**Web Application Enumeration**

**------------------------------------------------------------------------------------**

* + **Web application Enumeration**
    - **Steps**
    - **Content Discovery Vs Assets Discovery**
    - **Subdomain Enumeration**
    - **Finding Live Subdomains**
    - **visual Recon (Screenshots)**
    - **Waybackurl Enum**
    - **Analyzing web technologies**
    - **Websites Crawling**
    - **Directory Brute forcing**
    - **Parameter discovery (discover new/hidden parameters)**
    - **JavaScript Analysis**
    - **Virtual hosts scanning (Discovery)**
    - **Large Scope Reconnaissance (Asset Discovery)**
    - **Others**
    - **Frameworks & Automated Tools**
  + **SSL scanning**

**------------------------------------------------------------------------------------**

* + **Web application Enumeration**
    - **Steps**
      * **Medium Scope Reconnaissance**
        + Step 1:

Subdomain Enumeration

* + - * + Step 2:

Getting Live Subdomains

* + - * + Step 3:

visual Recon (aquatone)

WayBackUrl enum

* + - * + Step 4:

Analyzing web technologies

Website Crawling

Directory Bruteforcing

* + - * + Step 5:

Check hidden Parameters

Check JavaScript codes

* + - * **Large Scope Reconnaissance** 
        + Autonomous system number (ASN) Recon
        + Acquisitions
      * **Others** 
        + GitHub leaks
        + s3 buckets
        + subdomain takeover
      * **Automated Tools**
      * **Resources**
    - **Content Discovery Vs Assets Discovery** 
      * **Assets Discovery**
        + Asset discovery consists of finding all the digital online assets owned by a particular company. For example: Google owns google.com, YouTube.com, nest.com (acquisition), and a ton of other assets that may be a part of their bug bounty program. You can usually find these through searching on Google, Crunchbase, or Wikipedia. Each of these assets have a domain and each domain may have its own subdomains. For example Google.com has mail.google.com, drive.google.com, cloud.google.com, store.google.com and more.
        + Services like crt.sh, censys.io and shodan …etc. allow you to search through their data and look for certificates for a particular organization. If a subdomain doesn’t have a SSL Certificate we may rely on google or on tools that enumerate subdomains via Bruteforcing

IP addresses

domain names/certificate transparency (crt.sh)

subdomains

open ports

* + - * **Content Discovery**
        + Content discovery helps us to find additional/sensitive folders, endpoints (files), documentation, or features that may not be directly accessible on the site. This also includes finding administration panels, files with sensitive debug/diagnosis information, API documentation, or hidden features available to for users with higher privileges. This can be done by crawling the site using Burp Suite or by using other tools.

Files or Folders

Hidden Endpoints or Functionality

* + - **Subdomain Enumeration**
      * **Overview**
        + **What is Subdomain Enumeration?**

Sub-domain enumeration is the process of finding sub-domains for one or more domain(s). It is an essential part of the reconnaissance phase.

* + - * + **Subdomain Enumeration Techniques**

Passive Enumeration (Subdomain Scrapping)

Subdomain Enumeration by Google Dorking

Subdomain Enumeration by Certificate Transparency (CT)

Subdomain Enumeration by Online DNS tools and services (DNS Aggregators)

Subdomain Enumeration by FDNS Dataset

Subdomain Scrapping Using Tools

Amass

Subfinder

Active Enumeration

Subdomain brute forcing

Subdomain Enumeration by HTTP(CSP) Header

Subdomain Enumeration by Zone Transfer

Subdomain Enumeration by Dnssec walking

* + - * **Passive Subdomain Enumeration (Subdomain Scrapping)**
        + **Subdomain Enumeration by Google Dorking**

Search engines like Google and Bing supports various advanced search operators to refine search queries. These operators are often referred to as “Google dorks”.

We can use “site:” operator in Google search to find all the sub-domains that Google has found for a domain. Google also supports additional minus operator to exclude sub-domains that we are not interested in Example

“site:\*.wikimedia.org -www -store -jobs -uk”

Google dorks are useful for finding new subdomains

site:wikipedia.org

site:\*.wikipedia.org -www -store -jobs -uk

Bing also support dorks and can give others results

site:

When you use the Google Dork: site:\*.example.com, NEVER forget to check

Site: \*.\*.example.com

site: \*.\*.\*.example.com

**Also** You can Google the:

Copyright text

Terms of service text

Privacy Policy text

**Example**

"© 2006-2020 Company, Corp." -site:\*.domain.com inurl:company

"© 2019 twitch Interactive, Inc." inurl:twitch

"© 2019 twitch Interactive, Inc." inurl:twitch -www.domain.com

* + - * + **Subdomain Enumeration by Certificate Transparency (CT)**

When an SSL/TLS (Secure Sockets Layer/Transport Layer Security) certificate is created for a domain by a CA (Certificate Authority), CA's take part in what's called "Certificate Transparency (CT) logs". These are publicly accessible logs of every SSL/TLS certificate created for a domain name (Certificate Authority (CA) has to publish every SSL/TLS certificate they issue to a public log). The purpose of Certificate Transparency logs is to stop malicious and accidentally made certificates from being used. We can use this service to our advantage to discover subdomains belonging to a domain

An SSL/TLS certificate usually contains domain names, sub-domain names and email addresses. This makes them a treasure trove of information for attackers.

The easiest way to lookup certificates issued for a domain is to use search engines that collect the CT logs and let’s anyone search through them. Few of the popular ones are listed below -

<https://crt.sh/?q=%25taregt.com>

curl -s https://crt.sh/\?q\=**%25.example.com**\&output\=json | jq -r '.[].name\_value' | sed 's/\\*\.//g' | sort -u

https://censys.io/

https://developers.facebook.com/tools/ct/

<https://google.com/transparencyreport/https/ct/>

The downside of using CT for sub-domain enumeration is that the domain names found in the CT logs may not exist anymore and thus they can’t be resolved to an IP address. You can use tools like massdns in conjunction with CT logs to quickly identify resolvable domain names.

# ct.py - extracts domain names from CT Logs(shipped with massdns)

# massdns - will find resolvable domains & adds them to a file

./ct.py icann.org | ./bin/massdns -r resolvers.txt -t A -q -a -o -w icann\_resolvable\_domains.txt –

Resources

https://blog.appsecco.com/certificate-transparency-the-bright-side-and-the-dark-side-8aa47d9a6616

* + - * + **Subdomain Enumeration by Online DNS tools and services**

**Overview**

Also known as subdomain Scraping or DNS Aggregators

There are a lot of the third party services that aggregate massive DNS datasets and look through them to retrieve sub-domains for a given domain.

VirusTotal runs its own passive DNS replication service, built by storing DNS resolutions performed when visiting URLs submitted by users. In order to retrieve the information of a domain you just have to put domain name in the search bar

DNSdumpster is another interesting tool that can find potentially large number of sub-domains for a given domain. DNSDumpster has a good feature that it can represent the collected information graphically. DNSDumpster displayed the data from the table earlier as a graph. You can see the DNS and MX branching to their respective servers and also showing the IP addresses

**Example**

VirusTotal Passive DNS replication

Netcraft - Search DNS

https://pentest-tools.com/

<https://dnsdumpster.com/>

<http://dnsgoodies.com>

<https://www.shodan.io/>

<https://censys.io>

* + - * + **Subdomain Enumeration by FDNS Dataset**

There are projects that gather Internet wide scan data and make it available to researchers and the security community. The datasets published by these projects are a treasure trove of sub-domain information.

Forward DNS dataset is published as part of Project Sonar. This data is created by extracting domain names from a number of sources and then sending an ANY query for each domain. The data format is a gzip-compressed JSON file. We can parse the dataset to find sub-domains for a given domain. The dataset is massive though (20+GB compressed, 300+GB uncompressed)

Command to parse & extract sub-domains for a given domain

curl -silent https://scans.io/data/rapid7/sonar.fdns\_v2/20170417-fdns.json.gz | pigz -dc | grep “.icann.org” | jq

wget https://opendata.rapid7.com/sonar.fdns\_v2/2019-11-29-1574985929-fdns\_a.json.gz

cat 20170417-fdns.json.gz | pigz -dc | grep ".target.org" | jq

—

SonarSearch is a project built around the Rapid7 Sonar's database

https://github.com/Cgboal/SonarSearch

An API instance is available at <https://omnisint.io/>. Crobat is a CLI tool designed to query this API

crobat -h

Usage of crobat:

-r string

Perform reverse lookup on IP address or CIDR range. Supports files and quoted lists

-s string

Get subdomains for this value. Supports files and quoted lists

-t string

Get tlds for this value. Supports files and quoted lists

-u Ensures results are unique, may cause instability on large queries due to RAM requirements

* + - * + **Subdomain Scrapping Using Tools**

**Overview**

Doing this manually is a headache so the next set of tools scrape domain information from all sorts of projects that expose databases of URLs or domains

**Amass**

Amass enum -d domain.com

./amass enum --passive -d example.com -o result1.txt

**Subfinder**

Installation

apt-get install golang

go install -v github.com/subfinder/subfinder@latest

cd subfinder

go build

Usage

./subfinder -d example.com -o result.txt -v

**Assetfinder**

Assetfinder -subs-only domain.com | tee -a all\_sub.txt

**Sublist3r**

Installation

git clone https://github.com/aboul3la/Sublist3r

cd Sublist3r

pip install -r requirements.txt

alias sublist3r='python3 /path/to/Sublist3r/sublist3r.py -d '

Usage

Python3 sublist3r.py -v -d [website] -o result.txt

**Github-subdomain.py**

This script is good for scraping GitHub for subdomains

Github-subdomain.py -d “domain.com” -t github\_token > domain.txt

Github-subdomain.py -d “example.com” -t ghp\_5QzenXw4OQPEsSO6AuhFIL1kZQUc5b48FApS > domain.txt

To Generate a GitHub token

go to your github > click on your picture on the top right > click on settings > go to developer settings > go to personal access tokens

ghp\_5QzenXw4OQPEsSO6AuhFIL1kZQUc5b48FApS

**Dnsrecon**

dnsrecon -d microsoft.com -g

-g: perform Google enumeration with standard enumeration

**Theharvester**

theharvester -d microsoft.com -b google -l 200 -f /root/Desktop/msresults.html

We will use google as the search engine, we will limit the results to 200, and also store the results into a HTML file

* + - * **Active Subdomain Enumeration**
        + **Subdomain Brute forcing**

**Overview**

Brute force means guessing possible combinations of the target until the expected output is discovered. So, in the subdomain context, the brute-forcing is to try the possible combination of words, alphabets, and numbers before the main domain in order to get a subdomain that is resolved to IP address. Sometimes subdomains are not indexed on search engines and are not available on online DNS aggregators sites in that case brute forcing is the best way to find out the subdomains which may have been forgotten by the organization. It is like a treasure for an adversary.

Two techniques for subdomain enumeration using brute force:

Dictionary brute force

In the dictionary brute force, we directly use the wordlist to a brute force domain name to find valid subdomains. Example Tools:

Sublister

Amass

Subbrute

Knockpy

Fierce

Subfinder

Aquatone

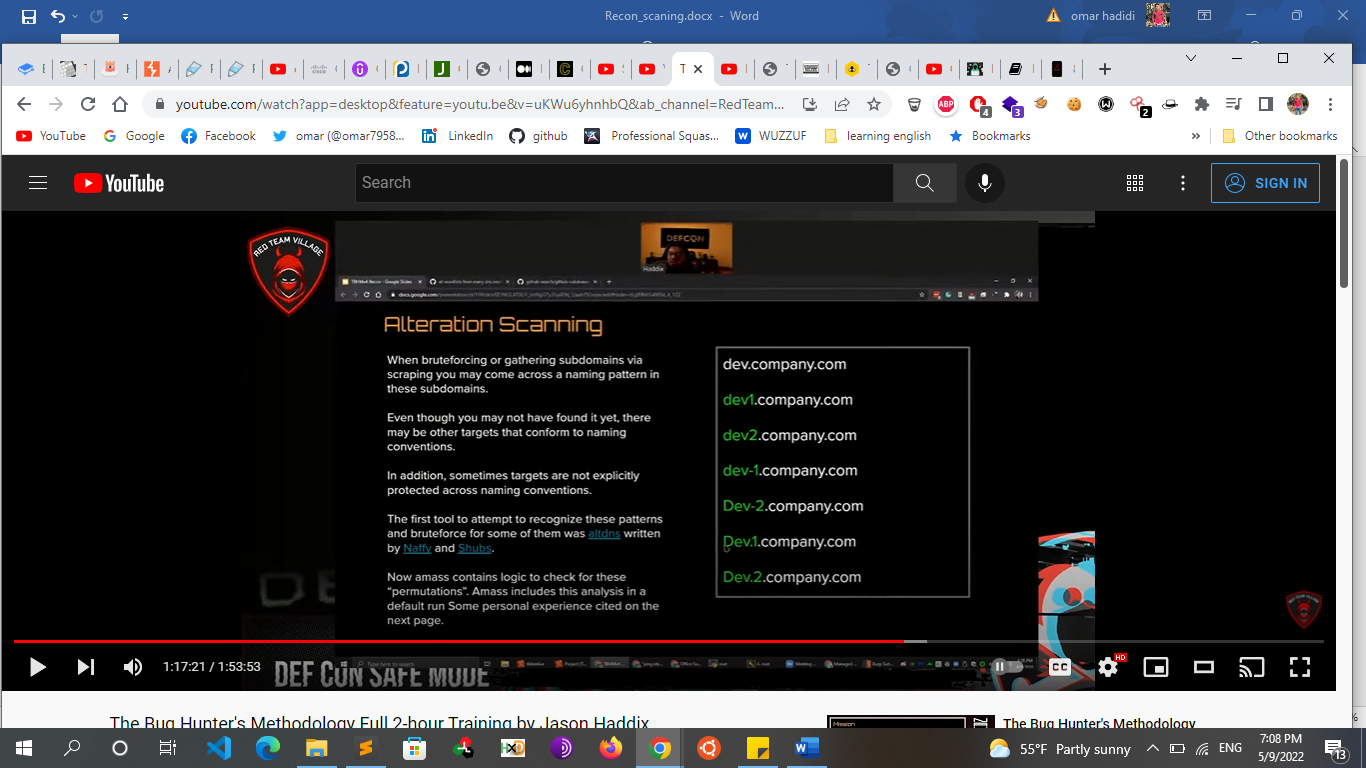
Bluto-Old

DNS-Discovery

Dnssearch

Permutation brute force

In permutation brute force, we create a new resolved subdomain list from already known subdomains/domains by using permutation, mutation, and alteration with a wordlist.



Tool:

Altdns

dnsgen

**Dictionary Bruteforce Tools**

**Sublist3r**

Installation

git clone https://github.com/aboul3la/Sublist3r

cd Sublist3r

pip install -r requirements.txt

alias sublist3r='python3 /path/to/Sublist3r/sublist3r.py -d '

Usage

Python3 sublist3r.py -v -d [website] -b -o result.txt

**Amass**

Installation

https://github.com/OWASP/Amass/releases

unzip amass\_v3.1.9\_linux\_amd64.zip

cd amass\_v3.1.9\_linux\_amd64

Usage

amass enum --active -d example.com -o result1.txt

Amass enum -brute -d example.com -src

**Knokpy**

Installation

Git clone <https://github.com/guelfoweb/knock.git>

Usage

Python Knockpy.py [website]

Python Knockpy.py [website] -w subdomains.txt

**Subbrute**

python subbrute.py [website]

**massdns**

./subbrute.py /root/work/bin/all.txt $TARGET.com | ./bin/massdns -r resolvers.txt -t A -a -o -w massdns\_output.txt

**Fierce**

Fierce -dns target.com

Fierce -dns target.com -wordlist <name.txt>

**Gobuster**

the "dns" mode. This allows Gobuster to brute-force subdomains.

gobuster -m dns -u google.com -w -w /usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt

-t [nb]: set threads number

**-fw option** to enable force processing parameter to continue the attack even if there is any Wildcard Domain

gobuster -m dns -u google.com -t 100 -w /usr/share/wordlists/dirb/common.txt -i

show subdomains with their ip

-cn : show their cname

**WFUZZ**

Wfuzz -c -f sub-fighter -w wordlist.txt -u “http:// example.com” -H “HOST:FUZZ.example.com”

**Recon-ng**

Netcraft Module

Show workspaces

Workspaces add google.com

Use netcraft

Show options

Set source google.com

Run

Show hosts

Resolve Module (find ip for each subdomain)

Use recon/hosts-hosts/resolve

Show options

Run

Show hosts

Freegeoip (find location of each server)

Use freegeoip

Show options

Run

Show hosts

Report

Use reporting/html

Show options

Set CREATOR omar

Set CUSTOMER Hackerone

Set FILENAME /root

run

**Notes**

Wordlists

Daniel Miessler’s DNS Discovery.

Commonspeak2

https://github.com/assetnote/commonspeak2-wordlists/blob/master/subdomains/subdomains.txt

Jhaddix’s all.txt

The all.txt is a collection of all the different wordlist used by all the different sub-domain bruteforcing tools. You can find it over here Jason Haddix' subdomain compilation

<https://gist.githubusercontent.com/jhaddix/86a06c5dc309d08580a018c66354a056/raw/f58e82c9abfa46a932eb92edbe6b18214141439b/all.txt>

* + - * **Subdomain Enumeration by HTTP Header**
        + Content Security Policy (CSP) is a response header that tells the browser from what sources it is allowed to include and execute resources.
        + It is like a filter where sources are mentioned and in sources, the domains and subdomains are mentioned. An adversary may use this information to enumerate more subdomains and other domains that are allowed by the organization.

<https://github.com/0xbharath/domains-from-csp>

* + - * + Sometimes you will find pages that only return the header Access-Control-Allow-Origin when a valid domain/subdomain is set in the Origin header. In these scenarios, you can abuse this behavior to discover new subdomains.

ffuf -w subdomains-top1million-5000.txt -u http://10.10.10.208 -H 'Origin: http://FUZZ.crossfit.htb' -mr "Access-Control-Allow-Origin" -ignore-body

* + - * **Subdomain Enumeration by Zone Transfer**
        + DNS zone transfer is the process of replication DNS database or DNS records from the primary name server to the secondary name server.

Zone transfer is a type of DNS transaction where a DNS server passes a copy of full or part of it’s zone file to another DNS server

* + - * + The DNS zone transfer functionality used by an adversary only when the primary name server is configured to replicate the zone information to any server. An adversary acts as a slave and asks the master for a copy of the zone records. If zone transfers are not securely configured, anyone can initiate a zone transfer against a nameserver and get a copy of the zone file. By design, zone file contains a lot of information about the zone and the hosts that reside in the zone
        + Only a list of pre-defined hosts should be able to perform this operation. However, it's sometimes possible to retrieve information from internal zones by asking publicly available servers.
        + **Types of information gathered by zone transfer**

Subdomains

Email address (Could help in phishing)

Current serial number (Change in serial no. may indicate changes in the organization)

Email server (Could help in phishing)

Name server

Location may be disclosed by LOC record

The IP address of the website

Services running in the organization by SRV record

* + - * + Example

dig +multi AXFR @ns1.insecuredns.com insecuredns.com

dig AXFR int @z.hackycorp.com

dig AXFR internal\_zone @z.hackycorp.com

dnsrecon -a -d tesla.com

* + - * **Subdomain Enumeration by Dnssec walking**
        + Due to the way non-existent domains are handled in DNSSEC, it is possible to “walk” the DNSSEC zones and enumerate all the domains in that zone. You can learn more about this technique from here.
        + For DNSSEC zones that use NSEC records, zone walking can be performed using tools like ldns-walk
        + → zone walking

ldns-walk @ns1.insecuredns.com insecuredns.com

ldns-walk insecuredns.com

* + - * + Some DNSSEC zones use NSEC3 records which uses hashed domain names to prevent attackers from gathering the plain text domain names. An attacker can collect all the sub-domain hashes and crack the hashes offline. Tools like nsec3walker, nsec3map help us automate the collecting NSEC3 hashes and cracking the hashes. Once you install nsec3walker, you can use the following commands to enumerate sub-domains of NSEC3 protected zone

# Collect NSEC3 hashes of a domain

./collect icann.org > icann.org.collect

cat icann.org.unhash | grep "icann" | awk '{print $2;}'

# Undo the hashing, expose the sub-domain information.

./unhash < icann.org.collect > icann.org.unhash

# Listing only the sub-domain part from the unhashed data

cat icann.org.unhash | grep "icann" | awk '{print $2;}'

* + - * + **Resources**

https://niiconsulting.com/checkmate/2020/10/active-subdomain-enumeration-part-2/

* + - * **Notes**
        + collect all these informations from subdinder + amass + crtfinder + sublist3r + google\_dorks etc… and collect all of them into one text file all\_subdomains.txt Now we have 1 text file contains all subdomains: **all\_subdomains.txt**

use altdns to collect subdomains form subdomains, for example sub.sub.sub.domain.com. collect all these info into text file altdns\_subdomain.txt

* + - * + **pass all\_subdomains.txt to 2 directions**

Finding Live Subdomains (httpx/httprobe)

Next section

Port Scanning: nmap or masscan to scan all ports and discover open ports + try to brute force these open ports if you see that these ports may be brute forced, use **brute-spray** to brute force these credentials collect all the results into text file nmap\_results.txt (run it on VPS because take lot time)

Method 1

Nmap -p- -sC -iL subdomains.txt -oN scaned-port.txt

nmap -sV -iL subdomains.txt -oN scaned-port.txt --script=vuln

Method 2

Cat subdomains.txt | xargs -nl host | grep “has address” | cut -d “ “ -f4 | sort -u > ipfile

Manual way to Get ip addresses for port scanning

Massscan -p1-65535 -iL $ipfile —max-rate 1800 -oG $outPutfile

* + - **Finding Live Subdomains**
      * **Overview** 
        + Pass the text file over httpx or httprobe , these tools will filter all subdomains and return only live subdomains which works on ports 80 and 443
        + take these live subdomains and collect them into separate file live\_subdomains.txt. Now we have 2 text files all\_subdomains.txt + live\_subdomains.txt
      * **Tools**
        + **HTTProbe**

**Example**

cat subdomains | httprobe | tee -a live\_domains

* + - * + **HTTPX**

**Example**

cat subdomains.txt | httpx -o live-subdomains.txt

* + - * **Notes**
        + **pass live\_subdomains.txt to 2 directions**

take the live\_subdomains.txt file and pass it over waybackurls tool to collect all links which related to all live subdomains and collect all these links into new file waybackurls.txt

in the next section

Also pass it to a visual recon tool to take screenshots

In the Next section

* + - * + **Or you can skip these 2 steps and pass it directly over directory brutefocing**
        + Now we have 3 text files all\_subdomains.txt + live\_subdomains.txt+ waybackurls.txt. take all subdomains text file and pass it over dirsearch or ffuf to discover all hidden directories.

<https://community.ibm.com/database_conf.txt>

* + - * + Also use live\_subdomains.txt and search for credentials in GitHub by using automated tools like GitHound or by manual search collect all these informations into text file GitHub\_search.txt
    - **visual Recon (Screenshots)**
      * Now that you have discovered all the web servers present in the scope (among the IPs of the company and all the domains and subdomains) you probably don't know where to start. So, let's make it simple and start just taking screenshots of all of them. Just by taking a look at the main page you can find weird endpoints that are more prone to be vulnerable.
      * Get all the domains in one file, make sure all domains are listed only once (uniq command in Linux). Provide the list to a tool which will make a screen shot of every domain:
        + Aquatone
        + EyeWitness
        + webscreenshot.py
      * **Eyewitness**
        + An example of usage:

sudo ./EyeWitness.py -f live\_subdomains.txt --prepend-https --web

sudo ./EyeWitness.py -f live\_subdomains.txt --headless --prepend-https

The --prepend-https options make sure it will take a screen shot of port 80 (HTTP) and 443(HTTPS)

WebScreenshot

wmap : Chrome extension.

* + - * **Aquatone**
        + When you've used aquatone-discover to gather sub-domains, you can use the aquatone-gather to make screenshots of the sub-domains. Make sure you first run aquatone-scan.
        + Requirements For Aquatone

install google chrome

Method 1

wget https://dl.google.com/linux/direct/google-chrome-stable\_current\_amd64.deb

sudo apt install ./google-chrome-stable\_current\_amd64.deb

Method 2

sudo apt-get update

sudo apt-get install chromium-browser

or sudo apt-get install chromium

or sudo add-apt-repository ppa:saiarcot895/chromium-beta

using snapd

snap install chromium

which chromium

aquatone -chrome-path /snap/bin/chromium

* + - * + install aquatone

Method 1

Wget <https://github.com/michenriksen/aquatone/releases/download/v1.7.0/aquatone_linux_amd64_1.7.0.zip>

unzip aquatone\_linux\_amd64\_1.7.0.zip

./aquatone –help

sudo mv aquatone /usr/local/bin/

Method 2

apt-get install gem

gem install aquatone

* + - * + Usage

cat live\_subdomains.txt | aquatone

python3 -m HTTPServer 8000

open the output folder in browser to see the html report & screenshots

* + - * **webscreenshot**
        + Requirements for WebScreenshot

pip install webscreenshot pip install selenium

* + - * + Usage

python webscreenshot.py -i live\_domains output example

* + - * **Note**
        + 401/403 response

After all the screenshots are made, go through the list and search for the ones which gave you a 401/403 response. Copy and paste these and throw them in a list. Use the waybackmachine and check if you can find some directories. Maybe the organization forgot to put the right permissions on certain directories or files. Tools u can use for this:

WayBackUrls

WayBackUnifier

ReconCat

* + - **Waybackurl Enum**
      * **Overview**
        + Archive.org is another great resource to find old robots.txt files that may contain old endpoints and other sensitive information, find older versions of the site in order to analyze the source and gather more information, and find other old and forgotten subdomains and dev environments. We can do all of this by just visiting Archive.org, searching for our target, picking an older date (maybe from a year or two ago), and clicking around the website.
        + Archive.org is also a great place to find older javascript files that may have still be available to read from. Using this method, we can find more outdated functionalities and endpoints.
        + After gathering a list of old and new javascript files, we can create a full list of all the endpoints mentioned in those javascript files using **JSParser:**

https://github.com/nahamsec/JSParser

* + - * + This could also be automated (and it has been). Using waybackurl.py and waybackrobots.txt, we can find all the information above by running one of the scripts and waiting for the results.

WaybackMachine (or WaybackUnifier, a wrapper around Waybackmachine) Given a URL, it queries Waybackmachine for all its versions, tracks the unique parts from each version & creates a unified file that contains these unique parts

* + - * + You can use it on:

robots.txt (trick shared by @zseano)

API documentation pages (trick shared by @filedescriptor)

JS files

Look for old endpoints & leaked API keys (it doesn’t matter if the syntax is messed up)

HTML pages to find comments disclosing sensitive information, more JS code, old endpoints, old input names

* + - * + You can also uses as CLI tool

https://github.com/tomnomnom/waybackurls

cat live\_domains.txt | waybackurls > urls

* + - * **Notes**
        + Cache pages

http://cachedview.com/

<https://www.giftofspeed.com/cache-checker/>

* + - * + **Pass the waybackurl.txt to 2 directions**

JS parser: use grep to collect all js files form waybackurls.txt as cat waybackurls.txt | grep js > js\_files.txt

Also pass waybackurls.txt file over gf tool and use gf-patterns to filter the links to possible vulnerable links, for example if the link has parameter like ?user\_id= so this link may be vulnerable to SQLI or IDOR, if the link has parameter like ?page= so this link may be vulnerable to lfi. collect all this vulnerable links into directory vulnerable\_links.txt and into this directory have separated text files for all vulnerable links gf\_sqli.txt , gf\_idor.txt ,etc.

Also pass all\_subdomain.txt + waybackurls.txt + vulnerable\_links.txt to nuclei “Automation scanner” to scan all of them

* + - **Analyzing web technologies**
      * **Overview**
      * **Using Tools** 
        + Wapplyzer extension — Wappalyzer is a browser extension that uncovers the technologies used on websites. It detects CMS, e-commerce platforms, web servers, frameworks, analytics tool and many more.
        + Netcraft- It will also allow you to detect back-end technologies using by your targets.
        + Built with- This will let you know that which technology your target is using on their backend.
        + What web- this is a next-generation web scanner that identifies technologies used by websites. This tool comes preinstalled in your Linux.

whatweb –v [website]

* + - * + Dmitry Tool

dmitry -winsep [[website]](http://example.com)

dmitry -winsepo result.txt [[website]](http://example.com)

* + - * **Using HTTP Headers**
        + When we make requests to the web server, the server returns various HTTP headers. These headers can sometimes contain useful information such as the webserver software and possibly the programming/scripting language in use. Using this information, we could find vulnerable versions of software being used.
      * **Knowing the Framework Through Favicon** 
        + The favicon is a small icon displayed in the browser's address bar or tab used for branding a website. Sometimes when frameworks are used to build a website, a favicon that is part of the installation gets leftover, and if the website developer doesn't replace this with a custom one, this can give us a clue on what framework is in use. OWASP host a database of common framework icons that you can use to check against the targets favicon https://wiki.owasp.org/index.php/OWASP\_favicon\_database. Once we know the framework stack, we can use external resources to discover more about it
        + **Example:**

If you run the following command , it will download the favicon and get its md5 hash value which you can then lookup on the

https://wiki.owasp.org/index.php/OWASP\_favicon\_database.

curl https://static-labs.tryhackme.cloud/sites/favicon/images/favicon.ico | md5sum

* + - * **Notes**
        + **Programing languages used**

Wappalyzer extension / whatweb / through error / inspecting extensions in URLs

* + - * + **Web servers used**

Wappalyzer extension / whatweb / through error

* + - * + **Database’s software used**

Wappalyzer extension / through error

* + - * + **Server operating system**

Wappalyzer extension / NSE

* + - **Websites Crawling** 
      * **What is Crawling?**
        + Crawling a website is the process of browsing a website in a methodical or automated manner to enumerate all the resources encountered along the way. It gives us the structure of the website we are auditing and an overview of the potential attack surface for our future tests. A crawler finds files and folders in a website because these appear in web page links, comments or forms.
        + Parallel to brute-forcing for sensitive assets, spidering can help you get a picture of a site that, without a sitemap
      * **Manually Crawling:**
        + try to browse the entire app in a normal way by visiting every link , submitting every form , without having to take notes or screenshots. Burp allows you to link your browser to the application's proxy, where it will then keep a record of all the pages you visit as you step through the site
        + **Robots.txt**

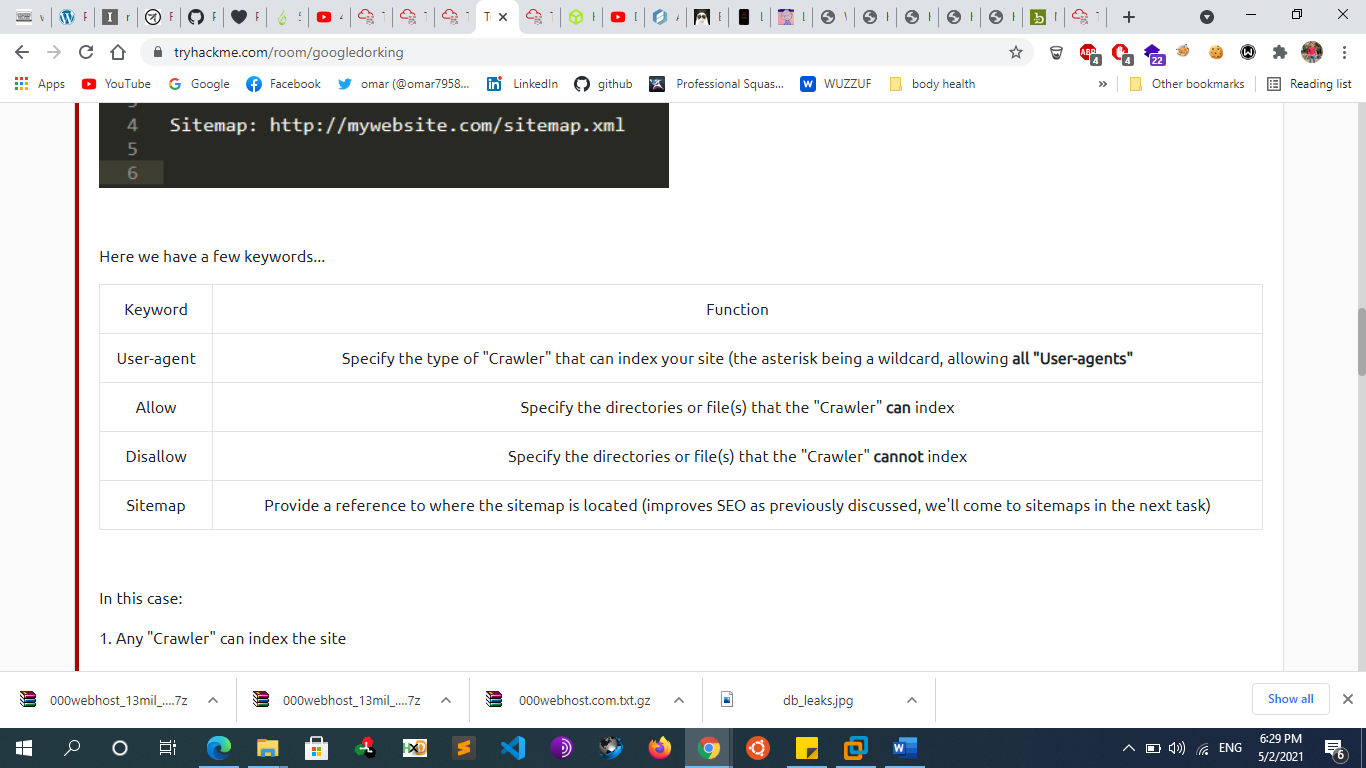
The robots.txt file is used to tell web spiders how to crawl a website. To avoid having confidential information indexed and searchable, webmasters often use this file to tell spiders to avoid specific pages. This is done using the keyword Disallow.

As an attacker, it's always a good idea to check the content of the robots.txt file and visit all the pages that are "disallowed" to find sensitive information

Example:

user-agent: \* Disallow: /page.php

This means disallow indexing for this page for all user-agents

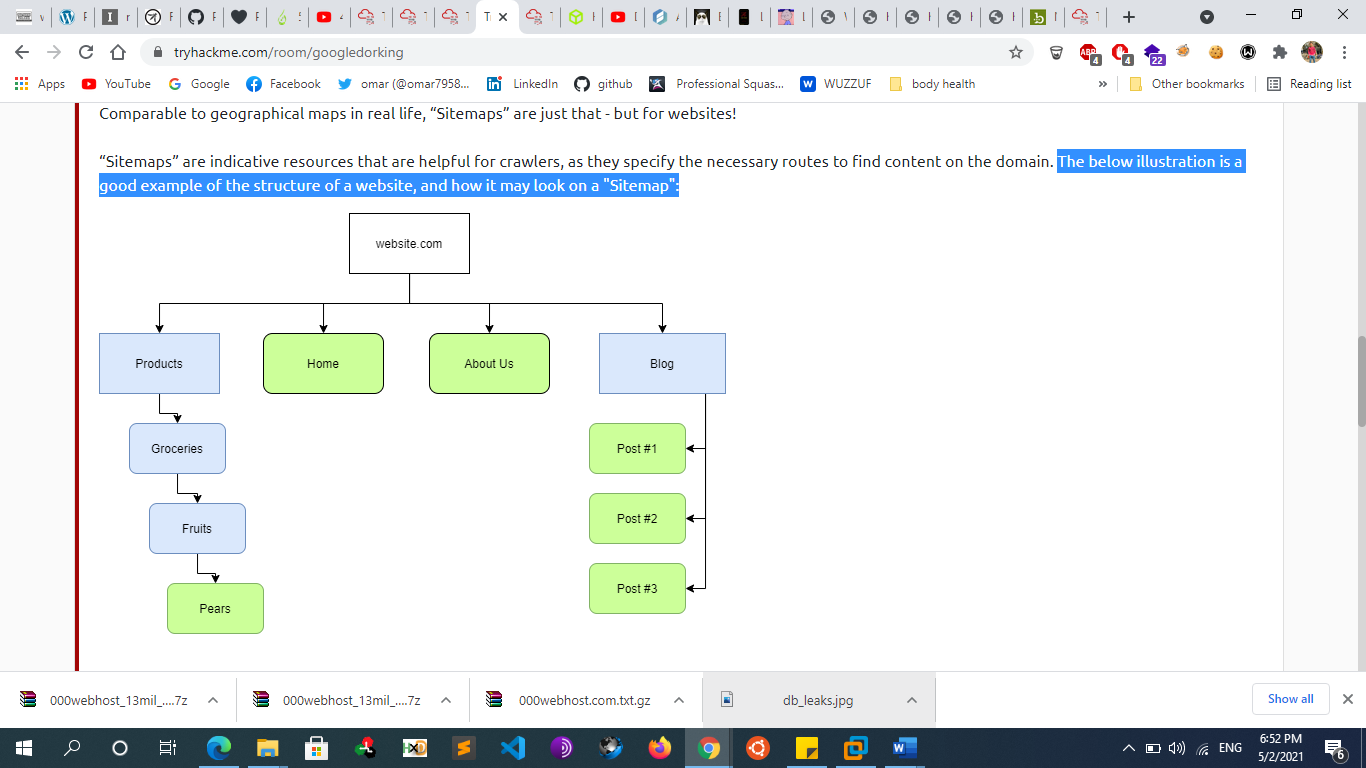


* + - * + **Sitemaps**

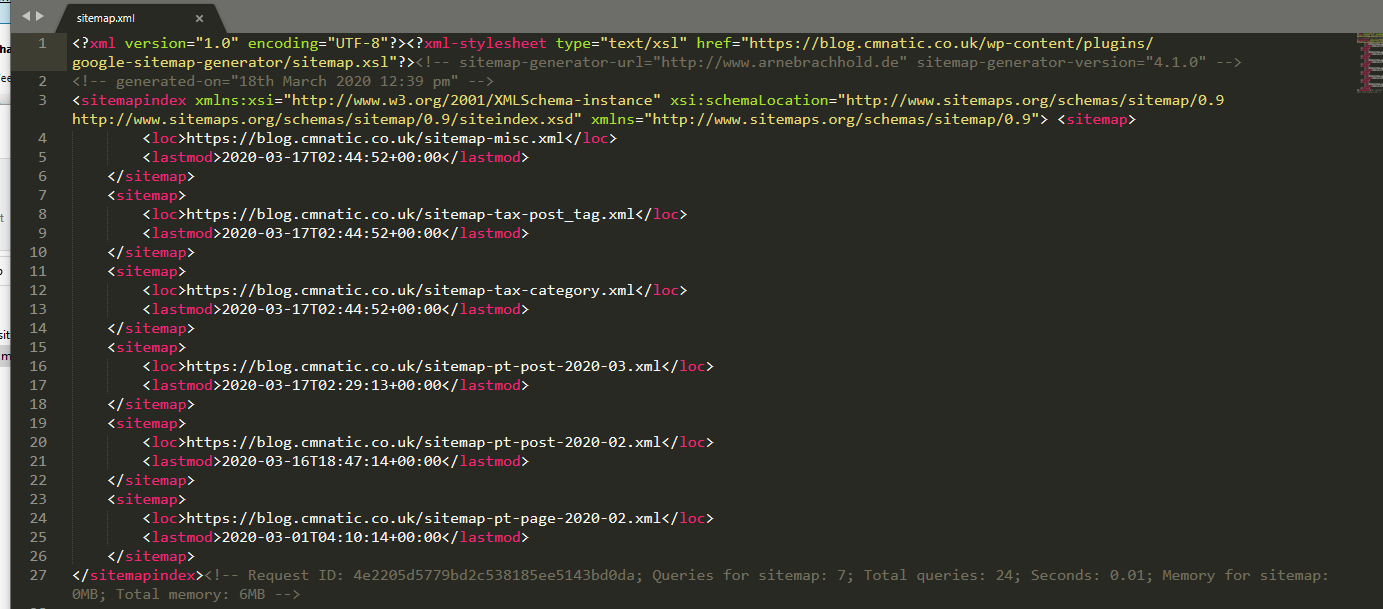
Unlike the robots.txt file, which restricts what search engine crawlers can look at, the sitemap.xml file gives a list of every file the website owner wishes to be listed on a search engine. These can sometimes contain areas of the website that are a bit more difficult to navigate to or even list some old webpages that the current site no longer uses but are still working behind the scenes.

Comparable to geographical maps in real life, “Sitemaps” are just that - but for websites!“Sitemaps” are indicative resources that are helpful for crawlers, as they specify the necessary routes to find content on the domain

The below illustration is a good example of the structure of a website, and how it may look on a "Sitemap":



“Sitemaps” don't look like this in the real world. They look something much more similar to this:



The presence of "Sitemaps" holds a fair amount of weight in influencing the "optimization" and favorability of a website. As we discussed in the "Search Engine Optimization" task, these maps make the traversal of content much easier for the crawler!  Resources like "Sitemaps" are extremely helpful for "Crawlers" as the necessary routes to content are already provided! All the crawler has to do is scrape this content - rather than going through the process of manually finding and scraping. Think of it as using a wordlist to find files instead of randomly guessing their names!

Sitemaps aren't always available d and they're always limited to what the site wants to show you d but they can be useful starting points for further investigation.

* + - * + **Others**

1- robots.txt: The robots.txt file is used to tell web spiders how to crawl a website. To avoid having confidential information indexed and searchable, webmasters often use this file to tell spiders to avoid specific pages. This is done using the keyword Disallow

2- 404 pages: Not Found/404 pages can leak information about the web stack used by a company or application. It also allows you to detect files that exists when you start bruteforcing directory. This is why it is important to check what the 404 page looks like.

3- security.txt: is a proposed standard for websites' security information that is meant to allow security researchers to easily report security vulnerabilities.The standard prescribes a text file called "security.txt" in the well known location ".well-known/security.txt", similar in syntax to robots.txt but intended to be read by humans wishing to contact a website's owner about security issues. security.txt files have been adopted by Google, GitHub, LinkedIn, and Facebook.

* + - * + **Locating Administration Consoles**

Web servers often ship with remote administration web applications, or consoles, which are accessible via a particular URL and often listening on a specific TCP port. Two common examples are the manager application for Tomcat and phpMyAdmin for MySQL hosted at /manager/html and /phpmyadmin respectively.

* + - * **Automatic Crawling:**
        + **Proxy Scanners**

burp spider or zap, webscarab

Crawling in Burp

Add to Scope the URL that You want

(can be one domain or multiple domains)

Right click on the target in sitemap

Click on Scan > Choose Crawl

Click on Scan Configuration on the Left > Choose on library > and choose the best configuration that suit you

“Crawl Strategy - Faster” is good

“Never Stop Crawl due to application errors”

Then Start

Extra Step

Before starting you can go to Resource Pool > give it a name and make it 50 for maximum concurrent requests

* + - * + **Metasploit**

use auxiliary/crawler/msfcrawler

msf auxiliary(msfcrawler) > set rhosts www.example.com

msf auxiliary(msfcrawler) > exploit

* + - **Directory Brute forcing**
      * **Wfuzz**
        + wfuzz -w ~/Code/SecLists/Discovery/Web-Content/SVNDigger/all.txt --hc 404 http://webscantest.com/FUZZ
        + python wfuzz.py -c -z file,wordlist/general/big.txt --hc 404 http://vulnerable/FUZZ

-c to output with colors.

-z file,wordlist/general/big.txt tells wfuzz to use the file wordlists/general/big.txt as a dictionary to brute force the remote directories' name.

--hc 404 tells wfuzz to ignore the response if the response code is 404 (Page not Found)

http://vulnerable/FUZZ tells wfuzz to replace the word FUZZ in the URL by each value found in the dictionary.

* + - * + wfuzz -c -w /usr/share/wfuzz/wordlist/general/megabeast.txt $ip:60080/?FUZZ=test
        + wfuzz -c --hw 114 -w /usr/share/wfuzz/wordlist/general/megabeast.txt $ip:60080/?page=FUZZ
        + wfuzz -c -w /usr/share/wfuzz/wordlist/general/common.txt "$ip:60080/?page=mailer&mail=FUZZ"
      * **ffuf**
        + apt install golang
        + git clone <https://github.com/ffuf/ffuf.git>
        + ffuf -w /usr/share/wordlists/dirbuster/directory-list.txt -u https//:target/FUZZ ->by using FUZZ keyword
      * **Dirsearch**
        + git clone <https://github.com/maurosoria/dirsearch.git>
        + ./dirsearch.py -u <http://testphp.vulnweb.com/>
        + **./dirsearch.py -u** [**[website]**](http://hostname.com) **-e php,html –x 400,401,403,404**
        + ./dirsearch.py -u [[website]](http://hostname.com) -e {aspx,php,…}
        + ./dirsearch.py -u [[website]](http://hostname.com) -e php –r

-> run recursively

* + - * + ./dirsearch.py -u [[website]](http://hostname.com) -e php,html –x 403,404 –r

-> exclude certain http response code

* + - * + ./dirsearch.py -u http://testphp.vulnweb.com/ -s 10

-s : delay between every request

* + - * + Output format

./dirsearch.py -u http://testphp.vulnweb.com/ --simple-report=report

./dirsearch.py -u http://testphp.vulnweb.com/ --plain-text-report=report

./dirsearch.py -u http://testphp.vulnweb.com/ --json-report=report

./dirsearch.py -u http://testphp.vulnweb.com/ --xml-report=report

./dirsearch.py -u http://testphp.vulnweb.com/ --markdown-report=report

./dirsearch.py -u http://testphp.vulnweb.com/ --csv-report=report

* + - * + ./dirsearch.py -u http://testphp.vulnweb.com/ -q

-q : remove unnecessarily banners

* + - * + ./dirsearch.py -u http://testphp.vulnweb.com/ -m HEAD

Change request method

* + - * **Dirbuster**
        + DirBuster is a file/directory brute-forcer. It's written in Java and programmed by the members of the OWASP community. It's a GUI application and comes with Kali Linux. DirBuster supports multithreading and is capable of brute-forcing targets at insane speeds.
        + https://www.hackingarticles.in/comprehensive-guide-on-dirbuster-tool/
      * **Gobuster**
        + Download

apt-get install gobuster

Dirbuster has a "dir" mode that allows the user to enumerate website directories. This is useful when you are performing a penetration test and would like to see what the directory structure of a website is

* + - * + Usage

gobuster dir -u [ip] -w /usr/share/wordlists/dirb/common.txt

**-o option: save result in a file**

**-e option** provides a more significant result, as it Prints complete URL when extracting any file or directories

**-r Option: follow redirection**

**-n Option** “no status” mode, it prints the output of the results without displaying the status code.

**-l option** enables content-length parameter which displays the size of the response. A Content-Length header is a number denoting and the exact byte length of the HTTP body for extracted file or directory.

**-q** **Option**: quiet mode which disable banner

**-c ; --cookies option:** Cookies to use for requests

The -k flag is special because it has an important use during penetration tests and captures the flag events.if HTTPS is enabled, you will most likely encounter an invalid cert error In instances like this, if you try to run Gobuster against this without the-k flag, it won't return anything and will most likely error out with something gross and will leave you sad. Don't worry though, easy fix! Just add the -k flag to your scan and it will bypass this invalid certification and continue scanning and deliver the goods!

Gobuster dir -u [ip] -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x .php,.html

-x: searches the given extension files

gobuster dir -u [ip] -w /usr/share/wordlists/dirb/common.txt -a Mozilla/5.0 -fw

-a : specify a user agent

gobuster dir -u [ip] -w /usr/share/wordlists/dirb/common.txt -s 200,301

-s: show responses from these status codes

gobuster dir -u [ip] dvwa -w /usr/share/wordlists/dirb/common.txt -f

appending a forward slash

gobuster dir -u [ip] -w /usr/share/wordlists/dirb/common.txt -U test -P test

HTTP AUTHORIZATION Basic (-u username: password)

-P ; --password: Password for Basic Auth

gobuster dir -u http://192.168.1.108/ -w /usr/share/wordlists/dirb/common.txt –p 192.168.1.108:3129

use proxy

* + - * **Dirb**
        + Dirb[http://example.com](http://INSERTIPADDRESS)
        + dirb [http://example.com](http://INSERTIPADDRESS) -r -o output.txt

-r : not recursive

The dirb scan, by default, scans the directories recursively. It means it scans a directory and then traverses inside that directory to scan for more subdirectories.

* + - * + dirb http://192.168.1.106/dvwa/ -X .php

searches the given extension files

* + - * + dirb http://192.168.1.106/dvwa/ -N 302

Ignore Unnecessary Status-Code

* + - * + dirb http://192.168.1.106/dvwa -z  100

there is some environment that cannot handle the flood created by the dirb scan, so it is important that we delay the scan for some time. -z : the time is provided on the scale of milliseconds

* + - * + dirb http://testphp.vulnweb.com/login.php -u  test:test

HTTP AUTHORIZATION Basic (-u username: password)

* + - * + dirb http://192.168.1.106/dvwa -H .php

**–H parameter** with specific extension, for example .php along with target URL it will enumerate all files or directories named with php

* + - * + dirb http://192.168.1.105/bwapp/portal.php -t

in order to run the dirb tool, we will have to add a forward slash (/) at the end of the URL to be accepted in dirb. In order to check that we need to try one attack on URL ending without any forward slash.

dirb http://192.168.1.105/bwapp/portal.php

You will observe that the scan doesn’t get executed successfully because of the lack of the forward slash, Try this attack once again with the same command with some changes so in order to run that command we have to add –t in the previous command.

* + - * + dirb http://192.168.1.108 –p 192.168.1.108:3129

-p : proxy

* + - * **Metasploit**
        + use auxiliary/scanner/http/brute\_dirs
        + use auxiliary/scanner/http/dir\_scanner
      * **Notes**
        + collect and filter all the results to show only 2xx, 3xx, 403 response codes from the tool itself (use -h to know how to filter the results) collect all these information into text file hidden\_directories.txt and try to discover the leakage data or the forbidden pages and try to bypass them
        + If you find a .git file some information can be extracted
        + If you find a .env information such as api keys, dbs passwords and other information can be found.
        + **403 Forbidden/Basic Authentication/401 Unauthorized (bypass)**

Manually

1. GET site.com => redirecting to site.com/login

2. GET site.com/login => 403

3. GET site.com //login => 200

using httpx:

cat hosts.txt | httpx -path /login -p 80,443,8080,8443 -mc 401,403 -silent -t 300 | unfurl format %s://%d | httpx -path //login -mc 200 -t 300 -nc -silent

* + - * + **Wordlists**

The quality of the wordlists you're using to brute-force-discover hidden content is important. There are many wordlists you can use here:

Jhaddix Content\_discovery\_all.txt

http://gist.github.com/jhaddix/b80ea67d85c13206125806f0828f4d10

The one(s) provided with the Tools

/usr/share/wordlists/dirbuster/directory-list-2.3-\*.txt

/usr/share/wordlists/dirbuster/directory-list-1.0.txt

/usr/share/wordlists/dirb/big.txt

/usr/share/wordlists/dirb/common.txt

/usr/share/wordlists/dirb/small.txt

/usr/share/wordlists/dirb/extensions\_common.txt - Useful for when fuzzing for files!

Seclist : Which contains several wordlists with several purposes (DNS, passwords, usernames, payloads, SQLi injection scripts, XSS snippets, and other generally malicious input,...)

<https://github.com/danielmiessler/SecLists/tree/master/Discovery/Web-Content>

raft-large-directories-lowercase.txt

directory-list-2.3-medium.txt

RobotsDisallowed/top10000.txt

Assetnote wordlists

https://github.com/carlospolop/Auto\_Wordlists/blob/main/wordlists/bf\_directories.txt

https://github.com/random-robbie/bruteforce-lists

https://github.com/google/fuzzing/tree/master/dictionaries

https://github.com/six2dez/OneListForAll

https://github.com/random-robbie/bruteforce-lists

* + - **Parameter discovery (discover new/hidden parameters)**
      * You can use tools like Arjun, parameth, x8 and Param Miner to discover hidden parameters. If you can, you could try to search hidden parameters on each executable web file.
        + **Arjun**

git clone https://github.com/s0md3v/Arjun

sudo python3 setup.py install

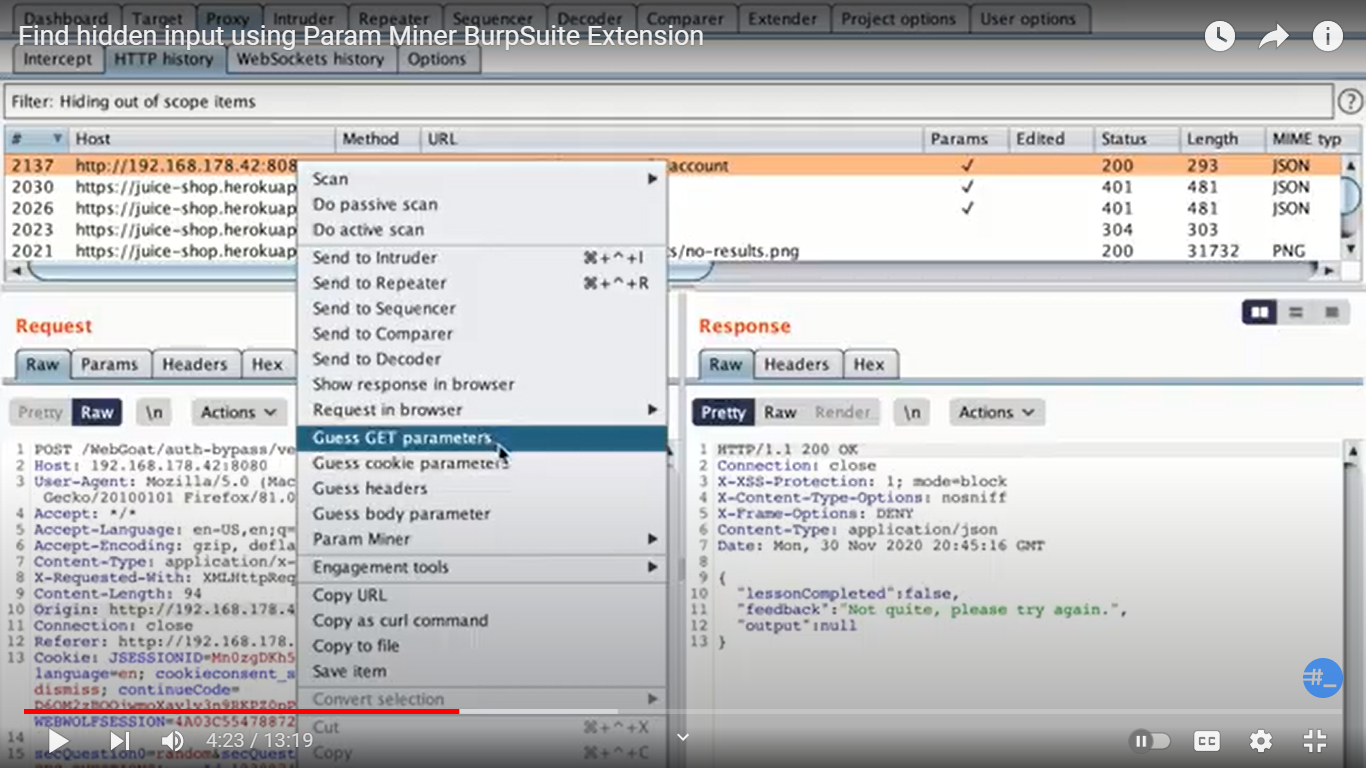
arjun -u [URL] -m Get

* + - * + **Param-miner burp suite Extension**

Step 1:

Choose a request and right click -> Guess Get Params

You can also Guess Cookie parameters and headers etc..



Step 2:

Configure the attack by adding the right wordlist & thread



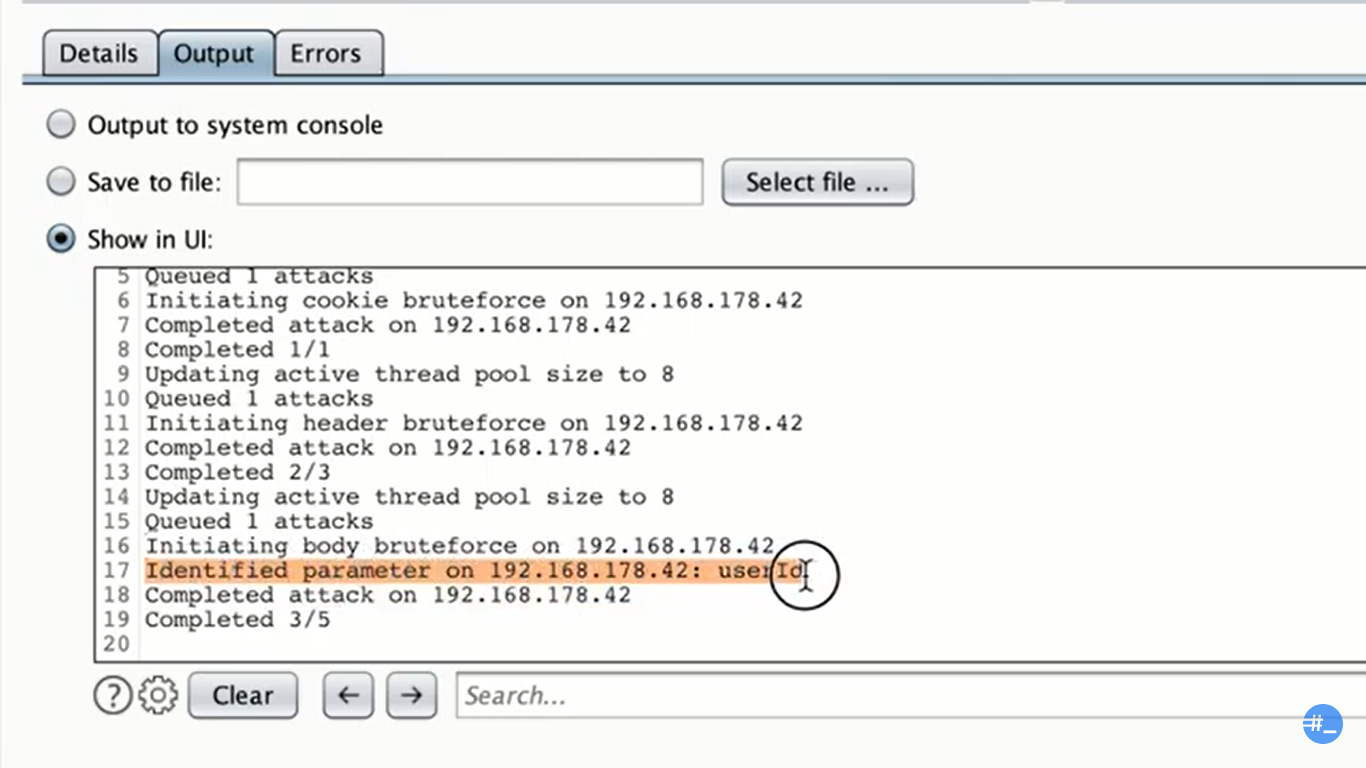
Step 3:

Press ok

You can watch the background process using Logger++ extension

Step 4:

Check result in the extension output. Got to Extender Tab -> Extension subtab -> click on the param miner extension -> output



* + - * + **Paramspider**

Python3 paramspider.py --domain domain.com --exclude svg,js,css,jpg --output domainParam.txt

* + - * + **Parameth**

[https://github.com/maK-/parameth]

* + - * + **Waybackurl**

Cat subdomains.txt | waybackurls | grep “=” | tee -a domain\_params.txt

* + - * **Wordlists**
        + Arjun all default wordlists:

https://github.com/s0md3v/Arjun/tree/master/arjun/db\_\_

* + - * + Param-miner “params”:

https://github.com/PortSwigger/param-miner/blob/master/resources/params\_\_

* + - * + Assetnote “parameters\_top\_1m”:

https://wordlists.assetnote.io/\_\_

* + - * + nullenc0de “params.txt”:

https://gist.github.com/nullenc0de/9cb36260207924f8e1787279a05eb773

* + - **JavaScript Analysis**
      * **Overview**
        + Another tip is always check the JavaScript code. It could be possible that you'll find new endpoints, hardcoded credentials, hardcoded JWT signing key, ... that may not be visible to the users Especially when they are making use of a CMS based on JavaScript.
        + 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.

* + - * **Analyzing JavaScript**
        + **Tools to make your life easier:**

LinkFinder

LinkFinder is a python script written to discover endpoints and their parameters in JavaScript files. This way penetration testers and bug hunters are able to gather new, hidden endpoints on the websites they are testing.

Installation

$ git clone https://github.com/GerbenJavado/LinkFinder.git

$ cd LinkFinder

$ python setup.py install

$ pip3 install -r requirements.txt

JSParser

A python 2.7 script using Tornado and JSBeautifier to parse relative URLs from JavaScript files. Useful for easily discovering AJAX requests when performing security research or bug bounty hunting.

$ python setup.py install

$ python handler.py

relative-url-extractor

* + - * + **Manually get all the used JavaScript files using BurpSuite:**

Method 1

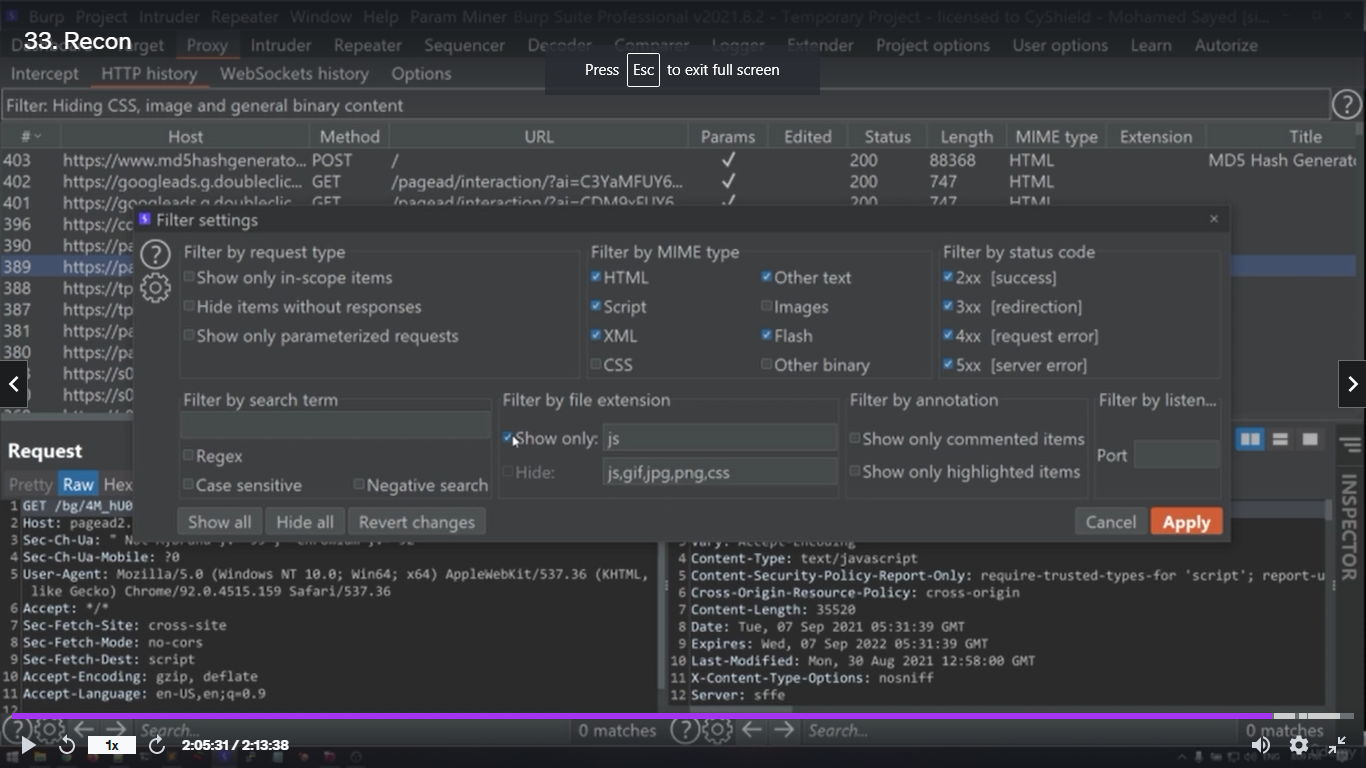
In BurpSuite, Go to Target tab.

Right click on the subdomain (where you want to analyze the JS from)

Click on Engagements tools

Select Find scripts

Method 2



CTRL + A to select all the scripts Burp has found.

Right click and select "Copy Selected URLS"

Paste them into a file and run a command like uniq. This to remove the duplicates.

Cat js.txt | sort -u

Paste all the urls in LinkFinder/JSParser.

* + - * + Via these tools, you can easily find new endpoints
      * **Parsing Deobfuscated JS Notes**
        + JavaScript Deobfuscator and Unpacker:

<https://lelinhtinh.github.io/de4js/>

https://matthewfl.com/unPacker.html

https://www.dcode.fr/javascript-unobfuscator

* + - * + JsFuck obfuscator / deobfuscator

(javascript with chars:"[]!+" )

<https://ooze.ninja/javascript/poisonjs/>

https://www.cleancss.com/javascript-obfuscate/index.php

http://www.jsfuck.com/

* + - * + Javascript Beautifier:

<http://jsbeautifier.org/>

http://jsnice.org/

* + - * + TrainFuck:

+72.+29.+7..+3.-67.-12.+55.+24.+3.-6.-8.-67.-23.

* + - * + In several occasions you will need to understand regular expressions used, this will be useful:

https://regex101.com/

* + - * **Notes**
        + You should also check discovered JS files with RetireJS or JSHole to find if it's vulnerable.
        + use grep to collect all js files form waybackurls.txt as cat waybackurls.txt | grep js > js\_files.txt

save all the results to js\_files.txt

* + - * + Find hidden GET parameters in javascript files

assetfinder example.com | gau | egrep -v '(.css|.png|.jpeg|.jpg|.svg|.gif|.wolf)' | while read url; do vars=$(curl -s $url | grep -Eo "var [a-zA-Z0-9]+" | sed -e 's,'var','"$url"?',g' -e 's/ //g' | grep -v '.js' | sed 's/.\*/&=xss/g'); echo -e "\e[1;33m$url\n\e[1;32m$vars"; done

* + - * + you can analyze these files manually or use automation tools (I recommend manual scan, see references)
    - **Virtual hosts scanning (Discovery)**
      * **FFUF**
        + ffuf -w /usr/share/wordlists/wfuzz/general/common.txt -u https://hackycorp.com/ -H "Host: FUZZ.hackycorp.com" -fs 4242

**-fs** switch, which tells ffuf to ignore any results that are of the specified size.

* + - * **Gobuster**
        + "vhost" Mode

the "vhost" mode. This allows Gobuster to brute-force virtual hosts. Virtual hosts are different websites on the same machine. In some instances, they can appear to look like sub-domains, but don't be deceived! Virtual Hosts are IP based and are running on the same server. This is not usually apparent to the end-user. On an engagement, it may be worthwhile to just run Gobuster in this mode to see if it comes up with anything. virtual hosts would be a good way to hide a completely different website if nothing turned up on your main port 80/443 scan.

* + - * + **Example**

gobuster vhost -u http://example.com -w /usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt

-w /usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt

This will tell Gobuster to do a virtual host scan http://example.com using the selected wordlist

vhostbrute.py --url="example.com" --remoteip="10.1.1.15" --base="www.example.com" --vhosts="vhosts\_full.list"

From https://github.com/allyshka/vhostbrute

VHostScan -t example.com

#https://github.com/codingo/VHostScan

* + - **Large Scope Reconnaissance (Asset Discovery)**
      * **Overview**
        + The First Thing in Recon Methodology is to identify our root Domains (Seeds) and this can be done by multiple Ways

Checking the Scope of the program at the Bug Bounty Platform

Acquisitions

Autonomous system number (ASN) Recon

Other Methods

Reverse Whois (Linked Discovery)

Use reverse whois lookups to search for other entries (organisation names, domains...) related to the first one (this can be done recursively)

Shodan

Fav icon Search

* + - * **Company acquisitions**
        + **We can investigate company’s acquisition on sites like**

Crunchbase

Wikipedia

Google

* + - * + **Crunchbase**

CrunchBase is a database where you can find information about organizations:

Companies

People

Investors and financial information

Companies Acquisitions

The power of CrunchBase is grounded on the concept of anyone being able to edit information in it.

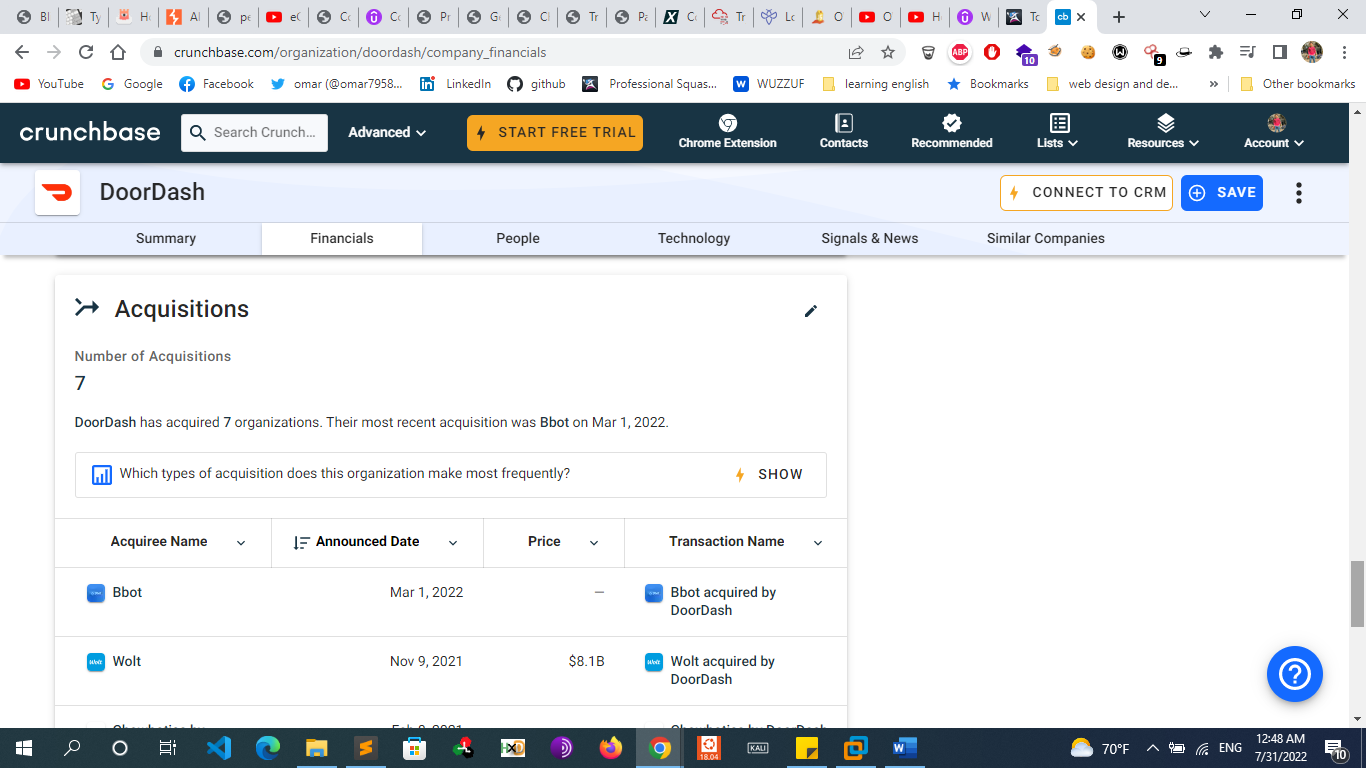
We will use Crunchbase to find Company Acquisitions.

Enter the company or a person into the search bar at the top of the page

Go to the acquisitions tab



look at the acquisitions. In this example we see that DoorDash has 7 Acquisitions



* + - * + **Wikipedia**

Other option is to visit google and the Wikipedia page of the main company and search for acquisitions.

* + - * **Autonomous system number (ASN) Recon**
        + **What is ASN**

An Autonomous System (AS) is a set of Internet routable IP prefixes belonging to a network or a collection of networks that are all managed, controlled and supervised by a single entity or organization. An AS utilizes a common routing policy controlled by the entity. The AS is assigned a globally unique 16 digit identification number known as the autonomous system number or ASN by the Internet Assigned Numbers Authority (IANA).

Autonomous System Number (ASN) is a globally unique identifier that defines a group of one or more IP prefixes run by one or more network operators that maintain a single, clearly-defined routing policy. These groups of IP prefixes are known as autonomous systems.

Autonomous systems numbered one to 64511 are available by IANA for global use. The 64512 to 65535 series is reserved for private and reserved purposes. Autonomous Systems were introduced to regulate networking organizations such as Internet Service Providers (ISP), educational institutions and government agencies.

Border Gateway Protocol (BGP) is the protocol that manages the routed peerings, prefix advertisement and routing of packets between different autonomous systems across the Internet. BGP uses the ASN to uniquely identify each system. In effect, BGP is the routing protocol for AS paths across the Internet.

* + - * + **Getting ASN numbers and IP ranges**

Using Tools

ASNLookup

<https://github.com/yassineaboukir/Asnlookup>

Which utilize maxmind.com dataset

Metabigor

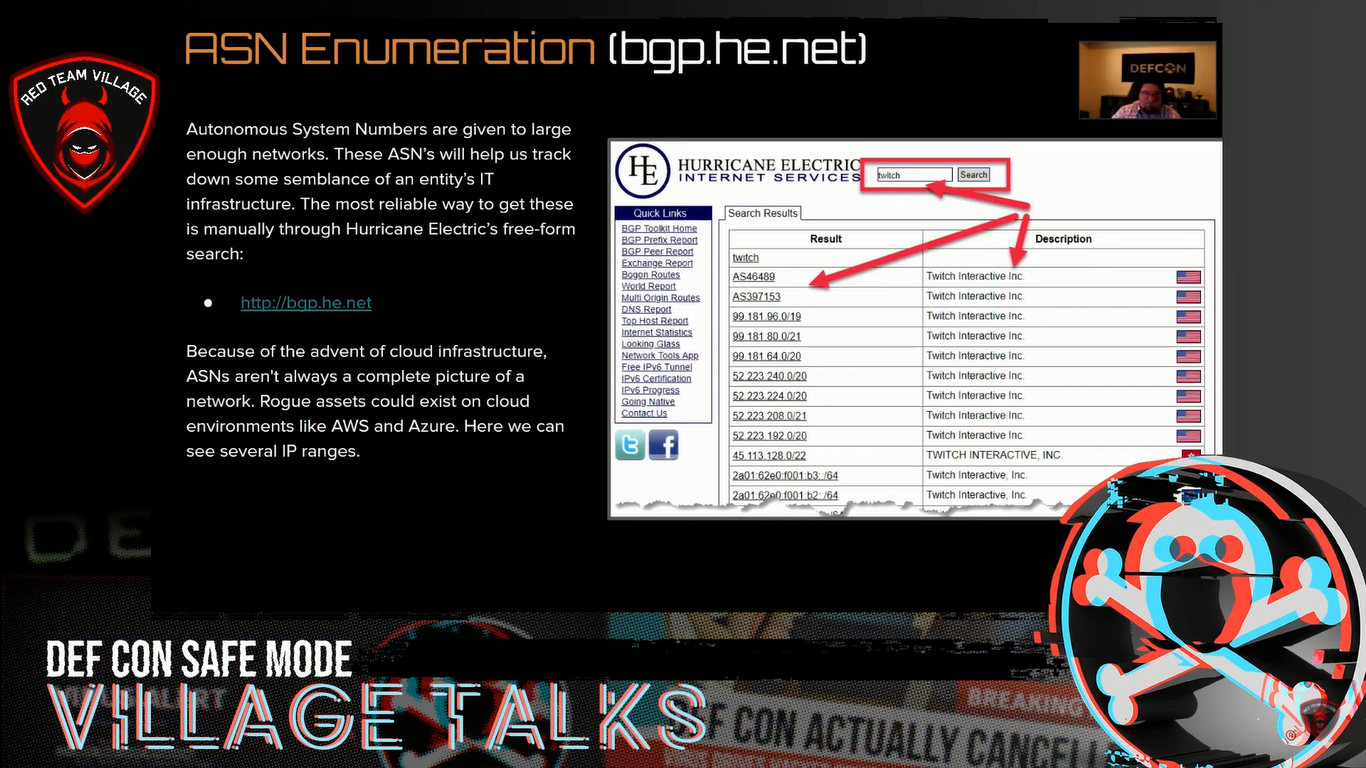
<https://github.com/j3ssie/metabigor>

Which will fetch ASN data From ASNLOOKUP.com and <https://bgp.he.net/>

Online

The most reliable way to get these ASN is manually through

<https://bgp.he.net/>



Here we got 2 ASN numbers if we clicked one of them, we will get the IP ranges of each ASN.

So after getting the Acquisitions of the company we will try to grab the ASN/IP ranges of Each one. So now we have ASN numbers and IP ranges of the Company and this is an example how can we organize what we found so far



* + - * + **Getting Root/Seed Domains From ASN**

For discovering more seed domains we want to scan the whole ASN with a port scanner on return any root domains we see in SSL certficates, etc.

We can do this with Amass intel Amass is written by Jeff Foley and the amass team

Amass intel module collects open-source intelligence for the target organization. It allows you to find root domain names associated with it.

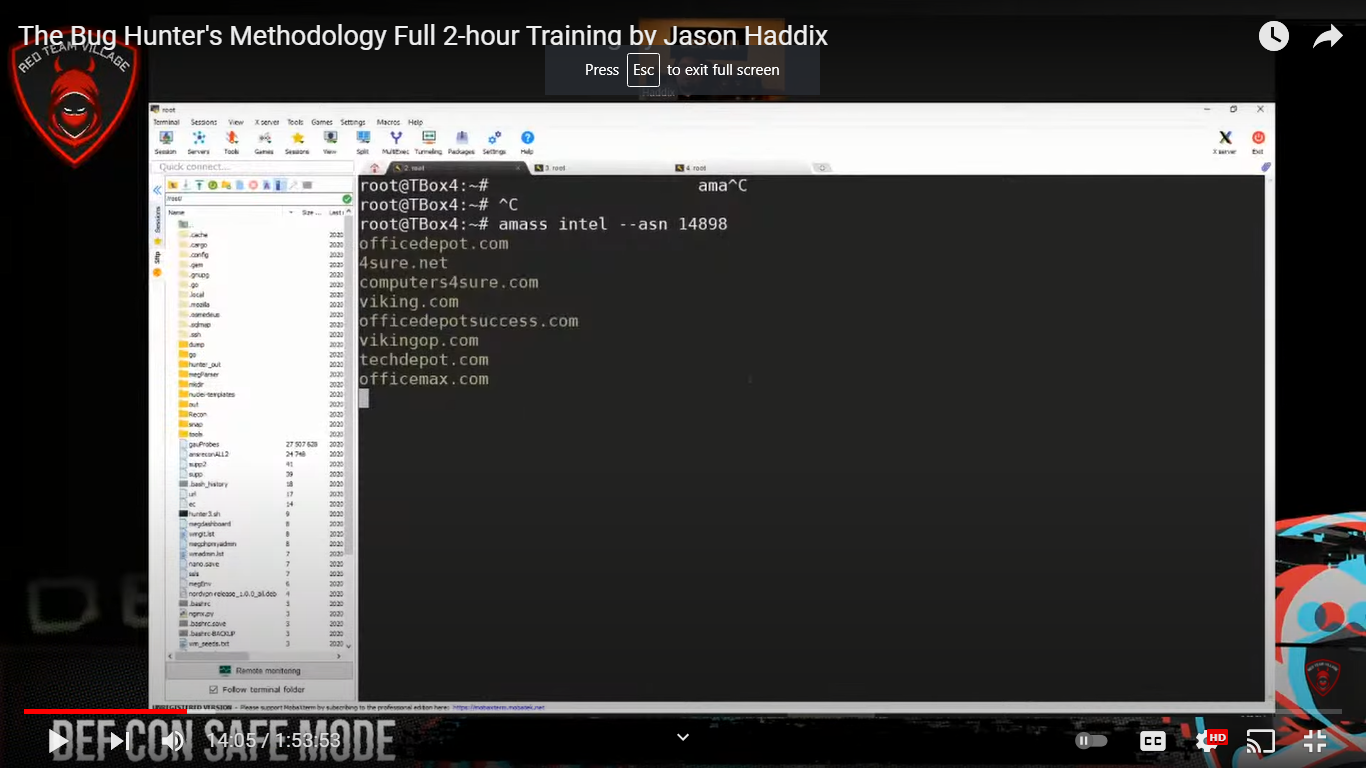
amass intel -org <org-name>

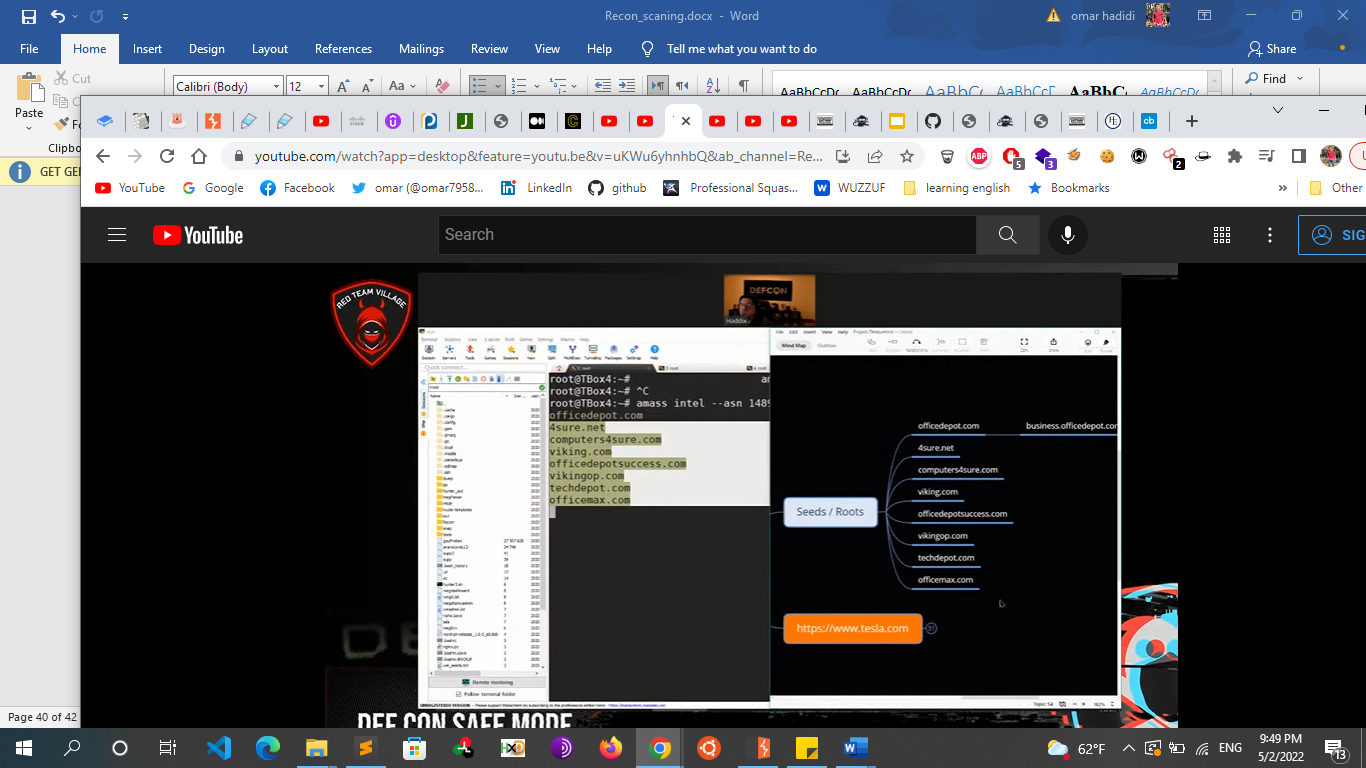
amass intel -asn <asn>

Amass Intel -asn 45646

amass intel -asn 8911,50313,394161

Example





So now We have Gathered ASN numbers , IP ranges and Root Domains

* + - * + **Notes**

Note: Because of the advent of cloud infrastructure, ASNs may not provide a complete picture of a network. Assets could also exist on cloud environments like AWS, GCP, and Azure.

Bash Script

amass intel -org yahoo -max-dns-queries 2500 | awk -F, '{print $1}' ORS=',' | sed 's/,$//' | xargs -P3 -I@ -d ','

amass intel -asn @ -max-dns-queries 2500

This is what the command does in detail:

Get a list of ASNs (Autonomous System Numbers) of the target organization (e.g. yahoo)

Extract only a list of AS numbers and separate them by comma

For each identified ASN, find list of domain names that are associated with the ASN

* + - * **Other Methods**
        + **Reverse WHOIS (Linked Discovery)**

Inside a **whois** you can find a lot of interesting **information** like **organisation name**, **address**, **emails**, phone numbers... But which is even more interesting is that you can find **more assets related to the company** if you perform **reverse whois lookups by any of those fields** (for example other whois registries where the same email appears). Reverse WHOIS leverage these registries and allow us to perform lookups based on that additional information You can use online tools like:

Whoxy Whois

https://www.whoxy.com/

Reverse Whois Lookup

https://www.reversewhois.io/

DomainEye Reverse Whois

https://domaineye.com/reverse-whois

domainIQ - Comprehensive domain name intelligence.

https://www.domainiq.com/

**Automated Tools**

DomLink

you can automate this task using DomLink (requires a whoxy API key).

python domlink.py -A <whoxy-api-key> -C <org-name> -o target.out.txt

python domlink.py -d <domain.com > -o target.out.txt

Whoxy API

curl --no-progress-meter "https://api.whoxy.com/?key=<whoxy-api-key>&reverse=whois&name=<org-name>" | jq

* + - * + **Trackers (Ads/Analytics)**

If find the same ID of the same tracker in 2 different pages you can suppose that both pages are managed by the same team.

For example, if you see the same Google Analytics ID or the same Adsense ID on several pages.

There are some pages that let you search by these trackers and more:

BuiltWith

Sitesleuth

Publicwww

SpyOnWeb

* + - * + **Shodan**

Shodan is a search engine for Internet-connected devices. Try filter by organization: org:<org-name> or hostname: hostname: <domain>.

Use other techniques like shodan “org” and “ssl” filters to search for other assets (the ssl trick can be done recursively).

ssl:”trade name”

ssl:”Facebook Inc.”

* + - * + **Favicon hashes**

Did you know that we can find related domains and sub domains to our target by looking for the same favicon icon hash

A more fringe technique of discovering a brands assets is taking their favicon and hashing it. You can then take this hash and search shodan for it. You can also scan IP ranges and cloud blocks to find assets with the same hash. In addition you can make hashes of commonly used admin portals or framework logins. This method is useful when an org has modified URL paths and maybe your scanners are only looking for a specific path. If the org has changed it all you won't find it, but org's rarely change the favicon for the frameworks.

**Tools**

FavFreak by Devansh Batham (OxAsmOd3us)

cat urts. txt | python3 favfreak.py -o output

search org: "Target"

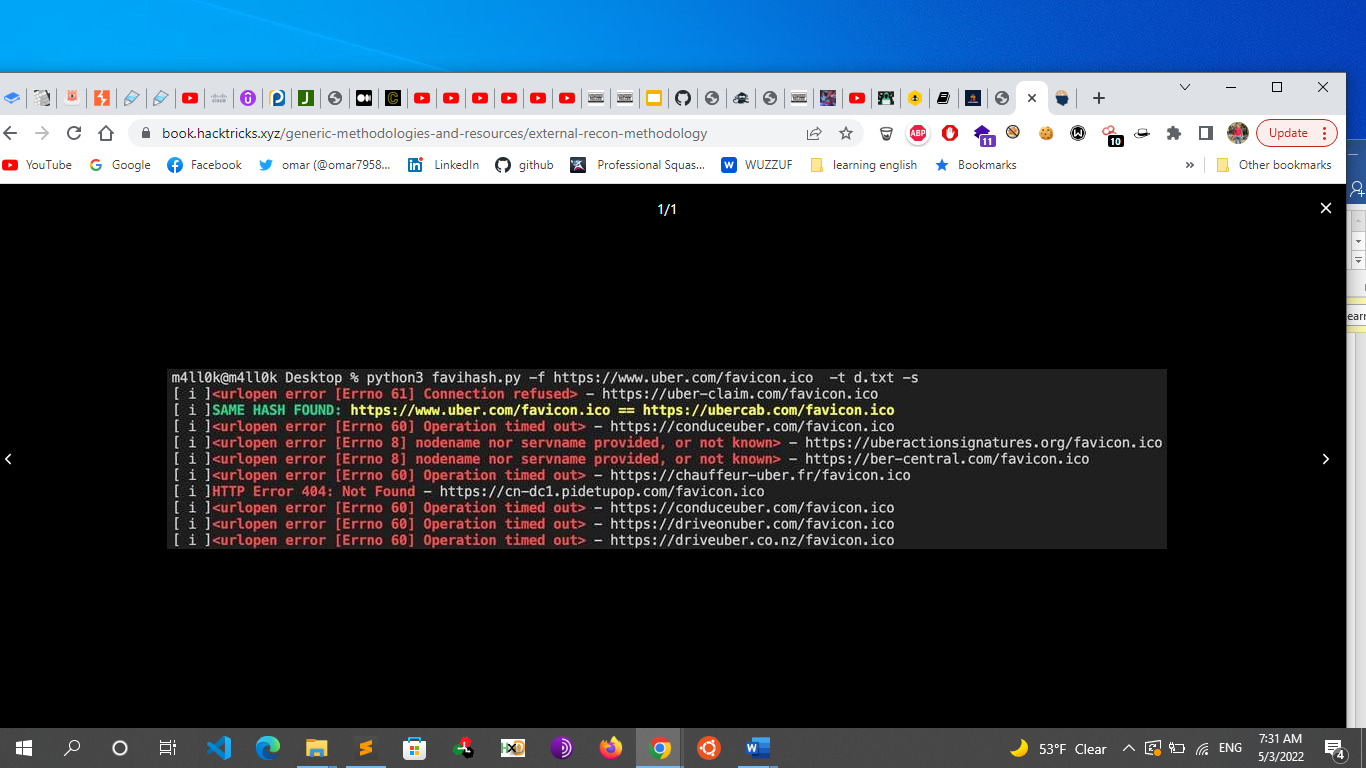
avicon.hash:116323821

port

Favihash.py

cat my\_targets.txt | xargs -I %% bash -c 'echo "http://%%/favicon.ico"' > targets.txt

python3 favihash.py -f https://target/favicon.ico -t targets.txt -s



* + - * **Notes**
        + At this point we known all the assets inside the scope, so if you are allowed you could launch some vulnerability scanner (Nessus, OpenVAS) over all the hosts.
        + Also, you could launch some port scans or use services like shodan to find open ports and depending on what you find you should take a look in this book to how to pentest several possible services running.
        + Also, It could be worth it to mention that you can also prepare some default username and passwords lists and try to bruteforce services with

https://github.com/x90skysn3k/brutespray.

* + - **Others**
      * **Detecting WAF Used**
        + To check which WAF is used on a certain subdomain, I make use of wafwoof.
        + **Wafwoof**

Wafwoof –L -> list all waf supported by wafwoof

Wafwoof -a [website]

* + - * + **Nmap**

nmap -p 80,443 --script=http-waf-detect [website]

* + - * **GitHub Recon**
        + **Overview**

Github is also a great place to look for credentials and private API keys. The hardest thing about this method is getting creative when it comes down to looking for different keys.

GitHub Dorking uses specific search keys to find sensitive information in public repositories. This is similar to Google Dorking

while doing subdomain enumeration we can start on dorking on GitHub for some finds

* + - * + **GitHub Dorks Example**

"Company" password

"Company" secret

"Company" credentials

"Company" token

"Company" config

"Company" key

"Company" pass

"Company" login

"Company" pwd

"Company" ssh\_auth\_password

"Company" ssh\_user

"Company" ssh\_pasword

"Company" ftp

"Company" security\_credentials

"Company" connectionstring

"Company" JDBC

"Company" send\_key,send\_keys

* + - * + **You can find more GitHub dorks**

https://github.com/KathanP19/HowToHunt/blob/master/Recon/Github\_Dorking.md

<https://github.com/techgaun/github-dorks/blob/master/github-dorks.txt>

https://book.hacktricks.xyz/generic-methodologies-and-resources/external-recon-methodology/github-leaked-secrets

* + - * + **Tools**

For larger repos, we can use several tools to help automate some of the searching, such as Gitrob and Gitleaks. Recon-ng also has several modules for searching GitHub. Most of these tools require an access token to use the source code hosting provider’s API.

Gitrob

https://github.com/michenriksen/gitrob

Gitleaks

Gitdorker

<https://github.com/obheda12/GitDorker>

Github-dorks

<https://github.com/techgaun/github-dorks>

git-all-secrets

https://github.com/anshumanbh/git-all-secrets

truffleHog

https://github.com/dxa4481/truffleHog

git-secrets

https://github.com/awslabs/git-secrets

github-dorks

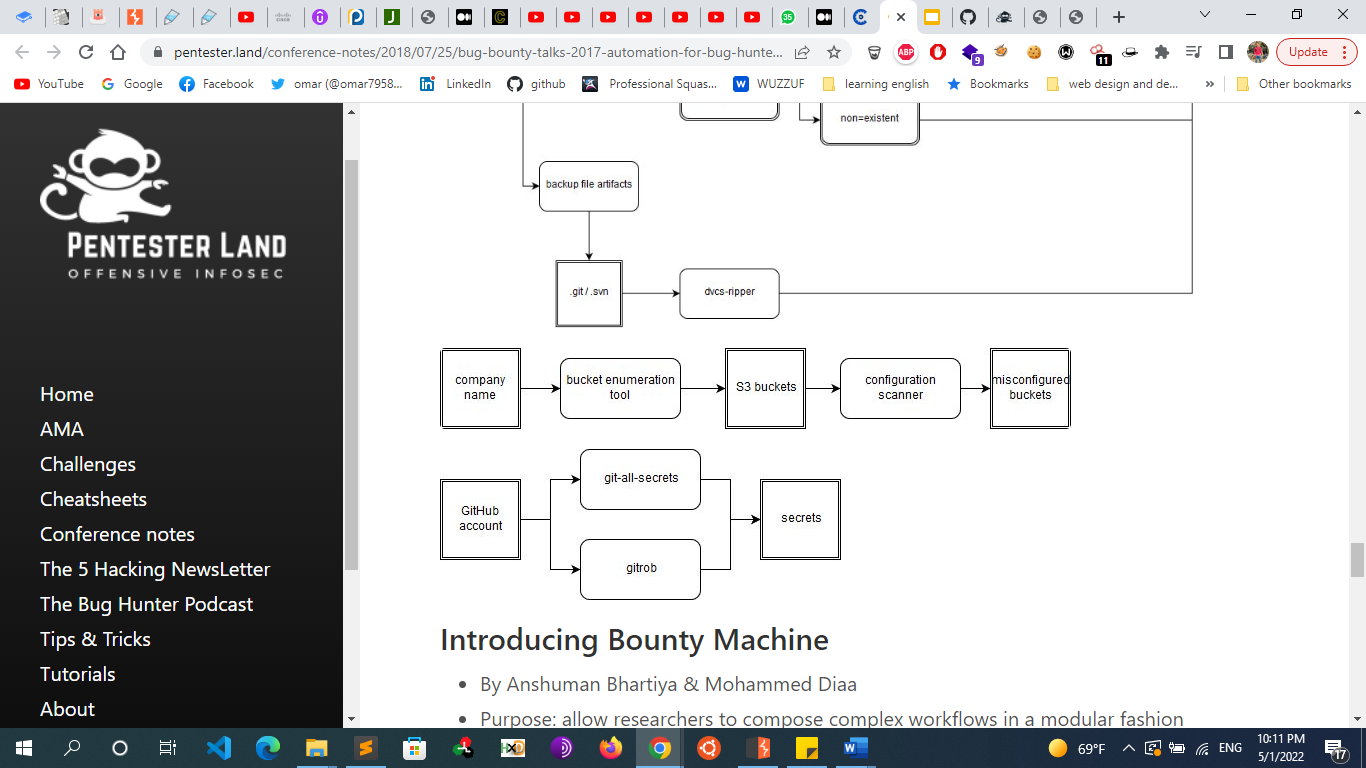
https://github.com/techgaun/github-dorks

gitGraber

https://github.com/hisxo/gitGraber

gittyleaks

https://github.com/kootenpv/gittyleaks



* + - * + **Notes**

You can also search for leaked secrets in all open repository platforms using: https://searchcode.com/?q=auth\_key

* + - * **S3 Buckets Leaks**
        + AWS has become a huge asset to thousands of different companies. Some companies just use AWS s3 buckets to host their content, while others use it to deploy and host their application. S3 Buckets are a storage service provided by Amazon AWS, allowing people to save files and even static website content in the cloud accessible over HTTP and HTTPS. The format of the S3 buckets is http(s)://**{name}.**[**s3.amazonaws.com**](http://s3.amazonaws.com/) where {name} is decided by the owner, such as [tryhackme-assets.s3.amazonaws.com](http://tryhackme-assets.s3.amazonaws.com/). S3 buckets can be discovered in many ways, such as finding the URLs in the website's page source, GitHub repositories, or even automating the process.

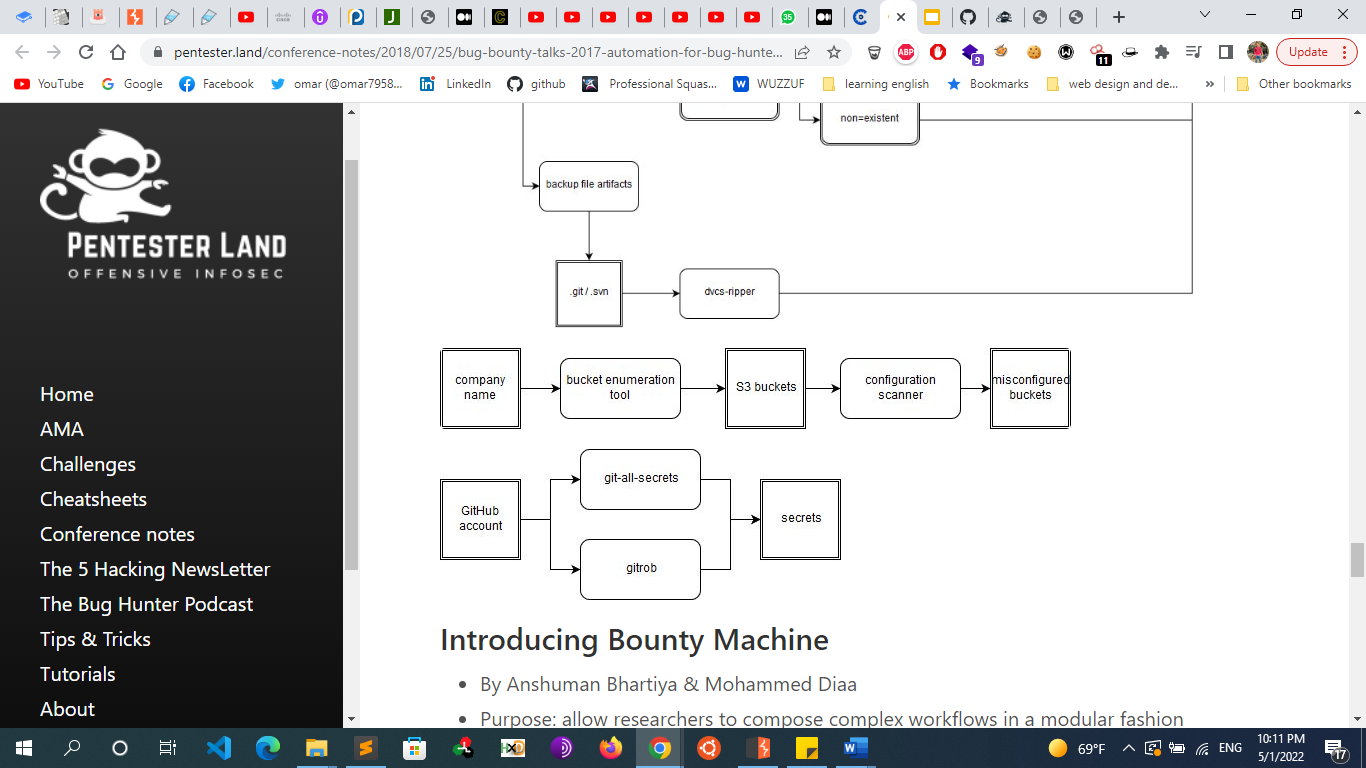
One common automation method is by using the company name followed by common terms such as **{name}**-assets, **{name}**-www, **{name}**-public, **{name}**-private, etc.

* + - * + Like any other resource, AWS could also be misconfigured or abused by criminals. For example, it's often possible to find misconfigured s3 buckets that allow a user outside of the organization to read and write files on a bucket belonging to that company.
        + The owner of the files can set access permissions to either make files public, private and even writable. Sometimes these access permissions are incorrectly set and inadvertently allow access to files that shouldn't be available to the public.
        + These properties can be discovered by combining a few different methods:

Using a google dork to find them: site:s3.amazonaws.com + hackme.tld

We can look them up on github: “hackme.tld” + “s3”

We can bruteforce AWS to find specific s3 buckets and automate this to speed it up

* + - * + 
        + **Note**

Amazon Web Services Storage Service (S3) allows file owners to set permissions on files. Historically, the rules "Any users" wasn't well explained and lead a lot of people to think only people in their Amazon account could access a file. However, this was allowing any AWS account to access the file

* + - * + **Tools**

Awscli

AWS CLI is useful for verifying or testing the permissions of the AWS S3 buckets, Creating Buckets and Read other buckets data. AWS Account needed to use CLI

apt install awscli

aws configure (paste your Access key and secret key)

aws s3 ls s3://assets.hackycorp.com/key2.txt

aws s3 cp s3://assets.hackycorp.com/key2.txt key2.txt

Lazys3

<https://github.com/nahamsec/lazys3>

LazyS3 is another tool which I use almost frequently to find the staging, sandboxed, dev and production buckets.

sandcastle [https://github.com/0xSearches/sandcastle]

slurp

S3scanner

teh\_s3\_bucketeer

* + - * + dork

site:s3.amazonaws.com inurl:site

* + - * + note

Another tip it's not because a bucket is not publicly readable that the permissions to write or delete a file into/from the bucket are correctly configured. Always test writing into the bucket.

https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/buckets

* + - * **Subdomain Takeover**
        + When you have a main list of all the subdomains, you can start looking for subdomain takeovers. You can provide your list of subdomains to a tool like [SubOver](https://github.com/Ice3man543/SubOver). I highly recommended resource is <https://github.com/EdOverflow/can-i-take-over-xyz>. In here you'll find the default messages provided by several services which can lead to a subdomain takeover.
        + **subdomain takeover tools**

aquatone-takeover

When you are using **aquatone**, you can use **aquatone-takeover**.

takeover

<https://github.com/m4ll0k/takeover>

Subover

SubOver -l subdomains.txt

Subjack

<https://github.com/haccer/subjack>

subjack -w live\_subdomains.txt -a

Nuclei

Nuclei -l live\_subdomains.txt -t subdomain-takeover/\*

* + - * + **Resources**

https://github.com/EdOverflow/can-i-take-over-xyz

* + - **Frameworks & Automated tools**
      * If could be recon is not really your thing. That's all right. Several hunters have open sourced their automation at this point and you can choose one that fits you and use it without worrying too much. I usually classify recon frameworks in rough tiers:
        + C-Tier: automation built around scripting up other tools in bash or python. Step based, no workflow.

Few techniques. Little extensibility.

* + - * + B-Tier: automation writing a few of their own modules. Some GUl or advanced workflow. Medium number of techniques. Runs point-in-time. Flat files.
        + A-Tier: (maybe) automation writing all their own modules. Has GUI. Runs iterativley. Manages data via db.
        + S-Tier: automation writing their own modules. Has GUI. Runs iterativley. Manages data via db. Scales across multiple boxes. Sends alerts to user. Uses novel techniques and iterates quickly. ML + Al.
      * There are several tools out there that will perform part of the proposed actions against a given scope.
        + C-Tier

https://github.com/AdmiralGaust/bountyRecon

https://github.com/offhourscoding/recon

https://github.com/Sambal0x/Recon-tools

https://github.com/JoshuaMart/AutoRecon

https://github.com/yourbuddy25/Hunter

https://github.com/venom26/recon/blob/master/ultimate\_recon.sh

https://gist.github.com/dwisiswant0/5f647e3d406b5e9

84e6d69d3538968cd

* + - * + B-Tier

https://github.com/capt-meelo/LazyRecon

is a tool written by naham sec that combine many recon tools in one tool

https://github.com/Screetsec/Sudomy

https://github.com/phspade/Automated-Scanner

https://github.com/devanshbatham/Gorecon

https://github.com/shmilylty/OneForAll https://github.com/LordNeoStark/tugarecon

https://github.com/SolomonSklash/chomp-scan

https://github.com/TypeError/domained

* + - * + A-Tier

https://github.com/Edu4rdSHL/findomain

https://github.com/SilverPoision/Rock-ON

https://github.com/epi052/recon-pipeline

* + - * + S-Tier

[**https://github.com/yogeshojha/rengine**](https://github.com/yogeshojha/rengine)

https://github.com/j3ssie/Osmedeus

hunterSuite.io

Jaeles

Intrigue.io

AssetNote

* + - **Resources**
      * **Videos**
        + https://m.youtube.com/watch?v=YhUiAH5SIqk

Naham sec building ubuntu box

* + - * + https://www.youtube.com/watch?v=l8iXMgk2nnY&ab\_channel=ST%C3%96K

stok: linux terminal tools for bug bounty pentest and redteams with @tomnomnom

* + - * + https://www.youtube.com/playlist?list=PLKAaMVNxvLmAkqBkzFaOxqs3L66z2n8LA

naham sec recon sunday

* + - * + https://m.youtube.com/watch?feature=youtu.be&v=uKWu6yhnhbQ

The Bug Hunter's Methodology Full 2-hour Training by Jason Haddix

* + - * + https://m.youtube.com/watch?feature=youtu.be&v=gIz\_yn0Uvb8

The Bug Hunter's Methodology by Jason Haddix 4.02

* + - * + https://www.youtube.com/watch?v=p4JgIu1mceI&ab\_channel=Nahamsec

The Bug Hunter's Methodology by Jason Haddix 4.0 (Nahamsec Channel)

* + - * + https://youtu.be/Qw1nNPiH\_Go

LevelUp 0x02 - Bug Bounty Hunter Methodology v3 2018

* + - * + https://youtu.be/VtFuAH19Qz0

How To Shot Web - Jason Haddix's talk from DEFCON23 2015

* + - * + <https://securib.ee/beelog/the-best-bug-bounty-recon-methodology/>

Multiple videos

* + - * **Blogs**
        + <https://pentester.land/cheatsheets/2019/04/15/recon-resources.html>
        + <https://infosecwriteups.com/recon-methodology-for-bug-hunting-e623120a7ca6>
        + <https://medium.com/soulsecteam/advanced-recon-automation-subdomains-case-1-9ffc4baebf70>
        + <https://systemweakness.com/simple-recon-methodology-a9ff080b40c8>
        + <https://book.hacktricks.xyz/generic-methodologies-and-resources/external-recon-methodology>
        + https://book.hacktricks.xyz/network-services-pentesting/pentesting-web
        + <https://github.com/Quikko/Recon-Methodology>
        + https://www.hackerone.com/ethical-hacker/how-recon-and-content-discovery
        + https://orwaatyat.medium.com/my-methodology-in-recon-and-find-bugs-my-methodology-in-hunting-using-phone-ccc9fe06dd2d
        + https://pentester.land/conference-notes/2018/07/25/bug-bounty-talks-2017-automation-for-bug-hunters.html
        + <https://gist.github.com/cyberheartmi9/1ac77d171d9b9dc9a5be45fa4f4c8dcb>
        + https://docs.google.com/presentation/d/1PCnjzCeklOeGMoWiE2IUzlRGOBxNp8K5hLQuvBNzrFY/edit#slide=id.g129ec7274d\_1\_381
        + <https://speakerdeck.com/e11i0t_4lders0n/the-bug-hunters-recon-methodology?slide=11>
        + https://mavericknerd.github.io/knowledgebase/BugBountyRecon/
        + https://gowthams.gitbook.io/bughunter-handbook/list-of-vulnerabilities-bugs/recon-and-osint/recon
* **SSL scanning**
  + **Sslyze**
    - sslyze --regular [www.example.com](http://www.example.com)
  + **Sslscan** 
    - Sslscan [website]
  + **Nmap** 
    - nmap --script ssl-enum-ciphers -p [port] [hostname]
  + **tls-scan** 
    - tls-scan -c [[website]](http://search.yahoo.com) --all --pretty
  + **Testssl.sh**
    - git clone –depth 1 <https://github.com/drwetter/testssl.sh.git>
    - ./testssl.sh [website]
  + [**https://pentest-tools.com/network-vulnerability-scanning/ssl-tls-scanner**](https://pentest-tools.com/network-vulnerability-scanning/ssl-tls-scanner)
  + **Ssllabs.com**
  + **Heartbleed**
    - cat list.txt | while read line ; do echo "QUIT" | openssl s\_client -connect $line:443 2>&1 | grep 'server extension "heartbeat" (id=15)' || echo $line: safe; done
      * Note that the Heartbleed (CVE-2014-0160) leads to a leak of server memory content and disclosure of sensitive information.