Recon Everything

SACHIN GROVER Follow
Oct 8 · 28 min read

Bug Bounty Hunting Tip #1- Always read the Source Code

Approach a Target

• Ideally you're going to be wanting to choose a program that has a wide scope. You're also going to be wanting to look for a bounty program that has a wider range of vulnerabilities within scope.

• Mining information about the domains, email servers and social network connections.

• Dig in to website, check each request and response and analysis that, try to understand their infrastructure such as how they're handling sessions/authentication, what type of CSRF protection they have (if any).

• Use negative testing to through the error, this Error information is very helpful for me to finding internal paths of the website. Give time to understand the flow of the application to get a better idea of what type of vulnerabilities to look for.

• Start to dig into using scripts for wordlist bruteforcing endpoints. This can help with finding new directories or folders that you may not have been able to find just using the website.
This tends to be private admin panels, source repositories they forgot to remove such as /.git/ folders, or test/debug scripts. After that check each form of the website then try to push client side attacks. Use multiple payloads to bypass client side filters.

• Start early. As soon as a program is launched, start hunting immediately, if you can.

• Once you start hunting, take a particular functionality/workflow in the application and start digging deep into it. I have stopped caring about low hanging fruits or surface bugs. There is no point focusing your efforts on those.

• So, let’s say an application has a functionality that allows users to send emails to other users.

• Observe this workflow/requests via a proxy tool such as Burp. Burp is pretty much the only tool I use for web app pen testing.

• Create multiple accounts because you would want to test the emails being sent from one user to another. If you haven’t been provided multiple accounts, ask for it. Till date, I have not been refused a second account whenever I have asked for it.

• Now, if you are slightly experienced, after a few minutes of tinkering with this workflow, you will get a feeling whether it might have something interesting going on or not. This point is difficult to explain. It will come with practice.

• If the above is true, start fuzzing, breaking the application workflow, inserting random IDs, values, etc. wherever possible. 80% of the time, you will end up noticing weird behavior.

• The weird behavior doesn't necessarily mean you have found a bug that is worth reporting. It probably means you have a good chance so you should keep digging into it more.

• There is some research that might be required as well. Let's say you found that a particular version of an email server is being used that is outdated. Look on the internet for known vulnerabilities against it. You might encounter a known CVE with a known exploit. Try that exploit and see what happens (provided you are operating under the terms and conditions of the bug bounty).

• There might be special tools that are required. Explore into that, if possible. Remember, Burp is a swiss army knife but you might have to use certain specific tools in certain cases. Always, be aware of that.

• After spending a few hours on this, if you think you have exhausted all your options and are not getting anything meaningful out of it, stop and move on. Getting hung up on something is the biggest motivation killer but that doesn't mean you are giving up. Get back to it later if something else comes up. Make a note of it.

• Something that has worked for me is bounds checking on parameters, pick a parameter that has an obvious effect on the flow of the application.
For example, if a field takes a number (lets call it ID for lulz).
What happens if:
-you put in a minus number?
-you increment or decrement the number?
-you put in a really large number?
-you put in a string or symbol characters?
-you try traverse a directory with …/
-you put in XSS vectors?
-you put in SQLI vectors?
-you put in non-ascii characters?
-you mess with the variable type such as casting a string to an array
-you use null characters or no value

I would then see if I can draw any conclusions from the outcomes of these tests,
-see if I can understand what is happening based on an error
-is anything broken or exposed

-can this action affect other things in the app.

• Focus on site functionality that has been redesigned or changed since a previous version of the target. Sometimes, having seen/used a bounty product before, you will notice right away any new functionality. Other times you will read the bounty brief a few times and realize that they are giving you a map. Developers often point out the areas they think they are weak in. They/us want you to succeed. A visual example would be new search functionality, role based access, etc. A bounty brief example would be reading a brief and noticing a lot of pointed references to the API or a particular page/function in the site.

• If the scope allows (and you have the skillset) test the crap out of the mobile apps. While client side bugs continue to grow less severe, the API's/web-endpoints the mobile apps talk to often touch partsof the application you wouldn't have seen in a regular workflow. This is not to say client side bugs are not reportable, they just become low severity issues as the mobile OS's raise the bar security-wise.

• So after you have a thorough "feeling" for the site you need to mentally or physically keep a record of workflows in the application. You need to start asking yourself questions like these:

• Does the page functionality display something to the users? (XSS,Content Spoofing, etc)

• Does the page look like it might need to call on stored data?

• (Injections of all type, Indirect object references, client side storage)

• Does it (or can it) interact with the server file system? (Fileupload vulns, LFI, etc)

• Is it a function worthy of securing? (CSRF, Mixed-mode)

• Is this function a privileged one? (logic flaws, IDORs, priv escalations)++

• Where is input accepted and potentially displayed to the user?

• What endpoints save data?

• Any file upload functionality?

• What type of authentication is used?

Steps to take when approaching a target

1. Check/Verify target's scope (*.example.com)

2. Find subdomains of target (Refer Subdomain tools mentioned in the article)

3. Run masscan

4. Check which domains resolve

5. Take Screenshot

6. Do Content Discovery (by bruteforcing the files and directories on a particular domain/subdomain)

Web Tools:
https://pentest-tools.com/
https://virustotal.com/
https://www.shodan.io/
https://crt.sh/?q=%25target.com
https://dnsdumpster.com/
https://censys.io
http://dnsgoodies.com

Recon

• Recon shouldn't just be limited to finding assets and outdated stuff. It's also understanding the app and finding functionality that's not easily accessible. There needs to be a balance between recon and good old hacking on the application in order to be successful — @NahamSec

Subdomain enumeration tools:

- It is recommended to go through the github links for usage of tools.

• **Enumerating Domains**

a. Vertical domain corelation (all the subdomain of a domain) (maps.google.com)
b. Horizontal domain corelation (like google.com, google.cz, youtube.com)

1. **Sublist3r** — https://github.com/aboul3la/Sublist3r

Setup:

> *git clone https://github.com/aboul3la/Sublist3r.git*
> *sudo pip install -r requirements.txt*

Usage:

– To enumerate subdomains of specific domain:

> *python sublist3r.py -d example.com*

Alias:

> *alias sublist3r='python /path/to/Sublist3r/sublist3r.py -d '*
>
> *alias sublist3r-one=". <(cat domains | awk '{print "sublist3r "$1 " -o " $1 ".txt"}')"*

2. **subfinder** — https://github.com/subfinder/subfinder

Setup:

> *go get github.com/subfinder/subfinder*

Usage:

> *subfinder -d freelancer.com*

To find domain recursively:

> *subfinder -d <domain> -recursive -silent -t 200 -v -o <outfile>*

For using bruteforcing capa kibilities, you can use -b flag with -w option to specify a wordlist.

> *./subfinder -d freelancer.com -b -w jhaddix_all.txt -t 100 — sources censys — set-settings CensysPages=2 -v*

The -o command can be used to specify an output file.

3. **findomain —** https://github.com/Edu4rdSHL/findomain

You can monitor the subdomains and provide the webhooks to get notifications on Slack and discord.

Setup:

> *$ wget https://github.com/Edu4rdSHL/findomain/releases/latest /download/findomain-linux*
> *$ chmod +x findomain-linux*

Usage:

> *findomain -t example.com*

4. **assetfinder —** https://github.com/tomnomnom/assetfinder

– Find domains and subdomains potentially related to a given domain.

Setup:

> *go get -u github.com/tomnomnom/assetfinder*

Usage:

> *assetfinder [— subs-only] <domain>*
>
> *5. **Amass:***

Setup:

> *go get -u github.com/OWASP/Amass/…*

Usage:

> *amass enum -o subdomains.txt -d output_file.txt*

6. **censys-enumeration** — https://github.com/0xbharath/censys-enumeration

– This is the most important steps, because the subdomains names that you find here, you cannot find from other bruteforce tools because your wordlist does not have pattern that are available in all the subdomains or does not have keyword like gateway or payment which are part of subdomain.

> *search query — 443.https.tls.certificate.parsed.extensions.subject_alt_name.dns_names:snapchat.com*

A script to extract subdomains/emails for a given domain using

SSL/TLS certificates dataset on Censys

Setup:

– Clone this repo

> *$ git clone git@github.com:yamakira/censys-enumeration.git*

– Install dependencies

> *$ pip install -r requirements.txt*

– Get Censys API ID and Censys API secret by creating a account on https://censys.io

– Add Censys API ID and Censys API secret as CENSYS_API_ID & CENSYS_API_SECRET respectively to the OS environment variables. On Linux you can use a command similar to following to do this

> *$ export CENSYS_API_SECRET=”iySd1n0l2JLnHTMisbFHzxClFuE0"*

Usage:

> *$ python censys_enumeration.py — no-emails — verbose — outfile results.json domains.txt*

7. **altdns —** https://github.com/infosec-au/altdns

– It generates the possible combinations of original domain with the *words* from the wordlist (*example*).

Setup:

> *pip install py-altdns*

Usage:

> *# python altdns.py -i input_domains.txt -o ./output/path -w altdns/words.txt -i subdomains.txt -o data_output -w words.txt -s results_output.txt*

8. **Massdns:** https://github.com/blechschmidt/massdns

Setup:

> *git clone https://github.com/blechschmidt/massdns.git*
> *cd massdns*
> *make*

Usage:

> *./bin/massdns [options] [domainlist]*

Resolve all A records from domains within domains.txt using the resolvers within resolvers.txt in lists and store the results within results.txt:

> *$ ./bin/massdns -r lists/resolvers.txt -t A domains.txt > results.txt*

9. **domains-from-csp** — https://github.com/0xbharath/domains-from-csp

– Content-Security-Policy header allows us to create a whitelist of sources of trusted content, and instructs the browser to only execute or render resources from those domains(sources).

Setup:

> *$ git clone git@github.com:yamakira/censys-enumeration.git*
> *$ pipenv install*

Usage:

> *# python csp_parser.py target_url*
> *# python csp_parser.py target_url — resolve*

10. **Using SPF record of DNS —** https://github.com/0xbharath/assets-from-spf/

– A Python script to parse netblocks & domain names from SPF(Sender Policy Framework) DNS record

– For every parsed asset, the script will also find and print Autonomous System Number(ASN) details

Setup:

> *$ git clone git@github.com:yamakira/assets-from-spf.git*
> *$ pipenv install*

Usage:

– Parse the SPF record for assets but don't do ASN enumeration

> *$ python assets_from_spf.py target_url*

– Parse the SPF record for assets and do ASN enumeration

> *$ python assets_from_spf.py target_url — asn*

Get ASN Number:

– Autonomous System Number (ASN) -> http://bgp.he.net -> check for example tesla.com and checkin Prefixes V4 to get the IP range or

> *$ curl -s http://ip-api.com/json/192.30.253.113 | jq -r .as*

AS36459 GitHub, Inc.

– The ASN numbers found can be used to find netblocks of the domain.

– We can use advanced WHOIS queries to find all the IP ranges that belong to an ASN

> *$ whois -h whois.radb.net — '-i origin AS36459' | grep -Eo "([0–9.]+){4}/[0–9]+" | uniq*

There is an Nmap script to find IP ranges that belong to an ASN that https://nmap.org/nsedoc/scripts/targets-asn.html

> *$ nmap — script targets-asn — script-args targets-asn.asn=17012 > paypal.txt*

Clean up the output from the above nmap result, take all the IPs in a file and then run version scanning on them or masscan on them.

> *nmap -p- -sV -iL paypal.txt -oX paypal.xml*

– you can use dig

> *$ dig AXFR @<nameserver> <domain_name>*

11. **Certspotter** — https://certspotter.com/api/v0/certs?domain=hackerone.com

– Good for vertical and horizontal corelation

– you can get domain names, subdomain names

– email address in a certificate

12. **Crt.sh** — https://crt.sh/?q=%25domain.com

13. **knockpy** — https://github.com/guelfoweb/knock.git

Setup:

> *$ sudo apt-get install python-dnspython*
> *$ git clone https://github.com/guelfoweb/knock.git*
> *Set your virustotal API_KEY:*
> *$ nano knockpy/config.json*
> *$ sudo python setup.py install*

Usage:

> *$ knockpy domain.com -w wordlist.txt*

1. **Shodan** -

Ports:8443, 8080
Title: "Dashboard[Jenkins]"
Product: Tomcat
Hostname: example.com
Org: google
ssl:Google

2. **Viewdns.info — Reverse whois lookup —** if you know the "email id " in the registrar of a domain and you want to check what other domains are registered with the same email id you can use this site.

Get email address using — $ whois <domain.com>

3. **Sublert —** https://github.com/yassineaboukir/sublert

• Sublert is a security and reconnaissance tool which leverages certificate transparency to automatically monitor new subdomains deployed by specific organizations and issued TLS/SSL certificate.

Setup: https://medium.com/@yassineaboukir/automated-monitoring-of-subdomains-for-fun-and-profit-release-of-sublert-634cfc5d7708

> *$ git clone https://github.com/yassineaboukir/sublert.git && cd sublert*
> *$ sudo pip3 install -r requirements.txt*

Usage:

Let's add PayPal for instance:

> *$ python sublert.py -u paypal.com*

Let's make Sublert.py executable:

> *$ chmod u+x sublert.py*

Now, we need to add a new Cron job to schedule execution of Sublert at given time. To do it, type:

> *$ Crontab -e*

Add the following line at the end of the Cron file:

> *0 */12 * * * cd /root/sublert/ && /usr/bin/python3 sublert.py -r -l >> /root/sublert/sublert.log 2>&1*

Jason Haddix (https://twitter.com/jhaddix/status/972926512595746816?lang=en)
The lost art of LINKED target discovery w/ Burp Suite:

1) Turn off passive scanning
2) Set forms auto to submit
3) Set scope to advanced control and use string of target name (not a normal FQDN)
4) Walk+browse, then spider all hosts recursively!
5) Profit (more targets)!

Content Discovery Tools (Directory Bruteforcing)

- Use robots.txt to determine the directories.

- Also spider the host for API endpoints.

- you see an open port on 8443

- Directory brute force

- /admin/ return 403

- You bruteforce for more files/direcotries on /admin/

- and let's say /admin/users.php return 200

- Repeat on other domain, ports, folders etc

1. **ffuf** — https://github.com/ffuf/ffuf

– A fast web fuzzer written in Go.

Setup:

> *go get github.com/ffuf/ffuf*

Usage:

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

Virtual host discovery (without DNS records):

- Start by figuring out the response length of false positive:

> *$ curl -s -H "Host:nonexistent.ffuf.io.fi" http://ffuf.io.fi | wc -c*

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

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

GET parameter fuzzing:

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

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

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

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

POST data fuzzing:

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

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

1. **dirsearch —** https://github.com/maurosoria/dirsearch.git or https://github.com/Damian89/dirsearch

Setup:

> *git clone https://github.com/Damian89/dirsearch.git*
> *cd dirsearch*
> *python3 dirsearch.py -u <URL> -e <EXTENSION>*
> *python3 dirsearch.py -e php,txt,zip -u https://target -w db/dicc.txt — recursive -R 2*

Wordlist:

> *$ wget https://gist.githubusercontent.com/EdOverflow /c4d6d8c43b315546892aa5dab67fdd6c /raw/7dc210b17d7742b46de340b824a0caa0f25cf3cc/open_redirect _wordlist.txt*

Alias:

> *alias dirsearch='python3 /path/to/dirsearch/dirsearch.py -u '*
> *alias dirsearch-one=". <(cat domains | awk '{print "dirsearch "$1 " -e *"}')"*
> *alias openredirect=". <(cat domains | awk '{print "dirsearch "$1 " -w /path/to/dirsearch/db/openredirectwordlist.txt -e *"}')"*

2. **Dirbuster**

3. **Gobuster —** https://github.com/OJ/gobuster

Setup:

> *go get github.com/OJ/gobuster*

Usage:

> *gobuster dir -u https://mysite.com/path/to/folder -c 'session=123456' -t 50 -w common-files.txt -x .php,.html*

4. **wfuzz** — https://github.com/xmendez/wfuzz/

Setup:

> *pip install wfuzz*

Usage:

> *$ wfuzz -w wordlist/general/common.txt — hc 404 http://testphp.vulnweb.com/FUZZ*

5. **Burp Intruder**

6. **Burp Scanner**

Screenshot Tools:

• Look at the headers to see which security options are in place, for example looking for presence of X-XSS-Protection: or X-Frame-Options: deny.

• Knowing what security measures are in place means you know your limitations.

1. **Aquatone —** https://github.com/michenriksen/aquatone

Setup:

> *go get -u github.com/michenriksen/aquatone*

Usage:

> *cat hosts.txt | aquatone -out ~/aquatone/example.com*

2. **Eyewitness:** https://github.com/FortyNorthSecurity/EyeWitness

Setup:

> *$ git clone https://github.com/FortyNorthSecurity/EyeWitness.git*

Navigate into the setup directory
Run the setup.sh script

Usage:

> *./EyeWitness -f urls.txt — web*
>
> *./EyeWitness -x urls.xml — timeout 8 — headless*

3. **Webscreenshot:** https://github.com/maaaaz/webscreenshot

Setup:

> *$ apt-get update && apt-get install phantomjs*
> *$ pip install webscreenshot*

Usage:

> *$ python webscreenshot.py -i list.txt -v*

• Once this is done, we use a tool called epg-prep (https://www.npmjs.com/package/epg-prep) to create thumbnails to do so, simply run: epg-prep uber.com
This will allow us to view the created pictures using express-photo-gallery.

• In a final step, use the express-gallery-script from the bottom of this blogpost and save it as yourname.js. All you need to do is to change the folder name inside the script: app.use('/photos', Gallery('uber.com', options)); the folder name in this case is set uber.com but depending on which target you look at it may be different. Once you've done that you can simply run the script using node yourname.js. This will create a webserver listening on Port 3000 with an endpoint called /photos. So to access this you simply type: http://yourserverip:3000/photos to get a nice overview of the subdomains you have enumerated

> ***System Tools***
> *apt update && apt upgrade*
> *curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -*
> *apt install -y git wget python python-pip phantomjs xvfb screen slurm gem phantomjs imagemagick graphicsmagick nodejs*

Requirements for WebScreenshot

> *pip install webscreenshot*
> *pip install selenium*

Requirements for express-photo-gallery

> *sudo npm install -g npm*
> *npm install express-photo-gallery*
> *npm install express*
> *npm install -g epg-prep*

express-photo-gallery Script

> *JavaScript*
> *var express = require('express');*
> *var app = express();*
>
> *var Gallery = require('express-photo-gallery');*
>
> *var options = {*
> *title: 'My Awesome Photo Gallery'*
> *};*
>
> *app.use('/photos', Gallery('uber.com', options));*
>
> *app.listen(3000);*

Check CMS

1. Wappalyzer browser extension

2. Builtwith — https://builtwith.com/

3. Retire.js for old JS library

4. Ghostery

WAF

Look out for WAFs, you can use WafW00f for that https://github.com/sandrogauci/wafw00f

Popular Google Dorks Use(finding Bug Bounty Websites)

> *site:.eu responsible disclosure*
> *inurl:index.php?id=*
> *site:.nl bug bounty*
> *"index of" inurl:wp-content/ (Identify Wordpress Website)*
> *inurl:"q=user/password" (for finding drupal cms)*

Wordlists/Payloads

raft-large-words.txt, https://github.com/danielmiessler/SecLists

content*discovery*all.txt from jhaddix: https://gist.github.com/jhaddix /b80ea67d85c13206125806f0828f4d10

all.txt from jhaddix — https://gist.github.com/jhaddix /f64c97d0863a78454e44c2f7119c2a6a

PayloadAllTheThings — https://github.com/swisskyrepo /PayloadsAllTheThings

XSS Payloads- http://www.xss-payloads.com/
XSS Payloads — https://github.com/Pgaijin66/XSS-Payloads /blob/master/payload.txt
SQL Injection Payloads — https://github.com/trietptm/SQL-Injection-Payloads
Google-Dorks Payloads — https://gist.github.com/clarketm /919457847cece7ce40323dc217623054

Extracting vhosts

Web Tool — https://pentest-tools.com/information-gathering/find-virtual-hosts
Virtual host scanner — https://github.com/jobertabma/virtual-host-discovery

git clone https://github.com/jobertabma/virtual-host-discovery.git
ruby scan.rb — ip=192.168.1.101 — host=domain.tld

Port Scan

• Scan each individual IP address associated with their subdomains and having the output saved to a file

• Look for any services running on unusual ports or any service running on default ports which could be vulnerable (FTP, SSH, etc). Look for the version info on services running in order to determine whether anything is outdated and potentially vulnerable

1. **Masscan** : https://github.com/robertdavidgraham/masscan

This is an Internet-scale port scanner. It can scan the entire Internet in under 6 minutes, transmitting 10 million packets per second, from a single machine.

Setup:

> *$ sudo apt-get install git gcc make libpcap-dev*
> *$ git clone https://github.com/robertdavidgraham/masscan*
> *$ cd masscan*
> *$ make -j8*

This puts the program in the masscan/bin subdirectory. You'll have to manually copy it to something like /usr/local/bin if you want to install it elsewhere on the system.

Usage:

Shell script to run *dig*

- Because Masscan takes only IPs as input, not DNS names
- Use it to run Masscan against either a name domain or an IP range

```
#!/bin/bash
strip=$(echo $1|sed ‘s/https\?:\/\///’)
echo “”
echo
“#######################################
############”
host $strip
echo
“#######################################
############”
echo “”
masscan -p1–65535 $(dig +short $strip|grep -oE “\b([0–9]{1,3}\.)
{3}[0–9]{1,3}\b”|head -1) — max-rate 1000 |& tee $strip_scan
```

> *Usage: masscan -p1–65535 -iL $TARGET_LIST — max-rate 10000 -oG $TARGET_OUTPUT*
> *# masscan -p80,8000–8100 10.0.0.0/8*
> *# masscan 10.0.0.0/8 -p80 — banners — source-ip 192.168.1.200*

1. Nmap: https://nmap.org/book/man.html

Github For Recon

• Github is extremely helpful in finding Sensitive information regarding the targets. Access-keys, password, open endings, s3 buckets, backup files, etc. can be found on public GitHub repositories.

• Look for below things during a general first assessment(taken from edoverflow):

– API and key. (Get some more endpoints and find API keys.)

– token

– secret

– TODO

– password

– vulnerable 😜

– http:// & https://

Then I will focus on terms that make me smile when developers mess things up:

– CSRF

– random

– hash

– MD5, SHA-1, SHA-2, etc.

– HMAC

Github Recon Tools

1. **gitrob:** https://github.com/michenriksen/gitrob

– Gitrob is a tool to help find potentially sensitive files pushed to public repositories on Github. Gitrob will clone repositories belonging to a user or organization down to a configurable depth and iterate through the commit history and flag files that match signatures for potentially sensitive files. The findings will be presented through a web interface for easy browsing and analysis.

Setup:

> *$ go get github.com/michenriksen/gitrob*

Usage:

gitrob [options] target [target2] … [targetN]

1. **shhgit —** https://github.com/eth0izzle/shhgit

– Shhgit finds secrets and sensitive files across GitHub code and Gists committed in near real time by listening to the GitHub Events API.

Setup:

> *$ go get github.com/eth0izzle/shhgit*

Usage:

• To configure it check the github page.

• Unlike other tools, you don't need to pass any targets with shhgit. Simply run $ shhgit to start watching GitHub commits and find secrets or sensitive files matching the included 120 signatures.

Alternatively, you can forgo the signatures and use shhgit with a search query, e.g. to find all AWS keys you could use

> *shhgit — search-query AWS_ACCESS_KEY_ID=AKIA*

2. **Trufflehog:** https://github.com/dxa4481/truffleHog

– Searches through git repositories for high entropy strings and secrets, digging deep into commit history.

Setup:

> *pip install truffleHog*

Usage:

> *$ truffleHog — regex — entropy=False https://github.com/dxa4481/truffleHog.git*

3. **git-all-secrets —** https://github.com/anshumanbh/git-all-secrets

– It clones public/private github repo of an org and user belonging to org and scan them.

– Clones gist belonging to org and users of org.

Setup:

> *git clone https://github.com/anshumanbh/git-all-secrets.git*

Usage:

> *docker run — rm -it abhartiya/tools_gitallsecrets — help*
> *docker run -it abhartiya/tools_gitallsecrets -token=<> -org=<>*

4. **gitGraber —** https://github.com/hisxo/gitGraber

– monitor GitHub to search and find sensitive data in real time for different online services such as: Google, Amazon, Paypal, Github, Mailgun, Facebook, Twitter, Heroku, Stripe.

Setup:

> *git clone https://github.com/hisxo/gitGraber.git*
> *cd gitGraber*
> *pip3 install -r requirements.txt*

Usage:

> *python3 gitGraber.py -k wordlists/keywords.txt -q "uber" -s*

We recommend creating a cron that will execute the script regulary :

> */15 * * * * cd /BugBounty/gitGraber/ && /usr/bin/python3 gitGraber.py -k wordlists/keywords.txt -q “uber” -s >/dev/null 2>&1

Do it manually:

• A quick Google “Gratipay GitHub” should return Gratipay’s org page on GitHub. Then from there I am going to check what repos actually belong to the org and which are forked. You can do this by selecting the Type: dropdown on the right hand side of the page. Set it to Sources.

• Now, I am going to take a look at the different languages that the projects are written in. My favourite language is Python so I might start focusing on Python projects, but for recon I will mostly just keep note of the different languages.

• After that I will start using the GitHub search bar to look for specific keywords.

org:gratipay hmac

• There are 4 main sections to look out for here.

– Repositories is nice for dedicated projects related to the keyword. For example, if the keyword is “password manager”, I might find they are building a password manager.

– Code is the big one. You can search for classic lines of code that cause security vulnerabilities across the whole organization.

– Commits is not usually my favourite area to look at manually, but if I see a low number I might have a quick look.

– Issues this is the second biggest and will help you all with your recon. **This is the gold mine.**

Companies share so much information about their infrastructure in issue discussions and debates. Look for domains and subdomains in those tickets.

Chris: "Oh, hey John. We forgot to add this certificate to this domain: vuln.example.com."

noted

- "company.com" "dev"
- "dev.company.com"
- "company.com" API_key
- "company.com" password

• “api.company.com” authorization

• others

Read every JS File

Sometimes, Javascript files contain sensitive information including various secrets or hardcoded tokens. It’s always worth to examine JS files manually.
Find following things in Javascript.

• AWS or Other services Access keys

• AWS S3 buckets or other data storage buckets with read/write permissions.

• Open backup sql database endpoints

• Open endpoints of internal services.

JS File Parsing

1. **JSParser**: https://github.com/nahamsec/JSParser

Setup:

> *$ git clone https://github.com/nahamsec/JSParser.git*
> *$ python setup.py install*

Usage:

Run handler.py and then visit *http://localhost:8008*.

> *$ python handler.py*

1. **LinkFinder:** https://github.com/GerbenJavado/LinkFinder

LinkFinder is a python script written to discover endpoints and their parameters in JavaScript files

Setup:

> *$ git clone https://github.com/GerbenJavado/LinkFinder.git*
> *$ cd LinkFinder*
> *$ pip3 install -r requirements.txt*
> *$ python setup.py install*

Usage:

• Most basic usage to find endpoints in an online JavaScript file and output the HTML results to results.html:

> *python linkfinder.py -i https://example.com/1.js -o results.html*

• CLI/STDOUT output (doesn't use jsbeautifier, which makes it very fast):

> *python linkfinder.py -i https://example.com/1.js -o cli*

• Analyzing an entire domain and its JS files:

> *python linkfinder.py -i https://example.com -d*

• Burp input (select in target the files you want to save, right click, Save selected items, feed that file as input):

> *python linkfinder.py -i burpfile -b*

• Enumerating an entire folder for JavaScript files, while looking for endpoints starting with /api/ and finally saving the results to results.html:

> *python linkfinder.py -i 'Desktop/*.js' -r ^/api/ -o results.html*

1. **getJS** — https://github.com/003random/getJS

– A tool to fastly get all javascript sources/files

Setup:

> *go get github.com/003random/getJS*

Usage:

> *cat domains.txt | getJS |tojson*

To feed urls from a file use:

> *$ getJS -input=domains.txt*

2. **InputScanner** — https://github.com/zseano/InputScanner

– A tool designed to scrape a list of URLs and scrape input names (id if no name is found). This tool will also scrape .js urls found on each page (for further testing).

Setup:

Somewhere to run PHP. Recommended to use LAMP/XAMPP locally so you can just run the PHP from your computer locally. You can grab XAMPP from here: https://www.apachefriends.org/index.html.

- Clone in /var/www

> *git clone https://github.com/zseano/InputScanner.git*

Usage:

– Now you're setup, it's time to gather some URLs to test. Use Burp Spider to crawl.

– Once the spider has finished (or you stop it), right click on the host and click "Copy Urls in this host".

– Once copied, paste them into urls.txt. Now open payloads.txt and enter some payloads you wish to inject into each parameter (such as xss" xss' to test for the reflection of " and ' characters on iputs. This will help automate looking for XSS). This script will inject each payload into each parameter.. so the more payloads, the more requests you'll be sending.

– Now visit http://127.0.0.1/InputScanner/ and begin scan

– Once the scanner is complete you will be given 4 txt file outputs (see below). Use the BURP Intruder to import your lists and run through them.

– 4 files are outputted in the /outputs/ folder: JS-output.txt, GET-output.txt, POSTHost-output.txt, POSTData-output.txt.

• GET-output.txt is a file which can be easily imported into a BURP intruder attack (using the Spider type). Set the position in the header (GET §val§ HTTP/1.0) and run the attack. Make sure to play with settings and untick "URL-encode these characters", found on the Payloads tab. Currently the script will echo the HOST url, and I just mass-replace in a text editor such as Sublime. (Replace to null). You are free to modify the script for how you see fit.

• JS-output.xt contains a list of .js urls found on each page. The format is found@https://www.example.com/|https://www.example.com/eg.js|, and this is so you can easily load it into JS-Scan (another tool released by me) and it will let you know where each .js file was found as it scrapes.

• POSTHost-output.txt contains a list of HOST urls (such as https://www.google.com/) which is used for the "Pitchfork" burp intruder attack. Use this file along with POSTData-output.txt. Set attack type to "Pitch fork" and set one position in the header (same as Sniper attack above), and another at the bottom of the request (the post data sent). Make sure to set a Content-Length etc.

• POSTData-output.txt contains post data. (param1=xss"&param2=xss"&param3=xss")

3. **JS-Scan** — https://github.com/zseano/JS-Scan

– A tool designed to scrape a list of .js files and extract urls, as well as juicy information.

Setup:

1. Install LAMP/XAMPP Server.

2. InputScanner to scrape .js files

3. Clone repo:

> *git clone https://github.com/zseano/JS-Scan.git*

Usage:

– Import JS-output.txt file in this interface — http://127.0.0.1/JS-Scan/

WaybackUrl

• Searching for the targets webpages in waybackmachine, the following things can be found.
Old and abandoned JS files.
Old API endpoints.
Abandoned CDN's Endpoints.
Abandoned Subdomains.

Dev & staging endpoint with juicy info in source code comments.
If you are getting 403 on a page, you can also search that 403 pages of targets in way back machine sometimes, you will find them open with helpful information.

1. **waybackurls —** https://github.com/tomnomnom/waybackurls

– Fetch all the URLs that the Wayback Machine knows about for a domain.

Setup:

> *go get github.com/tomnomnom/waybackurls*

Usage:

> *cat domains.txt | waybackurls > urls*

1. **waybackunifier** — https://github.com/mhmdiaa/waybackunifier

– WaybackUnifier allows you to take a look at how a file has ever looked by aggregating all versions of this file, and creating a unified version that contains every line that has ever been in it.

Setup:

> *go get github.com/mhmdiaa/waybackunifier*

Usage:

Syntax:

-concurrency int
Number of requests to make in parallel (default 1)
-output string
File to save results in (default “output.txt”)
-sub string
list of comma-separated substrings to look for in snapshots
(snapshots will only be considered if they contnain one of them)
(default “Disallow,disallow”)
-url string
URL to unify versions of (without protocol prefix) (default
“site.com/robots.txt”)

Webmap

S3 Tools

1. **SubOver** — https://github.com/Ice3man543/SubOver

– A Powerful Subdomain Takeover Tool

Setup:

> *go get github.com/Ice3man543/SubOver*

Usage:

> *./SubOver -l subdomains.txt*

1. **subjack** — https://github.com/haccer/subjack

– Subjack is a Subdomain Takeover tool written in Go designed to scan a list of subdomains concurrently and identify ones that are able to be hijacked

– Subjack will also check for subdomains attached to domains that don't exist (NXDOMAIN) and are **available to be registered**. No need for dig ever again

Setup:

> *go get github.com/haccer/subjack*

Usage:

> *./subjack -w subdomains.txt -t 100 -timeout 30 -o results.txt -ssl*

1. **TakeOver-v1** — https://github.com/samhaxr/TakeOver-v1

– It gives the CNAME of all the subdomains from a file

Setup:

> *git clone https://github.com/samhaxr/TakeOver-v1.git*
>
> *Usage:*
>
> *./takeover.sh subdomain.txt*

1. **subzy** — https://github.com/LukaSikic/subzy

• Subdomain takeover tool which works based on matching response fingerprings from *can-i-take-over-xyz*

Setup:

> *go get -u -v github.com/lukasikic/subzy*
> *go install -v github.com/lukasikic/subzy*

Usage:

> *List of subdomains*
>
> *./subzy -targets list.txt*
> *Single or few subdomains*
>
> *./subzy -target test.google.com*
> *./subzy -target test.google.com,https://test.yahoo.com*

Other/Interesting Tools

1. **Parameth** — https://github.com/maK-/parameth

– This tool can be used to brute discover GET and POST parameters

– Often when you are busting a directory for common files, you can identify scripts (for example test.php) that look like they need to be passed an unknown parameter. This hopefully can help find them.

Setup:

> *git clone https://github.com/maK-/parameth.git*
> *virtualenv venv*
> *. ./venv/bin/activate*
> *pip install -u -r requirements.txt*

Usage:

> *./parameth.py -u http://example.com/test.php*

2. **Arjun** — https://github.com/s0md3v/Arjun

– HTTP parameter discovery suite.

Setup:

> *https://github.com/s0md3v/Arjun.git*

Usage:

– Scanning a single URL

To find GET parameters, you can simply do:

> *python3 arjun.py -u https://api.example.com/endpoint — get*

Similarly, use — post for POST and — json to look for JSON parameters.

– Scanning multiple URLs

A list of URLs stored in a file can be test by using the — urls option as follows

> *python3 arjun.py — urls targets.txt — get*

3. **fuxploitder —** https://github.com/almandin/fuxploider

– File upload vulnerability scanner and exploitation tool.

Setup:

> *git clone https://github.com/almandin/fuxploider.git*
> *cd fuxploider*
> *pip3 install -r requirements.txt*

Usage:

To get a list of basic options and switches use :

> *python3 fuxploider.py -h*

Basic example :

> *python3 fuxploider.py — url https://awesomeFileUploadService.com — not-regex "wrong file type"*

4. **Syborg —** https://github.com/MilindPurswani/Syborg

– Recursive DNS Subdomain Enumerator with dead-end avoidance system

Setup:

Clone the repo using the git clone command as follows:

> *git clone https://github.com/MilindPurswani/Syborg.git*

Resolve the Dependencies:

> *pip3 install -r requirements.txt*

Usage:

> *python3 syborg.py yahoo.com*

At times, it is also possible that Syborg will hit High CPU Usage and that can cost you a lot if you are trying to use this tool on your VPS. Therefore to limit that use another utility called Cpulimit

> *cpulimit -l 50 -p $(pgrep python3)*

This tool can be downloaded as follows:

> *sudo apt install cpulimit*

5. **dnsgen —** https://github.com/ProjectAnte/dnsgen

– Generates combination of domain names from the provided input.

– Combinations are created based on wordlist. Custom words are extracted per execution.

Setup:

> *pip3 install dnsgen*

..or from GitHub:

> *git clone https://github.com/ProjectAnte/dnsgen*
> *cd dnsgen*
> *pip3 install -r requirements.txt*
> *python3 setup.py install*

Usage:

> *$ dnsgen domains.txt (domains.txt contains a list of active domain names)*

Combination with massdns:

> *$ cat domains.txt | dnsgen — | massdns -r /path/to/resolvers.txt -t A -o J — flush 2>/dev/null*

6. **SSRFmap —** https://github.com/swisskyrepo/SSRFmap

– Automatic SSRF fuzzer and exploitation tool

– SSRFmap takes a Burp request file as input and a parameter to fuzz.

Setup:

> *$ git clone https://github.com/swisskyrepo/SSRFmap*
> *$ cd SSRFmap/*
> *$ pip3 install -r requirements.txt*

Usage:

First you need a request with a parameter to fuzz, Burp requests works well with SSRFmap. They should look like the following. More examples are available in the **/data** folder.

```
POST /ssrf HTTP/1.1
Host: 127.0.0.1:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:62.0)
Gecko/20100101 Firefox/62.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*
```

```
/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://mysimple.ssrf/
Content-Type: application/x-www-form-urlencoded
Content-Length: 31
Connection: close
Upgrade-Insecure-Requests: 1

url=https%3A%2F%2Fwww.google.fr
```

Use the -m followed by module name (separated by a , if you want to launch several modules).

> *# Launch a portscan on localhost and read default files*
> *python ssrfmap.py -r data/request.txt -p url -m readfiles,portscan*

7. **nip.io** — https://nip.io/

– Dead simple wildcard DNS for any IP Address

Stop editing your etc/hosts file with custom hostname and IP address mappings.

nip.io allows you to do that by mapping any IP Address to a hostname using the following formats:

– 10.0.0.1.nip.io maps to 10.0.0.1

– 192–168–1–250.nip.io maps to 192.168.1.250

8. **CORS Scanner —** https://github.com/chenjj/CORScanner

– Fast CORS misconfiguration vulnerabilities scanner

Setup:

> *git clone https://github.com/chenjj/CORScanner.git*

– Install dependencies

> *sudo pip install -r requirements.txt*

Usage:

– To check CORS misconfigurations of specific domain:

> *python cors_scan.py -u example.com*

– To check CORS misconfigurations of specific URL:

> *python cors_scan.py -u http://example.com/restapi*

– To check CORS misconfiguration with specific headers:

> *python cors_scan.py -u example.com -d “Cookie: test”*

– To check CORS misconfigurations of multiple domains/URLs:

> *python cors_scan.py -i top_100_domains.txt -t 100*

9. **Blazy —** https://github.com/s0md3v/Blazy

– Blazy is a modern login bruteforcer which also tests for CSRF, Clickjacking, Cloudflare and WAF .

Setup:

> *git clone https://github.com/UltimateHackers/Blazy*
> *cd Blazy*
> *pip install -r requirements.txt*

Usage:

> *python blazy.py*

10. **XSStrike —** https://github.com/s0md3v/XSStrike

– Most advanced XSS scanner.

Setup:

> *git clone https://github.com/s0md3v/XSStrike.git*

Usage:

> *Scan a single URL*
> *Option: -u or — url*
>
> *Test a single webpage which uses GET method.*
>
> *python xsstrike.py -u "http://example.com/search.php?q=query"*
>
> *Supplying POST data*
> *python xsstrike.py -u "http://example.com/search.php" — data "q=query"*

11. **Commix — https://github.com/commixproject/commix**

– Automated All-in-One OS command injection and exploitation tool.

Setup:

> *git clone https://github.com/commixproject/commix.git commix*

Usage:

> *https://github.com/commixproject/commix/wiki/Usage-Examples*
>
> *# python commix.py — url="http://192.168.178.58/DVWA-1.0.8/vulnerabilities/exec/#" — data="ip=127.0.0.1& Submit=submit" — cookie="security=medium; PHPSESSID=nq30op434117mo7o2oe5bl7is4"*

12. **Bolt —** https://github.com/s0md3v/Bolt

– A dumb CSRF scanner

Setup:

> *git clone https://github.com/s0md3v/Bolt.git*

Usage:

Scanning a website for CSRF using Bolt is as easy as doing

> *python3 bolt.py -u https://github.com -l 2*

1. **bass —** https://github.com/Abss0x7tbh/bass

– Bass grabs you those "extra resolvers" you are missing out on when performing Active DNS
enumeration. Add anywhere from 100–6k resolvers to your

“resolver.txt”

Setup:

> *git clone https://github.com/Abss0x7tbh/bass.git*
> *cd bass*
> *pip3 install -r requirements.txt*

Usage:

> *python3 bass.py -d target.com -o output/file/for/final_resolver_list.txt*

1. **meg —** https://github.com/tomnomnom/meg

• meg is a tool for fetching lots of URLs but still being ‘nice’ to servers.

It can be used to fetch many paths for many hosts; fetching one path for all hosts before moving on to the next path and repeating.

Setup:

> *go get -u github.com/tomnomnom/meg*

Usage:

Given a file full of paths:

/robots.txt
/.well-known/security.txt
/package.json

And a file full of hosts (with a protocol):

http://example.com
https://example.com
http://example.net

meg will request each *path* for every *host*:

▶ meg — verbose paths hosts

> *meg <endpoint> <host>*
> *$ meg / https://edoverflow.com*

The latter command requests the top-level directory for https://edoverflow.com (https://edoverflow.com/). It is important to note, that protocols most be specified; meg does not automatically prefix hosts. If you happen to have a list of targets without protocols, make sure to sed the file and add the correct protocol.

> *$ sed 's#^#http://#g' list-of-hosts > output*

By default meg stores all output in an out/ directory, but if you would like to include a dedicated output directory, all it takes is appending the output directory to your command as follows:

> *$ meg / https://edoverflow.com out-edoverflow/*

Say we want to pinpoint specific files that could either assist us further while targeting a platform or be an actual security issue in itself if exposed to the public, all it takes is a list of endpoints (lists/php) and a series of targets (targets-all). For this process, storing all pages that return a "200 OK" status code will help us sieve out most of the noise and false-positives (-s 200).

> *$ meg -s 200 \*
> *lists/php targets-all \*
> *out-php/ 2> /dev/null*

2. **tojson —** https://github.com/tomnomnom/hacks/tree/master/tojson

• Turn lines of stdin into JSON.

Setup:

> *go get -u github.com/tomnomnom/hacks/tojson*

Usage:

> *getJS -url=https://poc-server.com | tojson*
> *ls -l | tojson*

3. **interlace —** https://github.com/codingo/Interlace

• Easily turn single threaded command line applications into a fast, multi-threaded application with CIDR and glob support.

Setup:

> *$ git clone https://github.com/codingo/Interlace.git*
> *$ python3 setup.py install*

Usage:

Let's say we need to run Nikto (a basic, free web server vulnerability scanner) over a list of hosts:

luke$ cat targets.txt
hackerone.com
bugcrowd.com
yahoo.com
google.com

$ interlace -tL ./targets.txt -threads 5 -c "nikto — host _target_ > ./_target_-nikto.txt" -v

```
===================================
===========Interlace v1.2 by Michael Skelton
(@codingo_)============================
==================[13:06:16] [VERBOSE] [nikto —
host yahoo.com > ./yahoo.com-nikto.txt] Added after
processing[13:06:16] [VERBOSE] [nikto — host google.com >
./google.com-nikto.txt] Added after processing[13:06:16]
[VERBOSE] [nikto — host hackerone.com > ./hackerone.com-
nikto.txt] Added after processing[13:06:16] [VERBOSE] [nikto —
host bugcrowd.com > ./bugcrowd.com-nikto.txt] Added after
processing[13:06:16] [THREAD] [nikto — host google.com >
./google.com-nikto.txt] Added to Queue
```

Let's break this down a bit — here's the command I ran:

```
interlace -tL ./targets.txt -threads 5 -c "nikto — host _target_ >
./_target_-nikto.txt" -v
```

– interlace is the name of the tool.

– -tL ./targets.txt defines a file with a list of hosts.

– -threads 5 defines the number of threads.

– -c should be immediately followed by the command you want to run.

– “nikto — host _target_ > ./_target_-nikto.txt” is the actual command which will be run, note that instances of _target_ will be replaced with each line in the ./targets.txt file.

– -v makes it verbose.

Good Articles to Read

1. Subdomain Takeover by Patrik — https://0xpatrik.com/subdomain-takeover/, https://0xpatrik.com/takeover-proofs/

2. Subdomain Enumeration — https://0xpatrik.com/subdomain-enumeration-smarter/, https://0xpatrik.com/subdomain-enumeration-2019/

3. Can-I-take-over-xyz — https://github.com/EdOverflow/can-i-take-over-xyz

4. File Upload XSS — https://brutelogic.com.br/blog/file-upload-xss/

5. Serverless Toolkit for Pentesters — https://blog.ropnop.com/serverless-toolkit-for-pentesters/

6. Docker for Pentesters — https://blog.ropnop.com/docker-for-pentesters/

7. For coding — https://learnxinyminutes.com/

8. Android Security Lab Setup — https://medium.com/@ehsahil/basic-android-security-testing-lab-part-1-a2b87e667533

9. SSL Bypass — https://medium.com/@ved_wayal/hail-frida-the-universal-ssl-pinning-bypass-for-android-e9e1d733d29

10. Bypass Certificate Pinning — *https://blog.it-securityguard.com/the-stony-path-of-android-%F0%9F%A4%96-bug-bounty-bypassing-certificate-pinning/*

11. Burp macros and Session handling — https://digi.ninja/blog/burp_macros.php

12. Burp Extensions — https://blog.usejournal.com/bug-hunting-methodology-part-2-5579dac06150

13. JSON CSRF to form data attack — https://medium.com/@osamaavvan/json-csrf-to-formdata-attack-eb65272376a2

14. meg — https://edoverflow.com/2018/meg/

15. assetnote: https://github.com/tdr130/assetnote Push notifications for the new domain

16. interlace : https://medium.com/@hakluke/interlace-a-productivity-tool-for-pentesters-and-bug-hunters-automate-and-multithread-your-d18c81371d3d

17. http-desync-attacks-request-smuggling-reborn- https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn

Scripts

• the art of subdomain enumeration — https://github.com/appsecco/the-art-of-subdomain-enumeration

• Setup Bug Bounty tools : https://gist.github.com/LuD1161/66f30da6d8b6c1c05b9f6708525ea885

• ReconPi — https://github.com/x1mdev/ReconPi/tree/dev/v2.0

• TotalRecon — Insalls all the tools — https://github.com/vitalysim/totalrecon

• Auto Recon Bash Script — https://github.com/mehulpanchal007/venom

Recon My Way

1. Do Subdomain enumeration using amass, assetfinder, subfind

> *amass enum — passive -d <DOMAIN>*
>
> *assetfinder — subs-only <domain>*
>
> *subfinder -d freelancer.com*
>
> *subfinder -d <domain> -recursive -silent -t 200 -v -o <outfile>*

2. Use commonspeak2 wordlist to get probable permutations of above subdomains.

https://github.com/assetnote/commonspeak2-wordlists

To generate the possibilities, you can use this simple Python snippet:

```
scope = '<DOMAIN>'
wordlist = open('./commonspeak2.txt').read().split('\n')

for word in wordlist:
if not word.strip():
continue
print('{}.{}\n'.format(word.strip(), scope))
```

3. Use massdns to resolve all the above domains:

```
./bin/massdns -r lists/resolvers.txt -t A domains.txt > results.txt
```

4. To get the best resolvers.txt use bass tool:

```
python3 bass.py -d target.com -o output/file
/for/final_resolver_list.txt
```

5. Use dnsgen to generates combination of domain names from the provided input.

```
cat domains.txt | dnsgen — | massdns -r /path/to/resolvers.txt -t A
-o J — flush 2>/dev/null
```

6. Port Scan Using masscan and nmap for version scan

Shell script to run *dig*

– Because Masscan takes only IPs as input, not DNS names

– Use it to run Masscan against either a name domain or an IP range

```
#!/bin/bash
strip=$(echo $1|sed 's/https\?:\/\///')
echo “”
echo
“#######################################
###########”
host $strip
```

echo
“###”
echo “”
masscan -p1–65535 $(dig +short $strip|grep -oE “\b([0–9]{1,3}\.){3}[0–9]{1,3}\b”|head -1) — max-rate 1000 |& tee $strip_scan

or

nmap -iL list.txt -Pn -n -sn -oG output.txt
masscan -iL output.txt -p 0–65535 — max-rate 1000

or run massscan + nmap using below script

wget https://raw.githubusercontent.com/PacktPublishing/Mastering-Kali-Linux-for-Advanced-Penetration-Testing-Third-Edition/master/Chapter03/massNmap.sh

nmap scan with output in an nice xml file !
$ sudo nmap -sS -T4 -sC -oA my*report*name — stylesheet *https://raw.githubusercontent.com/honze-net/nmap-bootstrap-xsl/master/nmap-bootstrap.xsl* -iL subdomain.txt

7. Do github recon

8. Take screenshot using aquatone.

cat hosts.txt | aquatone -out ~/aquatone/example.com

9. Run ffuz or gobuster to directory bruteforce/content discovery on a particular domain/subdomain

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

10. Read JS file, get the endpoints, check if there is any secret token/key in JS files.

11. Use waybackurls to get old JS files, and 403 files.

– Generate wordlist using wayback

# curl -s “http://web.archive.org/cdx/search /cdx?url=hackerone.com/*&output=text&fl=original& collapse=urlkey" | sed ‘s/\//\n/g’ | sort -u | grep -v ‘svg\|.png \|.img\|.ttf\|http:\|:\|.eot\|woff\|ico\|css\|bootstrap \|wordpress\|.jpg\|.jpeg’ > wordlist

or

# curl -L http://xrl.us/installperlnix | bash
# curl -s “http://web.archive.org/cdx/search /cdx?url=hackerone.com/*&output=text&fl=original& collapse=urlkey" | sed ‘s/\//\n/g’ | sort -u | grep -v ‘svg\|.png

\|.img\|.ttf\|http:\|:\|.eot\|woff\|ico\|css\|bootstrap \|wordpress\|.jpg\|.jpeg' | perl -pe 's/\%(\w\w)/chr hex $1/ge' > wordlist.txt

12. One liner to import whole list of subdomains into Burp suite for automated scanning!

> *cat <file-name> | parallel -j 200 curl -L -o /dev/null {} -x 127.0.0.1:8080 -k -s*