92Patrick Austin

CS 445 Homework 1

10 February, 2018

See Figure 1 for the network configuration. The top left device, at 192.168.56.101, is used as the attacker. The top right device, at 291.168.56.102, is running web service through port 80. See Figure 2 for confirmation that each device is able to ping the other devices.

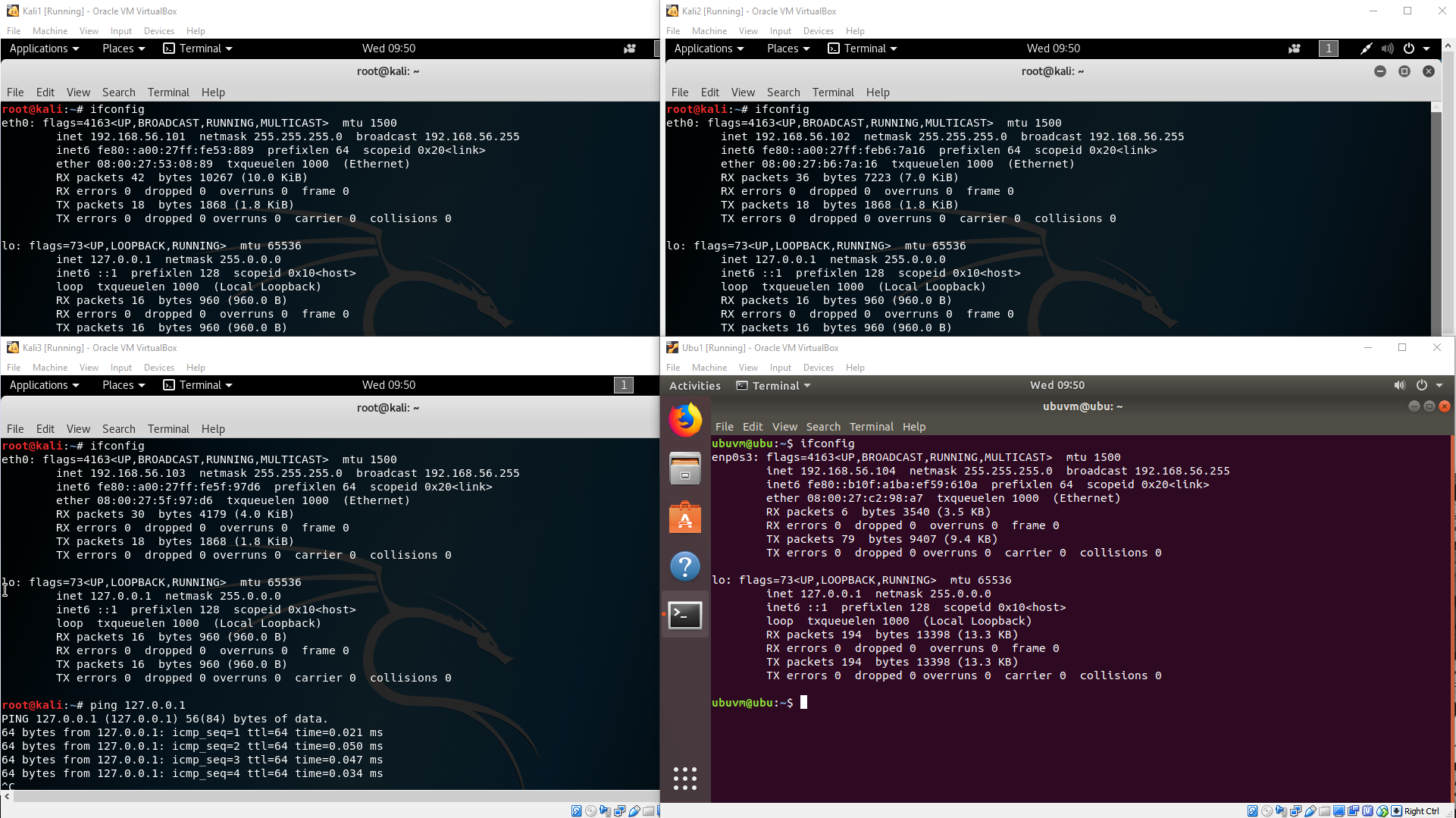
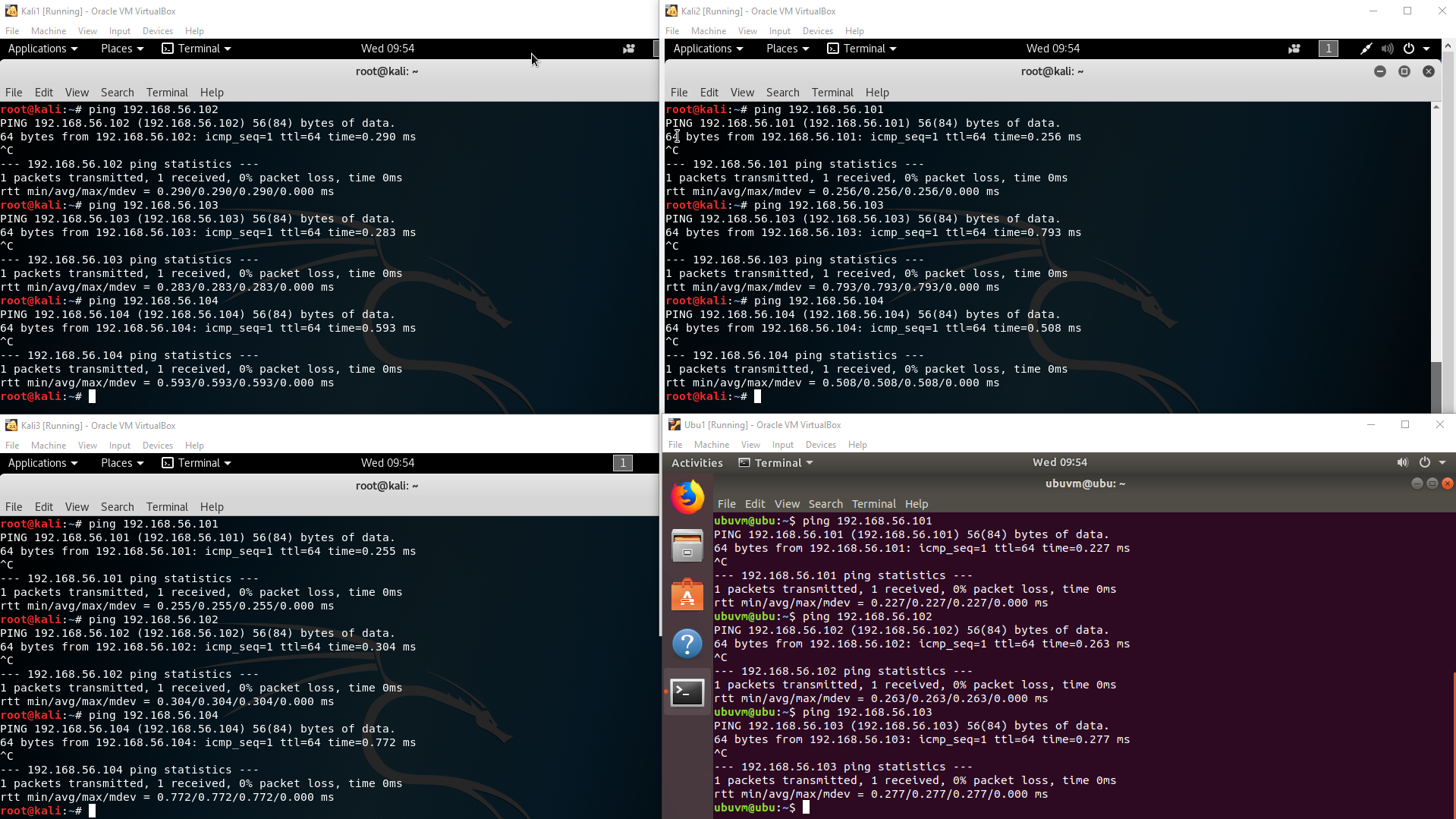


Figure 1: Testbed configuration, showing each device’s ifconfig information. The top left device, which will be used as the attacker, has address 192.168.56.101. The top right device has address 192.168.56.102. The bottom left device has address 192.168.56.103. The bottom right device, running Ubuntu, has address 192.168.56.104.

Figure 2: Testbed configuration, showing that each of the 4 devices is able to successfully ping the other 3 devices using the IP addresses described in Figure 1. 

1. To some extent this is an ambiguous question; sources online seem to use the term ‘ping sweep’ or ‘ping scan’ to refer to any of several related nmap options for host discovery. For that reason I will provide treatment of a few flavors of ping sweep command.

A command such as “nmap -sn 192.168.56.0/24” initiates host discovery on the specified subnet without conducting a subsequent port scan. This command combines the use of an ICMP echo request, a TCP SYN packet to port 443, a TCP ACK packet to port 80, and an ICMP timestamp request to do host discovery. Most sources seem to refer to this command, or an older variant of it which used the -sP parameter, when they discuss ping sweeps or scans. See Figure 3 for a screenshot of this scan being used.

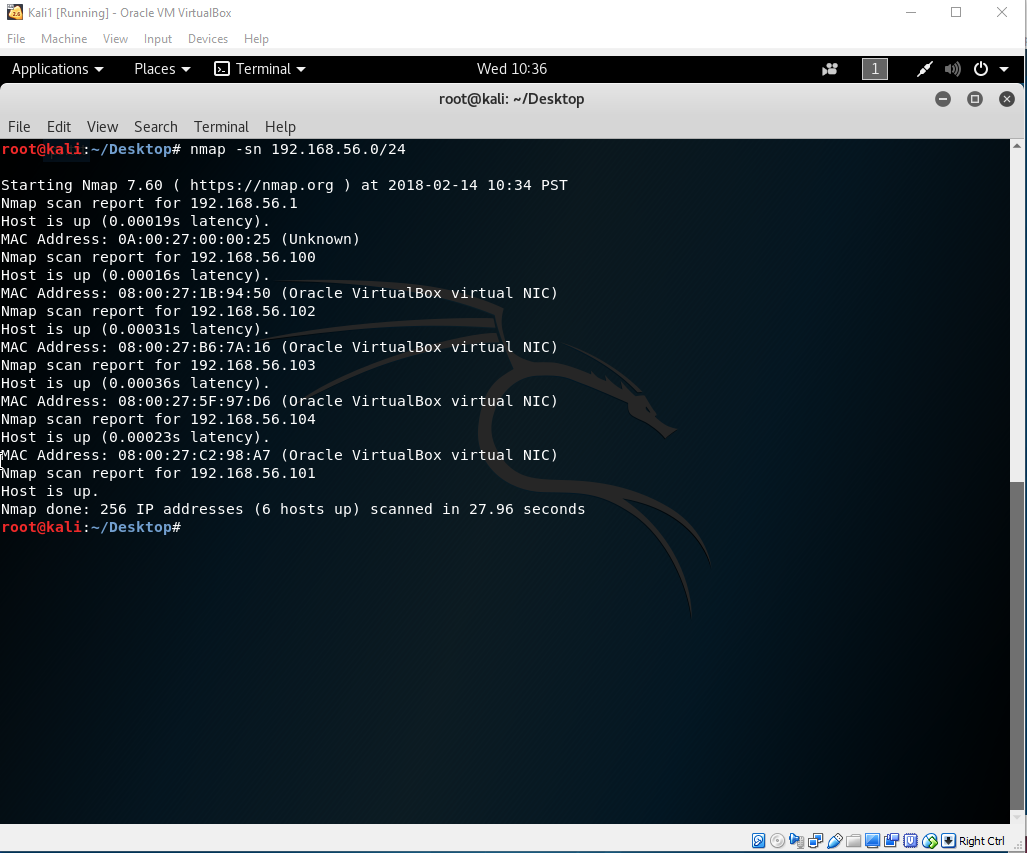


Figure 3: Successful usage of the command “nmap -sn 192.168.56.0/24” to perform a ping sweep.

If ping is taken more narrowly to mean host discovery via ICMP echo request only with no TCP involved, then the command “nmap -sn -PE 192.168.56.0/24” can be used to discover hosts with an echo request only, without using the TCP and ICMP timestamp components mentioned above. See Figure 4 for a screenshot of this scan being used.

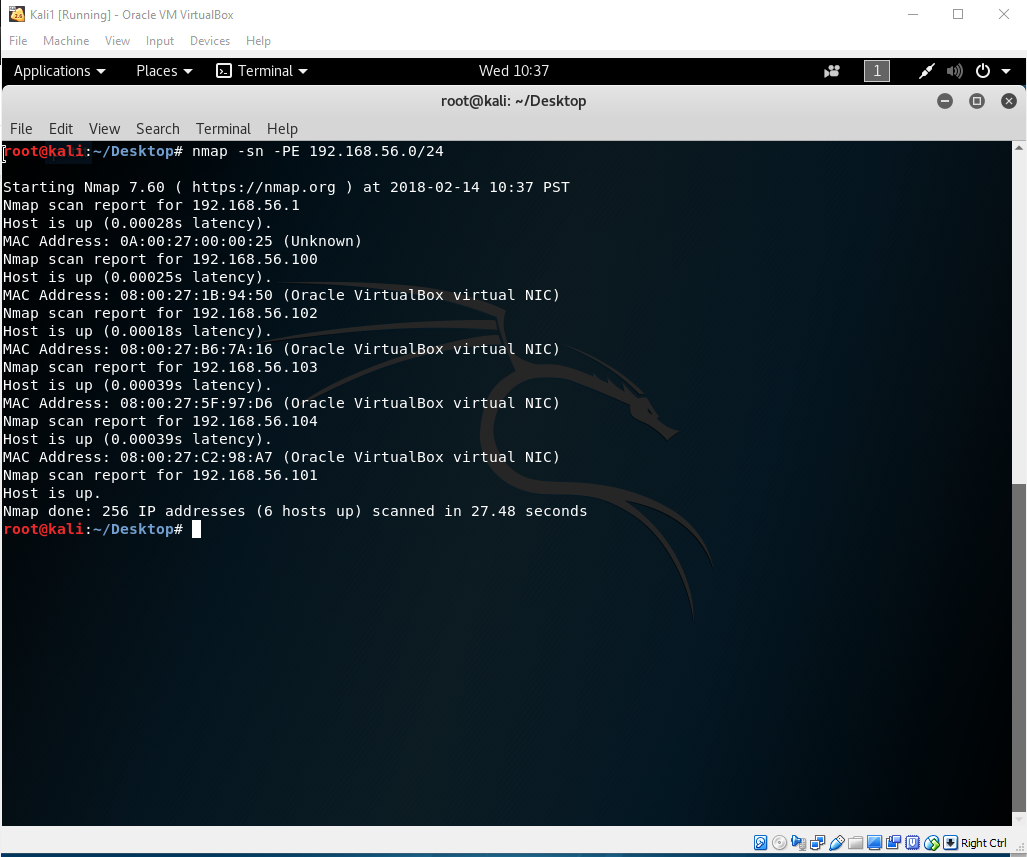


Figure 4: Successful usage of the command “nmap -sn -PE 192.168.56.0/24” to perform a ping sweep using only ICMP echo requests.

Sometimes sources discussing nmap online use pinging to mean e.g. an ARP scan on an ethernet LAN, which this simulation is emulating. This can be done with “nmap -sn -PR 192.168.56.0/24” as shown in Figure 5.

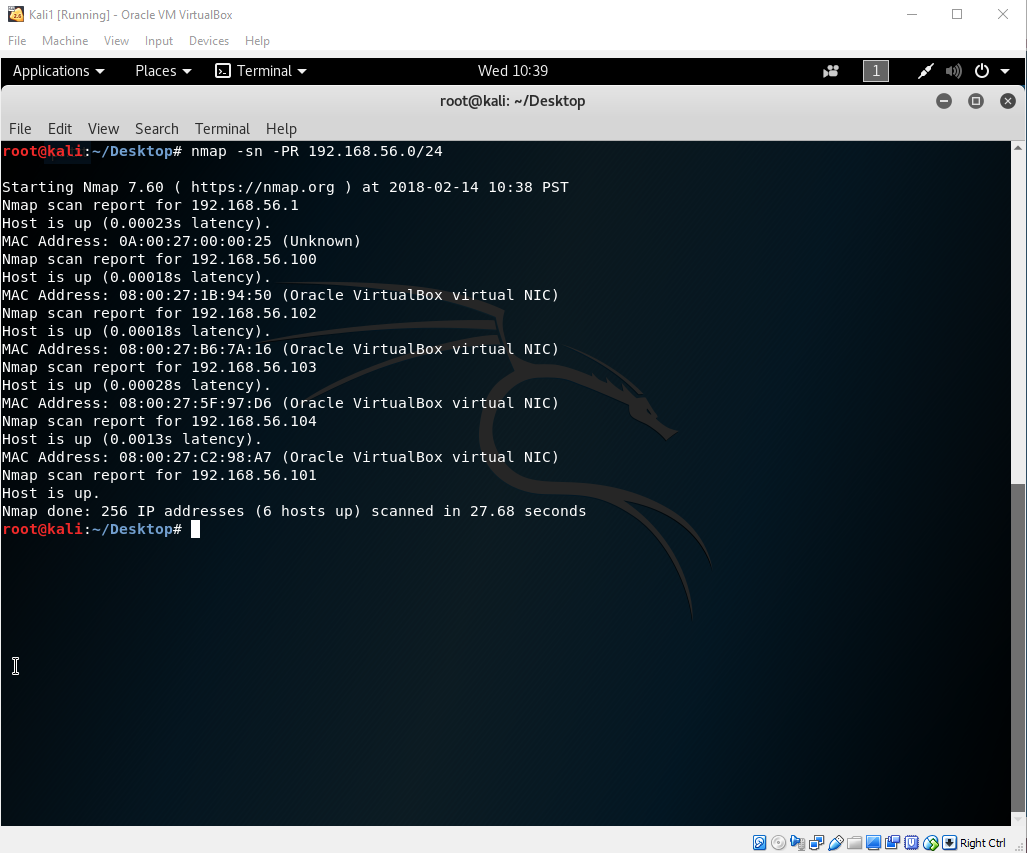


Figure 5: Successful usage of the command “nmap -sn -PR 192.168.56.0/24” to perform a ping sweep using an ARP scan.

All these scans were effective on this simple testbed. In real world situations different types of scans would have different performance in terms of time and against various types of countermeasure.

2. A file can be used to store IP addresses for probing with nmap quite easily. Figure 6 shows the file I created, ips.txt, which the attacker Kali at 192.168.56.101 could use to scan ports on the other devices detected in part 1.

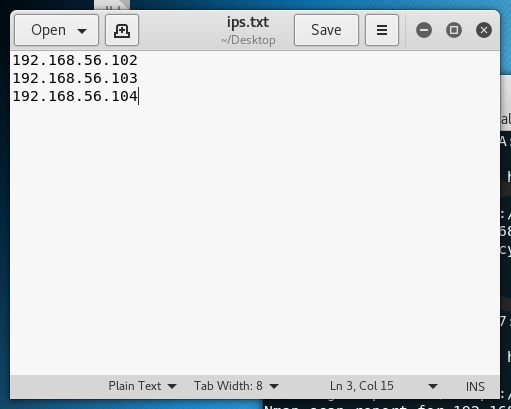


Figure 6: The simple structure of the IP address input file ips.txt, for use with nmap’s -iL command. As this file is intended to be run by the device at 192.168.56.101, that address is not included. This file was also used for address input in part 4.

Figure 7 shows the port scan done using the file, via the command “nmap -iL ips.txt”. From the results we can conclude that the only open port is port 80 on device 192.168.56.102, running http service.



Figure 7: Successful usage of the command “nmap -iL ips.txt” to scan the ports of the addresses specified in a input file. Note that port 80 is open on one of the devices, as specified.

3. Service version detection can be done in nmap with a command along the lines of “nmap -sV -iL ips.txt” in order to conduct version detection on all victim machines via file, as specified in part 2. Figure 8 shows this command in action. From the results we can conclude the version of Apache being used to run the http service on device 192.168.56.102, port 80.

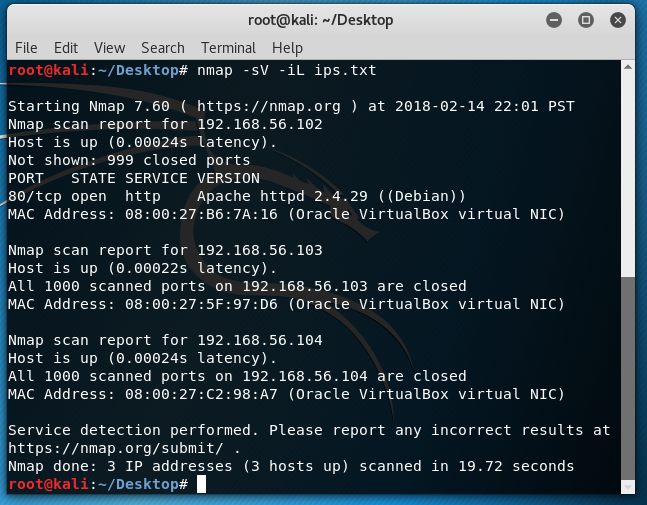
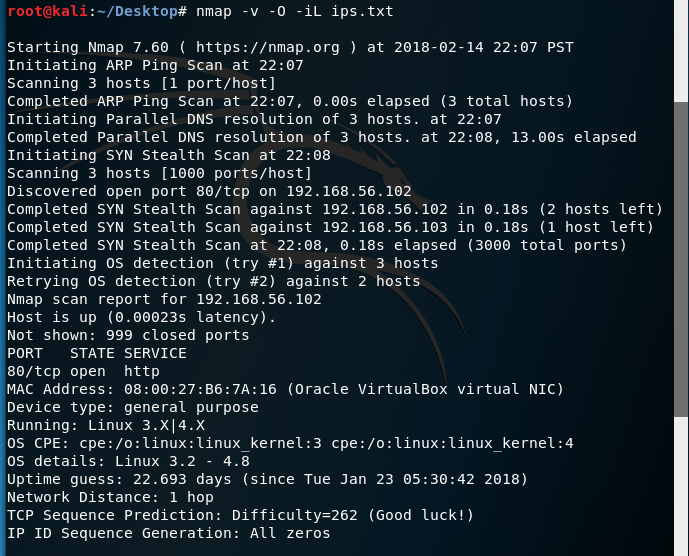


Figure 8: Successful usage of the command “nmap -sV -iL ips.txt” to conduct service version detection; results reveal the version of the Apache http service running on device 192.168.56.102. The target addresses are specified in ips.txt as in part 2.

OS detection can be done similarly via command “nmap -v -O -iL ips.txt”. See Figure 9 for output from this command. I note that due to the lack of available open ports and services, nmap could not determine the OS of two of the devices; only for the 192.168.56.102 device running http service could nmap make a guess as to OS, Linux 3.2-4.8.



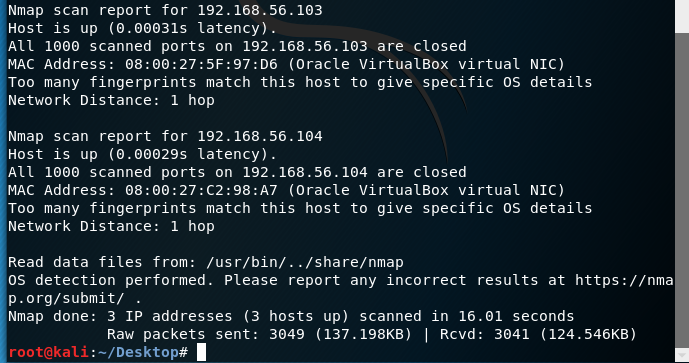


Figure 9: Successful usage of the command “nmap -v -O -iL ips.txt” to conduct OS detection; results fail against devices with all ports closed but succeed on the device running http service through port 80. The target addresses are specified in ips.txt as in part 2.

4. In order to scan 10 well-known ports at varying intervals and output their status to an easily readable file, I opted to create a simple executable bash script whose contents are shown in Figure 10. Please note that while the bash file, the input file, and a sample output file are available and can easily be provided upon request, I have opted not to include them in this submission due to the assignment document requesting a report only.

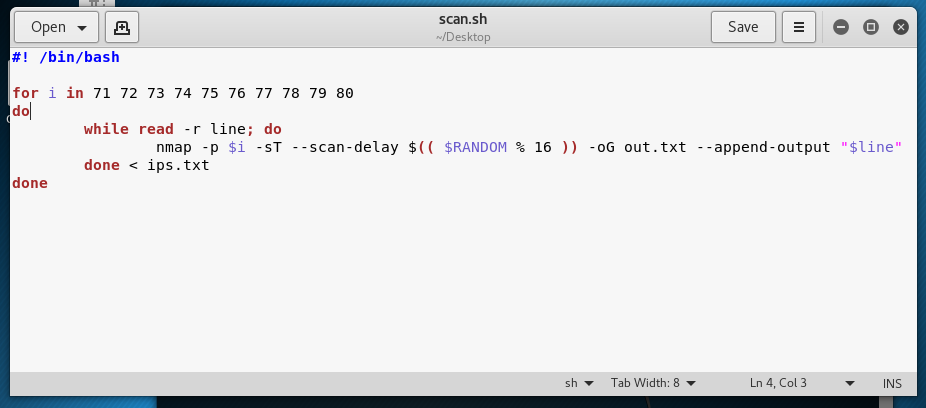


Figure 10: The simple bash script created to satisfy the requirements set out in problem 4. When compiled via the command “chmod u+x scan.sh” this script can easily be executed via “./scan.sh”.

The bash script scans ports 71-80 for all IP addresses specified in the input file ips.txt (endline separated), with a random delay of 0 to 15 seconds between each individual scan. Output is generated in out.txt. The ips.txt file used is identical to the one shown in Figure 4.

I opted to generate the output using the -oG command, which produces greppable output that I found most readable. -oN is another modifier option available for readable output, but given the structure of the problem (a new nmap call is needed for each individual port scan in order to generate a new random number for the time delay) I found too much redundant information was being printed. The -oG command reduced this redundancy significantly. Further removal of redundant information from the output file is an avenue that could be explored in more detail if need be by making a more complex bash script. Another option to simplify the output would be to settle for one call of nmap per port, i.e. one random delay value would be generated and repeatedly used for each of the ports. I opted to pursue more random behavior as this seemed most in the spirit of the question and would make for a less detectable scan; if this was not desired I throw myself at the mercy of the court.

Figure 11 shows the bash script in the process of being executed on the attacker Kali to scan ports 71-80 on the other devices in the network, with a random delay of 0 to 15 seconds between each port scan.

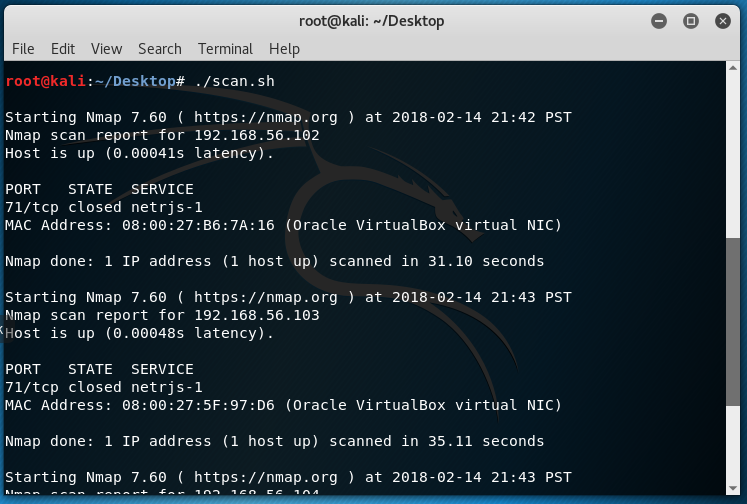


Figure 11: Port scan as specified in part 4 in progress via executable bash script. The script first determines the status of port 71 at each address listed in ips.txt, then determines the status of port 72, and so on, with a delay of 0-15 seconds between each scan.

Figure 12 shows a segment of the output file created by a successful run. As mentioned, this is in

greppable format.



Figure 12: A snippet of the output file created by the executable bash script in greppable format. This section shows scan results for ports 79 and 80 on each device; one of the devices has port 80 open, as specified.