recon-pipeline

Release 0.7.3

Aug 28, 2020

Contents

`recon-pipeline` was designed to chain together multiple security tools as part of a Flow-Based Programming paradigm. Each component is part of a network of “black box” processes. These components exchange data between each other and can be reconnected in different ways to form different applications without any internal changes.

CHAPTER 1

Getting Started

There are an accompanying set of blog posts detailing the development process and underpinnings of the pipeline. Feel free to check them out if you're so inclined, but they're in no way required reading to use the tool.

- *Installation Instructions* - How to install `recon-pipeline` and associated dependencies
- *Defining Target Scope* - How to define the scope of your scans (list of targets and a blacklist)
- *Running Scans* - Example scan of **tesla.com** using `recon-pipeline`
- *Viewing Scan Results* - How to view scan results
- *Using a Scheduler* - The Luigi schedulers and which to choose
- *Visualizing Tasks* - How to check on active tasks once they're running

1.1 Getting Started

1.1.1 Installation Instructions

There are two primary phases for installation:

- prior to the python dependencies being installed
- everything else

Manual Steps

First, the steps to get python dependencies installed in a virtual environment are as follows (and shown below)

Kali

```
sudo apt update
sudo apt install pipenv
```

Ubuntu 18.04/20.04

```
sudo apt update
sudo apt install python3-pip
pip install --user pipenv
echo "PATH=${PATH}:~/.local/bin" >> ~/.bashrc
bash
```

Both OSs After `pipenv` Install

```
git clone https://github.com/epi052/recon-pipeline.git
cd recon-pipeline
pipenv install
pipenv shell
```

Everything Else

After installing the python dependencies, the recon-pipeline shell provides its own *tools* command (seen below). A simple `tools install all` will handle all installation steps. Installation has **only** been tested on **Kali 2019.4 and Ubuntu 18.04/20.04**.

> **Ubuntu Note (and newer kali versions)**: You may consider running `sudo -v` prior to running `./recon-pipeline.py`. `sudo -v` will refresh your creds, and the underlying subprocess calls during installation won't prompt you for your password. It'll work either way though.

Individual tools may be installed by running `tools install TOOLNAME` where `TOOLNAME` is one of the known tools that make up the pipeline.

The installer does not maintain state. In order to determine whether a tool is installed or not, it checks the *path* variable defined in the tool's .yaml file. The installer in no way attempts to be a package manager. It knows how to execute the steps necessary to install and remove its tools. Beyond that, it's like Jon Snow, **it knows nothing**.

Current tool status can be viewed using `tools list`. Tools can also be uninstalled using the `tools uninstall all` command. It is also possible to individually uninstall them in the same manner as shown above.

Alternative Distros

In v0.8.1, an effort was made to remove OS specific installation steps from the installer. However, if you're using an untested distribution (i.e. not Kali/Ubuntu 18.04/20.04), meeting the criteria below **should** be sufficient for the auto installer to function:

- systemd-based system (`luigid` is installed as a systemd service)
- python3.6+ installed

With the above requirements met, following the installation steps above starting with `pipenv install` should be sufficient.

The alternative would be to manually install each tool.

Docker

If you have Docker installed, you can run the recon-pipeline in a container with the following commands:

```
git clone https://github.com/epi052/recon-pipeline.git
cd recon-pipeline
docker build -t recon-pipeline .
docker run -d \
    -v ~/docker/recon-pipeline:/root/.local/recon-pipeline \
    -p 8082:8082 \
    --name recon-pipeline \
    recon-pipeline
```

It is important to note that you should not lose any data during an update because all important information is saved to the `~/docker/recon-pipeline` location as specified by the `-v` option in the `docker run` command. If this portion of the command was not executed, data will not persist across container installations.

At this point the container should be running and you scan enter the shell with the following command:

```
docker exec -it recon-pipeline pipeline
```

Starting & Stopping

In the event that you need to start or stop the container, you can do so with the following commands after having run the installation commands above once:

```
docker start recon-pipeline
docker stop recon-pipeline
```

This is useful knowledge because Docker containers do not normally start on their own and executing the `docker run` command above again will result in an error if it is already installed.

Update

To update, you can run the following commands from inside the `recon-pipeline` folder cloned in the installation:

```
git pull
docker stop recon-pipeline
docker rm recon-pipeline
```

When complete, execute the inital installation commands again starting with `docker build`.

1.1.2 Defining Target Scope

New in v0.9.0: In the event you're scanning a single ip address or host, simply use `--target`. It accepts a single target and works in conjunction with `--exempt-list` if specified.

```
[db-1] recon-pipeline> scan HTBScan --target 10.10.10.183 --top-ports 1000
...
```

In order to scan more than one host at a time, the pipeline needs a file that describes the target's scope to be provided as an argument to the *–target-file* option. The target file can consist of domains, ip addresses, and ip ranges, one per line.

In order to scan more than one host at a time, the pipeline expects a file that describes the target's scope to be provided as an argument to the `--target-file` option. The target file can consist of domains, ip addresses, and ip ranges, one per line. Domains, ip addresses and ip ranges can be mixed/matched within the scope file.

```
tesla.com
tesla.cn
teslamotors.com
...
```

Some bug bounty scopes have expressly verboten subdomains and/or top-level domains, for that there is the `--exempt-list` option. The exempt list follows the same rules as the target file.

```
shop.eu.teslamotors.com
energysupport.tesla.com
feedback.tesla.com
...
```

1.1.3 Running Scans

All scans are run from within `recon-pipeline`'s shell. There are a number of individual scans, however to execute multiple scans at once, `recon-pipeline` includes wrappers around multiple commands. As of version 0.7.3, the following individual scans are available

- *`pipeline.recon.amass.AmassScan`*
- *`pipeline.recon.web.aquatone.AquatoneScan`*
- *`pipeline.recon.web.gobuster.GobusterScan`*
- *`pipeline.recon.masscan.MasscanScan`*
- *`pipeline.recon.nmap.SearchsploitScan`*
- *`pipeline.recon.web.subdomain_takeover.SubjackScan`*
- *`pipeline.recon.nmap.ThreadedNmapScan`*
- *`pipeline.recon.web.subdomain_takeover.TKOSubsScan`*
- *`pipeline.recon.web.waybackurls.WaybackurlsScan`*
- *`pipeline.recon.web.webanalyze.WebanalyzeScan`*

Additionally, two wrapper scans are made available. These execute multiple scans in a pipeline.

- *`pipeline.recon.wrappers.FullScan`* - runs the entire pipeline
- *`pipeline.recon.wrappers.HTBScan`* - nicety for hackthebox players (myself included) that omits the scans in FullScan that don't make sense for HTB

Example Scan

Here are the steps the video below takes to scan tesla[.]com.

Create a targetfile

```
# use virtual environment
pipenv shell

# create targetfile; a targetfile is required for all scans
mkdir /root/bugcrowd/tesla
cd /root/bugcrowd/tesla
echo tesla.com > tesla-targetfile

# create a blacklist (if necessary based on target's scope)
echo energysupport.tesla.com > tesla-blacklist
echo feedback.tesla.com >> tesla-blacklist
echo employeefeedback.tesla.com >> tesla-blacklist
echo ir.tesla.com >> tesla-blacklist

# drop into the interactive shell
/root/PycharmProjects/recon-pipeline/pipeline/recon-pipeline.py
recon-pipeline>
```

New as of v0.9.0: In the event you're scanning a single ip address or host, simply use `--target`. It accepts a single target and works in conjunction with `--exempt-list` if specified.

Create a new database to store scan results

```
recon-pipeline> database attach
   1. create new database
Your choice? 1
new database name? (recommend something unique for this target)
-> tesla-scan
[*] created database @ /home/epi/.local/recon-pipeline/databases/tesla-scan
[+] attached to sqlite database @ /home/epi/.local/recon-pipeline/databases/tesla-scan
[db-1] recon-pipeline>
```

Scan the target

```
[db-1] recon-pipeline> scan FullScan --exempt-list tesla-blacklist --target-file␣
↪tesla-targetfile --interface eno1 --top-ports 2000 --rate 1200
[-] FullScan queued
[-] TKOSubsScan queued
[-] GatherWebTargets queued
[-] ParseAmassOutput queued
[-] AmassScan queued
[-] ParseMasscanOutput queued
[-] MasscanScan queued
[-] WebanalyzeScan queued
[-] SearchsploitScan queued
[-] ThreadedNmapScan queued
[-] WaybackurlsScan queued
[-] SubjackScan queued
[-] AquatoneScan queued
[-] GobusterScan queued
[db-1] recon-pipeline>
```

Existing Results Directories and You

When running additional scans against the same target, you have a few options. You can either

- use a new directory

- reuse the same directory

If you use a new directory, the scan will start from the beginning.

If you choose to reuse the same directory, `recon-pipeline` will resume the scan from its last successful point. For instance, say your last scan failed while running nmap. This means that the pipeline executed all upstream tasks (amass and masscan) successfully. When you use the same results directory for another scan, the amass and masscan scans will be skipped, because they've already run successfully.

Note: There is a gotcha that can occur when you scan a target but get no results. For some scans, the pipeline may still mark the Task as complete (masscan does this). In masscan's case, it's because it outputs a file to `results-dir/masscan-results/` whether it gets results or not. Luigi interprets the file's presence to mean the scan is complete.

In order to reduce confusion, as of version 0.9.3, the pipeline will prompt you when reusing results directory.

```
[db-2] recon-pipeline> scan FullScan --results-dir testing-results --top-ports 1000 --
↪rate 500 --target tesla.com
[*] Your results-dir (testing-results) already exists. Subfolders/files may tell the␣
↪pipeline that the associated Task is complete. This means that your scan may start␣
↪from a point you don't expect. Your options are as follows:
   1. Resume existing scan (use any existing scan data & only attempt to scan what isn
↪'t already done)
   2. Remove existing directory (scan starts from the beginning & all existing␣
↪results are removed)
   3. Save existing directory (your existing folder is renamed and your scan proceeds)
Your choice?
```

1.1.4 Viewing Scan Results

As of version 0.9.0, scan results are stored in a database located (by default) at `~/.local/recon-pipeline/databases`. Databases themselves are managed through the *database* command while viewing their contents is done via *view*.

The view command allows one to inspect different pieces of scan information via the following sub-commands

- endpoints (gobuster results)
- nmap-scans
- ports
- searchsploit-results
- targets
- web-technologies (webanalyze results)

Each of the sub-commands has a list of tab-completable options and values that can help drilling down to the data you care about.

All of the subcommands offer a `--paged` option for dealing with large amounts of output. `--paged` will show you one page of output at a time (using `less` under the hood).

Chaining Results w/ Commands

All of the results can be **piped out to other commands**. Let's say you want to feed some results from `recon-pipeline` into another tool that isn't part of the pipeline. Simply using a normal unix pipe `|` followed by the next command will get that done for you. Below is an example of piping targets into gau

```
[db-2] recon-pipeline> view targets --paged
3.tesla.cn
3.tesla.com
api-internal.sn.tesla.services
api-toolbox.tesla.com
api.mp.tesla.services
api.sn.tesla.services
api.tesla.cn
api.toolbox.tb.tesla.services
...

[db-2] recon-pipeline> view targets | gau
https://3.tesla.com/pt_PT/model3/design
https://3.tesla.com/pt_PT/model3/design?redirect=no
https://3.tesla.com/robots.txt
https://3.tesla.com/sites/all/themes/custom/tesla_theme/assets/img/icons/favicon-
↪160x160.png?2
https://3.tesla.com/sites/all/themes/custom/tesla_theme/assets/img/icons/favicon-
↪16x16.png?2
https://3.tesla.com/sites/all/themes/custom/tesla_theme/assets/img/icons/favicon-
↪196x196.png?2
https://3.tesla.com/sites/all/themes/custom/tesla_theme/assets/img/icons/favicon-
↪32x32.png?2
https://3.tesla.com/sites/all/themes/custom/tesla_theme/assets/img/icons/favicon-
↪96x96.png?2
https://3.tesla.com/sv_SE/model3/design
...
```

view endpoints

An endpoint consists of a status code and the scanned URL. Endpoints are populated via gobuster.

Show All Endpoints

```
[db-2] recon-pipeline> view endpoints --paged
[200] http://westream.teslamotors.com/y
[301] https://mobileapps.teslamotors.com/aspnet_client
[403] https://209.133.79.49/analog.html
[302] https://209.133.79.49/api
[403] https://209.133.79.49/cgi-bin/
[200] https://209.133.79.49/client
...
```

Filter by Host

```
[db-2] recon-pipeline> view endpoints --host shop.uk.teslamotors.com
[402] http://shop.uk.teslamotors.com/
[403] https://shop.uk.teslamotors.com:8443/
[301] http://shop.uk.teslamotors.com/assets
[302] http://shop.uk.teslamotors.com/admin.cgi
[200] http://shop.uk.teslamotors.com/.well-known/apple-developer-merchantid-domain-
↪association
```

(continues on next page)

(continued from previous page)

```
[302] http://shop.uk.teslamotors.com/admin
[403] http://shop.uk.teslamotors.com:8080/
[302] http://shop.uk.teslamotors.com/admin.php
[302] http://shop.uk.teslamotors.com/admin.pl
[200] http://shop.uk.teslamotors.com/crossdomain.xml
[403] https://shop.uk.teslamotors.com/
[db-2] recon-pipeline>
```

Filter by Host and Status Code

```
[db-2] recon-pipeline> view endpoints --host shop.uk.teslamotors.com --status-code 200
[200] http://shop.uk.teslamotors.com/crossdomain.xml
[200] http://shop.uk.teslamotors.com/.well-known/apple-developer-merchantid-domain-
↪association
[db-2] recon-pipeline>
```

Remove Status Code from Output

Using `--plain` will remove the status-code prefix, allowing for easy piping of results into other commands.

```
[db-2] recon-pipeline> view endpoints --host shop.uk.teslamotors.com --plain
http://shop.uk.teslamotors.com/admin.pl
http://shop.uk.teslamotors.com/admin
http://shop.uk.teslamotors.com/
http://shop.uk.teslamotors.com/admin.cgi
http://shop.uk.teslamotors.com/.well-known/apple-developer-merchantid-domain-
↪association
http://shop.uk.teslamotors.com:8080/
http://shop.uk.teslamotors.com/crossdomain.xml
https://shop.uk.teslamotors.com:8443/
https://shop.uk.teslamotors.com/
http://shop.uk.teslamotors.com/admin.php
http://shop.uk.teslamotors.com/assets
[db-2] recon-pipeline>
```

Include Headers

If you'd like to include any headers found during scanning, `--headers` will do that for you.

```
[db-2] recon-pipeline> view endpoints --host shop.uk.teslamotors.com --headers
[302] http://shop.uk.teslamotors.com/admin.php
[302] http://shop.uk.teslamotors.com/admin.cgi
[302] http://shop.uk.teslamotors.com/admin
[200] http://shop.uk.teslamotors.com/crossdomain.xml
[403] https://shop.uk.teslamotors.com/
  Server: cloudflare
  Date: Mon, 06 Apr 2020 13:56:12 GMT
  Content-Type: text/html
  Content-Length: 553
  Retry-Count: 0
  Cf-Ray: 57fc02c788f7e03f-DFW
```

(continues on next page)

(continued from previous page)

```
[403] https://shop.uk.teslamotors.com:8443/
  Content-Type: text/html
  Content-Length: 553
  Retry-Count: 0
  Cf-Ray: 57fc06e5fcbfd266-DFW
  Server: cloudflare
  Date: Mon, 06 Apr 2020 13:59:00 GMT
[302] http://shop.uk.teslamotors.com/admin.pl
[200] http://shop.uk.teslamotors.com/.well-known/apple-developer-merchantid-domain-
↪association
[403] http://shop.uk.teslamotors.com:8080/
  Server: cloudflare
  Date: Mon, 06 Apr 2020 13:58:50 GMT
  Content-Type: text/html; charset=UTF-8
  Set-Cookie: __cfduid=dfbf45a8565fda1325b8c1482961518511586181530; expires=Wed, 06-
↪May-20 13:58:50 GMT; path=/; domain=.shop.uk.teslamotors.com; HttpOnly; SameSite=Lax
  Cache-Control: max-age=15
  X-Frame-Options: SAMEORIGIN
  Alt-Svc: h3-27=":443"; ma=86400, h3-25=":443"; ma=86400, h3-24=":443"; ma=86400, h3-
↪23=":443"; ma=86400
  Expires: Mon, 06 Apr 2020 13:59:05 GMT
  Cf-Ray: 57fc06a53887d286-DFW
  Retry-Count: 0
[402] http://shop.uk.teslamotors.com/
  Cf-Cache-Status: DYNAMIC
  X-Dc: gcp-us-central1,gcp-us-central1
  Date: Mon, 06 Apr 2020 13:54:49 GMT
  Cf-Ray: 57fc00c39c0b581d-DFW
  X-Request-Id: 79146367-4c68-4e1b-9784-31f76d51b60b
  Set-Cookie: __cfduid=d94fad82fbdc0c110cb03cbcf58d097e21586181289; expires=Wed, 06-
↪May-20 13:54:49 GMT; path=/; domain=.shop.uk.teslamotors.com; HttpOnly;␣
↪SameSite=Lax _shopify_y=e3f19482-99e9-46cd-af8d-89fb8557fd28; path=/; expires=Thu,␣
↪07 Apr 2022 01:33:13 GMT
  X-Shopid: 4232821
  Content-Language: en
  Alt-Svc: h3-27=":443"; ma=86400, h3-25=":443"; ma=86400, h3-24=":443"; ma=86400, h3-
↪23=":443"; ma=86400
  X-Content-Type-Options: nosniff
  X-Permitted-Cross-Domain-Policies: none
  X-Xss-Protection: 1; mode=block; report=/xss-report?source%5Baction%5D=index&source
↪%5Bapp%5D=Shopify&source%5Bcontroller%5D=storefront_section%2Fshop&source%5Bsection
↪%5D=storefront&source%5Buuid%5D=79146367-4c68-4e1b-9784-31f76d51b60b
  Server: cloudflare
  Content-Type: text/html; charset=utf-8
  X-Sorting-Hat-Shopid: 4232821
  X-Shardid: 78
  Content-Security-Policy: frame-ancestors *; report-uri /csp-report?source%5Baction
↪%5D=index&source%5Bapp%5D=Shopify&source%5Bcontroller%5D=storefront_section%2Fshop&
↪source%5Bsection%5D=storefront&source%5Buuid%5D=79146367-4c68-4e1b-9784-31f76d51b60b
  Retry-Count: 0
  X-Sorting-Hat-Podid: 78
  X-Shopify-Stage: production
  X-Download-Options: noopen
[301] http://shop.uk.teslamotors.com/assets
[db-2] recon-pipeline>
```

view nmap-scans

Nmap results can be filtered by host, NSE script type, scanned port, and product.

Show All Results

```
[db-2] recon-pipeline> view nmap-scans --paged
2600:9000:21d4:7800:c:d401:5a80:93a1 - http
===========================================

tcp port: 80 - open - syn-ack
product: Amazon CloudFront httpd :: None
nse script(s) output:
  http-server-header
    CloudFront
  http-title
    ERROR: The request could not be satisfied

...
```

Filter by product

```
[db-2] recon-pipeline> view nmap-scans --product "Splunkd httpd"
209.133.79.101 - http
=====================

tcp port: 443 - open - syn-ack
product: Splunkd httpd :: None
nse script(s) output:
  http-robots.txt
    1 disallowed entry
    /
  http-server-header
    Splunkd
  http-title
    404 Not Found
  ssl-cert
    Subject: commonName=*.teslamotors.com/organizationName=Tesla Motors, Inc./
↪stateOrProvinceName=California/countryName=US
    Subject Alternative Name: DNS:*.teslamotors.com, DNS:teslamotors.com
    Not valid before: 2019-01-17T00:00:00
    Not valid after:  2021-02-03T12:00:00
  ssl-date
    TLS randomness does not represent time
```

Filter by NSE Script

```
[db-2] recon-pipeline> view nmap-scans --nse-script ssl-cert --paged
199.66.9.47 - http-proxy
========================

tcp port: 443 - open - syn-ack
```

(continues on next page)

(continued from previous page)

```
product: Varnish http accelerator :: None
nse script(s) output:
  ssl-cert
    Subject: commonName=*.tesla.com/organizationName=Tesla, Inc./
↪stateOrProvinceName=California/countryName=US
    Subject Alternative Name: DNS:*.tesla.com, DNS:tesla.com
    Not valid before: 2020-02-07T00:00:00
    Not valid after:  2022-04-08T12:00:00

...
```

Filter by NSE Script and Port Number

```
[db-2] recon-pipeline> view nmap-scans --nse-script ssl-cert --port 8443
104.22.11.42 - https-alt
========================

tcp port: 8443 - open - syn-ack
product: cloudflare :: None
nse script(s) output:
  ssl-cert
    Subject: commonName=sni.cloudflaressl.com/organizationName=Cloudflare, Inc./
↪stateOrProvinceName=CA/countryName=US
    Subject Alternative Name: DNS:*.tesla.services, DNS:tesla.services, DNS:sni.
↪cloudflaressl.com
    Not valid before: 2020-02-13T00:00:00
    Not valid after:  2020-10-09T12:00:00
[db-2] recon-pipeline>
```

Filter by Host (ipv4/6 or domain name)

```
[db-2] recon-pipeline> view nmap-scans --host 2600:9000:21d4:3000:c:d401:5a80:93a1
2600:9000:21d4:3000:c:d401:5a80:93a1 - http
===========================================

tcp port: 80 - open - syn-ack
product: Amazon CloudFront httpd :: None
nse script(s) output:
  http-server-header
    CloudFront
  http-title
    ERROR: The request could not be satisfied

[db-2] recon-pipeline>
```

Include Command Used to Scan

The `--commandline` option will append the command used to scan the target to the results.

```
[db-2] recon-pipeline> view nmap-scans --host 2600:9000:21d4:3000:c:d401:5a80:93a1 --
↪commandline
2600:9000:21d4:3000:c:d401:5a80:93a1 - http
==========================================

tcp port: 80 - open - syn-ack
product: Amazon CloudFront httpd :: None
nse script(s) output:
  http-server-header
    CloudFront
  http-title
    ERROR: The request could not be satisfied
command used:
  nmap --open -sT -n -sC -T 4 -sV -Pn -p 80 -6 -oA /home/epi/PycharmProjects/recon-
↪pipeline/tests/data/tesla-results/nmap-results/nmap.
↪2600:9000:21d4:3000:c:d401:5a80:93a1-tcp 2600:9000:21d4:3000:c:d401:5a80:93a1

[db-2] recon-pipeline>
```

view ports

Port results are populated via masscan. Ports can be filtered by host and port number.

Show All Results

```
[db-2] recon-pipeline> view ports --paged
apmv3.go.tesla.services: 80
autodiscover.teslamotors.com: 80
csp.teslamotors.com: 443
image.emails.tesla.com: 443
marketing.teslamotors.com: 443
partnerleadsharing.tesla.com: 443
service.tesla.cn: 80
shop.uk.teslamotors.com: 8080
sip.tesla.cn: 5061
...
```

Filter by Host

```
[db-2] recon-pipeline> view ports --host tesla.services
tesla.services: 8443,8080
[db-2] recon-pipeline>
```

Filter by Port Number

```
[db-2] recon-pipeline> view ports --port-number 8443
tesla.services: 8443,8080
104.22.10.42: 8443,8080
104.22.11.42: 8443,8080
2606:4700:10::6816:a2a: 8443,8080
```

(continues on next page)

(continued from previous page)

```
2606:4700:10::6816:b2a: 8443,8080
[db-2] recon-pipeline>
```

view searchsploit-results

Searchsploit results can be filtered by host and type, the full path to any relevant exploit code can be shown as well.

Show All Results

```
[db-2] recon-pipeline> view searchsploit-results --paged
52.209.48.104, 34.252.120.214, 52.48.121.107, telemetry-eng.vn.tesla.services
=============================================================================
  local     | 40768.sh |  Nginx (Debian Based Distros + Gentoo) - 'logrotate' Local␣
↪Privilege
                       |    Escalation
  remote    | 12804.txt|  Nginx 0.6.36 - Directory Traversal
  local     | 14830.py |  Nginx 0.6.38 - Heap Corruption
  webapps   | 24967.txt|  Nginx 0.6.x - Arbitrary Code Execution NullByte Injection
  dos       | 9901.txt |  Nginx 0.7.0 < 0.7.61 / 0.6.0 < 0.6.38 / 0.5.0 < 0.5.37 / 0.4.
↪0 <
                       |    0.4.14 - Denial of Service (PoC)
  remote    | 9829.txt |  Nginx 0.7.61 - WebDAV Directory Traversal
  remote    | 33490.txt|  Nginx 0.7.64 - Terminal Escape Sequence in Logs Command␣
↪Injection
  remote    | 13822.txt|  Nginx 0.7.65/0.8.39 (dev) - Source Disclosure / Download
  remote    | 13818.txt|  Nginx 0.8.36 - Source Disclosure / Denial of Service
  remote    | 38846.txt|  Nginx 1.1.17 - URI Processing SecURIty Bypass
  remote    | 25775.rb |  Nginx 1.3.9 < 1.4.0 - Chuncked Encoding Stack Buffer Overflow
                       |    (Metasploit)
  dos       | 25499.py |  Nginx 1.3.9 < 1.4.0 - Denial of Service (PoC)
  remote    | 26737.pl |  Nginx 1.3.9/1.4.0 (x86) - Brute Force
  remote    | 32277.txt|  Nginx 1.4.0 (Generic Linux x64) - Remote Overflow
  webapps   | 47553.md |  PHP-FPM + Nginx - Remote Code Execution
...
```

Filter by Host

```
[db-2] recon-pipeline> view searchsploit-results --paged --host telemetry-eng.vn.
↪tesla.services
52.209.48.104, 34.252.120.214, 52.48.121.107, telemetry-eng.vn.tesla.services
=============================================================================
  local     | 40768.sh |  Nginx (Debian Based Distros + Gentoo) - 'logrotate' Local␣
↪Privilege
                       |    Escalation
  remote    | 12804.txt|  Nginx 0.6.36 - Directory Traversal
  local     | 14830.py |  Nginx 0.6.38 - Heap Corruption
  webapps   | 24967.txt|  Nginx 0.6.x - Arbitrary Code Execution NullByte Injection
  dos       | 9901.txt |  Nginx 0.7.0 < 0.7.61 / 0.6.0 < 0.6.38 / 0.5.0 < 0.5.37 / 0.4.
↪0 <
                       |    0.4.14 - Denial of Service (PoC)
  remote    | 9829.txt |  Nginx 0.7.61 - WebDAV Directory Traversal
```

(continues on next page)

(continued from previous page)

```
  remote    | 33490.txt|  Nginx 0.7.64 - Terminal Escape Sequence in Logs Command␣
↪Injection
  remote    | 13822.txt|  Nginx 0.7.65/0.8.39 (dev) - Source Disclosure / Download
  remote    | 13818.txt|  Nginx 0.8.36 - Source Disclosure / Denial of Service
  remote    | 38846.txt|  Nginx 1.1.17 - URI Processing SecURIty Bypass
  remote    | 25775.rb |  Nginx 1.3.9 < 1.4.0 - Chuncked Encoding Stack Buffer Overflow
                       |    (Metasploit)
  dos       | 25499.py |  Nginx 1.3.9 < 1.4.0 - Denial of Service (PoC)
  remote    | 26737.pl |  Nginx 1.3.9/1.4.0 (x86) - Brute Force
  remote    | 32277.txt|  Nginx 1.4.0 (Generic Linux x64) - Remote Overflow
  webapps   | 47553.md |  PHP-FPM + Nginx - Remote Code Execution
[db-2] recon-pipeline>
```

Filter by Type

```
[db-2] recon-pipeline> view searchsploit-results --paged  --type webapps
52.209.48.104, 34.252.120.214, 52.48.121.107, telemetry-eng.vn.tesla.services
============================================================================
  webapps   | 24967.txt|  Nginx 0.6.x - Arbitrary Code Execution NullByte Injection
  webapps   | 47553.md |  PHP-FPM + Nginx - Remote Code Execution
...
```

Include Full Path to Exploit Code

```
52.209.48.104, 34.252.120.214, 52.48.121.107, telemetry-eng.vn.tesla.services
============================================================================
  webapps |  Nginx 0.6.x - Arbitrary Code Execution NullByte Injection
          |  /home/epi/.recon-tools/exploitdb/exploits/multiple/webapps/24967.txt
  webapps |  PHP-FPM + Nginx - Remote Code Execution
          |  /home/epi/.recon-tools/exploitdb/exploits/php/webapps/47553.md
...
```

view targets

Target results can be filtered by type and whether or not they've been reported as vulnerable to subdomain takeover.

Show All Results

```
[db-2] recon-pipeline> view targets --paged

3.tesla.com
api-internal.sn.tesla.services
api-toolbox.tesla.com
api.mp.tesla.services
api.sn.tesla.services
api.tesla.cn
...
```

Filter by Target Type

```
[db-2] recon-pipeline> view targets --type ipv6 --paged
2600:1404:23:183::358f
2600:1404:23:188::3fe7
2600:1404:23:18f::700
2600:1404:23:190::700
2600:1404:23:194::16cf
...
```

Filter by Possibility of Subdomain Takeover

```
[db-2] recon-pipeline> view targets --paged --vuln-to-subdomain-takeover
[vulnerable] api-internal.sn.tesla.services
...
```

view web-technologies

Web technology results are produced by webanalyze. Web technology results can be filtered by host, type, and product.

Show All Results

```
[db-2] recon-pipeline> view web-technologies --paged
Varnish (Caching)
=================

   - inventory-assets.tesla.com
   - www.tesla.com
   - errlog.tesla.com
   - static-assets.tesla.com
   - partnerleadsharing.tesla.com
   - 199.66.9.47
   - onboarding-pre-delivery-prod.teslamotors.com
   - 2600:1404:23:194::16cf
   - 2600:1404:23:196::16cf
...
```

Filter by Technology Type

```
[db-2] recon-pipeline> view web-technologies --type "Programming languages"
PHP (Programming languages)
===========================

   - www.tesla.com
   - dummy.teslamotors.com
   - 209.10.208.20
   - 211.147.80.206
   - trt.tesla.com
   - trt.teslamotors.com
```

(continues on next page)

(continued from previous page)

```
   - cn-origin.teslamotors.com
   - www.tesla.cn
   - events.tesla.cn
   - 23.67.209.106
   - service.teslamotors.com

Python (Programming languages)
==============================

   - api-toolbox.tesla.com
   - 52.26.53.228
   - 34.214.187.20
   - 35.166.29.132
   - api.toolbox.tb.tesla.services
   - toolbox.teslamotors.com
   - 209.133.79.93

Ruby (Programming languages)
============================

   - storagesim.teslamotors.com
   - 209.10.208.39
...
```

Filter by Product

```
[db-2] recon-pipeline> view web-technologies --product OpenResty-1.15.8.2
OpenResty-1.15.8.2 (Web servers)
================================

   - links.tesla.com

[db-2] recon-pipeline>
```

Filter by Host

```
[db-2] recon-pipeline> view web-technologies --host api-toolbox.tesla.com
api-toolbox.tesla.com
=====================
   - gunicorn-19.4.5 (Web servers)
   - Python (Programming languages)
[db-2] recon-pipeline>
```

1.1.5 Manually interacting with the Database

If for whatever reason you'd like to query the database manually, from within the recon-pipeline shell, you can use the `py` command to drop into a python REPL with your current ReconShell instance available as `self`.

```
./pipeline/recon-pipeline.py
recon-pipeline> py
```

(continues on next page)

(continued from previous page)

```
Python 3.7.5 (default, Nov 20 2019, 09:21:52)
[GCC 9.2.1 20191008] on linux
Type "help", "copyright", "credits" or "license" for more information.

End with `Ctrl-D` (Unix) / `Ctrl-Z` (Windows), `quit()`, `exit()`.
Non-Python commands can be issued with: app("your command")

>>> self
<__main__.ReconShell object at 0x7f69f457f790>
```

Once in the REPL, the currently connected database is available as `self.db_mgr`. The database is an instance of *Database Manager* and has a `session` attribute which can be used to issue manual SQLAlchemy style queries.

```
>>> from pipeline.models.port_model import Port
>>> self.db_mgr.session.query(Port).filter_by(port_number=443)
<sqlalchemy.orm.query.Query object at 0x7f8cef804250>
>>>
```

1.1.6 Using a Scheduler

The backbone of this pipeline is spotify's luigi batch process management framework. Luigi uses the concept of a scheduler in order to manage task execution. Two types of scheduler are available, a **local** scheduler and a **central** scheduler. The local scheduler is useful for development and debugging while the central scheduler provides the following two benefits:

- Make sure two instances of the same task are not running simultaneously
- Provide *visualization* of everything that's going on

While in the `recon-pipeline` shell, running `tools install luigi-service` will copy the `luigid.service` file provided in the repo to its appropriate systemd location and start/enable the service. The result is that the central scheduler is up and running easily.

The other option is to add `--local-scheduler` to your *scan* command from within the `recon-pipeline` shell.

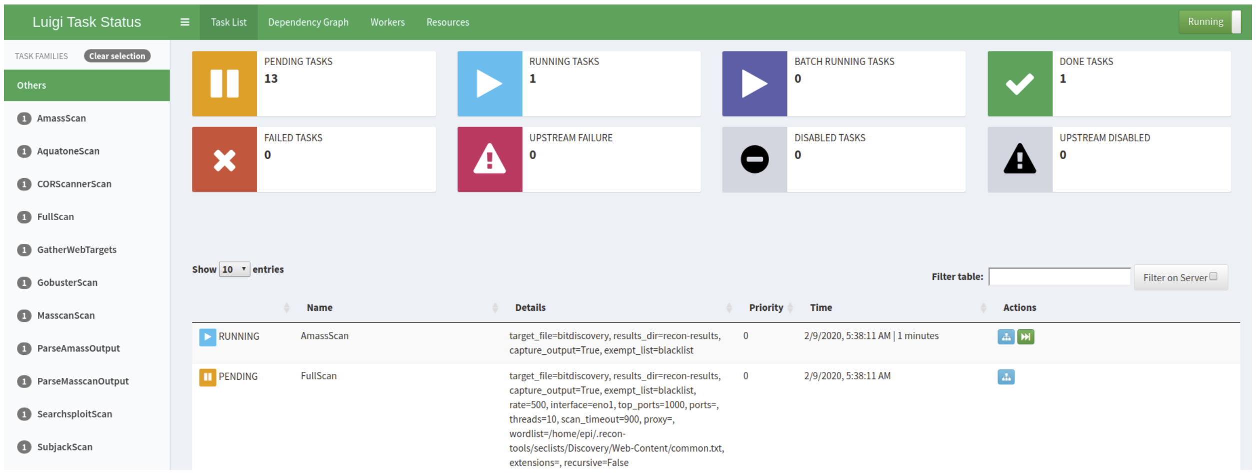
1.1.7 Visualizing Tasks

Setup

To use the web console, you'll need to *install the luigid service*. Assuming you've already installed `pipenv` and created a virtual environment, you can simply run the `tools install luigi-service` from within the pipeline.

Dashboard

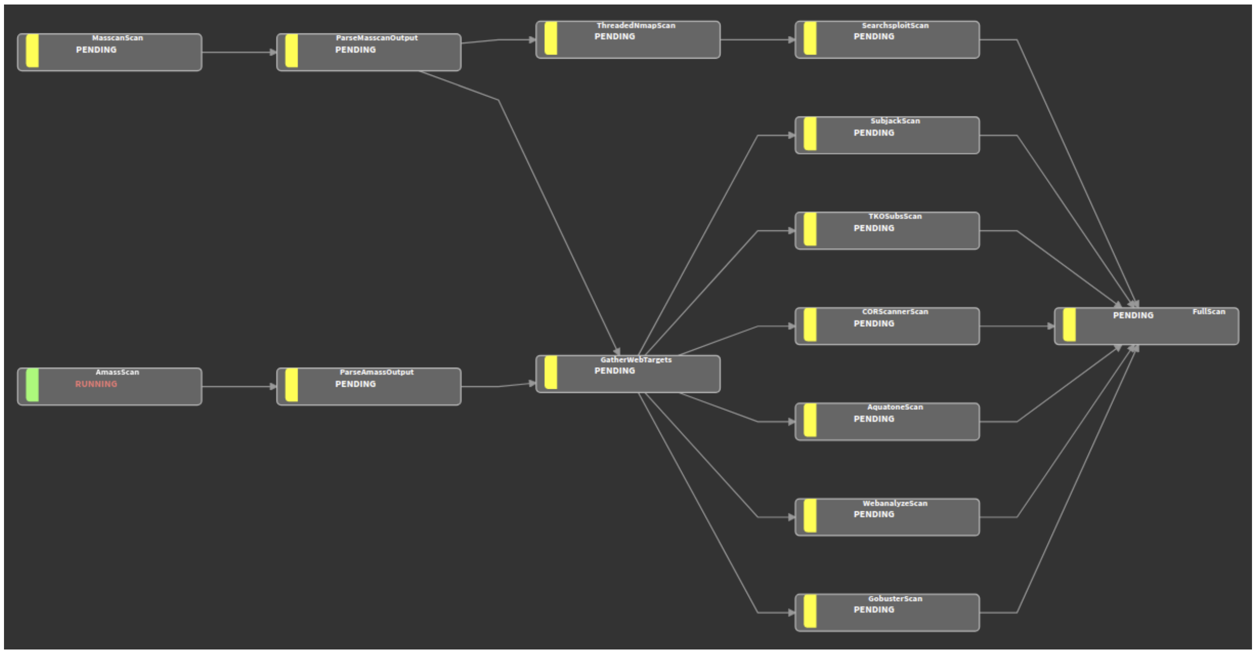
If you're using the *central scheduler*, you'll be able to use luigi's web console to see a dashboard style synopsis of your tasks.



	Name	Details	Priority	Time	Actions
RUNNING	AmassScan	target_file=bitdiscovery, results_dir=recon-results, capture_output=True, exempt_list=blacklist	0	2/9/2020, 5:38:11 AM | 1 minutes	
PENDING	FullScan	target_file=bitdiscovery, results_dir=recon-results, capture_output=True, exempt_list=blacklist, rate=500, interface=eno1, top_ports=1000, ports=, threads=10, scan_timeout=900, proxy=, wordlist=/home/epi/.recon-tools/seclists/Discovery/Web-Content/common.txt, extensions=, recursive=False	0	2/9/2020, 5:38:11 AM	

Dependency Graph

You can use the **Dependency Graph** link at the top of the dashboard to view your current task along with any up/downstream tasks that are queued.



Make it So

To view the console from within `recon-pipeline`, you can run the *status* command or add `--sausage` to your scan command at execution time. The web console runs on port **8082** by default, so at any time you can also just use your favorite browser to check it out manually as well.

There are an accompanying set of blog posts detailing the development process and underpinnings of the pipeline. Feel free to check them out if you're so inclined, but they're in no way required reading to use the tool.

- *Installation Instructions* - How to install `recon-pipeline` and associated dependencies
- *Defining Target Scope* - How to define the scope of your scans (list of targets and a blacklist)

- *Running Scans* - Example scan of **tesla.com** using `recon-pipeline`
- *Viewing Scan Results* - How to view scan results
- *Using a Scheduler* - The Luigi schedulers and which to choose
- *Visualizing Tasks* - How to check on active tasks once they're running

CHAPTER 2

Personalization

2.1 Making Changes to the pipeline

2.1.1 Add a New Scanner

The process of adding a new scanner is relatively simple. The steps are outlined below.

Create a tool definition file

This step isn't strictly necessary, but if you want the pipeline to know how to install/uninstall the tool your scanner uses, this is where that is defined. Tool definition files live in the `pipeline/tools` directory.

```
pipeline/
...
├── recon-pipeline.py
└── tools
    ├── amass.yaml
    ├── aquatone.yaml
    ...
```

Tool Definition Required Fields

Create a `.yaml` file with the following fields.

Field Name	Type	Description	Required
`commands`	Array of strings	Which commands to run to install the tool	True
`dependencies`	Array of strings	Each dependency must be defined in a separate definition file, as they'll be installed before the current defintion's tool	False
`environ`	Dictionary	Use this if you need to pass information via the environment to your tool (amass.yaml has an example)	False
`shell`	Boolean	true means each command in commands will be run via `/bin/sh -c` (see Popen's `shell` argument for more details)	False

Useful yaml Helpers

`pipeline.tools.loader` defines a few helpful functions to assist with dynamically creating values in yaml files as well as linking user-defined configuration values.

Dynamically creating strings and filesystem paths are handled by the following two functions.

- `!join` - join items in an array with a space character
- `!join_path` - join items in an array with a `/` character

In order to get values out of `pipeline.recon.config.py`, you'll need to use one of the yaml helpers listed below.

- `!get_default` - get a value from the `pipeline.recon.config.defaults` dictionary
- `!get_tool_path` - get a path value from the `pipeline.tools.tools` dictionary

Simple Example Tool Definition

The example below needs go to be installed prior to being installed itself. It then grabs the path to the `go` binary from `pipeline.tools.tools` by using `!get_tool_path`. After that, it creates a command using `!join` that will look like `/usr/local/go/bin/go get github.com/tomnomnom/waybackurls`. This command will be run by the `install waybackurls` command (or `install all`).

```
dependencies: [go]
go: &gobin !get_tool_path "{go[path]}"

commands:
- !join [*gobin, get github.com/tomnomnom/waybackurls]
```

If you're looking for a more complex example, check out `searchsploit.yaml`.

Write Your Scanner Class

You can find an abundance of information on how to write your scanner class starting with Part II of the blog posts tied to recon-pipeline's creation. Because scanner classes are covered in so much detail there, we'll only briefly summarize the steps here:

- Select `luigi.Task` or `luigi.ExternalTask` as your base class. Task allows more flexibility while ExternalTask is great for simple scans.
- Implement the `requires`, `output`, and either `run` (Task) or `program_args` (ExternalTask) methods

Add Your Scan to a Wrapper (optional)

If you want to run your new scan as part of an existing pipeline, open up `pipeline.recon.wrappers` and edit one of the existing wrappers (or add your own) to include your new scan. You should be able to import your new scan, and then add a `yield MyNewScan(**args)` in order to add it to the pipeline. The only gotcha here is that depending on what arguments your scan takes, you may need to strategically place your scan within the wrapper in order to ensure it doesn't get any arguments that it doesn't expect.

2.1.2 Create a New Wrapper Scan

If for whatever reason you want something other than FullScan, the process for defining a new scan is relatively simple. The `HTBScan` is a good example.

1. Define your new class, inheriting from **luigi.WrapperTask** and use the `inherits` decorator to include any scan you want to utilize

```
@inherits(SearchsploitScan, AquatoneScan, GobusterScan, WebanalyzeScan)
class HTBScan(luigi.WrapperTask):
    ...
```

2. Include all parameters needed by any of the scans passed to `inherits`

```
def requires(self):
    """ HTBScan is a wrapper, as such it requires any Tasks that it wraps. """
    args = {
        "results_dir": self.results_dir,
        "rate": self.rate,
        "target_file": self.target_file,
        "top_ports": self.top_ports,
        "interface": self.interface,
        "ports": self.ports,
        "exempt_list": self.exempt_list,
        "threads": self.threads,
        "proxy": self.proxy,
        "wordlist": self.wordlist,
        "extensions": self.extensions,
        "recursive": self.recursive,
    }
    ...
```

3. `yield` from each scan, keeping in mind that some of the parameters won't be universal (i.e. need to be removed/added)

```
def requires(self):
    """ HTBScan is a wrapper, as such it requires any Tasks that it wraps. """
    ...

    yield GobusterScan(**args)

    # remove options that are gobuster specific; if left dictionary unpacking to␣
↪other scans throws an exception
    for gobuster_opt in ("proxy", "wordlist", "extensions", "recursive"):
        del args[gobuster_opt]

    # add aquatone scan specific option
    args.update({"scan_timeout": self.scan_timeout})
```

(continues on next page)

(continued from previous page)

```
    yield AquatoneScan(**args)

    del args["scan_timeout"]

    yield SearchsploitScan(**args)
    yield WebanalyzeScan(**args)
```

There are a few things you can do to modify the pipeline to your own specifications:

- *Add a New Scanner*
- *Create a New Wrapper Scan*

CHAPTER 3

API Reference

3.1 Commands

`recon-pipeline` provides a handful of commands:

- *tools*
- *scan*
- *status*
- *database*
- *view*

All other available commands are inherited from cmd2.

3.1.1 tools

```
Usage: tools [-h] {install, uninstall, reinstall, list} ...
```

Sub-commands:

install

Install any/all of the libraries/tools necessary to make the recon-pipeline function

```
tools install [-h]
              {exploitdb, searchsploit, go, amass, webanalyze, waybackurls,
              luigi-service, masscan, subjack, tko-subs, seclists, gobuster,
              aquatone, recursive-gobuster, all}
```

Positional Arguments

tool	Possible choices: exploitdb, searchsploit, go, amass, webanalyze, waybackurls, luigi-service, masscan, subjack, tko-subs, seclists, gobuster, aquatone, recursive-gobuster, all which tool to install

uninstall

Remove the already installed tool

```
tools uninstall [-h]
                {exploitdb, searchsploit, go, amass, webanalyze, waybackurls,
                luigi-service, masscan, subjack, tko-subs, seclists, gobuster,
                aquatone, recursive-gobuster, all}
```

Positional Arguments

tool	Possible choices: exploitdb, searchsploit, go, amass, webanalyze, waybackurls, luigi-service, masscan, subjack, tko-subs, seclists, gobuster, aquatone, recursive-gobuster, all which tool to uninstall

reinstall

Uninstall and then Install a given tool

```
tools reinstall [-h]
                {exploitdb, searchsploit, go, amass, webanalyze, waybackurls,
                luigi-service, masscan, subjack, tko-subs, seclists, gobuster,
                aquatone, recursive-gobuster, all}
```

Positional Arguments

tool	Possible choices: exploitdb, searchsploit, go, amass, webanalyze, waybackurls, luigi-service, masscan, subjack, tko-subs, seclists, gobuster, aquatone, recursive-gobuster, all which tool to reinstall

list

Show status of pipeline tools

```
tools list [-h]
```

3.1.2 database

```
Usage: database [-h] {list, delete, attach, detach} ...
```

Sub-commands:

list

List all known databases

```
database list [-h]
```

delete

Delete the selected database

```
database delete [-h]
```

attach

Attach to the selected database

```
database attach [-h]
```

detach

Detach from the currently attached database

```
database detach [-h]
```

3.1.3 scan

```
Usage: scan [-h] (--target-file TARGET_FILE | --target TARGET)
            [--exempt-list EXEMPT_LIST] [--results-dir RESULTS_DIR]
            [--wordlist WORDLIST] [--interface INTERFACE] [--recursive]
            [--rate RATE] [--top-ports TOP_PORTS | --ports PORTS]
            [--threads THREADS] [--scan-timeout SCAN_TIMEOUT] [--proxy PROXY]
            [--extensions EXTENSIONS] [--sausage] [--local-scheduler]
            [--verbose]
            scantype
```

Positional Arguments

scantype	which type of scan to run

Named Arguments

--target-file	file created by the user that defines the target's scope; list of ips/domains
--target	ip or domain to target
--exempt-list	list of blacklisted ips/domains
--results-dir	directory in which to save scan results (default: recon-results) Default: "recon-results"
--wordlist	path to wordlist used by gobuster (default: /home/docs/.local/recon-pipeline/tools/seclists/Discovery/Web-Content/common.txt)
--interface	which interface masscan should use (default: tun0)
--recursive	whether or not to recursively gobust (default: False) Default: False
--rate	rate at which masscan should scan (default: 1000)
--top-ports	ports to scan as specified by nmap's list of top-ports (only meaningful to around 5000)
--ports	port specification for masscan (all ports example: 1-65535,U:1-65535)
--threads	number of threads for all of the threaded applications to use (default: 10)
--scan-timeout	scan timeout for aquatone (default: 900)
--proxy	proxy for gobuster if desired (ex. 127.0.0.1:8080)
--extensions	list of extensions for gobuster (ex. asp,html,aspx)
--sausage	open a web browser to Luigi's central scheduler's visualization site (see how the sausage is made!) Default: False
--local-scheduler	use the local scheduler instead of the central scheduler (luigid) (default: False) Default: False
--verbose	shows debug messages from luigi, useful for troubleshooting (default: False) Default: False

3.1.4 status

```
Usage: status [-h] [--port PORT] [--host HOST]
```

Named Arguments

--port	port on which the luigi central scheduler's visualization site is running (default: 8082) Default: "8082"
--host	host on which the luigi central scheduler's visualization site is running (default: localhost) Default: "127.0.0.1"

3.1.5 view

```
Usage: view [-h]
            {targets, web-technologies, endpoints, nmap-scans,
            searchsploit-results, ports} ...
```

Sub-commands:

targets

List all known targets (ipv4/6 & domain names); produced by amass

```
view targets [-h] [--vuln-to-subdomain-takeover]
             [--type {ipv4, ipv6, domain-name}] [--paged]
```

Named Arguments

--vuln-to-subdomain-takeover	show targets identified as vulnerable to subdomain takeover Default: False
--type	Possible choices: ipv4, ipv6, domain-name filter by target type
--paged	display output page-by-page (default: False) Default: False

web-technologies

List all known web technologies identified; produced by webanalyze

```
view web-technologies [-h] [--paged] [--host HOST] [--type TYPE]
                      [--product PRODUCT]
```

Named Arguments

--paged	display output page-by-page (default: False) Default: False
--host	filter results by host
--type	filter results by type
--product	filter results by product

endpoints

List all known endpoints; produced by gobuster

```
view endpoints [-h] [--headers] [--paged] [--plain]
               [--status-code STATUS_CODE] [--host HOST]
```

Named Arguments

--headers	include headers found at each endpoint (default: False) Default: False
--paged	display output page-by-page (default: False) Default: False
--plain	display without status-codes/color (default: False) Default: False
--status-code	filter results by status code
--host	filter results by host

nmap-scans

List all known nmap scan results; produced by nmap

```
view nmap-scans [-h] [--paged] [--commandline] [--host HOST]
                [--nse-script NSE_SCRIPT] [--port PORT] [--product PRODUCT]
```

Named Arguments

--paged	display output page-by-page (default: False) Default: False
--commandline	display command used to scan (default: False) Default: False
--host	filter results by host
--nse-script	filter results by nse script type ran
--port	filter results by port scanned
--product	filter results by reported product

searchsploit-results

List all known searchsploit hits; produced by searchsploit

```
view searchsploit-results [-h] [--paged] [--fullpath] [--host HOST]
                          [--type TYPE]
```

Named Arguments

--paged	display output page-by-page (default: False) Default: False
--fullpath	display full path to exploit PoC (default: False) Default: False
--host	filter results by host
--type	filter results by exploit type

ports

List all known open ports; produced by masscan

```
view ports [-h] [--paged] [--host HOST] [--port-number PORT_NUMBER]
```

Named Arguments

--paged	display output page-by-page (default: False) Default: False
--host	filter results by host
--port-number	filter results by port number

3.2 Database Manager

class pipeline.models.db_manager.**DBManager**(*db_location*)
Class that encapsulates database transactions and queries

add(*item*)
Simple helper to add a record to the database

add_ipv4_or_v6_address_to_target(*tgt*, *ipaddr*)
Simple helper that adds an appropriate IPAddress to the given target

close()
Simple helper to close the database session

get_all_endpoints()
Simple helper that returns all Endpoints from the database

get_all_exploit_types()
Simple helper that returns all exploit types reported by searchsploit

get_all_hostnames() → list
Simple helper to return all hostnames from Target records

get_all_ipv4_addresses() → list
Simple helper to return all ipv4 addresses from Target records

get_all_ipv6_addresses() → list
Simple helper to return all ipv6 addresses from Target records

get_all_nmap_reported_products()
Simple helper that returns all products reported by nmap

get_all_nse_script_types()
Simple helper that returns all NSE Script types from the database

get_all_port_numbers()
Simple helper that returns all Port.port_numbers from the database

get_all_targets()
Simple helper to return all ipv4/6 and hostnames produced by running amass

get_all_web_targets()
Simple helper that returns all Targets tagged as having an open web port

get_and_filter(*model*, *defaults=None*, ***kwargs*)
Simple helper to either get an existing record if it exists otherwise create and return a new instance

get_endpoint_by_status_code(*code*)
Simple helper that returns all Endpoints filtered by status code

get_endpoints_by_ip_or_hostname(*ip_or_host*)
Simple helper that returns all Endpoints filtered by ip or hostname

get_nmap_scans_by_ip_or_hostname(*ip_or_host*)
Simple helper that returns all Endpoints filtered by ip or hostname

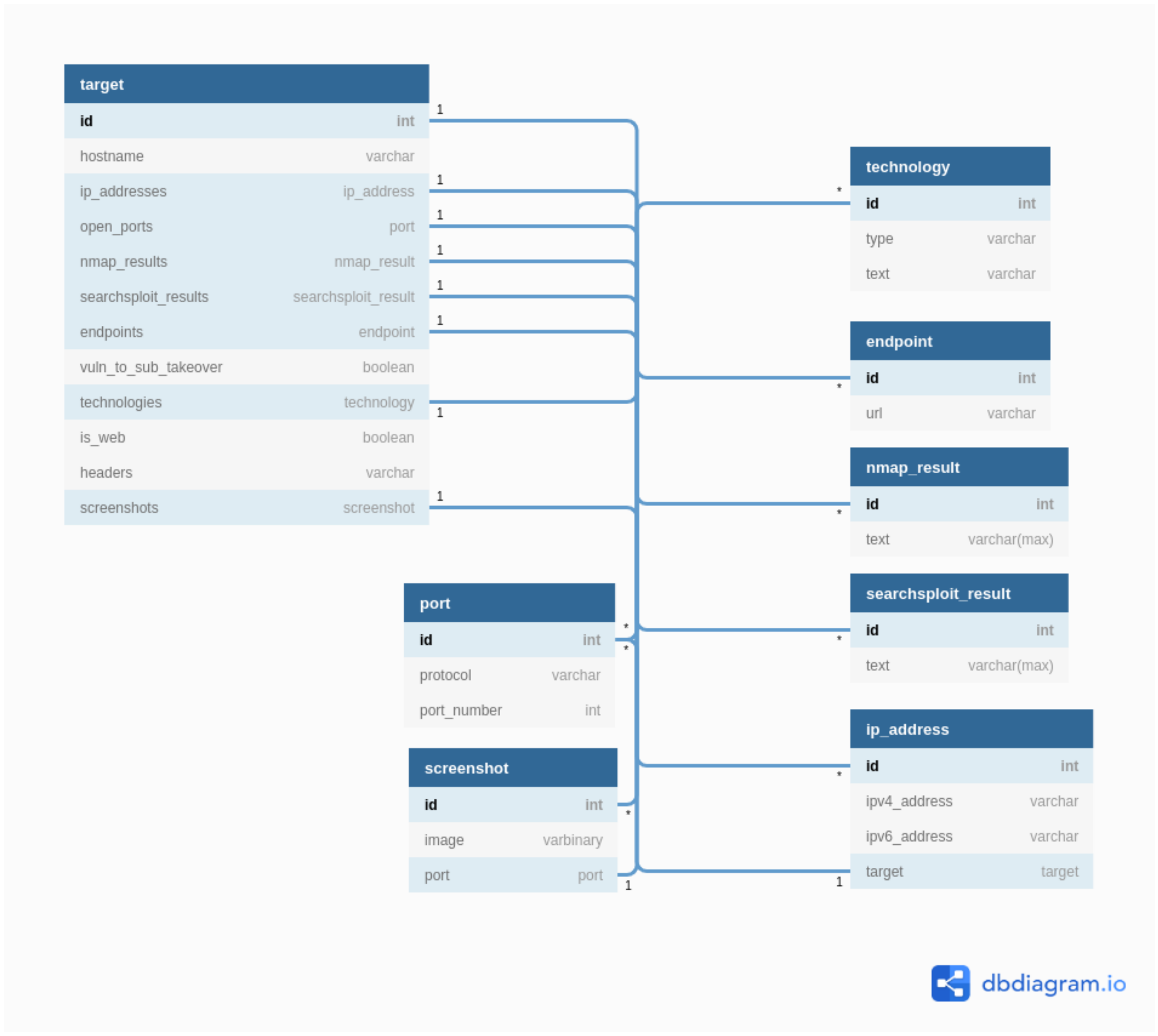
get_or_create(*model*, ***kwargs*)
Simple helper to either get an existing record if it exists otherwise create and return a new instance

get_or_create_target_by_ip_or_hostname(*ip_or_host*)
Simple helper to query a Target record by either hostname or ip address, whichever works

get_ports_by_ip_or_host_and_protocol(*ip_or_host*, *protocol*)
Simple helper that returns all ports based on the given protocol and host

get_status_codes()
Simple helper that returns all status codes found during scanning

3.3 Database Models



3.3.1 Target Model

class `pipeline.models.target_model.`**`Target`**`(**kwargs)`

Database model that describes a target; This is the model that functions as the "top" model.

Relationships: `ip_addresses`: one to many -> *`pipeline.models.ip_address_model.IPAddress`*

`open_ports`: many to many -> *`pipeline.models.port_model.Port`*

`nmap_results`: one to many -> *`pipeline.models.nmap_model.NmapResult`*

`searchsploit_results`: one to many -> *`pipeline.models.searchsploit_model.SearchsploitResult`*

`endpoints`: one to many -> *`pipeline.models.endpoint_model.Endpoint`* `technologies`: many to many -> *`pipeline.models.technology_model.Technology`*

`screenshots`: one to many -> *`pipeline.models.screenshot_model.Screenshot`*

3.3.2 Endpoint Model

class `pipeline.models.endpoint_model.`**`Endpoint`**(***kwargs*)

Database model that describes a URL/endpoint.

Represents gobuster data.

Relationships: `target`: many to one -> *`pipeline.models.target_model.Target`*

`headers`: many to many -> *`pipeline.models.header_model.Header`*

3.3.3 Header Model

class `pipeline.models.header_model.`**`Header`**(***kwargs*)

Database model that describes an http header (i.e. Server=cloudflare).

Relationships: `endpoints`: many to many -> `pipeline.models.target_model.Endpoint`

3.3.4 IP Address Model

class `pipeline.models.ip_address_model.`**`IPAddress`**(***kwargs*)

Database model that describes an ip address (ipv4 or ipv6).

Represents amass data or targets specified manually as part of the `target-file`.

Relationships: `target`: many to one -> *`pipeline.models.target_model.Target`*

3.3.5 Nmap Model

class `pipeline.models.nmap_model.`**`NmapResult`**(***kwargs*)

Database model that describes the TARGET.nmap scan results.

Represents nmap data.

Relationships: `target`: many to one -> *`pipeline.models.target_model.Target`*

`ip_address`: one to one -> *`pipeline.models.ip_address_model.IPAddress`*

`port`: one to one -> *`pipeline.models.port_model.Port`*

`nse_results`: one to many -> *`pipeline.models.nse_model.NSEResult`*

3.3.6 Nmap Scripting Engine Model

class `pipeline.models.nse_model.`**`NSEResult`**(***kwargs*)

Database model that describes the NSE script executions as part of an nmap scan.

Represents NSE script data.

Relationships: `NmapResult`: many to many -> *`pipeline.models.nmap_model.NmapResult`*

3.3.7 Port Model

class `pipeline.models.port_model.`**`Port`**(***kwargs*)

Database model that describes a port (tcp or udp).

Relationships: `targets`: many to many -> *`pipeline.models.target_model.Target`*

3.3.8 Screenshot Model

class `pipeline.models.screenshot_model.`**`Screenshot`**(***kwargs*)

Database model that describes a screenshot of a given webpage hosted on a `Target`.

Represents aquatone data.

Relationships: `port`: one to one -> *`pipeline.models.port_model.Port`*

`target`: many to one -> *`pipeline.models.target_model.Target`*

`endpoint`: one to one -> *`pipeline.models.endpoint_model.Endpoint`*

`similar_pages`: black magic -> *`pipeline.models.screenshot_model.Screenshot`*

3.3.9 Searchsploit Model

class `pipeline.models.searchsploit_model.`**`SearchsploitResult`**(***kwargs*)

Database model that describes results from running searchsploit –nmap TARGET.xml.

Represents searchsploit data.

Relationships: `target`: many to one -> *`pipeline.models.target_model.Target`*

3.3.10 Technology Model

class `pipeline.models.technology_model.`**`Technology`**(***kwargs*)

Database model that describes a web technology (i.e. Nginx 1.14).

Represents webanalyze data.

Relationships: `targets`: many to many -> *`pipeline.models.target_model.Target`*

3.4 Parsers

3.4.1 Amass Parser

class `pipeline.recon.amass.`**`ParseAmassOutput`**(**args*, ***kwargs*)

Read amass JSON results and create categorized entries into ip|subdomain files.

Parameters

- **`db_location`** – specifies the path to the database used for storing results *Required by upstream Task*
- **`target_file`** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*

- **`exempt_list`** – Path to a file providing blacklisted subdomains, one per line. *Optional by upstream Task*
- **`results_dir`** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

`output`()

Returns the target output files for this task.

Returns luigi.contrib.sqla.SQLAlchemyTarget

`requires`()

ParseAmassOutput depends on AmassScan to run.

TargetList expects target_file as a parameter. AmassScan accepts exempt_list as an optional parameter.

Returns luigi.ExternalTask - TargetList

`run`()

Parse the json file produced by AmassScan and categorize the results into ip|subdomain files.

An example (prettified) entry from the json file is shown below

{ "Timestamp": "2019-09-22T19:20:13-05:00", "name": "beta-partners.tesla.com", "domain": "tesla.com", "addresses": [

{ "ip": "209.133.79.58", "cidr": "209.133.79.0/24", "asn": 394161, "desc": "TESLA - Tesla"

}

], "tag": "ext", "source": "Previous Enum"

}

3.4.2 Web Targets Parser

class `pipeline.recon.web.targets.`**`GatherWebTargets`**(**args*, ***kwargs*)

Gather all subdomains as well as any ip addresses known to have a configured web port open.

Parameters

- **`db_location`** – specifies the path to the database used for storing results *Required by upstream Task*
- **`exempt_list`** – Path to a file providing blacklisted subdomains, one per line. *Optional by upstream Task*
- **`top_ports`** – Scan top N most popular ports *Required by upstream Task*
- **`ports`** – specifies the port(s) to be scanned *Required by upstream Task*
- **`interface`** – use the named raw network interface, such as "eth0" *Required by upstream Task*
- **`rate`** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **`target_file`** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **`results_dir`** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

output()
Returns the target output for this task.

Returns luigi.contrib.sqla.SQLAlchemyTarget

requires()
GatherWebTargets depends on ParseMasscanOutput and ParseAmassOutput to run.

ParseMasscanOutput expects rate, target_file, interface, and either ports or top_ports as parameters. ParseAmassOutput accepts exempt_list and expects target_file

Returns ParseMasscanOutput, str: ParseAmassOutput)

Return type dict(str

run()
Gather all potential web targets and tag them as web in the database.

3.4.3 Masscan Parser

class pipeline.recon.masscan.**ParseMasscanOutput**(**args*, ***kwargs*)
Read masscan JSON results and create a pickled dictionary of pertinent information for processing.

Parameters

- **top_ports** – Scan top N most popular ports *Required by upstream Task*
- **ports** – specifies the port(s) to be scanned *Required by upstream Task*
- **interface** – use the named raw network interface, such as "eth0" *Required by upstream Task*
- **rate** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **db_location** – specifies the path to the database used for storing results *Required by upstream Task*
- **target_file** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **results_dir** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

output()
Returns the target output for this task.

Naming convention for the output file is masscan.TARGET_FILE.parsed.pickle.

Returns luigi.local_target.LocalTarget

requires()
ParseMasscanOutput depends on Masscan to run.

Masscan expects rate, target_file, interface, and either ports or top_ports as parameters.

Returns luigi.Task - Masscan

run()
Reads masscan JSON results and creates a pickled dictionary of pertinent information for processing.

3.5 Scanners

3.5.1 Amass Scanner

class pipeline.recon.amass.**AmassScan**(**args*, ***kwargs*)

Run `amass` scan to perform subdomain enumeration of given domain(s).

Note: Expects **TARGET_FILE.domains** file to be a text file with one top-level domain per line.

Install:

```
sudo apt-get install -y -q amass
```

Basic Example:

```
amass enum -ip -brute -active -min-for-recursive 3 -df tesla -json amass.
↪tesla.json
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.amass AmassScan --
↪target-file tesla
```

Parameters

- **exempt_list** – Path to a file providing blacklisted subdomains, one per line.
- **db_location** – specifies the path to the database used for storing results *Required by upstream Task*
- **target_file** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **results_dir** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

output()

Returns the target output for this task.

Naming convention for the output file is amass.json.

Returns luigi.local_target.LocalTarget

requires()

AmassScan depends on TargetList to run.

TargetList expects target_file as a parameter.

Returns luigi.ExternalTask - TargetList

run()

Defines the options/arguments sent to amass after processing.

Returns list of options/arguments, beginning with the name of the executable to run

Return type list

3.5.2 Aquatone Scanner

class `pipeline.recon.web.aquatone.`**`AquatoneScan`**`(*args, **kwargs)`

Screenshot all web targets and generate HTML report.

Install:

```
mkdir /tmp/aquatone
wget -q https://github.com/michenriksen/aquatone/releases/download/v1.7.0/
↪aquatone_linux_amd64_1.7.0.zip -O /tmp/aquatone/aquatone.zip
unzip /tmp/aquatone/aquatone.zip -d /tmp/aquatone
sudo mv /tmp/aquatone/aquatone /usr/local/bin/aquatone
rm -rf /tmp/aquatone
```

Basic Example: `aquatone` commands are structured like the example below.

```
cat webtargets.tesla.txt | /opt/aquatone -scan-timeout 900 -threads 20
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.web.aquatone␣
↪AquatoneScan --target-file tesla --top-ports 1000
```

Parameters

- **`threads`** – number of threads for parallel aquatone command execution
- **`scan_timeout`** – timeout in miliseconds for aquatone port scans
- **`db_location`** – specifies the path to the database used for storing results *Required by upstream Task*
- **`exempt_list`** – Path to a file providing blacklisted subdomains, one per line. *Optional by upstream Task*
- **`top_ports`** – Scan top N most popular ports *Required by upstream Task*
- **`ports`** – specifies the port(s) to be scanned *Required by upstream Task*
- **`interface`** – use the named raw network interface, such as "eth0" *Required by upstream Task*
- **`rate`** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **`target_file`** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **`results_dir`** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

`output``()`

Returns the target output for this task.

Returns luigi.contrib.sqla.SQLAlchemyTarget

`parse_results``()`

Read in aquatone's .json file and update the associated Target record

`requires``()`

AquatoneScan depends on GatherWebTargets to run.

GatherWebTargets accepts exempt_list and expects rate, target_file, interface, and either ports or top_ports as parameters

Returns luigi.Task - GatherWebTargets

run()

Defines the options/arguments sent to aquatone after processing.

cat webtargets.tesla.txt | /opt/aquatone -scan-timeout 900 -threads 20

Returns list of options/arguments, beginning with the name of the executable to run

Return type list

3.5.3 Full Scanner

class `pipeline.recon.wrappers.`**FullScan**(**args*, ***kwargs*)

Wraps multiple scan types in order to run tasks on the same hierarchical level at the same time.

Note: Because FullScan is a wrapper, it requires all Parameters for any of the Scans that it wraps.

Parameters

- **`threads`** – number of threads for parallel gobuster command execution
- **`wordlist`** – wordlist used for forced browsing
- **`extensions`** – additional extensions to apply to each item in the wordlist
- **`recursive`** – whether or not to recursively gobust the target (may produce a LOT of traffic… quickly)
- **`proxy`** – protocol://ip:port proxy specification for gobuster
- **`exempt_list`** – Path to a file providing blacklisted subdomains, one per line.
- **`top_ports`** – Scan top N most popular ports
- **`ports`** – specifies the port(s) to be scanned
- **`interface`** – use the named raw network interface, such as "eth0"
- **`rate`** – desired rate for transmitting packets (packets per second)
- **`target_file`** – specifies the file on disk containing a list of ips or domains
- **`results_dir`** – specifes the directory on disk to which all Task results are written

requires()

FullScan is a wrapper, as such it requires any Tasks that it wraps.

3.5.4 Gobuster Scanner

class `pipeline.recon.web.gobuster.`**GobusterScan**(**args*, ***kwargs*)

Use `gobuster` to perform forced browsing.

Install:

```
go get github.com/OJ/gobuster
git clone https://github.com/epi052/recursive-gobuster.git
```

Basic Example:

```
gobuster dir -q -e -k -t 20 -u www.tesla.com -w /usr/share/seclists/Discovery/
↪Web-Content/common.txt -p http://127.0.0.1:8080 -o gobuster.tesla.txt -x␣
↪php,html
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.web.gobuster␣
↪GobusterScan --target-file tesla --top-ports 1000 --interface eth0 --proxy␣
↪http://127.0.0.1:8080 --extensions php,html --wordlist /usr/share/seclists/
↪Discovery/Web-Content/common.txt --threads 20
```

Parameters

- **`threads`** – number of threads for parallel gobuster command execution
- **`wordlist`** – wordlist used for forced browsing
- **`extensions`** – additional extensions to apply to each item in the wordlist
- **`recursive`** – whether or not to recursively gobust the target (may produce a LOT of traffic… quickly)
- **`proxy`** – protocol://ip:port proxy specification for gobuster
- **`exempt_list`** – Path to a file providing blacklisted subdomains, one per line. *Optional by upstream Task*
- **`db_location`** – specifies the path to the database used for storing results *Required by upstream Task*
- **`top_ports`** – Scan top N most popular ports *Required by upstream Task*
- **`ports`** – specifies the port(s) to be scanned *Required by upstream Task*
- **`interface`** – use the named raw network interface, such as "eth0" *Required by upstream Task*
- **`rate`** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **`target_file`** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **`results_dir`** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

`output``()`

Returns the target output for this task.

If recursion is disabled, the naming convention for the output file is gobuster.TARGET_FILE.txt Otherwise the output file is recursive-gobuster_TARGET_FILE.log

Results are stored in their own directory: gobuster-TARGET_FILE-results

Returns luigi.local_target.LocalTarget

`parse_results``()`

Reads in each individual gobuster file and adds each line to the database as an Endpoint

requires()

GobusterScan depends on GatherWebTargets to run.

GatherWebTargets accepts exempt_list and expects rate, target_file, interface, and either ports or top_ports as parameters

Returns luigi.Task - GatherWebTargets

run()

Defines the options/arguments sent to gobuster after processing.

Returns list of options/arguments, beginning with the name of the executable to run

Return type list

3.5.5 Hackthebox Scanner

class pipeline.recon.wrappers.**HTBScan**(**args*, ***kwargs*)

Wraps multiple scan types in order to run tasks on the same hierarchical level at the same time.

Note: Because HTBScan is a wrapper, it requires all Parameters for any of the Scans that it wraps.

Parameters

- **threads** – number of threads for parallel gobuster command execution
- **wordlist** – wordlist used for forced browsing
- **extensions** – additional extensions to apply to each item in the wordlist
- **recursive** – whether or not to recursively gobust the target (may produce a LOT of traffic. . . quickly)
- **proxy** – protocol://ip:port proxy specification for gobuster
- **exempt_list** – Path to a file providing blacklisted subdomains, one per line.
- **top_ports** – Scan top N most popular ports
- **ports** – specifies the port(s) to be scanned
- **interface** – use the named raw network interface, such as "eth0"
- **rate** – desired rate for transmitting packets (packets per second)
- **target_file** – specifies the file on disk containing a list of ips or domains
- **results_dir** – specifes the directory on disk to which all Task results are written

requires()

HTBScan is a wrapper, as such it requires any Tasks that it wraps.

3.5.6 Masscan Scanner

class pipeline.recon.masscan.**MasscanScan**(**args*, ***kwargs*)

Run `masscan` against a target specified via the TargetList Task.

Note: When specified, `--top_ports` is processed and then ultimately passed to `--ports`.

Install:

```
git clone https://github.com/robertdavidgraham/masscan /tmp/masscan
make -s -j -C /tmp/masscan
sudo mv /tmp/masscan/bin/masscan /usr/local/bin/masscan
rm -rf /tmp/masscan
```

Basic Example:

```
masscan -v --open-only --banners --rate 1000 -e tun0 -oJ masscan.tesla.json --
↪ports 80,443,22,21 -iL tesla.ips
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.masscan Masscan --
↪target-file tesla --ports 80,443,22,21
```

Parameters

- **`rate`** – desired rate for transmitting packets (packets per second)
- **`interface`** – use the named raw network interface, such as "eth0"
- **`top_ports`** – Scan top N most popular ports
- **`ports`** – specifies the port(s) to be scanned
- **`db_location`** – specifies the path to the database used for storing results *Required by upstream Task*
- **`target_file`** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **`results_dir`** – specifes the directory on disk to which all Task results are written *Required by upstream Task*
- **`exempt_list`** – Path to a file providing blacklisted subdomains, one per line. *Optional by upstream Task*

`output`()

Returns the target output for this task.

Naming convention for the output file is masscan.TARGET_FILE.json.

Returns luigi.local_target.LocalTarget

`run`()

Defines the options/arguments sent to masscan after processing.

Returns list of options/arguments, beginning with the name of the executable to run

Return type list

3.5.7 Searchsploit Scanner

class pipeline.recon.nmap.**SearchsploitScan**(**args, **kwargs*)

Run `searchcploit` against each `nmap*.xml` file in the **TARGET-nmap-results** directory and write results to disk.

Install: `searchcploit` is already on your system if you're using kali. If you're not using kali, refer to your own distributions instructions for installing `searchcploit`.

Basic Example:

```
searchsploit --nmap htb-targets-nmap-results/nmap.10.10.10.155-tcp.xml
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.nmap Searchsploit --
↪target-file htb-targets --top-ports 5000
```

Parameters

- **threads** – number of threads for parallel nmap command execution *Required by upstream Task*
- **db_location** – specifies the path to the database used for storing results *Required by upstream Task*
- **rate** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **interface** – use the named raw network interface, such as "eth0" *Required by upstream Task*
- **top_ports** – Scan top N most popular ports *Required by upstream Task*
- **ports** – specifies the port(s) to be scanned *Required by upstream Task*
- **target_file** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **results_dir** – specifies the directory on disk to which all Task results are written *Required by upstream Task*

output()

Returns the target output for this task.

Naming convention for the output folder is TARGET_FILE-searchsploit-results.

The output folder will be populated with all of the output files generated by any searchsploit commands run.

Returns luigi.local_target.LocalTarget

requires()

Searchsploit depends on ThreadedNmap to run.

TargetList expects target_file, results_dir, and db_location as parameters. Masscan expects rate, target_file, interface, and either ports or top_ports as parameters. ThreadedNmap expects threads

Returns luigi.Task - ThreadedNmap

run()

Grabs the xml files created by ThreadedNmap and runs searchsploit –nmap on each one, saving the output.

3.5.8 Subjack Scanner

class pipeline.recon.web.subdomain_takeover.**SubjackScan**(**args*, ***kwargs*)

Use `subjack` to scan for potential subdomain takeovers.

Install:

```
go get github.com/haccer/subjack
cd ~/go/src/github.com/haccer/subjack
go build
go install
```

Basic Example:

```
subjack -w webtargets.tesla.txt -t 100 -timeout 30 -o subjack.tesla.txt -ssl
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.web.subdomain_
↪takeover SubjackScan --target-file tesla --top-ports 1000 --interface eth0
```

Parameters

- **threads** – number of threads for parallel subjack command execution
- **db_location** – specifies the path to the database used for storing results *Required by upstream Task*
- **exempt_list** – Path to a file providing blacklisted subdomains, one per line. *Optional by upstream Task*
- **top_ports** – Scan top N most popular ports *Required by upstream Task*
- **ports** – specifies the port(s) to be scanned *Required by upstream Task*
- **interface** – use the named raw network interface, such as "eth0" *Required by upstream Task*
- **rate** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **target_file** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **results_dir** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

output()

Returns the target output for this task.

Returns luigi.contrib.sqla.SQLAlchemyTarget

parse_results()

Reads in the subjack's subjack.txt file and updates the associated Target record.

requires()

SubjackScan depends on GatherWebTargets to run.

GatherWebTargets accepts exempt_list and expects rate, target_file, interface, and either ports or top_ports as parameters

Returns luigi.Task - GatherWebTargets

run()

Defines the options/arguments sent to subjack after processing.

Returns list of options/arguments, beginning with the name of the executable to run

Return type list

3.5.9 ThreadedNmap Scanner

class pipeline.recon.nmap.**ThreadedNmapScan**(**args*, ***kwargs*)

Run `nmap` against specific targets and ports gained from the ParseMasscanOutput Task.

Install: `nmap` is already on your system if you're using kali. If you're not using kali, refer to your own distributions instructions for installing `nmap`.

Basic Example:

```
nmap --open -sT -sC -T 4 -sV -Pn -p 43,25,21,53,22 -oA htb-targets-nmap-
↪results/nmap.10.10.10.155-tcp 10.10.10.155
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.nmap ThreadedNmap --
↪target-file htb-targets --top-ports 5000
```

Parameters

- **threads** – number of threads for parallel nmap command execution
- **db_location** – specifies the path to the database used for storing results *Required by upstream Task*
- **rate** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **interface** – use the named raw network interface, such as "eth0" *Required by upstream Task*
- **top_ports** – Scan top N most popular ports *Required by upstream Task*
- **ports** – specifies the port(s) to be scanned *Required by upstream Task*
- **target_file** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **results_dir** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

output()

Returns the target output for this task.

Naming convention for the output folder is TARGET_FILE-nmap-results.

The output folder will be populated with all of the output files generated by any nmap commands run. Because the nmap command uses -oA, there will be three files per target scanned: .xml, .nmap, .gnmap.

Returns luigi.local_target.LocalTarget

parse_nmap_output()

Read nmap .xml results and add entries into specified database

requires()

ThreadedNmap depends on ParseMasscanOutput to run.

TargetList expects target_file, results_dir, and db_location as parameters. Masscan expects rate, target_file, interface, and either ports or top_ports as parameters.

Returns luigi.Task - ParseMasscanOutput

run()

Parses pickled target info dictionary and runs targeted nmap scans against only open ports.

3.5.10 TKOSubs Scanner

class pipeline.recon.web.subdomain_takeover.**TKOSubsScan**(**args*, ***kwargs*)

Use `tko-subs` to scan for potential subdomain takeovers.

Install:

```
go get github.com/anshumanbh/tko-subs
cd ~/go/src/github.com/anshumanbh/tko-subs
go build
go install
```

Basic Example:

```
tko-subs -domains=tesla.subdomains -data=/root/go/src/github.com/anshumanbh/
↪tko-subs/providers-data.csv -output=tkosubs.tesla.csv
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.web.subdomain_
↪takeover TKOSubsScan --target-file tesla --top-ports 1000 --interface eth0
```

Parameters

- **db_location** – specifies the path to the database used for storing results *Required by upstream Task*
- **exempt_list** – Path to a file providing blacklisted subdomains, one per line. *Optional by upstream Task*
- **top_ports** – Scan top N most popular ports *Required by upstream Task*
- **ports** – specifies the port(s) to be scanned *Required by upstream Task*
- **interface** – use the named raw network interface, such as "eth0" *Required by upstream Task*
- **rate** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **target_file** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **results_dir** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

output()

Returns the target output for this task.

Returns luigi.contrib.sqla.SQLAlchemyTarget

parse_results()
Reads in the tkosubs .csv file and updates the associated Target record.

requires()
TKOSubsScan depends on GatherWebTargets to run.

GatherWebTargets accepts exempt_list and expects rate, target_file, interface, and either ports or top_ports as parameters

Returns luigi.Task - GatherWebTargets

run()
Defines the options/arguments sent to tko-subs after processing.

Returns list of options/arguments, beginning with the name of the executable to run

Return type list

3.5.11 WaybackurlsScan Scanner

class pipeline.recon.web.waybackurls.**WaybackurlsScan**(**args*, ***kwargs*)
Fetch known URLs from the Wayback Machine, Common Crawl, and Virus Total for historic data about the target.

Install:

```
go get github.com/tomnomnom/waybackurls
```

Basic Example: `waybackurls` commands are structured like the example below.

```
cat domains.txt | waybackurls > urls
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.web.waybackurls␣
↪WaybackurlsScan --target-file tesla --top-ports 1000
```

Parameters

- **db_location** – specifies the path to the database used for storing results *Required by upstream Task*
- **exempt_list** – Path to a file providing blacklisted subdomains, one per line. *Optional by upstream Task*
- **top_ports** – Scan top N most popular ports *Required by upstream Task*
- **ports** – specifies the port(s) to be scanned *Required by upstream Task*
- **interface** – use the named raw network interface, such as “eth0” *Required by upstream Task*
- **rate** – desired rate for transmitting packets (packets per second) *Required by upstream Task*
- **target_file** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*

- **results_dir** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

output()
Returns the target output for this task.

Returns luigi.contrib.sqla.SQLAlchemyTarget

requires()
WaybackurlsScan depends on GatherWebTargets to run.

GatherWebTargets accepts exempt_list and expects rate, target_file, interface, and either ports or top_ports as parameters

Returns luigi.Task - GatherWebTargets

run()
Defines the options/arguments sent to waybackurls after processing.

3.5.12 Webanalyze Scanner

class pipeline.recon.web.webanalyze.**WebanalyzeScan**(**args*, ***kwargs*)
Use webanalyze to determine the technology stack on the given target(s).

Install:

```
go get -u github.com/rverton/webanalyze

# loads new apps.json file from wappalyzer project
webanalyze -update
```

Basic Example:

```
webanalyze -host www.tesla.com -output json
```

Luigi Example:

```
PYTHONPATH=$(pwd) luigi --local-scheduler --module recon.web.webanalyze␣
↪WebanalyzeScan --target-file tesla --top-ports 1000 --interface eth0
```

Parameters

- **threads** – number of threads for parallel webanalyze command execution
- **db_location** – specifies the path to the database used for storing results *Required by upstream Task*
- **exempt_list** – Path to a file providing blacklisted subdomains, one per line. *Optional for upstream Task*
- **top_ports** – Scan top N most popular ports *Required by upstream Task*
- **ports** – specifies the port(s) to be scanned *Required by upstream Task*
- **interface** – use the named raw network interface, such as “eth0” *Required by upstream Task*
- **rate** – desired rate for transmitting packets (packets per second) *Required by upstream Task*

- **`target_file`** – specifies the file on disk containing a list of ips or domains *Required by upstream Task*
- **`results_dir`** – specifes the directory on disk to which all Task results are written *Required by upstream Task*

`output`()

Returns the target output for this task.

Returns luigi.contrib.sqla.SQLAlchemyTarget

`parse_results`()

Reads in the webanalyze's .csv files and updates the associated Target record.

`requires`()

WebanalyzeScan depends on GatherWebTargets to run.

GatherWebTargets accepts exempt_list and expects rate, target_file, interface, and either ports or top_ports as parameters

Returns luigi.Task - GatherWebTargets

`run`()

Defines the options/arguments sent to webanalyze after processing.

Returns list of options/arguments, beginning with the name of the executable to run

Return type list

CHAPTER 4

Indices and tables

- genindex
- search

Index