projectdiscovery / **katana** Public

Code · Issues 71

8 Branches · 10 Tags · Go to file · Go to file · Code

ehsandeep Merge pull request #826 from zy9ard3/main · 9ba3bb8 · 3 months ago

.github	introduce passive crawling (#781)	3 months ago
.goreleaser	adding linux 386	3 months ago
cmd	introduce passive crawling (#781)	3 months ago
integration_tests	Add katana as lib (#205)	2 years ago
internal	version update	3 months ago
pkg	fixing hybrid redirect	3 months ago
.gitignore	send field with empty value	6 months ago
Dockerfile	fix: Dockerfile to reduce vulnera…	5 months ago
LICENSE.md	Initial Release	2 years ago
Makefile	Add katana as lib (#205)	2 years ago
README.md	Update README.md	3 months ago
SECURITY.md	Added SECURITY.md	2 years ago
go.mod	chore(deps): bump github.com/…	3 months ago
go.sum	chore(deps): bump github.com/…	3 months ago

About

A next-generation crawling and spidering framework.

#cli #crawler #headless #web-spider #spider-framework #gocrawler

- Readme
- MIT license
- Code of conduct
- Security policy
- Activity
- Custom properties
- **10k** stars
- **86** watching
- **514** forks

Report repository

Releases 10

v1.1.0 Latest
on Mar 26

+ 9 releases

Packages

No packages published

Contributors 36

+ 22 contributors

Languages

● **Go** 99.3% ● **Other** 0.7%

README · Code of conduct · More

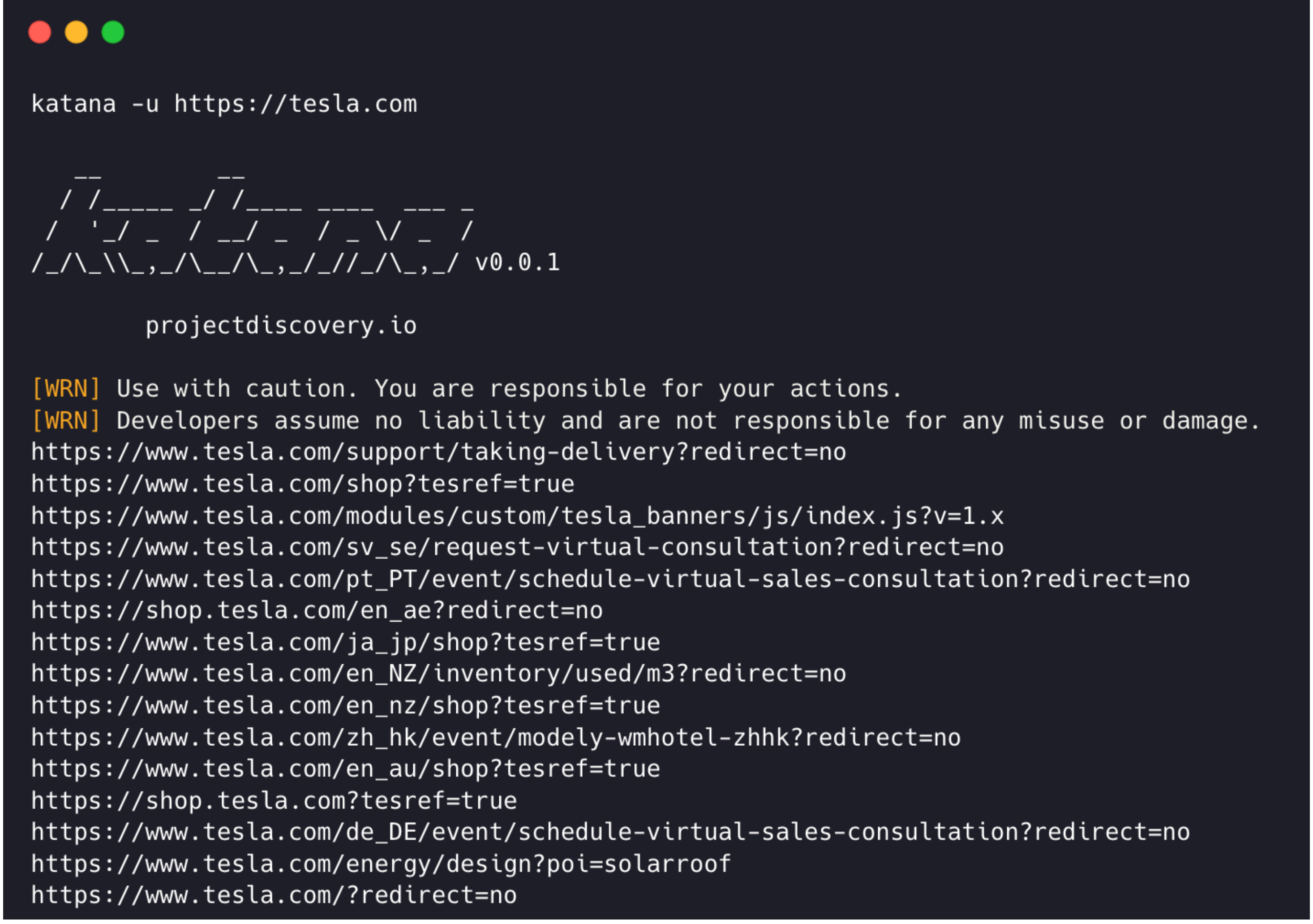
KATANA

A next-generation crawling and spidering framework

go report A+ · contributions welcome · release v1.1.0 · Follow @pdiscoveryio · chat 833 online

Features • Installation • Usage • Scope • Config • Filters • Join Discord

Features



```
katana -u https://tesla.com


   __        __
  / /_____ _/ /____ ____  ___ _
 /  '_/ _  / __/ _  / _ \/ _  /
/_/\_\\_,_/\__/\_,_/_//_/\_,_/ v0.0.1

        projectdiscovery.io

[WRN] Use with caution. You are responsible for your actions.
[WRN] Developers assume no liability and are not responsible for any misuse or damage.
https://www.tesla.com/support/taking-delivery?redirect=no
https://www.tesla.com/shop?tesref=true
https://www.tesla.com/modules/custom/tesla_banners/js/index.js?v=1.x
https://www.tesla.com/sv_se/request-virtual-consultation?redirect=no
https://www.tesla.com/pt_PT/event/schedule-virtual-sales-consultation?redirect=no
https://shop.tesla.com/en_ae?redirect=no
https://www.tesla.com/ja_jp/shop?tesref=true
https://www.tesla.com/en_NZ/inventory/used/m3?redirect=no
https://www.tesla.com/en_nz/shop?tesref=true
https://www.tesla.com/zh_hk/event/modely-wmhotel-zhhk?redirect=no
https://www.tesla.com/en_au/shop?tesref=true
https://shop.tesla.com?tesref=true
https://www.tesla.com/de_DE/event/schedule-virtual-sales-consultation?redirect=no
https://www.tesla.com/energy/design?poi=solarroof
https://www.tesla.com/?redirect=no
```

- Fast And fully configurable web crawling
- **Standard** and **Headless** mode
- **Active** and **Passive** mode
- **JavaScript** parsing / crawling
- Customizable **automatic form filling**
- **Scope control** - Preconfigured field / Regex
- **Customizable output** - Preconfigured fields
- INPUT - **STDIN**, **URL** and **LIST**
- OUTPUT - **STDOUT**, **FILE** and **JSON**

Installation

katana requires **Go 1.18** to install successfully. To install, just run the below command or download pre-compiled binary from release page.

```
go install github.com/projectdiscovery/katana/cmd/katana@latest
```

More options to install / run katana-

▶ Docker

▶ Ubuntu

Usage

```
katana -h
```

This will display help for the tool. Here are all the switches it supports.

```
Katana is a fast crawler focused on execution in automation
pipelines offering both headless and non-headless crawling.

Usage:
  ./katana [flags]

Flags:
INPUT:
   -u, -list string[]  target url / list to crawl
   -resume string      resume scan using resume.cfg
   -e, -exclude string[]  exclude host matching specified filter ('cdn', 'private-ips', cidr, ip, regex)

CONFIGURATION:
   -r, -resolvers string[]       list of custom resolver (file or comma separated)
   -d, -depth int                maximum depth to crawl (default 3)
   -jc, -js-crawl                enable endpoint parsing / crawling in javascript file
   -jsl, -jsluice                enable jsluice parsing in javascript file (memory intensive)
   -ct, -crawl-duration value    maximum duration to crawl the target for (s, m, h, d) (default s)
   -kf, -known-files string      enable crawling of known files (all,robotstxt,sitemapxml), a minimum depth of 3 is r
   -mrs, -max-response-size int  maximum response size to read (default 9223372036854775807)
   -timeout int                  time to wait for request in seconds (default 10)
   -aff, -automatic-form-fill    enable automatic form filling (experimental)
   -fx, -form-extraction         extract form, input, textarea & select elements in jsonl output
   -retry int                    number of times to retry the request (default 1)
   -proxy string                 http/socks5 proxy to use
   -H, -headers string[]         custom header/cookie to include in all http request in header:value format (file)
   -config string                path to the katana configuration file
   -fc, -form-config string      path to custom form configuration file
   -flc, -field-config string    path to custom field configuration file
   -s, -strategy string          Visit strategy (depth-first, breadth-first) (default "depth-first")
   -iqp, -ignore-query-params    Ignore crawling same path with different query-param values
   -tlsi, -tls-impersonate       enable experimental client hello (ja3) tls randomization
   -dr, -disable-redirects       disable following redirects (default false)

DEBUG:
   -health-check, -hc        run diagnostic check up
   -elog, -error-log string  file to write sent requests error log

HEADLESS:
   -hl, -headless                    enable headless hybrid crawling (experimental)
   -sc, -system-chrome               use local installed chrome browser instead of katana installed
   -sb, -show-browser                show the browser on the screen with headless mode
   -ho, -headless-options string[]   start headless chrome with additional options
   -nos, -no-sandbox                 start headless chrome in --no-sandbox mode
   -cdd, -chrome-data-dir string     path to store chrome browser data
   -scp, -system-chrome-path string  use specified chrome browser for headless crawling
   -noi, -no-incognito               start headless chrome without incognito mode
   -cwu, -chrome-ws-url string       use chrome browser instance launched elsewhere with the debugger listening at th
   -xhr, -xhr-extraction             extract xhr request url,method in jsonl output

PASSIVE:
   -ps, -passive                   enable passive sources to discover target endpoints
   -pss, -passive-source string[]  passive source to use for url discovery (waybackarchive,commoncrawl,alienvault)

SCOPE:
   -cs, -crawl-scope string[]       in scope url regex to be followed by crawler
   -cos, -crawl-out-scope string[]  out of scope url regex to be excluded by crawler
   -fs, -field-scope string         pre-defined scope field (dn,rdn,fqdn) or custom regex (e.g., '(company-staging.io
   -ns, -no-scope                   disables host based default scope
   -do, -display-out-scope          display external endpoint from scoped crawling

FILTER:
   -mr, -match-regex string[]       regex or list of regex to match on output url (cli, file)
   -fr, -filter-regex string[]      regex or list of regex to filter on output url (cli, file)
   -f, -field string                field to display in output (url,path,fqdn,rdn,rurl,qurl,qpath,file,ufile,key,valu
   -sf, -store-field string         field to store in per-host output (url,path,fqdn,rdn,rurl,qurl,qpath,file,ufile,k
   -em, -extension-match string[]   match output for given extension (eg, -em php,html,js)
   -ef, -extension-filter string[]  filter output for given extension (eg, -ef png,css)
   -mdc, -match-condition string    match response with dsl based condition
   -fdc, -filter-condition string   filter response with dsl based condition

RATE-LIMIT:
   -c, -concurrency int          number of concurrent fetchers to use (default 10)
   -p, -parallelism int          number of concurrent inputs to process (default 10)
   -rd, -delay int               request delay between each request in seconds
   -rl, -rate-limit int          maximum requests to send per second (default 150)
   -rlm, -rate-limit-minute int  maximum number of requests to send per minute

UPDATE:
   -up, -update                 update katana to latest version
   -duc, -disable-update-check  disable automatic katana update check

OUTPUT:
   -o, -output string              file to write output to
   -sr, -store-response            store http requests/responses
   -srd, -store-response-dir string  store http requests/responses to custom directory
   -or, -omit-raw                  omit raw requests/responses from jsonl output
   -ob, -omit-body                 omit response body from jsonl output
   -j, -jsonl                      write output in jsonl format
   -nc, -no-color                  disable output content coloring (ANSI escape codes)
   -silent                         display output only
   -v, -verbose                    display verbose output
   -debug                          display debug output
   -version                        display project version
```

Running Katana

Input for katana

katana requires **url** or **endpoint** to crawl and accepts single or multiple inputs.

Input URL can be provided using `-u` option, and multiple values can be provided using comma-separated input, similarly **file** input is supported using `-list` option and additionally piped input (stdin) is also supported.

URL Input

```
katana -u https://tesla.com
```

Multiple URL Input (comma-separated)

```
katana -u https://tesla.com,https://google.com
```

List Input

```
$ cat url_list.txt

https://tesla.com
https://google.com
```

```
katana -list url_list.txt
```

STDIN (piped) Input

```
echo https://tesla.com | katana
```

```
cat domains | httpx | katana
```

Example running katana -

```
katana -u https://youtube.com


   __        __
  / /_____ _/ /____ ____  ___ _
 /  '_/ _  / __/ _  / _ \/ _  /
/_/\_\\_,_/\__/\_,_/_//_/\_,_/ v0.0.1

      projectdiscovery.io

[WRN] Use with caution. You are responsible for your actions.
[WRN] Developers assume no liability and are not responsible for any misuse or damage.
https://www.youtube.com/
https://www.youtube.com/about/
https://www.youtube.com/about/press/
https://www.youtube.com/about/copyright/
https://www.youtube.com/t/contact_us/
https://www.youtube.com/creators/
https://www.youtube.com/ads/
https://www.youtube.com/t/terms
https://www.youtube.com/t/privacy
```

```
https://www.youtube.com/about/policies/
https://www.youtube.com/howyoutubeworks?utm_campaign=ytgen&utm_source=ythp&utm_medium=LeftNav&utm_content=txt&u=https
https://www.youtube.com/new
https://m.youtube.com/
https://www.youtube.com/s/desktop/4965577f/jsbin/desktop_polymer.vflset/desktop_polymer.js
https://www.youtube.com/s/desktop/4965577f/cssbin/www-main-desktop-home-page-skeleton.css
https://www.youtube.com/s/desktop/4965577f/cssbin/www-onepick.css
https://www.youtube.com/s/_/ytmainappweb/_/ss/k=ytmainappweb.kevlar_base.0Zo5FUcPkCg.L.B1.O/am=gAE/d=0/rs=AGKMywG5nh5
https://www.youtube.com/opensearch?locale=en_GB
https://www.youtube.com/manifest.webmanifest
https://www.youtube.com/s/desktop/4965577f/cssbin/www-main-desktop-watch-page-skeleton.css
https://www.youtube.com/s/desktop/4965577f/jsbin/web-animations-next-lite.min.vflset/web-animations-next-lite.min.js
https://www.youtube.com/s/desktop/4965577f/jsbin/custom-elements-es5-adapter.vflset/custom-elements-es5-adapter.js
https://www.youtube.com/s/desktop/4965577f/jsbin/webcomponents-sd.vflset/webcomponents-sd.js
https://www.youtube.com/s/desktop/4965577f/jsbin/intersection-observer.min.vflset/intersection-observer.min.js
https://www.youtube.com/s/desktop/4965577f/jsbin/scheduler.vflset/scheduler.js
https://www.youtube.com/s/desktop/4965577f/jsbin/www-i18n-constants-en_GB.vflset/www-i18n-constants.js
https://www.youtube.com/s/desktop/4965577f/jsbin/www-tampering.vflset/www-tampering.js
https://www.youtube.com/s/desktop/4965577f/jsbin/spf.vflset/spf.js
https://www.youtube.com/s/desktop/4965577f/jsbin/network.vflset/network.js
https://www.youtube.com/howyoutubeworks/
https://www.youtube.com/trends/
https://www.youtube.com/jobs/
https://www.youtube.com/kids/
```

Crawling Mode

Standard Mode

Standard crawling modality uses the standard go http library under the hood to handle HTTP requests/responses. This modality is much faster as it doesn't have the browser overhead. Still, it analyzes HTTP responses body as is, without any javascript or DOM rendering, potentially missing post-dom-rendered endpoints or asynchronous endpoint calls that might happen in complex web applications depending, for example, on browser-specific events.

Headless Mode

Headless mode hooks internal headless calls to handle HTTP requests/responses directly within the browser context. This offers two advantages:

- The HTTP fingerprint (TLS and user agent) fully identify the client as a legitimate browser
- Better coverage since the endpoints are discovered analyzing the standard raw response, as in the previous modality, and also the browser-rendered one with javascript enabled.

Headless crawling is optional and can be enabled using `-headless` option.

Here are other headless CLI options -

```
katana -h headless

Flags:
HEADLESS:
   -hl, -headless                    enable headless hybrid crawling (experimental)
   -sc, -system-chrome               use local installed chrome browser instead of katana installed
   -sb, -show-browser                show the browser on the screen with headless mode
   -ho, -headless-options string[]   start headless chrome with additional options
   -nos, -no-sandbox                 start headless chrome in --no-sandbox mode
   -cdd, -chrome-data-dir string     path to store chrome browser data
   -scp, -system-chrome-path string  use specified chrome browser for headless crawling
   -noi, -no-incognito               start headless chrome without incognito mode
   -cwu, -chrome-ws-url string       use chrome browser instance launched elsewhere with the debugger listening at th
   -xhr, -xhr-extraction             extract xhr requests
```

`-no-sandbox`

Runs headless chrome browser with **no-sandbox** option, useful when running as root user.

```
katana -u https://tesla.com -headless -no-sandbox
```

`-no-incognito`

Runs headless chrome browser without incognito mode, useful when using the local browser.

```
katana -u https://tesla.com -headless -no-incognito
```

`-headless-options`

When crawling in headless mode, additional chrome options can be specified using `-headless-options`, for example -

```
katana -u https://tesla.com -headless -system-chrome -headless-options --disable-gpu,proxy-server=http://127.0.0.1:8C
```

Scope Control

Crawling can be endless if not scoped, as such katana comes with multiple support to define the crawl scope.

`-field-scope`

Most handy option to define scope with predefined field name, `rdn` being default option for field scope.

- `rdn` - crawling scoped to root domain name and all subdomains (e.g. `*example.com`) (default)
- `fqdn` - crawling scoped to given sub(domain) (e.g. `www.example.com` or `api.example.com`)
- `dn` - crawling scoped to domain name keyword (e.g. `example`)

```
katana -u https://tesla.com -fs dn
```

`-crawl-scope`

For advanced scope control, `-cs` option can be used that comes with **regex** support.

```
katana -u https://tesla.com -cs login
```

For multiple in scope rules, file input with multiline string / regex can be passed.

```
$ cat in_scope.txt

login/
admin/
app/
wordpress/
```

```
katana -u https://tesla.com -cs in_scope.txt
```

`-crawl-out-scope`

For defining what not to crawl, `-cos` option can be used and also support **regex** input.

```
katana -u https://tesla.com -cos logout
```

For multiple out of scope rules, file input with multiline string / regex can be passed.

```
$ cat out_of_scope.txt

/logout
/log_out
```

```
katana -u https://tesla.com -cos out_of_scope.txt
```

`-no-scope`

Katana is default to scope `*.domain`, to disable this `-ns` option can be used and also to crawl the internet.

```
katana -u https://tesla.com -ns
```

`-display-out-scope`

As default, when scope option is used, it also applies for the links to display as output, as such **external URLs are default to exclude** and to overwrite this behavior, `-do` option can be used to display all the external URLs that exist in targets scoped URL / Endpoint.

```
katana -u https://tesla.com -do
```

Here is all the CLI options for the scope control -

```
katana -h scope

Flags:
SCOPE:
   -cs, -crawl-scope string[]       in scope url regex to be followed by crawler
   -cos, -crawl-out-scope string[]  out of scope url regex to be excluded by crawler
   -fs, -field-scope string         pre-defined scope field (dn,rdn,fqdn) (default "rdn")
   -ns, -no-scope                   disables host based default scope
   -do, -display-out-scope          display external endpoint from scoped crawling
```

Crawler Configuration

Katana comes with multiple options to configure and control the crawl as the way we want.

`-depth`

Option to define the `depth` to follow the urls for crawling, the more depth the more number of endpoint being crawled + time for crawl.

```
katana -u https://tesla.com -d 5
```

`-js-crawl`

Option to enable JavaScript file parsing + crawling the endpoints discovered in JavaScript files, disabled as default.

```
katana -u https://tesla.com -jc
```

`-crawl-duration`

Option to predefined crawl duration, disabled as default.

```
katana -u https://tesla.com -ct 2
```

`-known-files`

Option to enable crawling `robots.txt` and `sitemap.xml` file, disabled as default.

```
katana -u https://tesla.com -kf robotstxt,sitemapxml
```

`-automatic-form-fill`

Option to enable automatic form filling for known / unknown fields, known field values can be customized as needed by updating form config file at `$HOME/.config/katana/form-config.yaml`.

Automatic form filling is experimental feature.

```
katana -u https://tesla.com -aff
```

Authenticated Crawling

Authenticated crawling involves including custom headers or cookies in HTTP requests to access protected resources. These headers provide authentication or authorization information, allowing you to crawl authenticated content / endpoint. You can specify headers directly in the command line or provide them as a file with katana to perfrom authenticated crawling.

> **Note**: User needs to be manually perform the authentication and export the session cookie / header to file to use with katana.

`-headers`

Option to add a custom header or cookie to the request.

> Syntax of headers in the HTTP specification

Here is an example of adding a cookie to the request:

```
katana -u https://tesla.com -H 'Cookie: usrsess=AmljNrESo'
```

It is also possible to supply headers or cookies as a file. For example:

```
$ cat cookie.txt

Cookie: PHPSESSIONID=XXXXXXXXX
X-API-KEY: XXXXX
TOKEN=XX
```

```
katana -u https://tesla.com -H cookie.txt
```

There are more options to configure when needed, here is all the config related CLI options -

```
katana -h config

Flags:
CONFIGURATION:
   -r, -resolvers string[]       list of custom resolver (file or comma separated)
   -d, -depth int                maximum depth to crawl (default 3)
   -jc, -js-crawl                enable endpoint parsing / crawling in javascript file
   -ct, -crawl-duration int      maximum duration to crawl the target for
   -kf, -known-files string      enable crawling of known files (all,robotstxt,sitemapxml)
   -mrs, -max-response-size int  maximum response size to read (default 9223372036854775807)
   -timeout int                  time to wait for request in seconds (default 10)
   -aff, -automatic-form-fill    enable automatic form filling (experimental)
   -fx, -form-extraction         enable extraction of form, input, textarea & select elements
   -retry int                    number of times to retry the request (default 1)
   -proxy string                 http/socks5 proxy to use
   -H, -headers string[]         custom header/cookie to include in request
   -config string                path to the katana configuration file
   -fc, -form-config string      path to custom form configuration file
   -flc, -field-config string    path to custom field configuration file
   -s, -strategy string          Visit strategy (depth-first, breadth-first) (default "depth-first")
```

Connecting to Active Browser Session

Katana can also connect to active browser session where user is already logged in and authenticated. and use it for crawling. The only requirement for this is to start browser with remote debugging enabled.

Here is an example of starting chrome browser with remote debugging enabled and using it with katana -

step 1) First Locate path of chrome executable

Operating System	Chromium Executable Location	Google Chrome Executable Location
Windows	`C:\Program Files`	`C:\Program Files`

Operating System	Chromium Executable Location	Google Chrome Executable Location
(64-bit)	`(x86)\Google\Chromium\Application\chrome.exe`	`(x86)\Google\Chrome\Application\chrome.exe`
Windows (32-bit)	`C:\Program Files\Google\Chromium\Application\chrome.exe`	`C:\Program Files\Google\Chrome\Application\chrome.exe`
macOS	`/Applications/Chromium.app/Contents/MacOS/Chromium`	`/Applications/Google Chrome.app/Contents/MacOS/Google Chrome`
Linux	`/usr/bin/chromium`	`/usr/bin/google-chrome`

step 2) Start chrome with remote debugging enabled and it will return websocker url. For example, on MacOS, you can start chrome with remote debugging enabled using following command -

```
$ /Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --remote-debugging-port=9222


DevTools listening on ws://127.0.0.1:9222/devtools/browser/c5316c9c-19d6-42dc-847a-41d1aeebf7d6
```

> Now login to the website you want to crawl and keep the browser open.

step 3) Now use the websocket url with katana to connect to the active browser session and crawl the website

```
katana -headless -u https://tesla.com -cwu ws://127.0.0.1:9222/devtools/browser/c5316c9c-19d6-42dc-847a-41d1aeebf7d6
```

> **Note**: you can use `-cdd` option to specify custom chrome data directory to store browser data and cookies but that does not save session data if cookie is set to `Session` only or expires after certain time.

Filters

`-field`

Katana comes with built in fields that can be used to filter the output for the desired information, `-f` option can be used to specify any of the available fields.

```
   -f, -field string  field to display in output (url,path,fqdn,rdn,rurl,qurl,qpath,file,key,value,kv,dir,udir)
```

Here is a table with examples of each field and expected output when used -

FIELD	DESCRIPTION	EXAMPLE
`url`	URL Endpoint	`https://admin.projectdiscovery.io/admin/login?user=admin&password=admin`
`qurl`	URL including query param	`https://admin.projectdiscovery.io/admin/login.php?user=admin&password=admin`
`qpath`	Path including query param	`/login?user=admin&password=admin`
`path`	URL Path	`https://admin.projectdiscovery.io/admin/login`
`fqdn`	Fully Qualified Domain name	`admin.projectdiscovery.io`
`rdn`	Root Domain name	`projectdiscovery.io`
`rurl`	Root URL	`https://admin.projectdiscovery.io`
`ufile`	URL with File	`https://admin.projectdiscovery.io/login.js`
`file`	Filename in URL	`login.php`
`key`	Parameter keys in URL	`user,password`
`value`	Parameter values in URL	`admin,admin`
`kv`	Keys=Values in URL	`user=admin&password=admin`
`dir`	URL Directory name	`/admin/`
`udir`	URL with Directory	`https://admin.projectdiscovery.io/admin/`

Here is an example of using field option to only display all the urls with query parameter in it -

```
katana -u https://tesla.com -f qurl -silent

https://shop.tesla.com/en_au?redirect=no
https://shop.tesla.com/en_nz?redirect=no
https://shop.tesla.com/product/men_s-raven-lightweight-zip-up-bomber-jacket?sku=1740250-00-A
https://shop.tesla.com/product/tesla-shop-gift-card?sku=1767247-00-A
https://shop.tesla.com/product/men_s-chill-crew-neck-sweatshirt?sku=1740176-00-A
https://www.tesla.com/about?redirect=no
https://www.tesla.com/about/legal?redirect=no
https://www.tesla.com/findus/list?redirect=no
```

Custom Fields

You can create custom fields to extract and store specific information from page responses using regex rules. These custom fields are defined using a YAML config file and are loaded from the default location at `$HOME/.config/katana/field-config.yaml`. Alternatively, you can use the `-flc` option to load a custom field config file from a different location. Here is example custom field.

```yaml
- name: email
  type: regex
  regex:
  - '([a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9_-]+)'
  - '([a-zA-Z0-9+._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9_-]+)'

- name: phone
  type: regex
  regex:
  - '\d{3}-\d{8}|\d{4}-\d{7}'
```

When defining custom fields, following attributes are supported:

- **name** (required)

> The value of **name** attribute is used as the `-field` cli option value.

- **type** (required)

> The type of custom attribute, currenly supported option - `regex`

- **part** (optional)

> The part of the response to extract the information from. The default value is `response`, which includes both the header and body. Other possible values are `header` and `body`.

- group (optional)

> You can use this attribute to select a specific matched group in regex, for example: `group: 1`

Running katana using custom field:

```
katana -u https://tesla.com -f email,phone
```

`-store-field`

To compliment `field` option which is useful to filter output at run time, there is `-sf, -store-fields` option which works exactly like field option except instead of filtering, it stores all the information on the disk under `katana_field` directory sorted by target url.

```
katana -u https://tesla.com -sf key,fqdn,qurl -silent
```

```
$ ls katana_field/

https_www.tesla.com_fqdn.txt
https_www.tesla.com_key.txt
https_www.tesla.com_qurl.txt
```

The `-store-field` option can be useful for collecting information to build a targeted wordlist for various purposes, including but not limited to:

- Identifying the most commonly used parameters
- Discovering frequently used paths
- Finding commonly used files
- Identifying related or unknown subdomains

Katana Filters

`-extension-match`

Crawl output can be easily matched for specific extension using `-em` option to ensure to display only output containing given extension.

```
katana -u https://tesla.com -silent -em js,jsp,json
```

`-extension-filter`

Crawl output can be easily filtered for specific extension using `-ef` option which ensure to remove all the urls containing given extension.

```
katana -u https://tesla.com -silent -ef css,txt,md
```

`-match-regex`

The `-match-regex` or `-mr` flag allows you to filter output URLs using regular expressions. When using this flag, only URLs that match the specified regular expression will be printed in the output.

```
katana -u https://tesla.com -mr 'https://shop\.tesla\.com/*' -silent
```

`-filter-regex`

The `-filter-regex` or `-fr` flag allows you to filter output URLs using regular expressions. When using this flag, it will skip the URLs that are match the specified regular expression.

```
katana -u https://tesla.com -fr 'https://www\.tesla\.com/*' -silent
```

Advance Filtering

Katana supports DSL-based expressions for advanced matching and filtering capabilities:

- To match endpoints with a 200 status code:

```
katana -u https://www.hackerone.com -mdc 'status_code == 200'
```

- To match endpoints that contain "default" and have a status code other than 403:

```
katana -u https://www.hackerone.com -mdc 'contains(endpoint, "default") && status_code != 403'
```

- To match endpoints with PHP technologies:

```
katana -u https://www.hackerone.com -mdc 'contains(to_lower(technologies), "php")'
```

- To filter out endpoints running on Cloudflare:

```
katana -u https://www.hackerone.com -fdc 'contains(to_lower(technologies), "cloudflare")'
```

DSL functions can be applied to any keys in the jsonl output. For more information on available DSL functions, please visit the dsl project.

Here are additional filter options -

```
katana -h filter
```

```
Flags:
FILTER:
   -mr, -match-regex string[]      regex or list of regex to match on output url (cli, file)
   -fr, -filter-regex string[]     regex or list of regex to filter on output url (cli, file)
   -f, -field string               field to display in output (url,path,fqdn,rdn,rurl,qurl,qpath,file,ufile,key,valu
   -sf, -store-field string        field to store in per-host output (url,path,fqdn,rdn,rurl,qurl,qpath,file,ufile,k
   -em, -extension-match string[]  match output for given extension (eg, -em php,html,js)
   -ef, -extension-filter string[] filter output for given extension (eg, -ef png,css)
   -mdc, -match-condition string   match response with dsl based condition
   -fdc, -filter-condition string  filter response with dsl based condition
```

Rate Limit

It's easy to get blocked / banned while crawling if not following target websites limits, katana comes with multiple option to tune the crawl to go as fast / slow we want.

`-delay`

option to introduce a delay in seconds between each new request katana makes while crawling, disabled as default.

```
katana -u https://tesla.com -delay 20
```

`-concurrency`

option to control the number of urls per target to fetch at the same time.

```
katana -u https://tesla.com -c 20
```

`-parallelism`

option to define number of target to process at same time from list input.

```
katana -u https://tesla.com -p 20
```

`-rate-limit`

option to use to define max number of request can go out per second.

```
katana -u https://tesla.com -rl 100
```

`-rate-limit-minute`

option to use to define max number of request can go out per minute.

```
katana -u https://tesla.com -rlm 500
```

Here is all long / short CLI options for rate limit control -

```
katana -h rate-limit

Flags:
RATE-LIMIT:
   -c, -concurrency int          number of concurrent fetchers to use (default 10)
   -p, -parallelism int          number of concurrent inputs to process (default 10)
   -rd, -delay int               request delay between each request in seconds
   -rl, -rate-limit int          maximum requests to send per second (default 150)
   -rlm, -rate-limit-minute int  maximum number of requests to send per minute
```

Output

Katana support both file output in plain text format as well as JSON which includes additional information like, `source`, `tag`, and `attribute` name to co-related the discovered endpoint.

`-output`

By default, katana outputs the crawled endpoints in plain text format. The results can be written to a file by using the -output option.

```
katana -u https://example.com -no-scope -output example_endpoints.txt
```

`-jsonl`

```
katana -u https://example.com -jsonl | jq .
```

```
{
  "timestamp": "2023-03-20T16:23:58.027559+05:30",
  "request": {
    "method": "GET",
    "endpoint": "https://example.com",
    "raw": "GET / HTTP/1.1\r\nHost: example.com\r\nUser-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 11_1) AppleWebK
  },
  "response": {
    "status_code": 200,
    "headers": {
      "accept_ranges": "bytes",
      "expires": "Mon, 27 Mar 2023 10:53:58 GMT",
      "last_modified": "Thu, 17 Oct 2019 07:18:26 GMT",
      "content_type": "text/html; charset=UTF-8",
      "server": "ECS (dcb/7EA3)",
      "vary": "Accept-Encoding",
      "etag": "\"3147526947\"",
      "cache_control": "max-age=604800",
      "x_cache": "HIT",
      "date": "Mon, 20 Mar 2023 10:53:58 GMT",
      "age": "331239"
    },
    "body": "<!doctype html>\n<html>\n<head>\n    <title>Example Domain</title>\n\n    <meta charset=\"utf-8\" />\n
    "technologies": [
      "Azure",
      "Amazon ECS",
      "Amazon Web Services",
      "Docker",
      "Azure CDN"
    ],
    "raw": "HTTP/1.1 200 OK\r\nContent-Length: 1256\r\nAccept-Ranges: bytes\r\nAge: 331239\r\nCache-Control: max-age=
  }
}
```

`-store-response`

The `-store-response` option allows for writing all crawled endpoint requests and responses to a text file. When this option is used, text files including the request and response will be written to the **katana_response** directory. If you would like to specify a custom directory, you can use the `-store-response-dir` option.

```
katana -u https://example.com -no-scope -store-response
```

```
$ cat katana_response/index.txt

katana_response/example.com/327c3fda87ce286848a574982ddd0b7c7487f816.txt https://example.com (200 OK)
katana_response/www.iana.org/bfc096e6dd93b993ca8918bf4c08fdc707a70723.txt http://www.iana.org/domains/reserved (200 C
```

Note:

`-store-response` option is not supported in `-headless` mode.

Here are additional CLI options related to output -

```
katana -h output

OUTPUT:
   -o, -output string             file to write output to
   -sr, -store-response           store http requests/responses
   -srd, -store-response-dir string  store http requests/responses to custom directory
   -j, -json                      write output in JSONL(ines) format
   -nc, -no-color                 disable output content coloring (ANSI escape codes)
   -silent                        display output only
   -v, -verbose                   display verbose output
   -version                       display project version
```

Katana as a library

`katana` can be used as a library by creating an instance of the `Option` struct and populating it with the same options that would be specified via CLI. Using the options you can create `crawlerOptions` and so standard or hybrid `crawler`. `crawler.Crawl` method should be called to crawl the input.

```
package main

import (
	"math"

	"github.com/projectdiscovery/gologger"
	"github.com/projectdiscovery/katana/pkg/engine/standard"
	"github.com/projectdiscovery/katana/pkg/output"
	"github.com/projectdiscovery/katana/pkg/types"
)

func main() {
	options := &types.Options{
		MaxDepth:     3,             // Maximum depth to crawl
		FieldScope:   "rdn",         // Crawling Scope Field
		BodyReadSize: math.MaxInt,   // Maximum response size to read
		Timeout:      10,            // Timeout is the time to wait for request in seconds
		Concurrency:  10,            // Concurrency is the number of concurrent crawling goroutines
		Parallelism:  10,            // Parallelism is the number of urls processing goroutines
		Delay:        0,             // Delay is the delay between each crawl requests in seconds
		RateLimit:    150,           // Maximum requests to send per second
		Strategy:     "depth-first", // Visit strategy (depth-first, breadth-first)
		OnResult: func(result output.Result) { // Callback function to execute for result
			gologger.Info().Msg(result.Request.URL)
		},
	}
	crawlerOptions, err := types.NewCrawlerOptions(options)
```