Networks, Communications & Cyber Defence

Practical Session
UDP

Peter Norris
University of Warwick
October 2021

Contents

Starting assumptions

1: You have used Netkit **lstart** to launch a coordinated group of several virtual machines. Specifically, you are aware of how this group is coordinated via the **lab.conf** file.

2: You know how to shut down virtual machines using **lhalt** and **lcrash**.

3: You have used **ip / ifconfig** and **ping** within the virtual machines.

4: You have saved captured network traffic from a virtual machine using **tcpdump**.

5: You have viewed a packet capture file in the real Linux host using **wireshark**.

6: You are prepared to make accurate notes of what you do.

7: You will complete unfinished activities before the next timetabled lab session.

8: You will resolve what you do not understand by conducting your own careful (and ethically sound) experimentation and / or further reading.

UDP activities

Intended outcomes

9: Can pass data from a UDP client to a UDP listener in Linux using the **netcat** (aka **nc**) instruction.

10: Can analyse a captured pcap file using wireshark and explain the ports used in UDP datagrams.

11: Can explain how UDP datagrams are carried in IP packets which are in turn carried over the LAN in Ethernet frames.

12: Can adapt marginally incorrect instructions and / or Netkit configurations. (again; **there will be some deliberate errors**.)

Preparing your home directory

13: As in previous labs, we will not be storing pcap files for later analysis in your home directory. Ensure you have the necessary directory by executing:

```
14: makedir -p ~/nklabs/lab07
```

Launching the Netkit lab

15: Open a Linux terminal, make the **~/nklabs/** directory your current directory:

```
16: cd ~/nklabs/
17: pwd
```

18: As ever, use the **man** command to find out what a particular command does, however, note that **netcat** in the real host and virtual machines may be different versions with different switches and capabilities.

19: Copy the zipped archive **lab07.tar.gz** and save it in the **~/nklabs/** directory. This contains the configuration information for around 15 virtual machines split across two directories: **lab07a** directory and **lab07b** directory.

20: The overall arrangement of the virtual machines into subnets is almost identical to the lab06 arrangement. The lab **lab07a** contains the **W, X** and part of the **D** LAN. The lab **lab07b** contains the **Y** and **Z** subnets and the remainder of the **D** LAN.

21: Extract the the contents of **lab07.tar.gz** into the **~/nklabs** directory. Assuming your current directory is **~/nklabs** , this can be achieved by:

```
22: tar -xvcf lab07.tar.gz
```

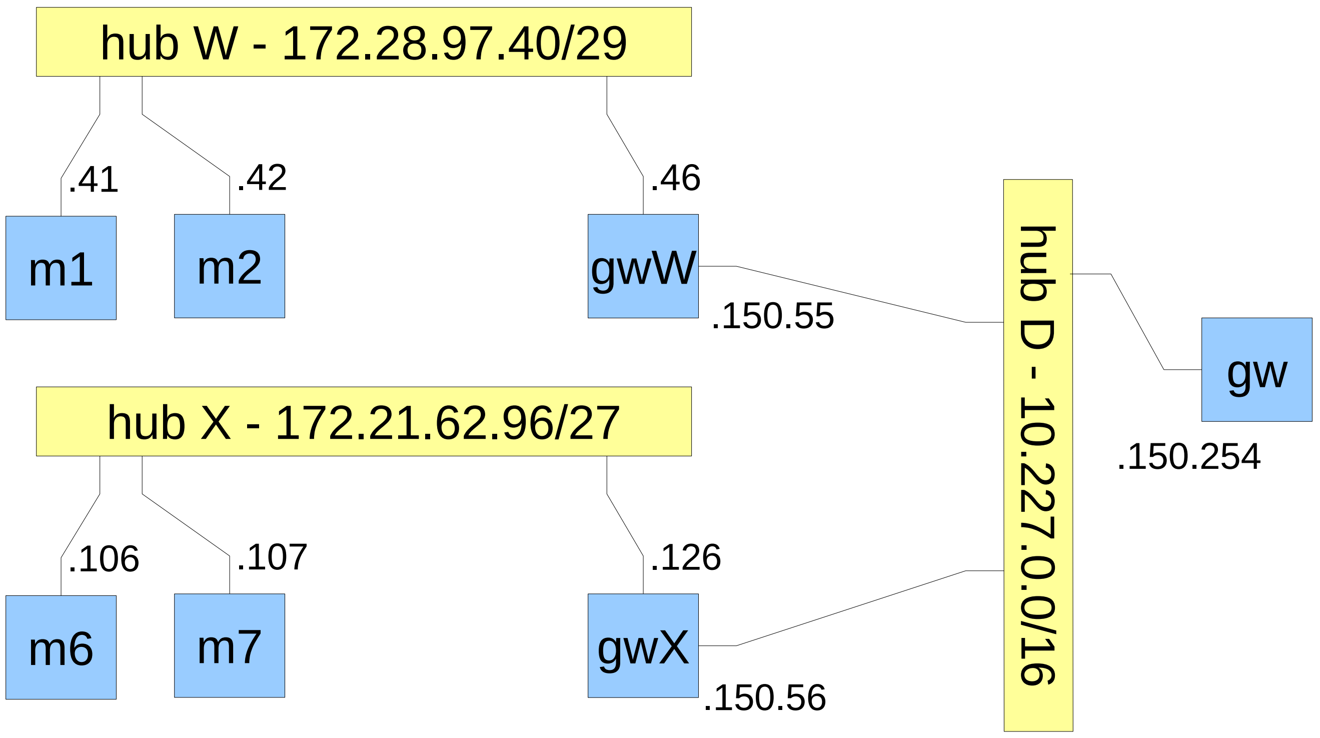
23: Place yourself in the newly created **lab07a** directory and list its contents:

```
24: cd lab07a
25: ls
```

26: Look in the **lab.conf** file. Confirm that it corresponds to the diagram of the network you have made last week in your notes and the one below.

27: With your current directory as the **~/nklabs/lab07a** directory (which contains the **lab.conf** file), launch the seven virtual machines using the instruction:

```
28: lstart
```



29: Look carefully at the virtual machines as they start. In particular record any errors you see.

Look at the configuration

30: Once all machines are up, use **tcpdump** on machines **m2, m7** and **gw** to store captured traffic in the files **m2-d1.pcap**, **m7-d1.pcap** and **gw-d1.pcap** in the **hostlab** directory. The instructions will be similar to the one below.

```
31:  tcpdump -v -s eth0 -w /labhost/m2-d1.pcap
```

32: Write down the exact instruction you used on each machine.

33: Now generate some UDP traffic. From machine **m1**, try and send a UDP datagram to port 51966 of gwW (you have a choice of IP addresses - note which you chose and why):

```
34:  netcat ???? 51966
```

35: Now let us start a listener first before we try and send a UDP packet. On machine **gwW** launch and on machine m7, launch the UDP listeners (The instructions below are definitely wrong. What is the problem?):

```
36:  netcat -l -p 51966    # on gwW
37:  netcat -l -p 57005    # on m7
```

38: Convert the port numbers to hex and note the value.

39: Go back to **m1** and send *hello m7* to **m7** in a UDP datagram:

```
40:  echo "hello m7" | netcat -q0 ????
41:  echo "hello gwW" | netcat -q0 ????
```

42: Note what the **-q** option should do in netcat. How does the -w option relate to the -q option? Is this useful for UDP? Experiment with / without the **-q** / **-w** options .
What does the pipe **|** symbol achieve in the shell?
What does the **echo** command do?

43: Stop the packet capture in **m2** and look at its content in wireshark. Make sure that you can find and explain the UDP datagrams in each of the activities above. Note for each case which client port was used. Is there a pattern? Compare the results you got with the results of a colleague and note similarities / differences.

Using the screen command

44: It is often useful to be able to look at two things concurrently in a netkit virtual machine. The **screen** command enables you to have several output terminals, only one of which is visible at a time. Think of them like a deck of cards. You can see the one that is on top and you can choose which one goes on top. Once you have started **screen**, *ctrl-a* followed by another character controls what screen does. For example,
ctrl-a c will create another window (you get one for free when you start **screen**)
ctrl-a n will move to the "next" window.
ctrl-a p will move to the "previous" window.

45: We will use screen to do several things on machine **m6**.

46: First, start a packet capture in **m7** writing to a pcap file **m7-d1.pcap** (put it in the right directory so it will still be available to you next week on any machine in the faculty).

```
47:  tcpdump etc etc
```

48: Now on **m6**, start screen (read the initial messages, then press whatever is suggested to get to the shell prompt).

```
49:  screen
```

50: On **m6** start a netcat listener on port 64206 (what is this in hex?)

```
51:  nc -u -l -p 64206
```

52: Create another widow on **m6** using **screen**:

```
53:  ctrl-a c
```

54: In the window you have just created, start another UDP listener on port 57005.

```
55: nc -u -l -q 57005
```

56: Check you can switch between the two windows on **m6**:

```
57: ctrl-a n
58: ctrl-a n
```

59: Create another widow on **m6**:

```
60: ctrl-a c
```

61: In the window you have just created, start another UDP listener, this time on port 51966:

```
62: nc -u -1 -p 51966
```

63: Create a fourth widow on **m6**:

```
64: ctrl-a c
```

65: In this window launch **netstat** to look for all IPv4 packets (this excludes Unix sockets from the display since these do not interest us):

```
66: netstat -an4
```

67: It would be useful if we could have a continuous view of the network status via **netstat** rather than the one-shot view. There is a **-c** option to continuously refresh the output every second. This however causes the display to scroll quite vigorously. Instead we will use the **watch** command on **m6**:

```
68: watch -n1 netstat -an4
```

69: Keep this netstat output as the visible screen on **m6**.

70: Now, on **m1**, **m2** and **gwW**, connect to the three UDP listeners you have running on **m6**. On **m1**, you should force the use of client port 11111 to connect to 64206, on **m2** use client port 22222 to connect to port 57005 and on **gwW** let the operating system choose the client port to connect to port 51966. (Do these need to be different or could they be the same?). For example, on **m1** this would be:

```
71: echo "m1m1m1" | nc -u 172.21.62.106 64206 -p 11111 -q0
```

72: Record what netstat shows as you send each datagram.

73: Stop the packet capture on m7 and explain the port number in every UDP datagram.

Using netcat UDP to transfer files

74: So far, we have used netcat interactively to make characters from the keyboard (stdin) at one end to appear on the screen (stdout) at the other end. It is however a trivial matter to redirect stdin from being the keyboard using < . It is similarly trivial to redirect stdout to a different file instead of the screen using > .

75: The text file we will send to the client when they connect will be the content of **/etc/serices** (at the listener's end). The client should save this to a file in the **/root** directory named **received-file**.

76: We will use **m1** as the listener to send the file using something like:

```
77: nc -ulp 49374 < /etc/services
```

78: Ensure you have packet capture running on **m2**.

79: On a machine (which is neither **m1** nor **m2**), get this file with:

```
80: nc -u -q0 172.28.97.41 49374 > received
```

81: We can therefore set up a simple UDP file transfer mechanism using this approach. Either end could be the sender but we will set the listener to always send the same file to any client that connects.

82: This however does not work. What is the problem? Can it be fixed?

Extension activity - DNS over UDP

83: Leave the first set of machines running which you started from the **lab07a** directory ie hubs **W** and **X** and some of hub D**.**

84: Debian provides multiple desktops aka workspaces. Select a new workspace (quickly switch between them using *ctrl alt right-arrow* or *ctrl alt left-arrow*) and in it open a terminal with the current directory set to **~/nklabs/lab07b**.

85: Start the netkit lab in the **lab07b** directory using **lstart**. This lab will also use hub **D** to create the overall configuration shown below, plus two Domain Name Servers (**dns1** and **dns2**) on hub **D** which are not shown on the diagram on the next page. Note that in this lab, addressing and routing is achieved using the **ip** instruction rather than **ifconfig** and **route**.

86: Start a packet capture on gw, then on m1 do a dns lookup:

```
87: nslookup m2.cyber.test
```

88: Stop the packet capture and explain what is happening in the UDP packets. Try typing *udp* into the wireshark filter - explain what happens.

Extension activity: TFTP over UDP

89: The Trivial File Transfer Protocol (TFTP - RFC 1350) uses UDP as its transport protocol and builds a small protocol on top of that. Typically it is used as part of a Preboot eXecution Environment (PXE) where the small footprint client gets the files it needs to boot, from a network file server via TFTP.

90: Gateway **gwZ** (in **lab07b**) has been configured in its startup file to function as a TFTP server. Some arbitrary files have been placed in the **/srv/tftp** directory for it to provide to requesting clients.

91: Start a packet captures on a machines **gw** and **m17**.

```
92:  tcpdump <etc etc>
```

93: Then on m1, launch a tftp client

```
94:  tftp 10.227.150.58
```

95: which will give you the tftp prompt **tftp>** Try entering the following (do not enter the “tftp>” prompt) to enable tracing of progress, increased verbosity, getting a particular file, and quitting the session.

```
96:  tftp> trace
97:  tftp> verbose
98:  tftp> get b/b3     # x/xn where x=a..f and n=1..6
99:  tftp> quitting
```

100: Alternatively, the same effect can be achieved by passing the interactive commands in via a pipe.

```
101:printf "trace \nverbose \nget b/b3 \nquit" \
102:| tftp  10.227.150.58
```

103: Stop the packet captures on the machines and look at the content. In particular, look how tftp uses UDP.

104: Experiment and explain.

References

105: Postel, J; User Datagram Protocol; RFC768; https://tools.ietf.org/html/rfc768