Reconnaissance: a eulogy in three acts

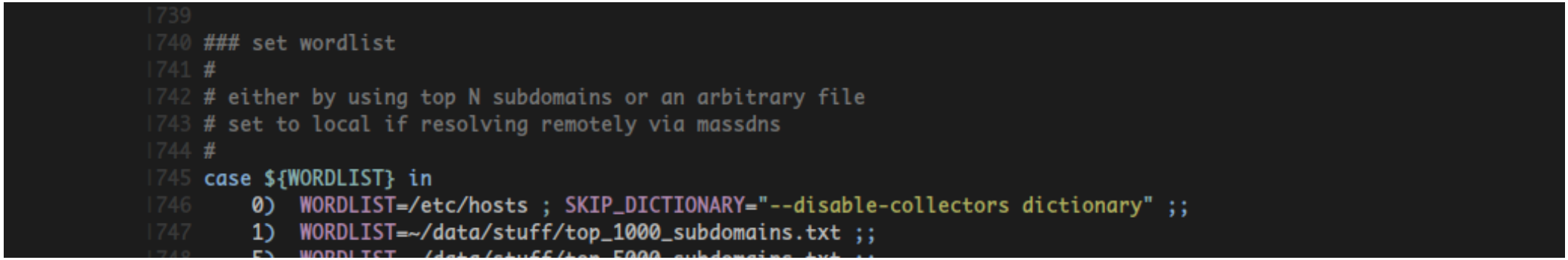
europa Follow
Feb 11, 2018 · 8 min read

```
### get non-optional arguments
#
shift $((OPTIND-1))

while [[ $# > 0 ]] && [[ ! ${1} =~ ^- ]] && [[ ! ${1} =~ ' ' ]]; do
    DOMAINS=("${DOMAINS[@]}" "${1}")
    shift
done

unset OPTIND

if [[ ${#DOMAINS[@]} < 1 || ${HELP} == true || ${NMAP_PORTS} =~ [^0-9n,\-] ]]; then
    echo "" >&2
    echo "[?] usage: recon [-a] [-z] [-e cf] [-p ports] [-t dsj] [-r -s x] [-w x] <domains...>" >&2
    echo "[?]    -a      full-auto (don't prompt user)" >&2
    echo "[?]    -e      esoteric enum (c|loudflare f|dns)" >&2
    echo "[?]    -p      ports to scan (n|map 1k, X,Y-Z)" >&2
    echo "[?]    -t      use a remote host for (d|iscover, s|can, j|uicy, r|edir, c|rlf)" >&2
    echo "[?]    -w      wordlist size (1, 5, 10) or path" >&2
    echo "[?]    -z      check dependencies" >&2
    echo "[?]    -r -s   resume / skip from step:" >&2
    echo "[?]            n|map     g|ather j|uicy  b|uckets t|akeover" >&2
    echo "[?]            r|edirect c|rlf   x|ss    s|qlmap" >&2
    echo "" >&2
    exit 1
fi
```



I used to hate Vim until I learned how to quit

When it comes to the topic of **reconnaissance**, more and more people are explaining their own methodologies as of late—people like Jason Haddix and Behrouz Sadeghipour have been my go-to reference while taking the first steps into the bug hunting world, with their "The Bug Hunter's Methodology" and "How to: Recon and Content Discovery". Reading the source code of some of the more and less-known tools, along with the aforementioned methodologies, has allowed me to build a base for an effective recon early on for free while giving me the chance to improve and expand on their knowledge further down the road.

I'm a big fan of automation, I've always been. I'm likely to spend hours honing my tools to achieve better and faster automation, in order to focus more on things that rely on logic using the time I've saved. My personal approach is to avoid monolithic solutions, where one tool does everything, preferring a wrapper-style approach instead: gluing different tools and methodologies together has proven effective (and remunerative) while also allowing me to give back to the community by improving said tools with pull requests to their creators when necessary.
Far from being presentable to the general public, I will not provide my 1,000+ LoC toolset; I will, however, share my own approach I make use of daily, with snippets here and there to help anyone find their own.

This will be a eulogy in three acts: **the gathering** of assets, **the examination** of said assets, and **the initial assessment** of the targets.

Act one: The Gathering

The first step I usually take is gathering as many subdomains as possible for a given target, as it's often time-consuming (even when automated), leaving me between 15 and 30 minutes to do some scavenging on Google and Github.

The **gathering** phase itself is divided into three steps: *"esoteric enumeration"*, using non-standard sources learned from the excellent talk from **BugCrowd's LevelUp 2017** and internet-wide FDNS data parsing from **Rapid7 Project Sonar**, wordlist-based subdomain discovery using **Blechschmidt's massdns** coupled with **Jason Haddix's list**, and standard subdomain enumeration using my own fork of Michael Henriksen's **Aquatone**.

In my examples, I will make use of Teslamotors assets for comparison when necessary.

Esoteric enumeration using Cloudflare

To be honest, I've found this approach to be just genial in its simplicity: after registering for a free account at **Cloudflare** we can add any domain to our list, Cloudflare itself will enumerate the available resources for us and give us a parseable JSON output with `A`, `AAAA`, and `CNAME` records in less than one minute. You can read about this approach on the LevelUp slides above, or by having a look at the source code used in the said talk.

```
~/aquatone/tesla.com
❯ grep -c -E ',(CNAME|A|AAAA),' *.csv
```

```
tesla_cn.csv:8
tesla_com.csv:16
teslamotors_com.csv:29
```

53 assets in one minute. Not bad! Thanks, Cloudflare!

Rapid7 Project Sonar FDNS enumeration

While deeply time-consuming (averaging 25 minutes per run), this step usually returns a bunch of resources that nobody would ever be able to find using vanilla approaches.
Project Sonar provides internet-wide DNS scan data for free at https://scans.io/study/sonar.fdns_v2—these 20GB+ gzipped archives contain `ANY` records for millions and millions of hosts, often exposing unseen or overlooked resources. By using just the right amount of Bash-fu it is possible to parse the huge archive for some juicy records:

```
# parse fdns file for domain.tld|domain.tld|…
pv 2018-01-28-1517126401-fdns_any.json.gz \
 | pigz -dc \
 | grep -P "\.(tesla\.com|tesla\.cn|teslamotors\.com)\"," \
```

```
 | jq -r '.name' \
 | tee fdns.lst

# retrieve unique subdomains without the domain
 rev fdns.lst \
 | cut -f3- -d'.' \
 | rev \
 | grep -Fv '*' \
 | sort -u
 | wc -l
82
```

82 more resources. Surely there will be some duplicates, but it's looking good.

Wordlist enumeration

This is the step that everyone knows about: glue together a good wordlist (in our case, jhaddix's aptly named `all.txt`), and an acceptable list of public resolvers, and soon enough you'll have a fairly large list of new assets at hand. Two issues arise here: first of all, not every provider is happy when one of its customers blasts DNS queries for millions of hosts; secondly, some (if not many) of the public resolvers will reply for non-existent subdomains, spoiling the list with fake results.

In order to tackle the first one I looked around for a nice VPS provider with a relaxed policy towards authorized pentesting; I remember reading in jhaddix's "Bug Hunter's Methodology" about DigitalOcean so the first thing I did (and you should as well) was opening a ticket on their support system, asking for their policy towards my activity. I gave them links to my profiles on HackerOne and BugCrowd, explained what I wanted to do and asked for their opinion. To be honest, I've never had such a pleasant experience with a customer service before (except for Amazon): while not being *officially supported* (your account can still be suspended if your behavior does not pertain the good etiquette of an authorized pentest), I got the green light in less than two days, all my questions had an answer and my account was "flagged" internally not to raise too many alerts during my daily scans. Having a droplet on **DigitalOcean** is probably the best investment I've made so far, as we'll see later. It's **imperative** to talk to their support team first, and provide them with proof of your good intentions, or **you will** get into troubles.

My script will execute the `massdns` scan through a remote SSH session and retrieve the results for further processing in less than 5 minutes.

```
# execute remotely
ssh ... ... \
    python ./bin/subbrute.py ./bin/lists/jhaddix.txt ${domains} \
    | massdns -r ./bin/lists/resolvers.txt -t A --verify-ip -o S \
        | cut -f1 -d' ' \
        | sed -r \"s/\.(${domains// /\|})\.//g\" \
        > massdns.out
```

As far as fake results go, instead of blindly trusting what the public resolvers tell me I prefer to parse the returned results, strip the main domain away, and prepare a sub-wordlist with all the returned entries, to be subsequently fed to **Aquatone's** dictionary module.

```
~/aquatone/tesla.com
❯ wc -l massdns.out
297 massdns.out
```

297 potential targets now available for the wordlist. **Aquatone** will take care of any fake result or wildcarded entry.

Aquatone discovery

The final step in this initial process comes from **Aquatone**'s Discovery modules. After registering with all the required metadata providers and requesting an **API key** for each, my script will locally run `aquatone-discover` for each and every target, feeding it the Cloudflare entries, FDNS results, and the `massdns` list as a comprehensive dictionary, in order to weed out old, unresolvable and fake entries in one sweep; all the while also adding legit entries from metadata sources such as **CertSH**, **Shodan**, **CertSpotter**, and many, many more.

```
# retrieve the resolved entries from the remote droplet
scp ... ... massdns.out .
```

```
# add cloudflare results to the list
grep --no-filename -E ',(A|AAAA|CNAME),' *.csv \
    | cut -f1 -d',' \
    | rev \
    | cut -f3- -d'.' \
    | rev \
    | sort -u \
    >> massdns.out
```

```
# add fdns results to the list
rev fdns.lst \
```

```
    | cut -f3- -d'.' \
    | rev \
    | grep -Fv '*' \
    | sort -u \
    >> massdns.out

# run aquatone-discover on each target
sort -u massdns.out \
    | awk NF \
    > x && \
    mv x massdns.out

# run locally
aquatone-discover \
    --ignore-private \
    --wordlist=massdns.out \
    -d target target ...
```

Of course, the code snippet above is just to further clarify the methodology, as Aquatone doesn't support multiple targets natively and you'll have to find your own approach to that.

Let's see how good we did!

```
~/aquatone/tesla.com
❯ wc -l hosts.txt
217 hosts.txt
```

217 unique targets. This is *way* more than some other approaches would result in.

Port scanning for available services

We now have a hosts list, and it's time for some old-fashioned port scanning. Aquatone offers a scan module, namely `aquatone-scan`, with varying degrees of complexity, and it surely does its job in a pinch—alas, it's extremely slow at it when compared to other approaches (such as **nmap**, or **masscan**), let alone the fact that it's essentially impossible to scan more than a thousand ports per host without waiting for hours the scan to complete.

Using `nmap` or `masscan` with a **good ports list** is a good approach, albeit still slow in the case of `nmap`, but might still rise some concerns from your home ISP (as it did happen to me). **DigitalOcean** saves the day once again: my script will send over an IP-only `hosts.txt` file from Aquatone to the remote

droplet and perform a `masscan` on every host for **all** of the 65,535 available ports in just a few minutes:

```
# scp a ip-only hosts.txt to the droplet
cut -f2 -d',' hosts.txt \
    | sort -u \
    > masscan.in

scp masscan.in ... ...
```

```
# execute remotely
ssh ... ... \
    sudo masscan -iL masscan.in -oG masscan.out \
         --open --max-rate 30000 -p 0-65535

# retrieve the remote file
scp ... ... masscan.out

~/aquatone/tesla.com
❯ grep Port: -c masscan.out
303
```

No complaints from any ISP, and a comprehensive list of open services.

Now we have **217** unique targets and **303** open services. Not every single one of those will reply with an HTTP service, so we'll need to weed them out.

Visual inspection with Aquatone's Gather module

Similar to **EyeWitness**, Aquatone offers a **Gather** module which, using an headless NightmareJS browser, will try and connect to every available service to retrieve useful metadata and a screenshot of the page for visual inspection. This is crucial: you *cannot* think you're going to be able to sift through thousands of targets one-by-one. You need to weed out the useless ones using your eyes only, without having to manually check every single one of them.

Aquatone's **Gather** module will only accept its own format as input, and since we didn't use `aquatone-scan` but `masscan` instead we'll have to convert its output to something Aquatone will be able to handle. I'll gladly share this snippet of code because it took me a while to figure out, and I believe it might help a lot:

```
ips=($(grep -Po '([0-9]{1,3}\.){3}[0-9]{1,3}' masscan.out \
        | sort -u))
for ip in ${ips[@]}; do
    ports="$(grep -F "${ip} " masscan.out \
```

```
            | grep -Po '[0-9]{1,5}(?=/open/tcp)')"
    echo "${ip},${ports//[^0-9]/,}" >> open_ports.txt
done
```

This should produce an Aquatone-compatible `open_ports.txt` file which should let the **Gather** module do its thing properly. We'll let this run locally, as the droplet isn't as powerful as we'd need it to be in order to run so many headless browser instances without rolling over and dying.
My own fork of Aquatone will also produce an `alive.txt` file containing all the replying hosts in their full URI form, as we'll need that in the second act of the recon process.

```
~/aquatone/tesla.com
❯ wc -l alive.txt
269 alive.txt
```

269 available web services spread across **217** targets.

Checking for sub-domains takeovers

Before beginning with the second act of the recon process, I'll check the found targets against common hosting services looking for forgotten `CNAME` records. Aquatone offers a **Takeover** module covering many of the popular services, and I've implemented a few more in my own fork, using pointers from ice3man543's SubOver.

```
~/aquatone/tesla.com
❯ aquatone-takeover --list-detectors | grep -Fc Service.
32
```

As my script will only make one folder to keep everything in one place, running individual Aquatone modules will not pose a challenge, as there's only one `hosts.txt` and `hosts.json` for all the targets.

Concluding Act One: moving forward

We now have a comprehensive and visual list of targets and resources, we'll see how to approach them in the second part of this series: looking for sensitive files using an optimized wordlist, cloud storage "buckets" going

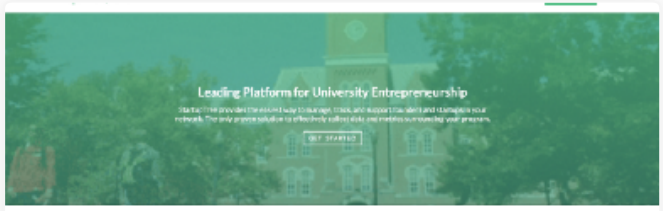
beyond the lazys3 and teh_s3_bucketeers approach, looking for *sleeping subdomains*, and implementing more of Thomas WB's "Passive-ish Recon Techniques" and Jon Bottarini's "It's in the little things" methodologies in our workflow.

Ad maiora!

Medium | Bug Bounty | Recon | Pentesting | Reconnaissance

europa

Security researcher and strength athlete



Mike Bond

Jun 1, 2018 · 14 min rea

233

Sai Krishna Kothap...

May 18, 2018 · 7 min r

1.4K

Akshay 'Ax' Sharma

Jun 19, 2018 · 3 min re

388

Responses

Write a response...

Show all responses