds).

```
ffuf -w /path/to/wordlist -u
https://target/FUZZ -maxtime 60
```

When working with recursion, you can control the maxtime **per job** using `-maxtime-job`. This will stop the current job after a given time (in seconds) and continue with the next one. New jobs are created when the recursion functionality detects a subdirectory.

```
ffuf -w /path/to/wordlist -u
https://target/FUZZ -maxtime-job 60 -
recursion -recursion-depth 2
```

It is also possible to combine both flags limiting the per job maximum execution time as well as the overall execution time. If you do not use recursion then both flags behave equally.

Using external mutator to produce test cases

For this example, we'll fuzz JSON data that's sent over POST. Radamsa is used as the mutator.

When `--input-cmd` is used, ffuf will display matches as their position. This same position value will be available for the callee as an environment variable `$FFUF_NUM`. We'll use this position value as the seed for the mutator. Files example1.txt and example2.txt contain valid JSON payloads. We are matching all the responses, but filtering out response code `400 - Bad request`:

```
ffuf --input-cmd 'radamsa --seed $FFUF_NUM
example1.txt example2.txt' -H "Content-
Type: application/json" -X POST -u
https://ffuf.io.fi/FUZZ -mc all -fc 400
```

It of course isn't very efficient to call the mutator for each payload, so we can also pre-generate the payloads, still using Radamsa as an example:

```
# Generate 1000 example payloads
radamsa -n 1000 -o %n.txt example1.txt
example2.txt

# This results into files 1.txt ...
1000.txt
# Now we can just read the payload data in
a loop from file for ffuf

ffuf --input-cmd 'cat $FFUF_NUM.txt' -H
"Content-Type: application/json" -X POST -u
https://ffuf.io.fi/ -mc all -fc 400
```

Configuration files

When running ffuf, it first checks if a default configuration file exists. Default path for a `ffufrc` file is `$XDG_CONFIG_HOME/ffuf/ffufrc`. You can configure one or multiple options in this file, and they will be applied on every subsequent ffuf job. An example of ffufrc file can be found here.

A more detailed description about configuration file locations can be found in the wiki: https://github.com/ffuf/ffuf/wiki/Configuration

The configuration options provided on the command line override the ones loaded from the default `ffufrc` file. Note: this does not apply for CLI flags that can be provided more than once. One of such examples is `-H` (header) flag. In this case, the `-H` values provided on the command line will be *appended* to the ones from the config file instead.

Additionally, in case you wish to use bunch of configuration files for different use cases, you can do this by defining the configuration file path using `-config` command line flag that takes the file path to the configuration file as its parameter.

Usage

To define the test case for ffuf, use the keyword `FUZZ` anywhere in the URL (`-u`), headers (`-H`), or POST data (`-d`).

```
Fuzz Faster U Fool - v2.1.0

HTTP OPTIONS:
  -H                  Header `"Name:
Value"`, separated by colon. Multiple -H
```

```
flags are accepted.
  -X                  HTTP method to use
  -b                  Cookie data
`"NAME1=VALUE1; NAME2=VALUE2"` for copy as
curl functionality.
  -cc                 Client cert for
authentication. Client key needs to be
defined as well for this to work
  -ck                 Client key for
authentication. Client certificate needs to
be defined as well for this to work
  -d                  POST data
  -http2              Use HTTP2 protocol
(default: false)
  -ignore-body        Do not fetch the
response content. (default: false)
  -r                  Follow redirects
(default: false)
  -raw                Do not encode URI
(default: false)
  -recursion          Scan recursively.
Only FUZZ keyword is supported, and URL (-
u) has to end in it. (default: false)
  -recursion-depth    Maximum recursion
depth. (default: 0)
  -recursion-strategy Recursion strategy:
"default" for a redirect based, and
"greedy" to recurse on all matches
(default: default)
  -replay-proxy       Replay matched
requests using this proxy.
  -sni                Target TLS SNI, does
not support FUZZ keyword
  -timeout            HTTP request timeout
in seconds. (default: 10)
  -u                  Target URL
  -x                  Proxy URL (SOCKS5 or
HTTP). For example: http://127.0.0.1:8080
or socks5://127.0.0.1:8080

GENERAL OPTIONS:
  -V                  Show version
information. (default: false)
  -ac                 Automatically
calibrate filtering options (default:
false)
  -acc                Custom auto-
calibration string. Can be used multiple
times. Implies -ac
  -ach                Per host
autocalibration (default: false)
  -ack                Autocalibration
keyword (default: FUZZ)
  -acs                Custom auto-
calibration strategies. Can be used
multiple times. Implies -ac
  -c                  Colorize output.
(default: false)
```

```
  -config             Load configuration
from a file
  -json               JSON output, printing
newline-delimited JSON records (default:
false)
  -maxtime            Maximum running time
in seconds for entire process. (default: 0)
  -maxtime-job        Maximum running time
in seconds per job. (default: 0)
  -noninteractive     Disable the
interactive console functionality (default:
false)
  -p                  Seconds of `delay`
between requests, or a range of random
delay. For example "0.1" or "0.1-2.0"
  -rate               Rate of requests per
second (default: 0)
  -s                  Do not print
additional information (silent mode)
(default: false)
  -sa                 Stop on all error
cases. Implies -sf and -se. (default:
false)
  -scraperfile        Custom scraper file
path
  -scrapers           Active scraper groups
(default: all)
  -se                 Stop on spurious
errors (default: false)
  -search             Search for a FFUFHASH
payload from ffuf history
  -sf                 Stop when > 95% of
responses return 403 Forbidden (default:
false)
  -t                  Number of concurrent
threads. (default: 40)
  -v                  Verbose output,
printing full URL and redirect location (if
any) with the results. (default: false)

MATCHER OPTIONS:
  -mc                 Match HTTP status
codes, or "all" for everything. (default:
200-299,301,302,307,401,403,405,500)
  -ml                 Match amount of lines
in response
  -mmode              Matcher set operator.
Either of: and, or (default: or)
  -mr                 Match regexp
  -ms                 Match HTTP response
size
  -mt                 Match how many
milliseconds to the first response byte,
either greater or less than. EG: >100 or
<100
  -mw                 Match amount of words
in response
```

```
FILTER OPTIONS:
  -fc                 Filter HTTP status
codes from response. Comma separated list
of codes and ranges
  -fl                 Filter by amount of
lines in response. Comma separated list of
line counts and ranges
  -fmode              Filter set operator.
Either of: and, or (default: or)
  -fr                 Filter regexp
  -fs                 Filter HTTP response
size. Comma separated list of sizes and
ranges
  -ft                 Filter by number of
milliseconds to the first response byte,
either greater or less than. EG: >100 or
<100
  -fw                 Filter by amount of
words in response. Comma separated list of
word counts and ranges

INPUT OPTIONS:
  -D                  DirSearch wordlist
compatibility mode. Used in conjunction
with -e flag. (default: false)
  -e                  Comma separated list
of extensions. Extends FUZZ keyword.
  -enc                Encoders for
keywords, eg. 'FUZZ:urlencode b64encode'
  -ic                 Ignore wordlist
comments (default: false)
  -input-cmd          Command producing the
input. --input-num is required when using
this input method. Overrides -w.
  -input-num          Number of inputs to
test. Used in conjunction with --input-cmd.
(default: 100)
  -input-shell        Shell to be used for
running command
  -mode               Multi-wordlist
operation mode. Available modes:
clusterbomb, pitchfork, sniper (default:
clusterbomb)
  -request            File containing the
raw http request
  -request-proto      Protocol to use along
with raw request (default: https)
  -w                  Wordlist file path
and (optional) keyword separated by colon.
eg. '/path/to/wordlist:KEYWORD'

OUTPUT OPTIONS:
  -debug-log          Write all of the
internal logging to the specified file.
  -o                  Write output to file
  -od                 Directory path to
store matched results to.
  -of                 Output file format.
```

```
Available formats: json, ejson, html, md,
csv, ecsv (or, 'all' for all formats)
(default: json)
  -or                  Don't create the
output file if we don't have results
(default: false)

EXAMPLE USAGE:
  Fuzz file paths from wordlist.txt, match
all responses but filter out those with
content-size 42.
  Colored, verbose output.
    ffuf -w wordlist.txt -u
https://example.org/FUZZ -mc all -fs 42 -c
-v

  Fuzz Host-header, match HTTP 200
responses.
    ffuf -w hosts.txt -u
https://example.org/ -H "Host: FUZZ" -mc
200

  Fuzz POST JSON data. Match all responses
not containing text "error".
    ffuf -w entries.txt -u
https://example.org/ -X POST -H "Content-
Type: application/json" \
      -d '{"name": "FUZZ", "anotherkey":
"anothervalue"}' -fr "error"

  Fuzz multiple locations. Match only
responses reflecting the value of "VAL"
keyword. Colored.
    ffuf -w params.txt:PARAM -w
values.txt:VAL -u https://example.org/?
PARAM=VAL -mr "VAL" -c

  More information and examples:
https://github.com/ffuf/ffuf
```

Interactive mode

By pressing `ENTER` during ffuf execution, the process is paused and user is dropped to a shell-like interactive mode:

```
entering interactive mode
type "help" for a list of commands, or
ENTER to resume.
> help

available commands:
 afc  [value]             - append to
status code filter
 fc   [value]             - (re)configure
status code filter
 afl  [value]             - append to line
```

```
count filter
 fl   [value]             - (re)configure
line count filter
 afw  [value]             - append to word
count filter
 fw   [value]             - (re)configure
word count filter
 afs  [value]             - append to size
filter
 fs   [value]             - (re)configure
size filter
 aft  [value]             - append to time
filter
 ft   [value]             - (re)configure
time filter
 rate [value]             - adjust rate of
requests per second (active: 0)
 queueshow                - show job queue
 queuedel [number]        - delete a job in
the queue
 queueskip                - advance to the
next queued job
 restart                  - restart and
resume the current ffuf job
 resume                   - resume current
ffuf job (or: ENTER)
 show                     - show results
for the current job
 savejson [filename]      - save current
matches to a file
 help                     - you are looking
at it
>
```

in this mode, filters can be reconfigured, queue managed and the current state saved to disk.

When (re)configuring the filters, they get applied posthumously and all the false positive matches from memory that would have been filtered out by the newly added filters get deleted.

The new state of matches can be printed out with a command `show` that will print out all the matches as like they would have been found by `ffuf`.

As "negative" matches are not stored to memory, relaxing the filters cannot unfortunately bring back the lost matches. For this kind of scenario, the user is able to use the command `restart`, which resets the state and starts the current job from the beginning.