CSCI368 Network Security

School of Computing and Information Technology

University of Wollongong

Lab Set Up

Notice: This lab is partially built on SEED Labs

Objectives

- Set up VM lab environment which will be used later.
- Be familiar with frequently used docker commands for VM labs.
- Understand packet sniffer – for eavesdropping attacks.
- Apply packet analyser tools – Wireshark and tcpdump.

Part 1 VM Set Up

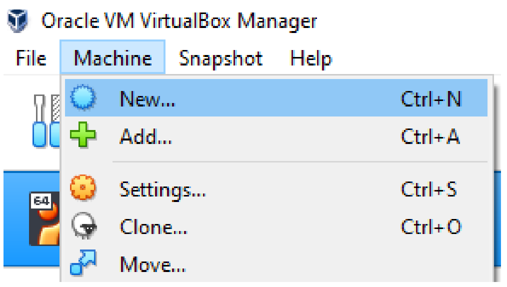
Task 1: Set up the VM environment (for Windows)

In this task, you will set up a virtual machine (VM) for some of the labs of this subject. Follow the instructions below to set up the environment. For Apple silicon machines, see here.

Required Software: VirtualBox version 7.0.8

Step 1. Download and Unzip VM lab image (SEED-Ubuntu20.04.zip).

Step 2. Create a new VM in VirtualBox



Step 3. Name the VM

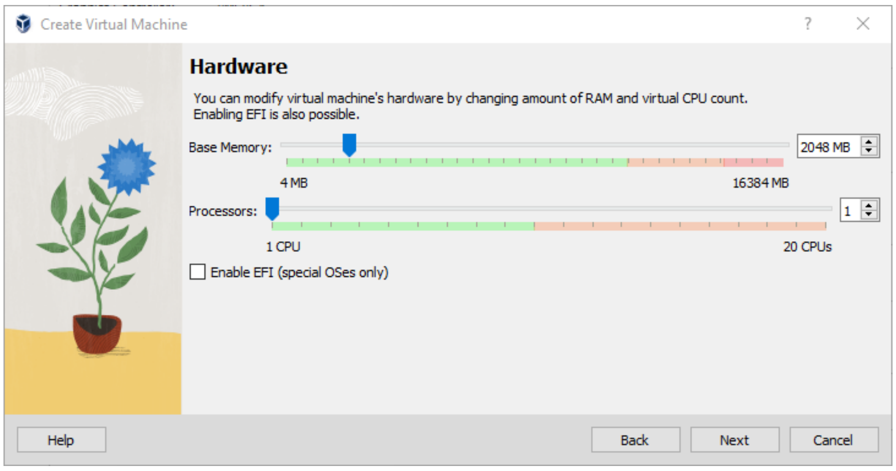
Name: **CSCI368-labs** (for example)

Type: **Linux**

Version: **Ubuntu 20.04 LTS (64-bit)**.

Then, **Next**.

Step 4. Set memory size to **2048 MB**, then **Next**. (You may allocate more than one processor.)



Create Virtual Machine

Hardware

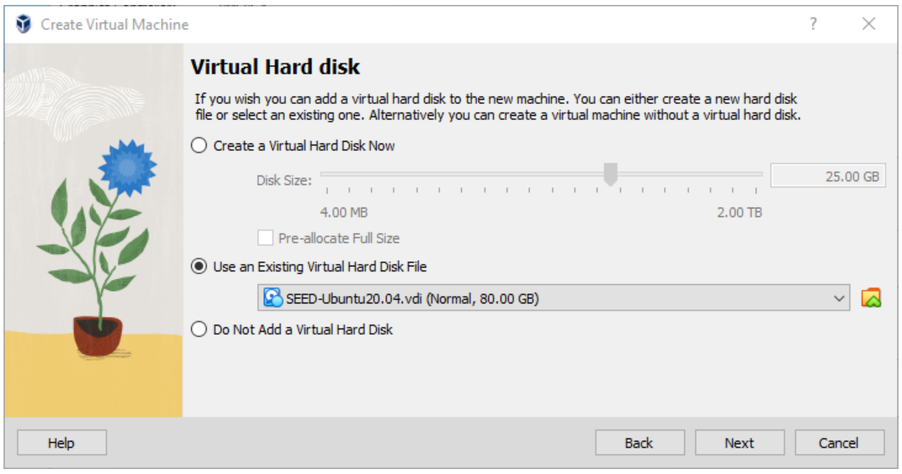
You can modify virtual machine's hardware by changing amount of RAM and virtual CPU count. Enabling EFI is also possible.

Base Memory: 2048 MB

Processors: 1

Enable EFI (special OSes only) [ ]

Step 5. Select “Use an existing virtual hard disc file” and select the VM lab image, then Create.



Create Virtual Machine

Virtual Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select an existing one. Alternatively you can create a virtual machine without a virtual hard disk.

○ Create a Virtual Hard Disk Now

Disk Size: 25.00 GB

Pre-allocate Full Size [ ]

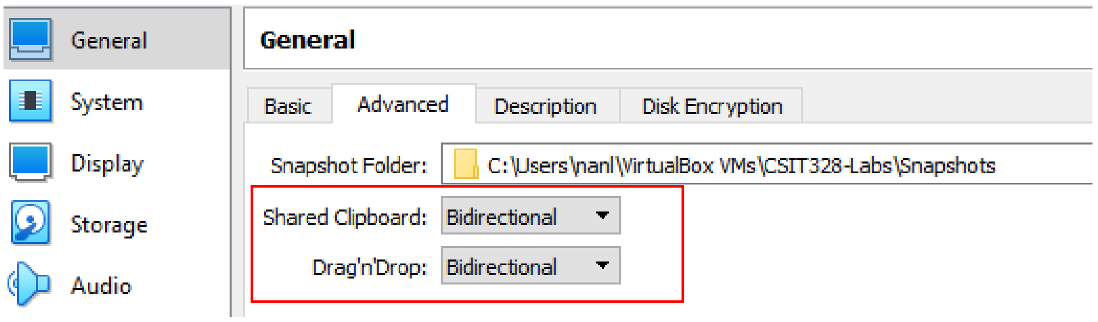
[✓ Use an Existing Virtual Hard Disk File]

SEED-Ubuntu20.04.vdi (Normal, 80.00 GB)

○ Do Not Add a Virtual Hard Disk

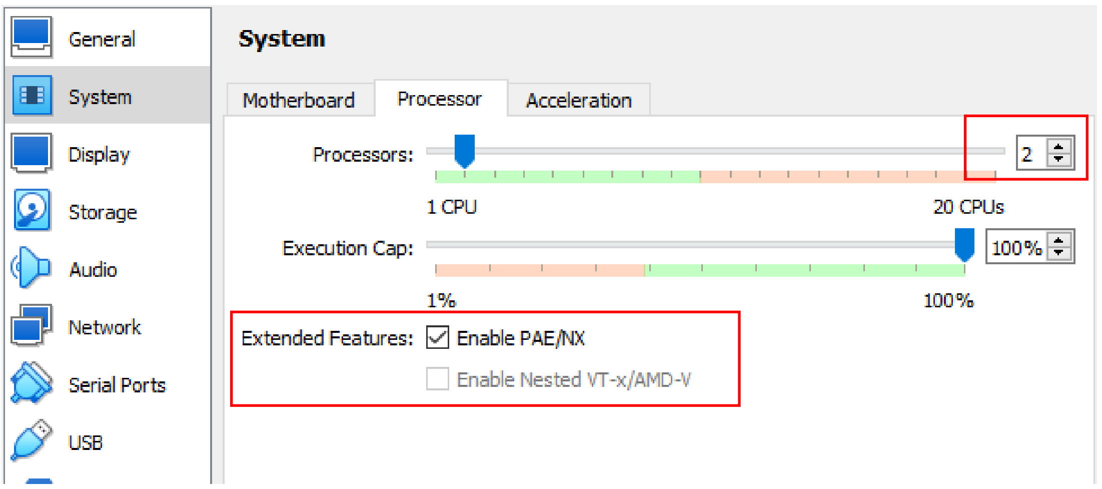
Step 6. Right click the created VM and go to the **Settings…**

Step 7. Configure **General → Advanced** tab as follows.



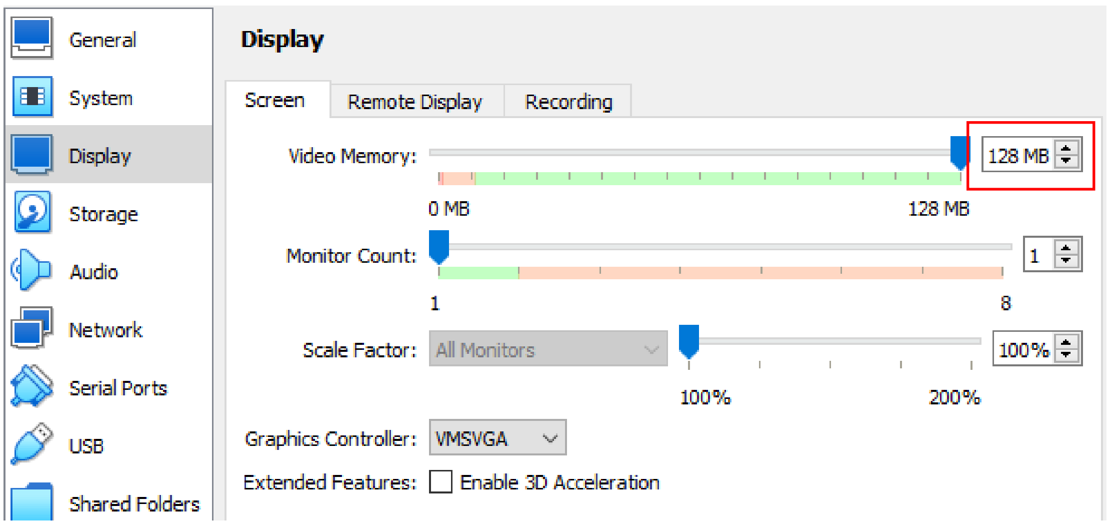
Snapshot Folder: C:\Users\nanl\VirtualBox VMs\CSIT328-Labs\Snapshots
Shared Clipboard: Bidirectional
Drag'n'Drop: Bidirectional

Step 8. Configure **System → Processor** tab as follows



Processors: 2
Execution Cap: 100%
Extended Features: ☑ Enable PAE/NX
☐ Enable Nested VT-x/AMD-V

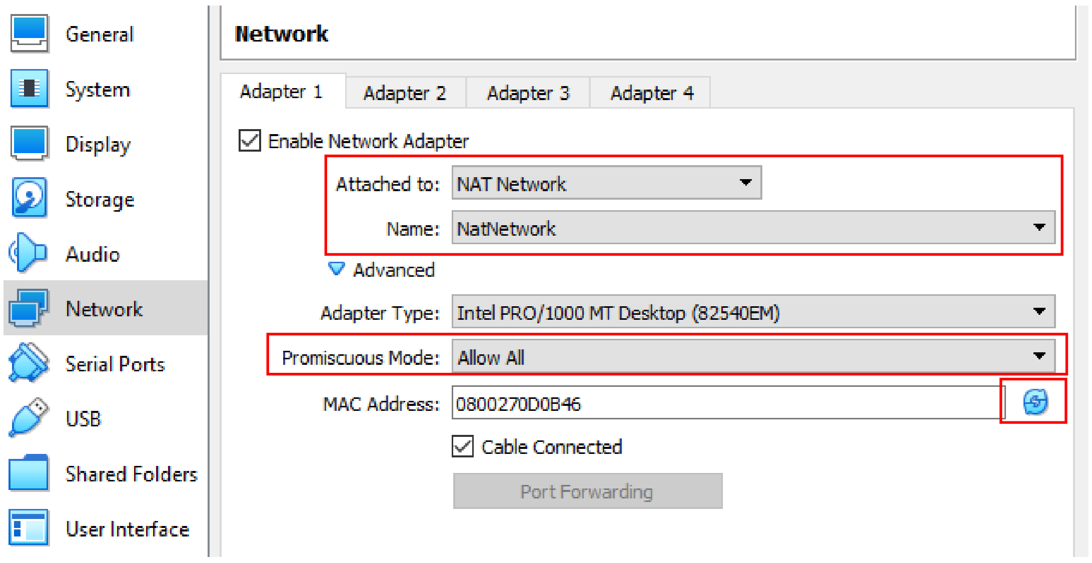
Step 9. Configure **Display → Screen** tab as follows AND click OK.



Video Memory: 128 MB
Monitor Count: 1
Scale Factor: All Monitors 100%
Graphics Controller: VMSVGA
Extended Features: ☐ Enable 3D Acceleration

Step 10. Add a new Nat Network adaptor. Go to the File menu -> Tools -> Network Manager -> Create.

Step 11. Go to VM Settings to configure **Network → Adapter 1** tab, then click **OK**



Network

☑ Enable Network Adapter

Attached to: NAT Network

Name: NatNetwork

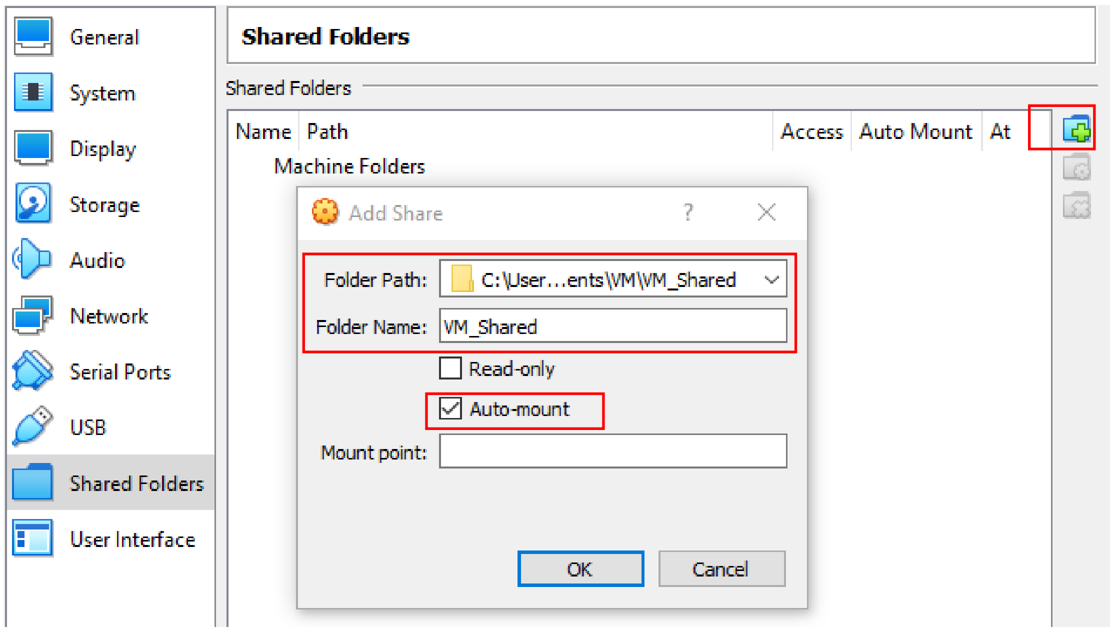
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Promiscuous Mode: Allow All

MAC Address: 0800270D0B46

☑ Cable Connected

Step 12. Configure **Shared Folders.** Choose a folder on your local computer, **Auto-mount**, then **OK**, and then **Ok** to complete the configuration.



Shared Folders

Folder Path: C:\User…ents\VM\VM_Shared

Folder Name: VM_Shared

☐ Read-only

☑ Auto-mount

Mount point:

Step 13. Start the VM (you may take a snapshot as well)

User: **SEED**

Password: **dees**

Step 14. Mount the shared folder. Open a terminal and use the following commands.

```
$ mkdir -p ~/Share
$ sudo mount -t vboxsf VM_Shared ~/Share
```

You should have completed the VM configuration.

Part 2 Introduction to Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

For more information about docker, please go to Docker overview.

You may use the frequent docker commands in all the labs.

```
$ docker-compose build                # Build the container image

$ docker-compose up                   # Start the container

$ docker-compose down                 # Shut down the container

$ docker ps                           # Show the running containers

$ docker network ls                   # List all the networks

$ docker container restart <id>       # Restart a container

$ docker container stop <id>          # Stop a container

$ docker container start <id>         # Start a container

$ docker exec -it <id> /bin/bash      # Run bash shell in a container

$ docker cp <file> <id>:<path>        # Copy a file from host to path

e.g., docker cp lab.py 02b:/home/     # 02b is a head of container id
```

Task 2.1.

Please try the above commands in Part 4, and for each command you execute, explain what it does and what you observe.

Part 3 Lab Environment - Introduction

The SEED VM has pre-build the lab environment and it will work with docker and containers to set up different labs. Simply speaking, the docker significantly simplifies the configuration effort of the VM environment. With the prepared script, we can quickly build multiple computers (through containers).

In this lab, we will create a LAN with three computers (Figure 1), Host A, Host B and Host M. The Host M will play as the attacker and Host A and B will be the victims.

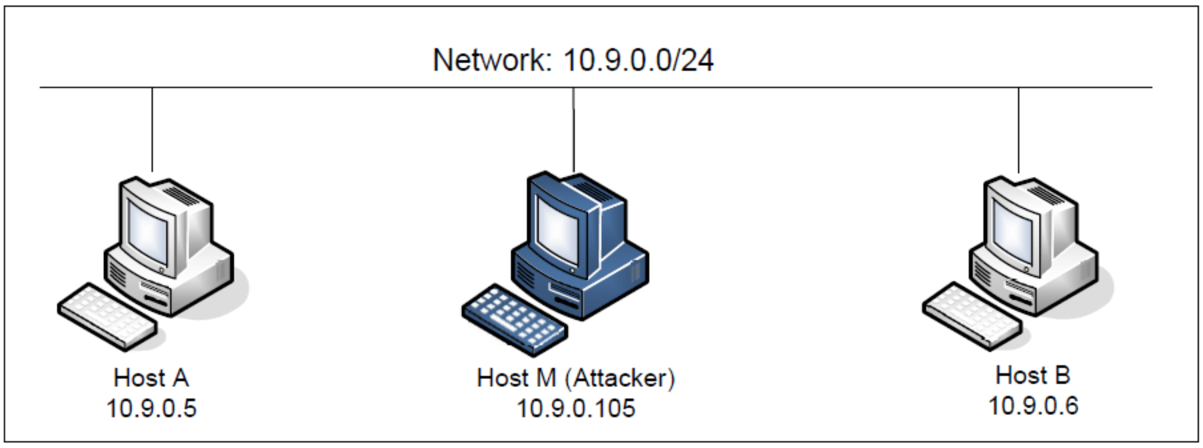


Figure 3.1. Lab environment setup

Note: Hosts A, B and M are all running in the VM. When the network is created, the VM will be automatically assigned an IP address 10.9.0.1. Therefore, the VM is also a part of the created LAN network.

Part 4 Setup and Test Containers

Task 2. Set up containers with docker.

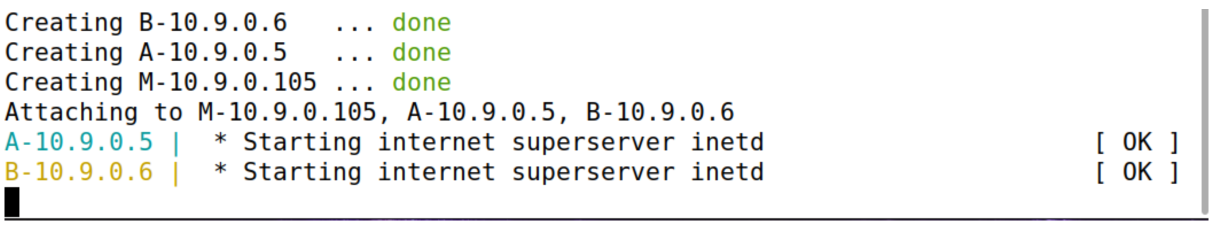
Step 1. Download the lab file from Moodle to your VM. (You may use the shared folder to transfer between the host (e.g., Windows) and the VM. DO NOT work (e.g., unzip, run) inside the shared folder.

Step 2. Unzip the lab file to **~/Documents/lab**

Step 3. Open a terminal and go to **~/Documents/lab** where contains the extracted **docker-compose.yml** file.

Step 4. Start the container image based on **docker-compose.yml** file.

```
$ docker-compose up
```

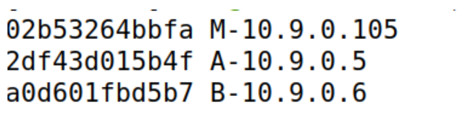


```
Creating B-10.9.0.6   ... done
Creating A-10.9.0.5   ... done
Creating M-10.9.0.105 ... done
Attaching to M-10.9.0.105, A-10.9.0.5, B-10.9.0.6
A-10.9.0.5 |  * Starting internet superserver inetd                    [ OK ]
B-10.9.0.6 |  * Starting internet superserver inetd                    [ OK ]
```

Step 5. Open a new terminal to obtain container IDs.

Note: The IDs will be different from each run. So, the following output is an example.

```
$ docker ps --format "{{.ID}} {{.Names}}"
```



```
02b53264bbfa M-10.9.0.105
2df43d015b4f A-10.9.0.5
a0d601fbd5b7 B-10.9.0.6
```

Step 6. Execute bash shell of a container.

```
$ docker exec -it <container id> /bin/bash
```

`<container id>` can be a head of the full id if it uniquely identifies the container. For example, the following command executes /bin/bash on the attacker's container M.

In this example, **02b5** is used for **02b53264bbfa**

```
$ docker exec -it 02b5 /bin/bash
```

We will control a container using `/bin/bash`

You can open multiple `/bin/bash` for the same container – in different terminals of the VM.

Part 5. Packet Sniffer and Tools

Packet sniffers (such as Wireshark) are applications that capture packets that are seen by a machine's network interface. Packet sniffers can be used to troubleshoot networks and also in application and network software development. Moreover, packet sniffers can be used in security attacks like eavesdropping.

Introduction

When a sniffer runs on a system, it grabs all the packets that come into and goes out of the Network Interface Card (NIC) of the machine on which the sniffer is installed. This means that, if the NIC is set to the promiscuous mode, then it will receive all the packets sent to the network if that network is connected by a hub (if the network is switched, this won't happen). A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a copy of packets that are sent/received from/by application and protocols executing on your machine. In cases where the adversary wants to sniff packets that are not intended to be sent to it (such as in switched networks), they may perform an ARP poisoning attack. This is an active attack which involves sending a spoofed Address Resolution Protocol message into the LAN, causing the other nodes to ”think” that the adversary is also a different one in that network. The adversary may use this attack to capture (or even alter) all packets within the LAN by impersonating the router, or they may choose to only target particular nodes.

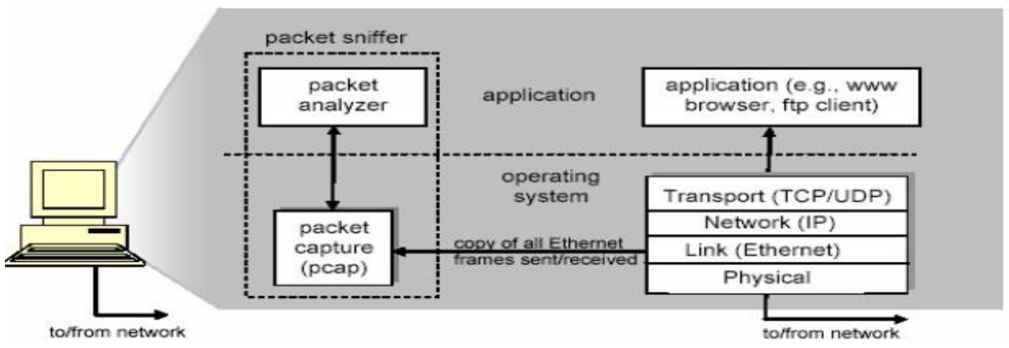


Figure 5.1: Packet Sniffer Structure

The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer, and consists of two parts. The packet capture library receives a copy of every link-layer frame that is sent from or received by your computer. Messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP are all eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Figure 1, the assumed physical media is an Ethernet network, and so all upper layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

The second component of a packet sniffer is the packet analyser, which displays the contents of all fields within a protocol message. In order to do so, the packet analyser must “understand” the structure of all messages exchanged by protocols. For example, the packet analyser understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. It understands the TCP segment structure, so it can extract the application protocol message contained in the TCP segment. Finally, if it understands the application protocol (typically if it is a common one such as HTTP) the packet analyser can extract some information from the application packet (such as reading the first bytes of an HTTP message, which must contain one of the strings "GET,""POST," or "HEAD").

Wireshark

Note: This information is extracted from Wireshark User's Guide. For more information go to www.wireshark.org/docs

Wireshark is a network packet analyser. A network packet analyser will capture network packets and display that packet data in as detailed a manner as possible.

You can think of a network packet analyser as a measuring device used to examine what's going on inside a network cable, just like a voltmeter is used by an electrician to examine what's going on inside an electric cable (but at a higher level, of course). In the past, such tools were either very expensive, proprietary, or both. However, open-source and freeware examples of these tools are now available. Wireshark is perhaps one of the best open-source packet analysers available today. Wireshark is an open-source software project, and is released under the GNU General Public License (GPL). You can freely use Wireshark on any number of computers you like, without worrying about license keys or fees or such. In addition, all source code is freely available under the GPL. Because of that, it is very easy for people to add new protocols to Wireshark, either as plugins, or built into the source, and they often do!

Using Wireshark – Understanding the Capture Menu

The Wireshark interface has five major components:

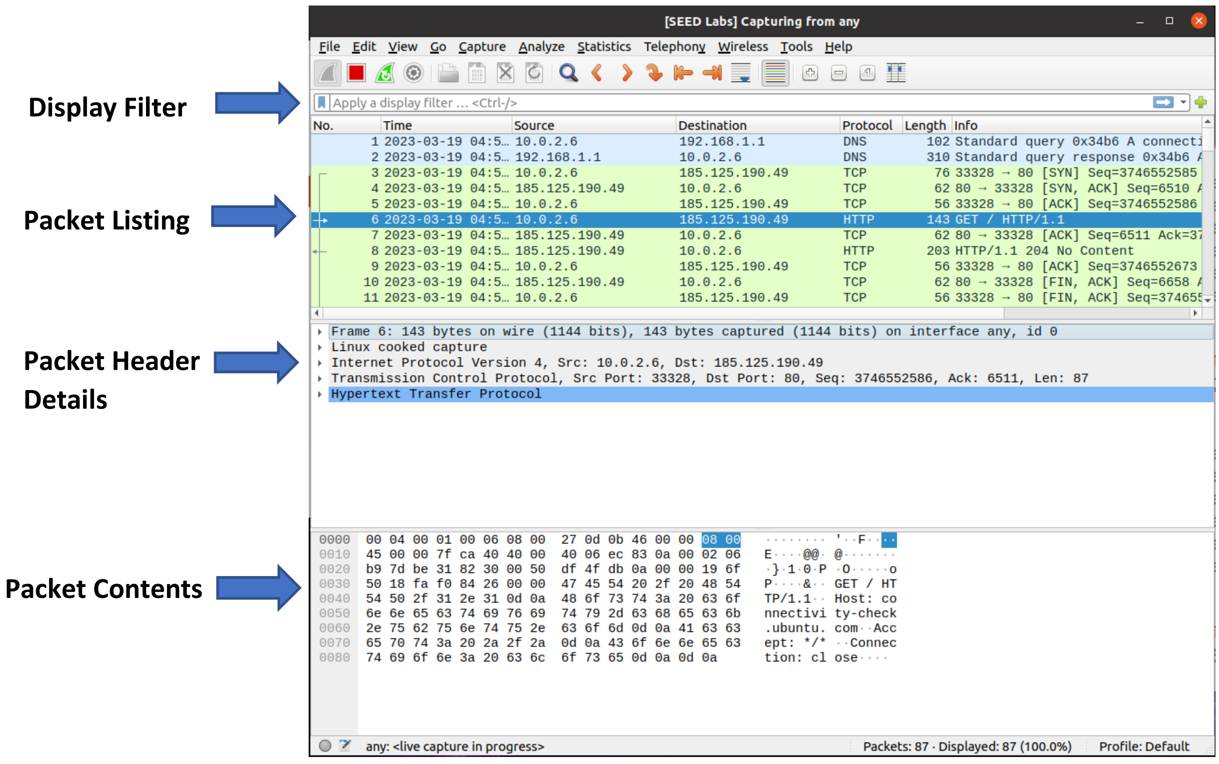
- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data and exit the Wireshark application. The Capture menu allows you to begin packet capture.
- The **packet-listing pane** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is not a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.
- The **packet-header details pane** provides details about the packet selected (highlighted) in the packet listing window. (To select a packet in the packet listing window, place the cursor over the packet's one-line summary in the packet listing window and click with the left mouse

button.). These details include information about the Ethernet frame and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the right-pointing or down-pointing arrowhead to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.

- The **packet-contents pane** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the Wireshark graphical user interface, is the **packet display filter field**, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows).

Capturing Packets Using Wireshark

Figure 2 shows the page of Wireshark once some packets have been captured.


Display Filter
Packet Listing
Packet Header Details
Packet Contents
[SEED Labs] Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time               Source           Destination      Protocol Length Info
   1 2023-03-19 04:5… 10.0.2.6         192.168.1.1      DNS       102 Standard query 0x34b6 A connecti
   2 2023-03-19 04:5… 192.168.1.1      10.0.2.6         DNS       310 Standard query response 0x34b6 A
   3 2023-03-19 04:5… 10.0.2.6         185.125.190.49   TCP        76 33328 → 80 [SYN] Seq=3746552585
   4 2023-03-19 04:5… 185.125.190.49   10.0.2.6         TCP        62 80 → 33328 [SYN, ACK] Seq=6510 A
   5 2023-03-19 04:5… 10.0.2.6         185.125.190.49   TCP        56 33328 → 80 [ACK] Seq=3746552586
   6 2023-03-19 04:5… 10.0.2.6         185.125.190.49   HTTP      143 GET / HTTP/1.1
   7 2023-03-19 04:5… 185.125.190.49   10.0.2.6         TCP        62 80 → 33328 [ACK] Seq=6511 Ack=37
   8 2023-03-19 04:5… 185.125.190.49   10.0.2.6         HTTP      203 HTTP/1.1 204 No Content
   9 2023-03-19 04:5… 10.0.2.6         185.125.190.49   TCP        56 33328 → 80 [ACK] Seq=3746552673
  10 2023-03-19 04:5… 185.125.190.49   10.0.2.6         TCP        62 80 → 33328 [FIN, ACK] Seq=6658 A
  11 2023-03-19 04:5… 10.0.2.6         185.125.190.49   TCP        56 33328 → 80 [FIN, ACK] Seq=374655

Frame 6: 143 bytes on wire (1144 bits), 143 bytes captured (1144 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.6, Dst: 185.125.190.49
Transmission Control Protocol, Src Port: 33328, Dst Port: 80, Seq: 3746552586, Ack: 6511, Len: 87
Hypertext Transfer Protocol

0000  00 04 00 01 00 06 08 00  27 0d 0b 46 00 00 08 00   ········ '··F····
0010  45 00 00 7f ca 40 40 00  40 06 ec 83 0a 00 02 06   E····@@· @·······
0020  b9 7d be 31 82 30 00 50  df 4f db 0a 00 00 19 6f   ·}·1·0·P ·O·····o
0030  50 18 fa f0 84 26 00 00  47 45 54 20 2f 20 48 54   P····&·· GET / HT
0040  54 50 2f 31 2e 31 0d 0a  48 6f 73 74 3a 20 63 6f   TP/1.1·· Host: co
0050  6e 6e 65 63 74 69 76 69  74 79 2d 63 68 65 63 6b   nnectivi ty-check
0060  2e 75 62 75 6e 74 75 2e  63 6f 6d 0d 0a 41 63 63   .ubuntu. com··Acc
0070  65 70 74 3a 20 2a 2f 2a  0d 0a 43 6f 6e 6e 65 63   ept: */* ··Connec
0080  74 69 6f 6e 3a 20 63 6c  6f 73 65 0d 0a 0d 0a      tion: cl ose····

any: <live capture in progress>    Packets: 87 · Displayed: 87 (100.0%)    Profile: Default
```

Figure 5.2. Wireshark Graphical User Interface.

The **packet listing pane** shows all the network packets that have been captured while you have been running Wireshark in capture mode – one packet per line. When you click on a packet, that packet is displayed in different ways in the lower two panes.

The **bottom packet contents** pane shows the packet in raw format as it was captured. This information is very low level and mostly not very interesting. You can see three vertical sections. The first column gives you the offset of the octet. The next 16 two-digit columns show you the octet contents in hexadecimal (that is four-bit numbers 0- 15 are represented by 0-9, then A-F). You should get used to hexadecimal numbers for communications work since it represents four-bit entities neatly and we will be using hexadecimal more in future labs. The next section shows the same data as ASCII characters. Wireshark does the best it can displaying the characters, but not all 8-bit patterns (hexadecimal codes) correspond to printable characters, especially codes < 0x20. Non-printable characters are simply represented by a dot '.'.

The **middle packet header details pane** is where you will look most. This pane displays Wireshark's interpretation of the packet. The lines here represent different layers of the network stack, albeit upside down to how we usually think of the network stack. The top line represents the physical layer. The second line the data-link layer where you will mainly see Ethernet packets, but it could be another data-link protocol. The third line is the network layer, which is mainly IP. The fourth line is the transport layer, normally TCP, but it could be UDP. If Wireshark finds an application protocol like HTTP that it can interpret, it will also give you information about that on the fifth line.

In the toolbar, you will find some handy shortcuts for starting and stopping captures. Since a capture can capture many packets, the filter text box allows you to enter criteria to display relevant packets.

Depending on how you would like to capture the packet, there are several ways to go about it.

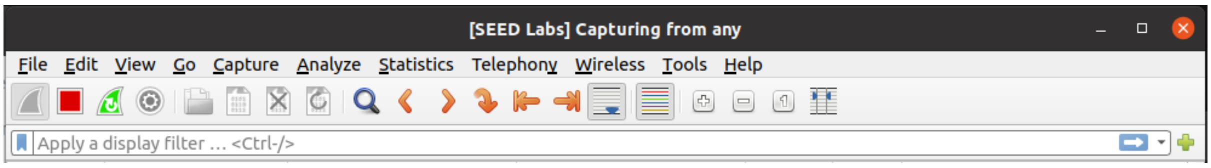
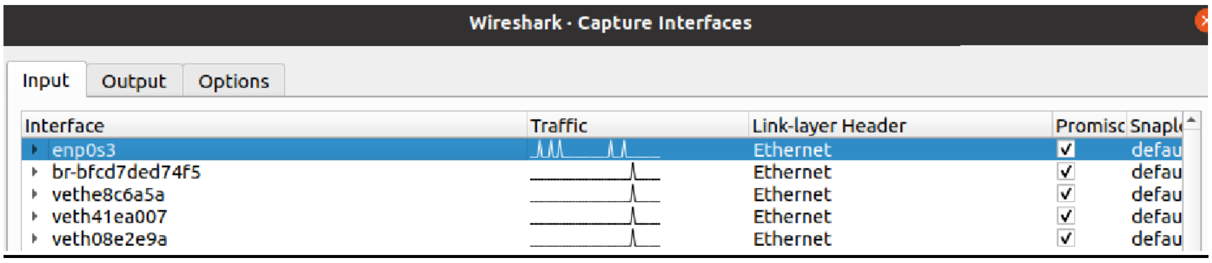

[SEED Labs] Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

Figure 5.3: Wireshark Menu Bar

In Figure 3, choose the "Options" sub-menu in the "Capture" drop-down listing. Once you choose this, you will have the listing of various network interfaces on your machine that can be seen from your computer's NIC(s). An example is shown in Figure 4, where we have five interfaces.



Interface	Traffic	Link-layer Header	Promisc	Snapl
enp0s3		Ethernet	✓	defau
br-bfcd7ded74f5		Ethernet	✓	defau
vethe8c6a5a		Ethernet	✓	defau
veth41ea007		Ethernet	✓	defau
veth08e2e9a		Ethernet	✓	defau

Figure 5.4: Interfaces

Task 5.1. Capture packets using Wireshark in the VM.

Step 1. In the VM, open a terminal and execute `/bin/bash` in Host A.

Step 2. In the VM, open the Wireshark from the Desktop of the VM. Choose the "**any**" interface and start capturing.

Step 3. In Host A, **ping** from Host A to Host B and see if the network is correctly configured.

Step 4. Stop Wireshark capturing. Explain the packets you captured.

- Find **ICMP** records that can show the Ping process from Host A to Host B.
- Look at the records and find Host A and B source MAC addresses, respectively.
- Use the **ifconfig** command to find the MAC addresses in Host A and B, respectively. Check if they are the same as your above interpretation.

tcpdump

In a command-line environment, tcpdump is a powerful packet analyser. It is a popular alternative tool of Wireshark. For more information, please see tcpdump documentation.

Task 5.2. Capture packets using tcpdump in a container

Step 1. In the VM, open a terminal and execute `/bin/bash` in Host B.

Step 2. In Host B, use **tcpdump** tool to capture all packets through the **eth0** interface.

```
$ tcpdump -i eth0 -n
```

Step 3. In Host A, **Ping** from Host A to Host B. Observe the records in Host B.

Note: Wireshark (in Task 5.1) captures all packets of the entire network (VM, Host A, B and M), because it runs in the VM. However, the **tcpdump** (in Task 5.2) only captures the packets through Host B.