CS325 Computer Security

Project 2 Part – Unsecure TCP/IP Communication with Sockets in Python

Posted: Monday 24th October.

Due: **11:59pm on Monday the 7th of November**

Introduction

For this project, you are going to build on the concepts learned in the previous project to build an application that permits unsecured communication over a network using the TCP/IP stack in Python using the Socket API.

What is TCP/IP

The **Transmission Control Protocol** (TCP) and the **Internet Protocol** (IP) are two key protocols in the **TCP/IP protocol stack**. These two protocols form a suite of communication protocols used to interconnect computing devices over a network. **TCP** defines how applications can create channels of communication across a network between application layer processes. **IP** defines how to address and route each packet across a network though devices and links to make sure it reaches its intended destination. Table 1 below shows how these form layers four and five of the five-layer model.

Table 1. The Five-Layer networking model

Layer #	Layer Name	Protocol	Protocol Data Unit	Addressing
5	Application	HTTP, SMTP, FTP etc..	Messages	n/a
4	Transport	TCP/UDP	Segments/Datagrams	Port numbers
3	Network	IP	Packets	IP Address
2	Data Link	Ethernet, Wi-Fi	Frames	Mac Address
1	Physical	100 Base T, 802.3ab, 802.11	Bits	n/a

When an application has a message to send over a network using **TCP**, it is done using **sockets**. A socket is a notional end point of a unique, dedicated communication link between two programs running over a network. A client will send a message over a socket with outgoing port number, and a recipient or server will receive a message from a socket with an inbound port number. The port numbers at each end will likely differ. Figure 1 below illustrates this.

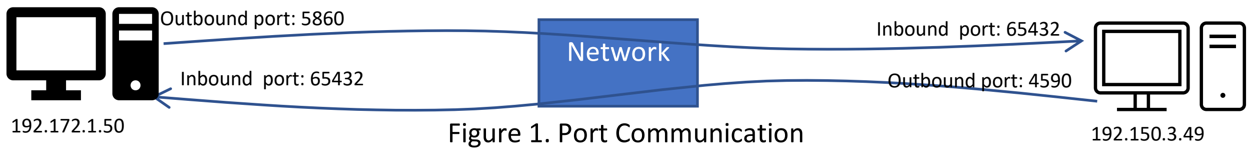


Figure 1. Port Communication

Each inbound and outbound **socket** has a unique **port number** allocated to it, which is then associated with an **IP address** to uniquely identify that computing device (IP Address) and that specific connection on that device (a sockets port number). A modern operating system will utilize many ports concurrently for communication purposes in any application that utilizes a connection to a network and the wider internet. **TCP** supports a range of port numbers from 0 to 65,535. Typically, the security features of an operating system, i.e. a firewall, will close (prevent the use of) inbound ports, as they can represent a security risk. To use an inbound port, it often has to be opened by configuring a firewall. Outbound ports are less dangerous and are generally not closed. Typically, port numbers less than 1024 are special (sometimes called privileged ports) in that they are reserved for standardized protocols implemented by an operating system, its software components, or user installed applications for common tasks. For example, port 80 is used by the **hypertext transfer protocol** (HTTP); port 22 is used by **Secure Shell** (SSH). Port numbers 1024 and above can be used by applications developers. However, even some of these port numbers can have standardized uses. Therefore, it is best to use a high port number because the chances of it being used by something else are less. Wikipedia maintains a list of known **TCP** port numbers: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

An **IP** address is a numeric label used to uniquely identify a computer on a network. An **IP v 4** address is 31 bits and is displayed as four numbers each from 0 to 255 separated by periods, such as: such as 192.172.52.4. This is called the dotted decimal notation. IP v 6 addresses are 64 bit, which creates a larger address space, but we will not get into that here. An **IP** address is either assigned to a computer manually by an administrator or automatically by a **Dynamic Host Configuration Protocol (DHCP)** server. Commonly, home internet routers incorporate a DHCP server because it is important that each device connected to that router is allocated a unique IP address. A router is responsible for forming a network of computing devices, permitting communication between the devices on that network, and for allowing each device to communicate with the network it is connected to, which is typically the network of your internet service provider. Remember that the internet is a network of networks. Therefore, it is also important that each network interface of the router itself has a unique IP address for the network it is connected to, which is typically a network defined by an internet service provider. It is also important to note that IP addresses cannot be used for ***authentication*** because the address of the sender can easily be spoofed. This is because IPv4 was defined in more innocent times.

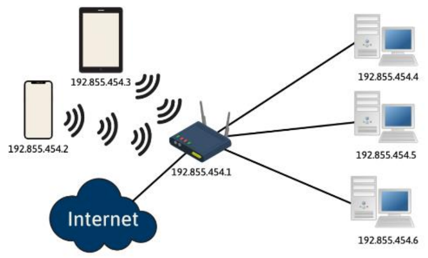


Figure 2. Network Communication

In this project, for simplicity's sake, we are only concerned about communication between devices on a single network. This can still pose the same security risks as the wider web: for example, a malicious employee on the same network, or a hacker in a coffee bar sniffing the packets of others on the public internet the venue provides.

TCP/IP Communication:

Python's **Socket API** facilitates network communication using TCP/IP. This API follows the conventions first used in the 1983 Berkley Software Distribution of Unix. These conventions have also been widely adopted in most mainstream operating systems (quite literally the code was copied). The diagram below illustrates how these conventions are used in communication between two computers using the terms **client** and **server** (see figure 3). Although the figure depicts a client and a server, the diagram holds true for peer-to-peer connections too. Also, there could be more than one client. In the case of peer-to-peer connections, one must assume the role of the server and the other the client, and these roles may alternate depending on who is initiating the communication. The sockets do not remain open but are open and used while an exchange of messages occurs. The sockets are then closed. If another message is to be sent, the process starts over again.

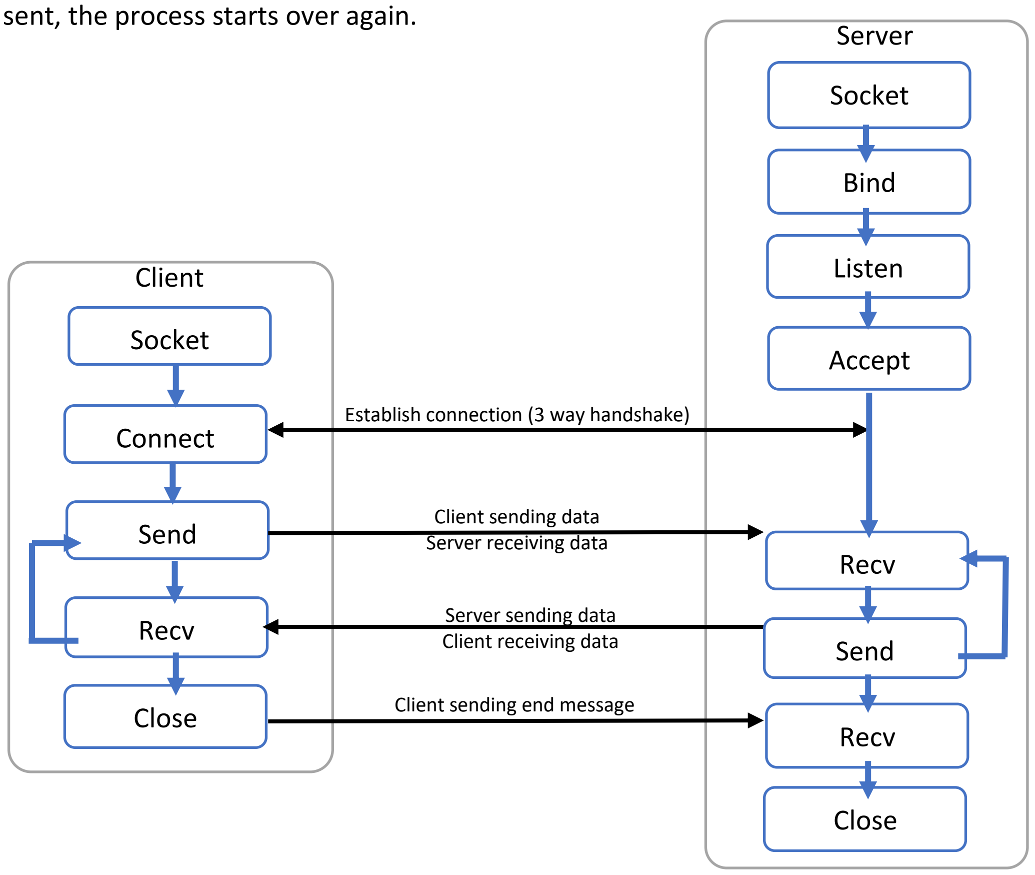


Figure 3: Communication between two computers

The functions as shown in the above diagram are present in Python's **socket API** and are described below.

- **socket()** creates a new socket of a specified type identified by a constant integer, and allocates resources to it.
- **bind((ip, port))** is used on the server side to associate the socket with a particular **IP** address (or all **IP** addresses) and a port number. It receives as a parameter a tuple containing an **IP** address and a port number
- **listen()** is used on the server side and causes a bound socket to go into a listening state.
- **accept()** is a blocking function which will wait until it accepts an incoming attempt to create the TCP connection from a client computer.
- **send() / sendTo()** are functions for sending a message over the **TCP** connection, typically as bytes.
- **recv() recvfrom()** are functions for receiving a message over the **TCP** connection, typically as bytes.
- **close()** causes the operating system to release resources allocated to a socket, but also causes a **client** to send an **end** message to a server so it can choose to close its socket if needed.

Other Prominent functions:

- **gethostbyname()** and **gethostbyaddr()** are used to resolve **IPv4** addresses and host names.
- **getaddrinfo() getnameinfo()** is used to receive host names and addresses.
- **select()** is used to suspend, waiting for one or more of a provided list of sockets to be ready to read, ready to write, or that have errors.
- **poll()** is used to check on the state of a socket in a set of sockets. The set can be tested to see if any socket can be written to, read from, or if an error occurred.
- **getsockopt()** is used to retrieve the current value of a particular socket option.
- **setsockopt()** is used to set a particular socket option for the specified socket.

More detail and information on other functions can be found in the Python documentation for the **socket** API: https://docs.python.org/3/library/socket.html#functions

Pre-Preparation

We are going to be communicating between a host and guest operating system to conveniently but realistically demonstrate communication between two computers on a network. Alternatively, you can make this work by communicating between two guest operating systems. This overcomes many of the complexities some of you will face in getting two computers to communicate over an enterprise network such as WCU's one. For security reasons, modern operating systems close inbound port numbers, only opening those port numbers used for specific purposes. Therefore, we need to open inbound ports on both the host and guest operating system, or any other operating system we wish to communicate with (such as another laptop).

The outbound port number we will use to initiate communication is: 65432

Follow the guidelines below on how to open ports for Windows, Linux and Mac OS. If you are working, as expected, with a virtual machine you will need to perform this twice, once for the guest and once for the host, or otherwise on each machine you intend to run your program on.

Opening Ports on Windows:

This guide is for Windows 10, but the basic principle should be the same on Windows 11. If not, I am sure a simple web search will answer this question.

Press the Windows key and type **Windows Defender Firewall**. When you see the application icon of the same name appear, click it. On the left of this window, you will find **Advanced Settings**. Click on this and the firewall rules window will open.

Click on the **inbound rules** link at the top left. Then, to the right, click on **new rule.** On the window that appears **select port** and then next. On the next screen **select TCP** and **Specific Local Ports** and in the text field enter **65432** and click next. On the next screen select **Allow the Connection** and click next. On the following window, ensure **Domain, Private** and **public** are checked and click next. Finally, provide a memorable and contextual name and description for this rule and click finish.

Opening Ports on Linux:

On Ubuntu and Debian Linux opening ports is much easier, as it can be done in the command line. Open a terminal and using super user privileges type the following, to open port 65432 to TCP traffic.

```
(on the Labtainers Virtual Machine sudo
is not needed, but likely this will be
needed on a real Linux install)
```

```
firewall-cmd --add-port 65432/tcp
```

Opening Ports on Mac OS:

Open the system preferences and select **security and privacy**. This icon should look like a house. Once open, click on Firewall. By default, Mac OS has the firewall turned off. If that is the case, no further intervention is needed. Mac OS secures its port on a per application basis rather than a per port basis. If the firewall is enabled, perform the following steps to open an inbound port.

- Open System Preferences > Security & Privacy > Firewall > Firewall Options.
- Click Add.
- Choose an application from the Applications folder and click Add.
- Ensure that the option next to the application is set to Allow incoming connections.
- Click OK.

However, as we are building our own application, it might be easier to temporarily turn the firewall off.

Adjust Virtual Machine Network Adapter Settings:

By default, the network settings of a virtual machine are set to Network Address Translation (NAT). This means your virtual machine can share the internet connection of its host OS but true bi-directional network communication is difficult. There are two options Host to guest communication, or two virtual machines for guest to guest communication.

Host to Guest Communication:

With the virtual machine turned off, open the **Virtual Box Manager** application and click on the settings for your **Labtainer Virtual Machine.** Click on the **network** section (to the left) and select the tab for **Adapter 1**. Make sure the adapter is enabled, and that **'Attached to:'** is set to Bridged Adapter. This will enable the guest and host operating systems to communicate as if they were two computers on the same network. If you are on WCUs network, this may not work due to port blocking by WCUs network administrators. If this does not work consider using a guest to guest solution.

Guear to Guest Communication:

Guest to guest communication is where a host OS maintains two virtual machines which communicate with each other over a virtual network. This isolates you from any of the complexities of WCUs network.

In this solution first prepare one virtual machine with the necessary software and open ports. Next, with the virtual machine turned off, in the Virtual Box manager application right click and clone the Lataniers virtual machine as a **full clone** giving the new clone a distinctive name. In the **network** settings of Virtual Box for both VMs ensure adapter 1 is enabled and set to 'Host-only Adapter'. To more easily tell them apart, set the background image of both instances to a difference image. Now each instance will have its own IP address and can communicate to the other as if they were two computers on the same network.

Discover the IP Address of Linux Computer/ Virtual Machine

Now that the VM is using a bridged connection; install the networking tools to see what network adapters are available. To do this, use the following command. If you are prompted for a password on the Labtainers virtual machine, it is password123.

```
    sudo apt install net-tools
```

Once the installation is complete, you should be able to type: **ifconfig**. This will list all the network adapters in the virtual machine. There will be several; look for the one labeled **enp0s3 (or similar)**. This will output the following information:

```
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.20  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 ea80::6ac7:8e05:f6da:ba2b  prefixlen 64  scopeid 0x20<link>
        ether 08:00:a7:b9:a9:9b  txqueuelen 1000  (Ethernet)
        RX packets 26  bytes 5518 (5.5 KB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 73  bytes 9012 (9.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

In this example, the **IPv4 Address** is shown on the second line next to **inet** as 192.168.1.20. The **IPv6 Address** is shown next to **inet** 6 on line 3, and the **MAC address** is shown on line 4 next to **ether**. It is the **IPv4 Address** which is important. Take a note of this address, as it will be important in this project.

Discover the IP Address of a Windows Host:

On a Windows machine open the command prompt. Press the win-key and type **cmd** to find it. Once open, type **ipconfig**. This command will list all of the adapters on your computer. Find the entries marked **Ethernet Adapter** (for wired connections) or **Wireless Lan Adapter** (for wireless connections) and find the **IPv4** addresses.

Discover the IP Address of a Mac OS Host:

Open the **system preferences** then select the **network settings**. By clicking on the list of adapters on the left, you should find your network address on the right. For Wi-Fi adapters it is written quite small, so you will have to look closely under the status message to see the **IP address**.

Below write the **IP addresses** of your host and guest operating systems.

Host IP Address:

Guest IP Address:

Other computer IP Address (Optional):

Now that we are set up, you are ready to begin learning how to communicate over a network in Python.

Network Communication in Python

Below are demonstrations of simple code to send and receive messages in Python. What you see here is an example of inter process communication (IPC). These examples align closely with Figure 3 discussed previously. Note these are example code fragments that should in use be placed inside functions. Read this section carefully, as you will be utilizing this code in a program.

Sending Messages (the client):

The basic sending of messages is quite a simple affair. The code below is explained underneath it.

```
1.   import socket
2.
3.   SERVER = "192.168.1.15" #IP Address of the recipient.
4.   PORT = 65432  # The port used by the server
5.
6.   with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
7.       s.settimeout(30.0)#Time out of 30 seconds if not received
8.       s.connect((SERVER, PORT))
9.       s.settimeout(None)#Always set timeout to none before sending.
10.      s.sendall(b"Hello, world")
11.      data = s.recv(1024)
```

On line 6 the socket is created as a stream socket (socket.SOCK_STEAM) using the **IPv4** Protocol (socket.AF_INET) and this referenced by variable **S**. On line 8 the socket's connect function is supplied with a tuple containing the address of the recipient (the server) and the port address on which the recipient is listening for a connection. On line 10 the message is sent as bytes. Noting that bytes are used is important, because if you have a string it should be converted to bytes using the **encode** function. Line 11 contains a thread blocking function that waits for the recipient to send data back. Once this is received, the program execution continues. In this instance, for brevity the data returned on line 11 is not printed or used, but this can easily be printed or otherwise used to check the message was received correctly. Note that if the message is not received within 30 seconds, a time-out occurs. To this end, note the use of timeouts on lines 7 and 9.

Note here the use of Python's **with** command. With is a kind of Python short hand; it is the same as writing **try** and **finally** blocks (or try and except depending on what is used with it). The example below demonstrates what the **with** command in this context translates to.

```
s= None
try:
     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((HOST, PORT))
    s.sendall(b"Hello, world")
    data = s.recv(1024)
finally:
    if s != None:
         s.close()

print(f"Received {data!r}")
```

Further reading on the with command can be found in the official Python documentation:
https://docs.python.org/3/reference/compound_stmts.html#the-with-statement

Receiving messages (the server):

To get you started, lets first look at a simple server. Below is a simple example program for the host:

```
1.  VM_IP = "192.168.1.15"
2.  ALL_IP = ""
3.  PORT = 65432  # Port to listen on (non-privileged ports are > 1023)
4.
5.  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
6.      s.bind((ALL_IP, PORT)) #Inner brackets define a tuple
7.      s.listen()
8.      conn, addr = s.accept()
9.      with conn:
10.         print(f"Connected by {addr}")
11.         exit = False
12.         while not exit:
13.             data = conn.recv(1024)
14.             if not data:
15.                 exit = True
16.             else:
17.                 print(data.decode(),end="#")
18.             conn.sendall(data)
```

On line 6 the socket is created as a stream socket (socket.SOCK_STEAM) using the IPv4 Protocol (socket.AF_INET) and this is referenced by variable S. On line 7 the socket is bound using a **tuple** containing the **IP address** and **port number** to be listened to. If the **IP address** supplied is an empty string, it will listen to all ports. Line 8 puts the socket into a listening state. Optionally, this command can also take a parameter specifying the maximum number of connections for supporting multiple clients; we shall not do this and depend on the default value, which varies by OS. Line 9 is a **blocking function**[1] that will wait for the reception of data on the specified **IP address** and **port** number.

When a connection is established the **accept** function will return an object and a string: **conn** represents the connection as an instance of socket, and **addr** is a tuple containing the **IP address** of the **client** and a **port number**. You will notice that this port number is not 65432. It is important to understand that this occurs because this is the **outbound port** number of the client. It is distinct from the inbound socket **65432** that the server is using to listen on.

Once the connection is established, the program enters a loop over blocking calls to the function **conn.recv()** which reads and returns the bytes sent by the client. The amount of data that can be received is limited by the buffer size parameter (bytes) and is therefore expected to be a relatively small power of two, for example 4096. If a longer message is required, it should be split up and sent over multiple messages, each sent individually. In this example, the loop breaks when it receives an empty string. Notice too how online the received data is being sent back to the client. In this example it is used so that the client's own **s.recv(1024)** function waits for a receipt. Other approaches are possible, such as sending an ending sequence of characters. In this example, timeouts are not used, but they could have been used, so that if no message was received after a specified time period, the connection would time out.

[1] A blocking function does not proceed to the next statement until its work is completed.

Task 1: Unsecure Two-Way Communication

The first task you are going to complete is to achieve 2-way communication between two computers.

Begin by creating an empty directory called project 2. Then initialize a git repository. Create a branch for **project 2**. Write and commit your code to a python module called **communication.py** on that new branch.

You will write a Python program that uses the **socket API** so that any computer running that program can choose to act as the client or server. The server will listen for communication from the host and the client will send a message to the host.

The program must have a **send** function, a **receive** function, and a **menu** function. Use port 65432 as the outbound port number. Once developed, test your solution by running it from the virtual machine (the guest OS) and on your desktop/laptop computer (the host OS). Once fully successful, use Git to **tag** your most recent commit with the label: **Task1.**

You are expected to use helper methods liberally for the processes of taking various inputs and validating them, and anything else that maximizes reuse. Each function should be **S**mall, do **O**ne thing, have **F**ew arguments, and aim for **A**bstraction level consistency (SOFA).

Menu Function:

The menu function should give the user three options: 1) send message, 2) receive message and 0) exit. If the user enters a value other than 0, 1 or 2, an error message should be presented.

```
===The Python Communicator===
1) send message
2) receive message
0) exit
Enter option: exit
Error, invalid input
Enter option: 0

Goodbye!
```

Send function:

The user is prompted to enter in a message of a maximum of 4096 characters. Once this has been achieved, they are asked to enter the IP address of a recipient. The program should send the message but handle any errors. If not received within 30 seconds, then it should time out gracefully.

Remember too that the sent information should be in the form of bytes, so you will need to convert any strings to bytes before sending them.

Once the recipient has received the communication, they should send back a final message “#<<END>>#” to confirm that the message was received. Use this to report “Message sent successfully” if this string was received or “Message receipt failed”, if the sending timed out or the returned message did not contain the expected string.

In sending information, be sure to handle errors; for example: setting reasonable timeouts and catching specific exceptions. For the purpose of catching specific exceptions, you may find not using the **with** syntax easier.

Example of Expected Successful Interaction:

```
Enter Message (max 4096 characters): Would you like green eggs and ham?

Enter Recipient IP: 192.168.sam

Error not a valid IPv4 Address:

Enter Recipient IP: 192.168.2

Error not a valid IPv4:

Enter Recipient IP: 192.168.1.52

Sending.

Message sent successfully.
```

Example of Expected Unsuccessful Interaction where IP address be resolved, or no response received:

```
Enter Message (max 4096 characters): I do not like green eggs and ham.

Enter Recipient IP: 192.168.0.99

Message sending error. Message not sent. (After timeout of 30 seconds.)
```

Receive function:

The receive function should simply listen for communication to be initiated via port 65432. For example:

```
Waiting for message on port 65432

Connected by (192.168.1.20)

Message:

I do not like green eggs and ham.

End of message.
```

To show the sender the message has been received and to complete the communication, send back the message “#<<END>>#” as bytes.

Introduction to Wireshark

Wireshark is a free, open source software tool for analyzing the packets on a network by monitoring the network traffic through a network interface. Since its release as Etherial in 1998, it has become the most popular network monitoring tool used in industry, education and hacking because of its versatility in monitoring Ethernet, wi-fi and Bluetooth traffic, its support for networking standards, and its usability. Wireshark is also available for Windows, Linux and MacOS.

Installation:

We will begin by installing Wireshark on the Ubuntu virtual machine. Open a terminal and type the following to update the package manager and then to install Wireshark. Remember the password is **password123**

```
sudo apt update

sudo apt install wireshark
```

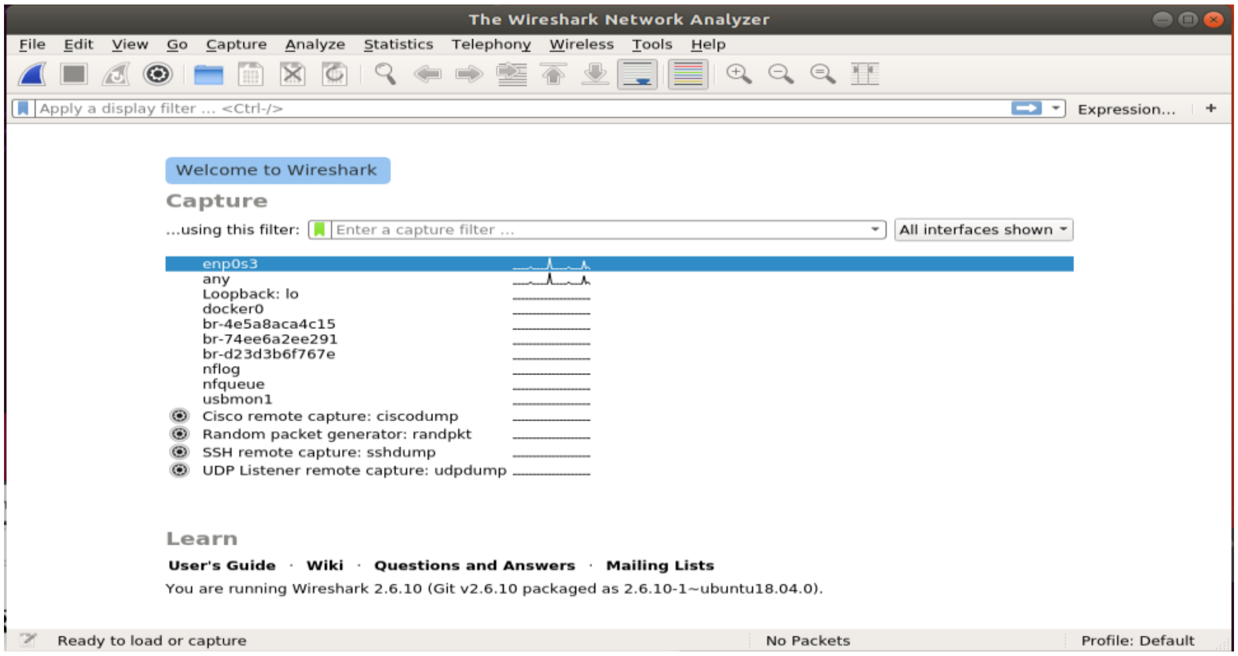
Using Wireshark:

If you open Wireshark from the Launcher, you will not have sufficient privileges to access the low level network data. Open a terminal and open Wireshark as a super user with the following:

```
sudo wireshark
```

When Wireshark opens, you should find a list of network interfaces. You need to find the network interface that corresponds with your network adapter. On the virtual machine this should be **enp0s3** as shown below in figure 4. You may only have to do this once the first time.

To begin capturing network traffic, with the adapter selected, click on the shark fin in the tool bar's left (or double click the adapter). You should see the app change to a list of network activity that shows what is happening on that adapter as shown in figure 5.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
Welcome to Wireshark
Capture
…using this filter:
Enter a capture filter …
All interfaces shown

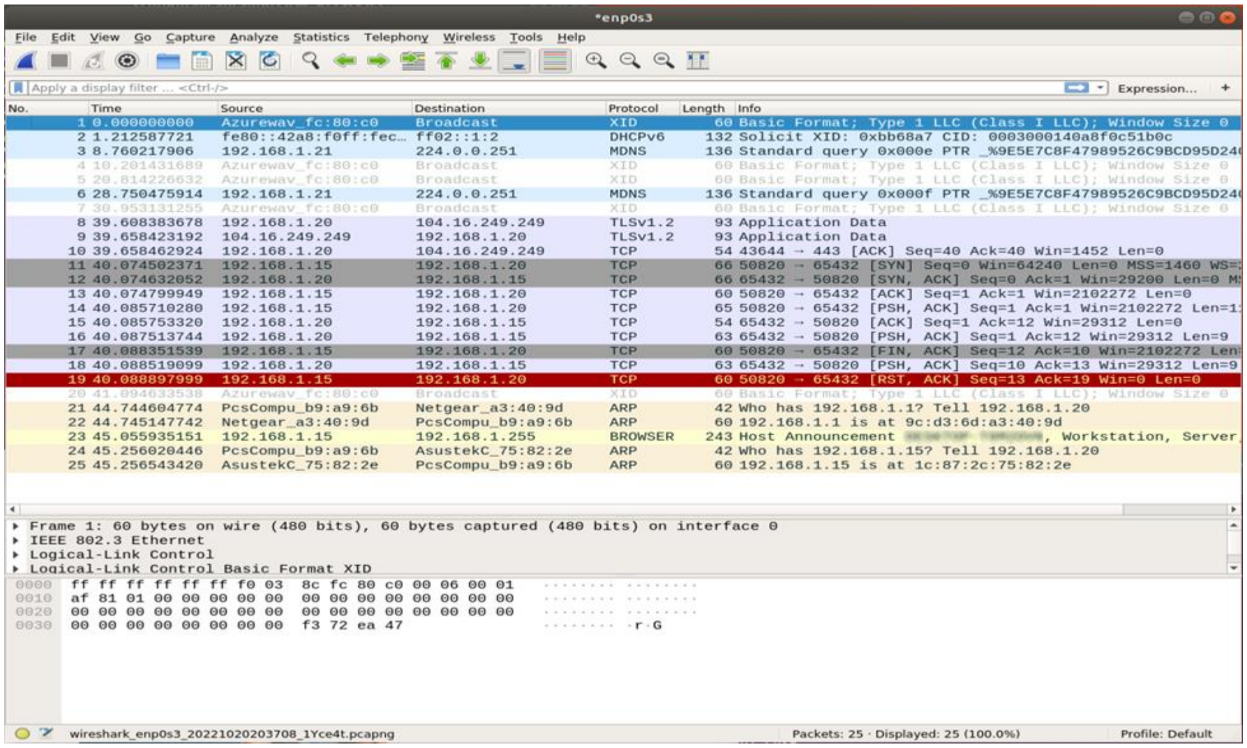
```
enp0s3
any
Loopback: lo
docker0
br-4e5a8aca4c15
br-74ee6a2ee291
br-d23d3b6f767e
nflog
nfqueue
usbmon1
Cisco remote capture: ciscodump
Random packet generator: randpkt
SSH remote capture: sshdump
UDP Listener remote capture: udpdump
```

Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists

You are running Wireshark 2.6.10 (Git v2.6.10 packaged as 2.6.10-1~ubuntu18.04.0).

Ready to load or capture   No Packets   Profile: Default

Figure 4. Wireshark Upon Starting.


*enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

```
No.   Time          Source                  Destination          Protocol  Length Info
    1 0.000000000   Azurewav_fc:80:c0       Broadcast            XID          60 Basic Format; Type 1 LLC (Class I LLC); Window Size 0
    2 1.212587721   fe80::42a8:f0ff:fec…    ff02::1:2            DHCPv6      132 Solicit XID: 0xbb68a7 CID: 0003000140a8f0c51b0c
    3 8.760217906   192.168.1.21            224.0.0.251          MDNS        136 Standard query 0x000e PTR _%9E5E7C8F47989526C9BCD95D24
    4 10.201431689  Azurewav_fc:80:c0       Broadcast            XID          60 Basic Format; Type 1 LLC (Class I LLC); Window Size 0
    5 20.814226632  Azurewav_fc:80:c0       Broadcast            XID          60 Basic Format; Type 1 LLC (Class I LLC); Window Size 0
    6 28.750475914  192.168.1.21            224.0.0.251          MDNS        136 Standard query 0x000f PTR _%9E5E7C8F47989526C9BCD95D24
    7 30.953131255  Azurewav_fc:80:c0       Broadcast            XID          60 Basic Format; Type 1 LLC (Class I LLC); Window Size 0
    8 39.608383678  192.168.1.20            104.16.249.249       TLSv1.2      93 Application Data
    9 39.658423192  104.16.249.249          192.168.1.20         TLSv1.2      93 Application Data
   10 39.658462924  192.168.1.20            104.16.249.249       TCP          54 43644 → 443 [ACK] Seq=40 Ack=40 Win=1452 Len=0
   11 40.074502371  192.168.1.15            192.168.1.20         TCP          66 50820 → 65432 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=
   12 40.074632052  192.168.1.20            192.168.1.15         TCP          66 65432 → 50820 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 M
   13 40.074799949  192.168.1.15            192.168.1.20         TCP          60 50820 → 65432 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
   14 40.085710280  192.168.1.15            192.168.1.20         TCP          65 50820 → 65432 [PSH, ACK] Seq=1 Ack=1 Win=2102272 Len=1
   15 40.085753320  192.168.1.20            192.168.1.15         TCP          54 65432 → 50820 [ACK] Seq=1 Ack=12 Win=29312 Len=0
   16 40.087513744  192.168.1.20            192.168.1.15         TCP          63 65432 → 50820 [PSH, ACK] Seq=1 Ack=12 Win=29312 Len=9
   17 40.088351539  192.168.1.15            192.168.1.20         TCP          60 50820 → 65432 [FIN, ACK] Seq=12 Ack=10 Win=2102272 Len
   18 40.088519099  192.168.1.20            192.168.1.15         TCP          63 65432 → 50820 [PSH, ACK] Seq=10 Ack=13 Win=29312 Len=9
   19 40.088897999  192.168.1.15            192.168.1.20         TCP          60 50820 → 65432 [RST, ACK] Seq=13 Ack=19 Win=0 Len=0
   20 41.094633538  Azurewav_fc:80:c0       Broadcast            XID          60 Basic Format; Type 1 LLC (Class I LLC); Window Size 0
   21 44.744604774  PcsCompu_b9:a9:6b       Netgear_a3:40:9d     ARP          42 Who has 192.168.1.1? Tell 192.168.1.20
   22 44.745147742  Netgear_a3:40:9d        PcsCompu_b9:a9:6b    ARP          60 192.168.1.1 is at 9c:d3:6d:a3:40:9d
   23 45.055935151  192.168.1.15            192.168.1.255        BROWSER     243 Host Announcement [redacted], Workstation, Server
   24 45.256020446  PcsCompu_b9:a9:6b       AsustekC_75:82:2e    ARP          42 Who has 192.168.1.15? Tell 192.168.1.20
   25 45.256543420  AsustekC_75:82:2e       PcsCompu_b9:a9:6b    ARP          60 192.168.1.15 is at 1c:87:2c:75:82:2e

▸ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ IEEE 802.3 Ethernet
▸ Logical-Link Control
▸ Logical-Link Control Basic Format XID

0000  ff ff ff ff ff ff f0 03  8c fc 80 c0 00 06 00 01   ........ ........
0010  af 81 01 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  f3 72 ea 47               ........ ·r·G

wireshark_enp0s3_20221020203708_1Yce4t.pcapng     Packets: 25 · Displayed: 25 (100.0%)     Profile: Default
```

Figure 5. Wireshark Capturing packets.

To stop the capture, press the red stop button, which then turns grey. There is not an explicit option to clear the captured data, except that you can go to the file menu and select close. This will return you to the list of adapters. You can also press the shark icon to start a new capture (you will be prompted to save or close).

For each packet the following information is shown:

No	The packet order starting with 1 as the first.
Time	Time relative to the start of the capture (other formats are available).
Source	The IP address of the source of the packet.
Destination	The IP address of where the packet is destined.
Protocol	The protocol the packet is part of.
Length	The length of the packet in bytes.
Info	Information about the packet's type, flags and data in the packet.

The capturing of packets illustrates the need for secure communication. If you were, for example, in a coffee shop or other public place with free a public Wi-Fi network, then with Wireshark you would be able to capture and analyze a lot of other people's traffic. Note, however, that doing so is illegal, especially if used for malicious purposes.

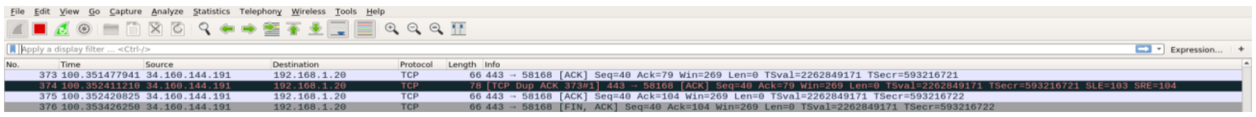
Filtering by Specific IP Addresses:

Often the amount of data captured is so vast that it is difficult to home in on the specific things you are looking for. This is where filtering can help. Table 2 shows some examples of the filtering available.

There are many more filters than this. To learn about them, see the Wireshark documentation online. The filter controls are directly above the list of packets where it says, '**apply a display filter**' (Figure 6).

Table 2. Some Filter Examples

Filter by	Description	Examples
Protocol	To define a filter so that we see only packets using the **tcp** protocol we can type **tcp** into that text field and press return. The same can be done for other protocols such as **udp** and **http**.	tcp udp http
IP Address	You can filter by specific **IP** addresses.	ip.addr == 192.178.12.35.
Port number	You can filter by **TCP port number**. The example to the right will allow us to see web traffic as **TCP.** Port 80 is used by the Hyper Text Transfer Protocol for web content.	tcp.port == 80
Search by packet content	If you started Wireshark and then directed a browser to Youtube.com you could type **tcp contains youtube**. Once filtered, if you looked at the binary data of one of those packets, you would see that it contains the string YouTube. You could do this to find a packet mentioning Youtube and then see its destination IP address. If you then wanted to see all the things going to that server, you would filter by **IP** address.	tcp contains youtube
Logical operators	You can use logical operators **and** and **or** (also && and || work) to combine expressions.	ip.addr == 224.168.1.15 and tcp.port == 22 ip.addr == 192.168.21.10 or ip.addr ==192.168.21.11
Searching for a range of addresses	It is also possible to use relational operators >=, >, <= and <.	ip.addr >= 192.168.0.0 and ip.addr <= 192.168.255.255

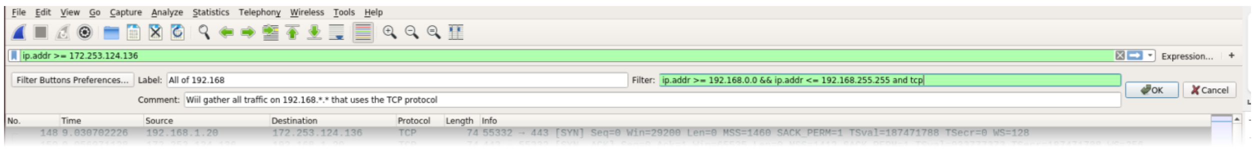

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time          Source          Destination   Protocol  Length Info
  373 100.351477941 34.160.144.191  192.168.1.20  TCP       66 443 → 58168 [ACK] Seq=40 Ack=79 Win=269 Len=0 TSval=2262849171 TSecr=593216721
  374 100.352411210 34.160.144.191  192.168.1.20  TCP       78 [TCP Dup ACK 373#1] 443 → 58168 [ACK] Seq=40 Ack=79 Win=269 Len=0 TSval=2262849171 TSecr=593216721 SLE=103 SRE=104
  375 100.352420825 34.160.144.191  192.168.1.20  TCP       66 443 → 58168 [ACK] Seq=40 Ack=104 Win=269 Len=0 TSval=2262849171 TSecr=593216722
  376 100.353426250 34.160.144.191  192.168.1.20  TCP       66 443 → 58168 [FIN, ACK] Seq=40 Ack=104 Win=269 Len=0 TSval=2262849171 TSecr=593216722
```

Figure 6. Defining a filter.

Saving a Filter:

It is also possible to save the filters. To the right of the filter entry text field, is a + button. Here you can enter a label, a filter and a description. Figure 7 shows what this looks like. Upon clicking ok, the saved filter appears to the right of the input field as a clickable button.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
Filter Buttons Preferences...
OK
Cancel

ip.addr >= 172.253.124.136

Label: All of 192.168

Filter: ip.addr >= 192.168.0.0 && ip.addr <= 192.168.255.255 and tcp

Comment: Wiil gather all traffic on 192.168.*.* that uses the TCP protocol

```
No.   Time          Source          Destination     Protocol  Length Info
  148 9.030702226   192.168.1.20    172.253.124.136 TCP       74 55332 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=187471788 TSecr=0 WS=128
```

Figure 7 Saving Filters

Analyzing a Packet:

To analyze a packet, click on it. Below the list of packets are two resizable areas. The first is the *packet details pane*. This provides additional information about what data the packet contains, such as **port numbers** and **payload** (data). Many of the sections in this handout are expandable.

Below the **packet details pane** is the **packet bytes pane.** This feature allows you to see how each packet is composed of bytes (there 8 bits in a byte). As you click on the items in the **packet details pane** the relevant bytes highlight and vice versa to enable deeper insights as to the information presented.

Notice how port numbers are not part of the packet list. That is because port numbers are a feature of **TCP** (and **UDP**) but not necessarily a feature of all packets. To view the port number, find a **TCP** packet in the list and select it. In the **packet details pane** is shown information about that packet. In that section, expand the section marked **Transmission Control Protocol.** It is there you will find the port number.

Another important feature shown for some **TCP** packets is **TCP Payload**. This is the actual data being exchanged; the payload is the bytes of a message. The content of the packet is also shown as bytes in the bottom section of Wireshark.

Saving the Captured Packets:

Wireshark can save the captured packets to file for later analysis. It supports the PCAP and PCAPNG formats as well as many other formats. When saving your packets, save them as PCAPNG as it is able to retain more information than the standard PCAP.

Once saved you can also load in the PCAP and PCAPNG files for later analysis or sharing.

Task 2 – Create a Filter

This project assumes you are using a Virtual Box in bridged networking to supply a second computer in addition to your host computer.

Set up Wireshark so that it has a filter looking for TCP packets from the IP address of the host and guest operating system so that you can capture the packets going back and forth between the two computers when using your communication application.

Type a copy of your filter below in a color other than black:

"ip.host == 192.168.122. and tcp"

Task 3 Analyze Communication

Using your new filter, capture packets while facilitating communication between the host and the guest operating system using your application:

1) Start your communication application on the host and guest operating system.
2) Start capturing packets in Wireshark.
3) First the host sends a message, "Hello guest operating system" that the guest receives.
4) Secondly, the guest sends a message, "Hello host operating system" that the host receives.
5) Stop the capturing and save this list of packets as a PCAPNG file called **capture1.pcapng** in the same folder as your Python program.

Remember too that the port number 65432 will be used, but you will need to carefully analyze the network traffic while using your application to see what other port numbers are used.

Type your responses to each question in a color such as green, blue or purple that stands out.

Q1: What port number(s) are used and in what context? For steps 3 and 4 describe for each communication in detail what TCP port numbers seem to be used to facilitate the communication and who is using them i.e. guest or host OS: the sender uses varying ports each time a message is sent, while the receiver has a static port. The TCP port numbers used to facilitate communication is the receivers port number.

Q3: Are the port numbers the same every time the programs are run? If they are not, what changes?

No, the same port numbers are not used every time. It changes to a port number located in the 40,000s.

Q3: How many packets are exchanged between the host and the guest operating system before the packet containing the payload is sent? 4 packets before the payload are sent.

Q4: How many packets are exchanged between the host and the guest operating system before the packet containing #<<END>># is sent? 1 packet is sent in between the #<<END>># and the Hello Host Operating System message

Q5: In that instance, who sends the packet containing #<<END>># the guest or the host OS and why?

The guest OS sends the packet containing #<<END>>#, and this is because it is the last message being sent so it is closing the connection from the receiver to the sender.

Q6: In one round of communication between the host and guest, is the packet containing #<<END>># the last packet sent? Give your answer and a detailed reason for your answer.

No, the #<<END>># packet is not the last packet sent. There are 3 other packets sent between the sender and receiver after the #<<END>># packet, and this is to terminate the connection between host and guest OS.

Q7: Some packets contain the flag SYN. Describe roughly what is going on at this point. This may require some additional research to answer.

The SYN packets enable the initial connection between the guest and host OS.

Q8: Assuming you knew how to generate packets using Wireshark, describe in detail how you might be able to disrupt or interfere with this communication?

You could interject your own packet into the communication and interject a message between the guest and host OS. This would affect the message integrity and authentication between the guest and host OS.

Q9: How long does the message being sent need to be before it uses more than one packet to send? Explain your answer. You may have to do some research to answer this.

The message being sent would need to be more than 1024 bytes before having to use multiple packets. This is because we set the data in the code to read in 1024 bytes per packet.

Q10: On your virtual machine, initiate a new capture session that only filters for TCP packets. Then take the built-in Firefox browser to **bbc.co.uk**. Once the page has fully loaded, stop the capture of packets in Wireshark. *Note, if you see the word Mozilla, that is referring to the Firebox browser.*

a) Pick a couple of fairly unique words from the home page (not including BBC). Can you find any packets with this content? Yes/No and explain your answer and say what words you looked for.

   We were looking for the words Sunday, 6 November. We were not able to find these words within the packets and that is because TCP packets do not appear to hold any usable data. TCP packets appear to generally hold links over holding actual data.

b) Do a filter **TCP** packet containing BBC and take a look at the first packet using the HTTP protocol. Besides your **IP** address, what information does that leak that a potential hacker might find useful?
   Aside from my IP address some useful information that could be useful to potential hackers could be your online certificate because then someone could potentially masquerade as you.

c) What did you find most surprising about the captured packets?
   We found that the TCP packets mostly containing links and the HTTP packets mostly contained actual page styling things.

Finishing and Submission:

Once you have answered the questions and committed the most recent version of the code to the GIT repository, tar up your project folder including communication.py, your **PCAPNG** file and git repository to a file called CS325_Project2a.tar

To tar your project folder, be in its parent directory and type the following (assuming your project folder is project 2).

```
tar -cvzf project2.tar.gz project2
```

Grading:

Starting with 100 points, we will remove 0..n points for each category below.

**Submissions with programs that do not run due to a syntax error will be given a 0 grade.

Task 1 Program: 0- 60%

Points will be deducted for errors under the following categories.

- Correct and complete Implementation of requirements,
- Resilience to user error and the prevention of crashes,
- Code style and reuse,
- Documentation and commenting,
- Answer to the question posed in the prior section,
- The attestation report, and if required your oral demonstration of the team project.

Task 2 Filter 0 - 10 %

Task 3 Questions and PCAPNG File 0-30%

Administrative

- This project was posted on Canvas's Assignments page at 6:15pm on October 24th 2022.
- This project is to be submitted to Agora using `handin` by **11:59pm on Monday the 7th of November**. You should tar the source code, the four directories (with files), and your answer to the question.
  ```
  handin.325.1 2 project2.tar.gz
  ```
- This project can be turned in late, but three points will be taken off for every date late (not counting weekends and holidays) up to a maximum of 30 late points. For example, if the project is turned in between 12:00am and 11:59pm on Tuesday 8th of November, it is one day late. You can turn it in up to a date near the end of the semester that will be announced later.
- You must work only with your assigned partner. You may talk with other students in the class about the concepts involved in doing the project, but anything involving actual code needs to just involve just your partner. In other words, you may not show your code to students outside of your pair, and you may not look at the code of another team.
- If you and your partner are experiencing difficulty working together, identify this quickly and see a course instructor for assistance.

Version Control:

V0.1 Draft Version.

V1.0 Final public version

V1.1 Added information in relation to guest-to-guest communication.