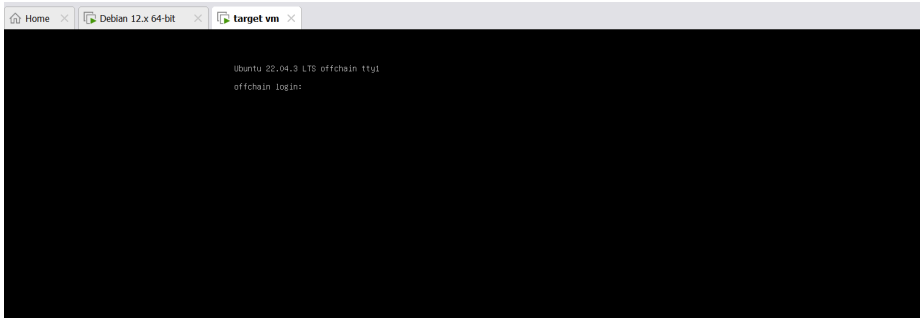
Lab 6: Penetration Testing AKOR JACOB TERUNGWA

Created	@March 15, 2025 8:49 PM
Attendance Required	☐

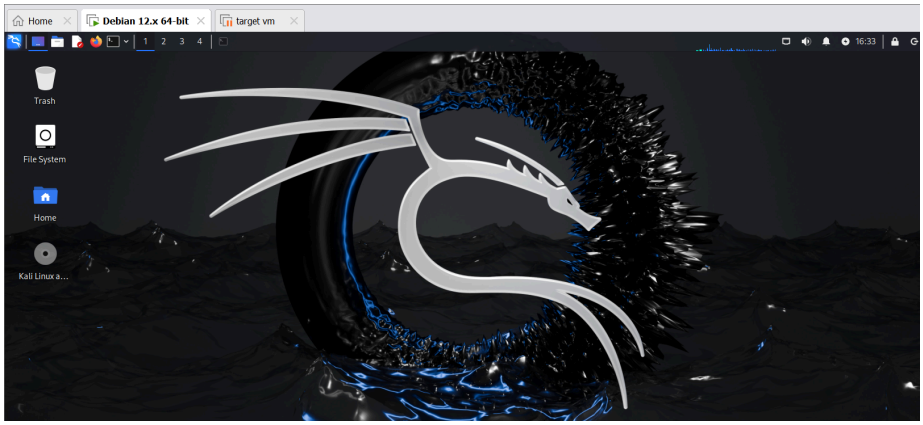
Lab Setup

1. Download and set up two VMs:

○ **Target VM (Ubuntu) with a vulnerable web application**



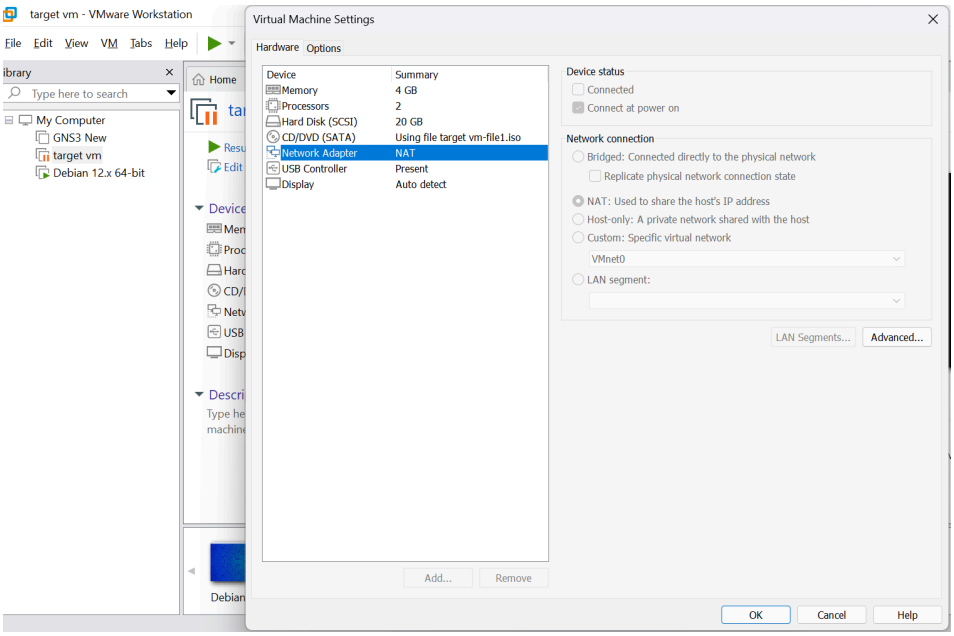
○ **Attacking VM (Kali Linux or another penetration testing distribution)**



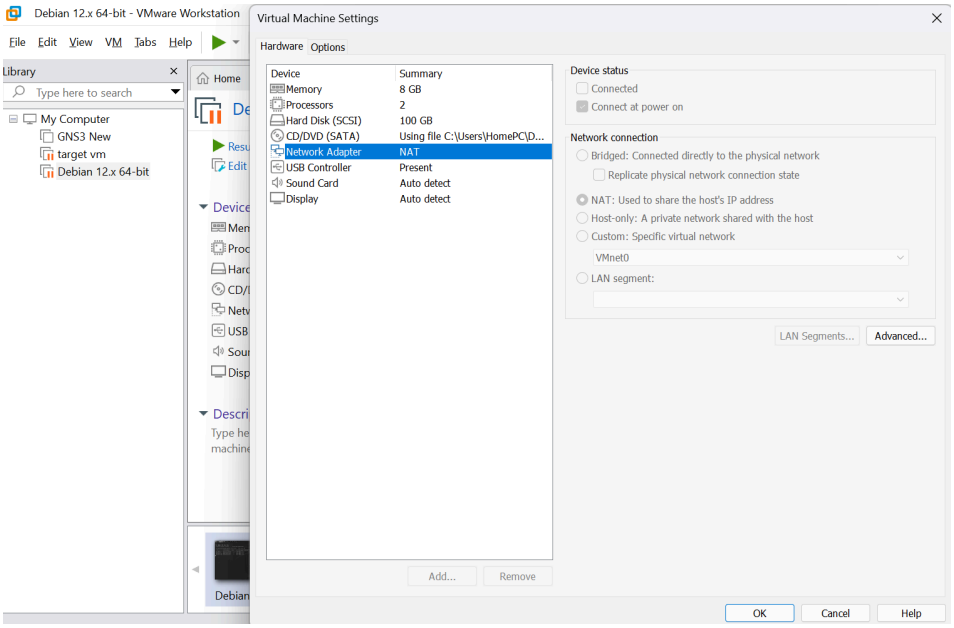
2. Ensure both VMs are on the same network.

Both the Target VM(Ubuntu) and the Attacking VM (Kali Linux) are on NAT, which means that both VMs are on the same network, as shown below.

This is the target VM



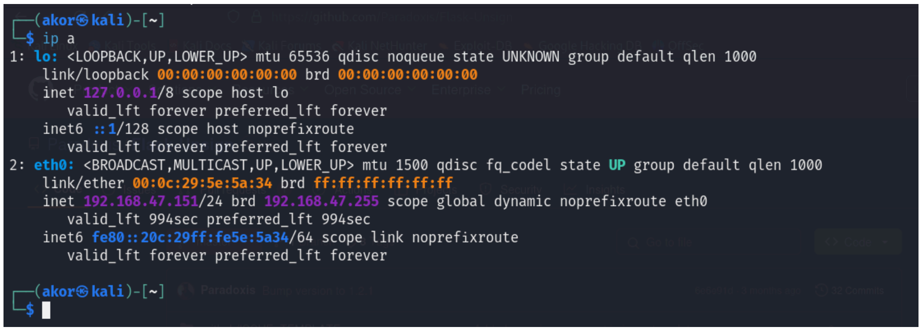
- Attacking VM



3. Identify the target VM's IP address.

To identify the target VM's IP address, first, I used the command below to know the IP and the subnet mask of the machine to identify
the network range.

- IP a

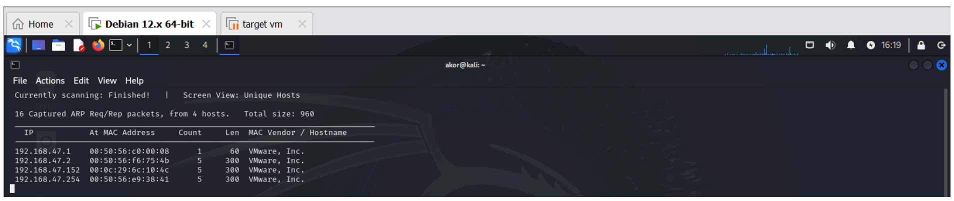


4. Open a terminal on the attacking VM to begin the challenge

From the attached picture above, as you can see, the subnet mask is 255.255.255.0 , therefore, the network range has to be 192.168.47.0/24 .

Then, I used `netdiscover` to discover active hosts on the network

- sudo netdiscover -r 192.168.47.0/24



From the output, the IP address of the target VM is 192.168.47.152.

2. I used `nmap` to perform a detailed scan **for Open Ports and Running Services** of the target VM using:
    - nmap -sV -sC -p- 192.168.47.152

The following ports and services were opened and running after I had performed a detailed scan for target VM.

- Port 22/tcp is opened ssh running OpenSSH 8.9p1 Ubuntu 3ubuntu0.3 (Ubuntu Linux; protocol 2.0)
- Port 80/tcp is opened http hosting Apache httpd 2.4.52
- Port 3131/tcp filtered netbookmark running Node.js Express framework .

3. **Analyzed the result and determined Potential Attack Vectors:**

Based on the open ports and services identified, here are the **potential attack vectors** I explored:

- **Port 22 - SSH (OpenSSH 8.9p1)**
  - **Attack Vector:** Brute-force or weak credentials.

  SSH is often targeted for brute-force attacks if weak or default credentials are used.
- **Port 80 - HTTP (Apache httpd 2.4.52)**
  - **Attack Vector:** Web application vulnerabilities.

  The web server runs Apache 2.4.52, which may have vulnerabilities or misconfigurations.
- **Port 3131 - Filtered (netbookmark)**
  - **Attack Vector:** Misconfigured or vulnerable service.

  The service on port 3131 is filtered, which means it may be behind a firewall or restricted to certain IPs.

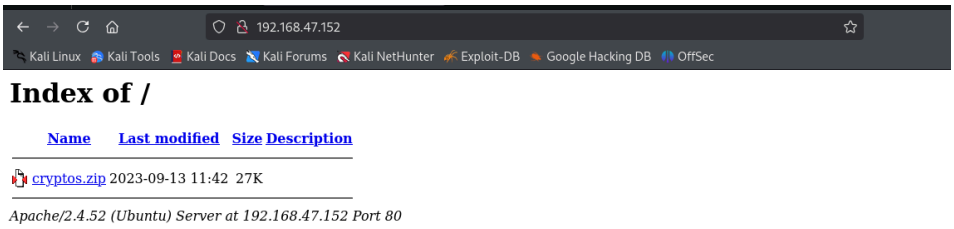
  Combined Attack Vectors
  - **Attack Vector:** Chaining vulnerabilities.

B. I analyzed the Web Application:

I identified the Web Application Running on the Target VM:

I opened a browser and navigated to the target IP(192.168.47.152) on a web server running on port 80

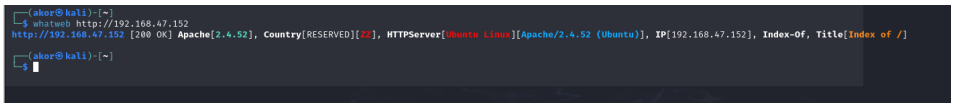
- http://192.168.47.152 :80



The index page image only listed a file named cryptos.zip as seen , indicating directory listing is enabled, a common misconfiguration.

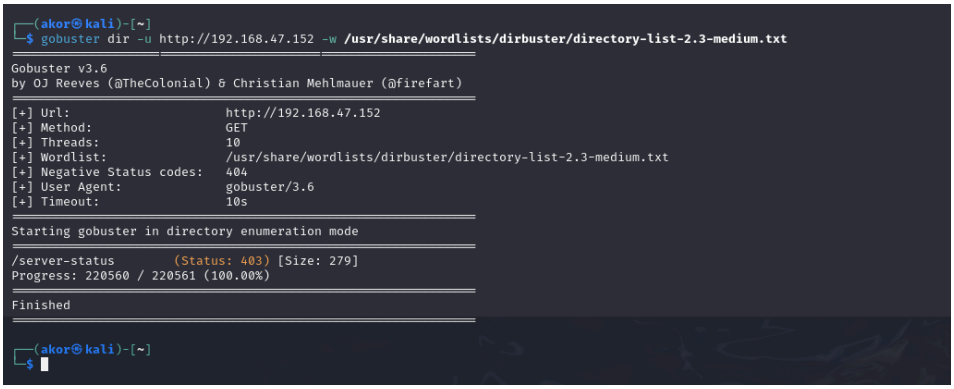
Then, I used a tool like `whatweb` to gather more information about the web application:

- `whatweb http://192.168.47.152`



I proceeded to obtain the Source Code to identify a Logic Flaw:

The source code is not directly available `;` therefore `,` `I` used a tool like `gobuster` to enumerate directories and files:

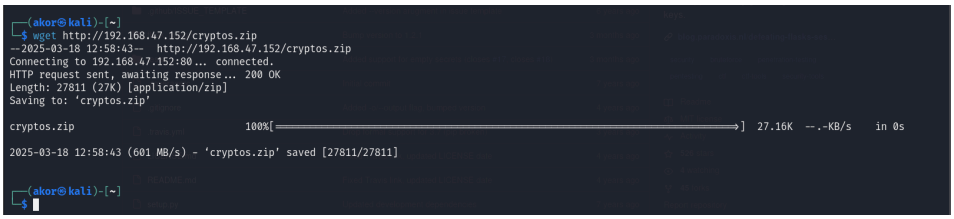


After this, I went ahead to obtain the Source Code itself to identify a Logic Flaw by downloading and extracting the cryptos.zip file.

The cryptos.zip file contains the source code of the web application

I downloaded the File using the command:

wget http://192.168.159.134/cryptos.zip



Then, I extracted the content of the zip file using the command:

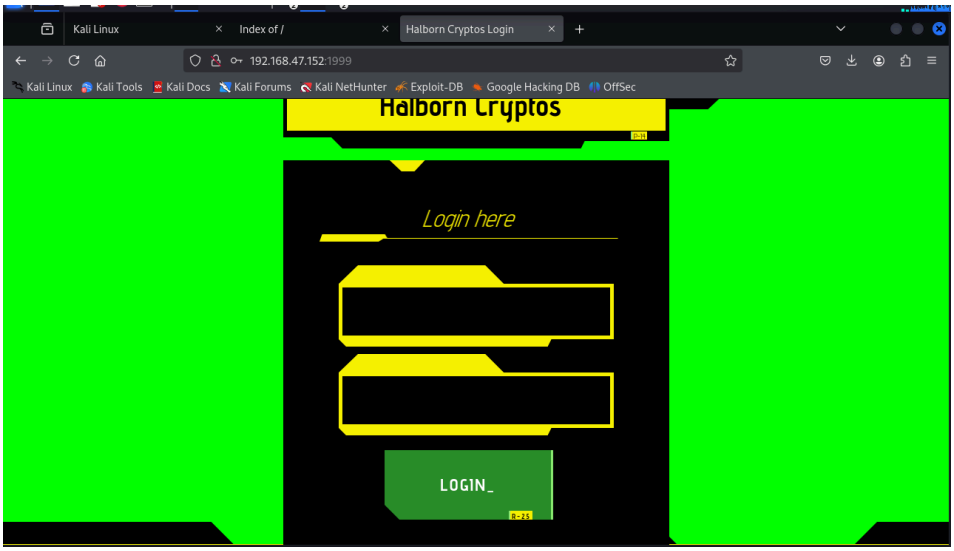
- unzip cryptos.zip

The extracted zip file basically provides the source code of the web application, secrets.py and run.py

Then, I exploited the JWT Secret Key Vulnerability:

- **Step 1: I identified the Target**

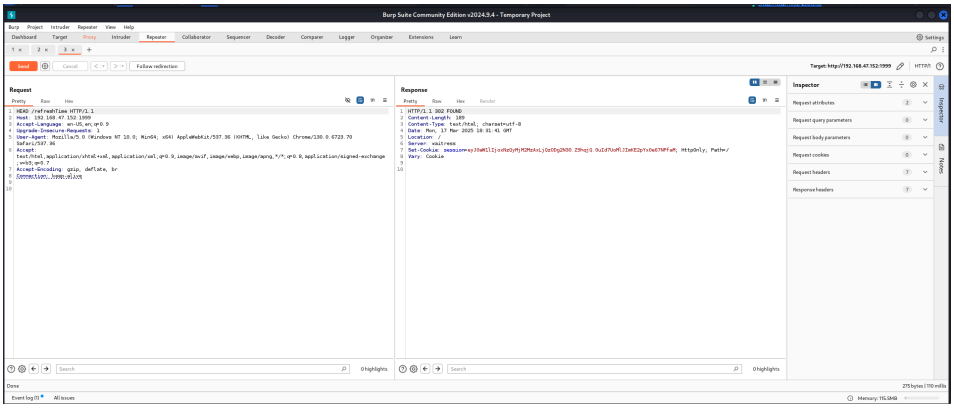
1. **Target IP and Port:**
   - Target IP: `192.168.47.152` (since the target is on the same network).
   - Port: `1999` (HTTP).
2. **I verified the Target:**
   - Then, opened a browser and navigate to:
     - http://192.168.47.152:1999



I landed at the login page of the Flask application.

- Step 2: **Registered a New User: By n**avigating to the registration page:

Step1: Here in Burp Suite, I visited a route/refreshTime to capture a cookie



In the Flask app, JWT tokens are signed using `token_jwt`, which is generated in `run.py`

Then, I created a Python Script(generate_wordlist.py) to generate the Wordlist

I used the command: python3 generate_wordlist.py to generate the wordlist

```
┌──(myenv)─(akor㉿kali)-[~/Downloads/cryptos/cryptos/flask]
└─$ python3 generate_wordlist.py

[+] Wordlist generated: jwt_wordlist.txt
```

Use various exploitation Tools to gain access

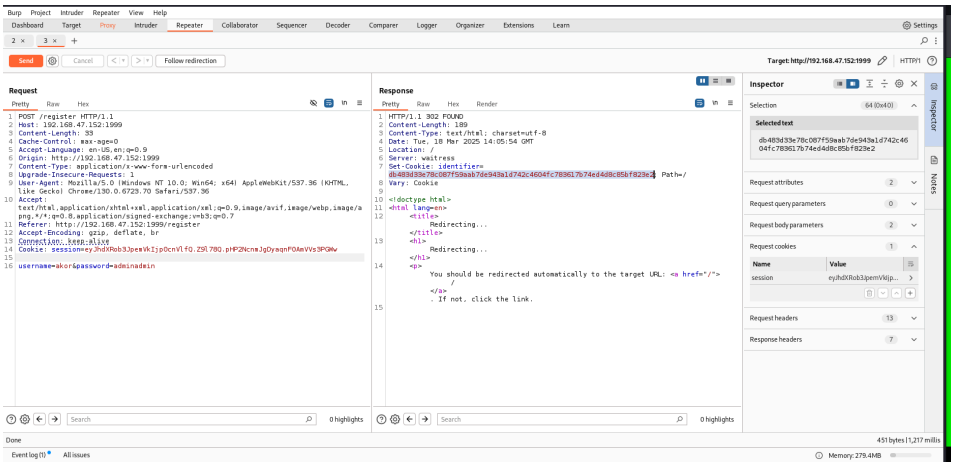
Flask uses the secret_key to sign session cookies which a 5-character hexadecimal key has only 1,048,576 possible combinations, making it feasible to brute-force and forge a valid cookie.

In other to do this, I needed to find the secret key, so i generated **all possible 5-character combinations** from the given alphabet (`abcdef0123456789`)

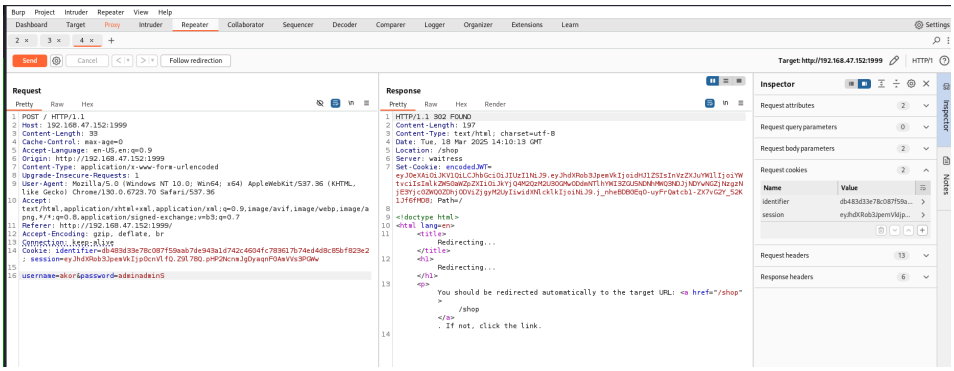
- flask-unsign --unsign --cookie "eyJ0aW1lIjoxNzQyMjM2MzAxLjQzODg2N30.Z9hqjQ.0uId7UoMlJImKE2pYx0e67NPfaM" --wordlist wordlist.txt --no-literal-eval

```
┌──(myenv)─(akor㉿kali)-[~/Downloads/cryptos/cryptos]
└─$ flask-unsign --unsign --cookie "eyJ0aW1lIjoxNzQyMzA1NDE2LjU5OTY1NDd9.Z9l4iA.K4dJZ-lVCn2a_LZBtH_lUlz52h4" --wordlist
wordlist.txt --no-literal-eval

[*] Session decodes to: {'time': 1742305416.5996547}
[*] Starting brute-forcer with 8 threads..
[+] Found secret key after 339328 attempts
b'52d24'

┌──(myenv)─(akor㉿kali)-[~/Downloads/cryptos/cryptos]
└─$
```

I proceeded by registering an account on the website which I needed session, username and password and I got an identifier: db483d33e78c087f59aab7de943a1d742c4604fc783617b74ed4d8c85bf823e2



logged into the account successfully and obtained JWT: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdXRob3JpemVkIjoidHJ1ZSIsInVzZXJuYW1lIjoiYWtvciIsImlkZW50aWZpZXIiOiJuyFrQatcb1-ZX7vG2Y_52K1Jf6fMD8



After logging in successfully, I was redirected accessing the shop page