|  |  | **Cybersecurity Lab, CSE 3140** |
| --- | --- | --- |
|  |  | **spring 2022** |
| **Network Protocols and Sniffing** | | |
| **Section # :** | | |
| **Team #:** | | |
| **Names:** | | |

A key way for an attacker to gain access to an organization's resources is through the network, which is usually connected to the internet. Organizations try to prevent as many attacks as possible in a timely manner. Therefore, it is critical to understand the architecture of the system, the network design, communication flows and how to protect against possible attacks. In this lab we’ll start to look at network security. Almost everything we do on computers requires speaking to some other computer over a network (often through the Internet). Attackers can observe this traffic and try to figure out what people are doing. ***Sniffing and Spoofing*** are core fundamental skills for most networking attacks. As data travels over the network, a sniffer program is capable of capturing and analyzing each protocol data unit. There are many programs that do sniffing and spoofing. An extensively used tool is [Wireshark](https://www.wireshark.org/) which we will be using during this lab. Wireshark is a software protocol analyzer, or "packet sniffer" application, used for network troubleshooting, analysis, software and protocol development, and education. At the most basic level, Wireshark monitors all network traffic that is visible to your computer. It has lots of features for making analysis of this traffic easier. You’ll need to use MobaXterm for this lab to get X windows forwarding. However, it is also very important to be able to build your own tool. At some point you may have a specific goal that is not supported by any tool. In this lab you will learn basic networking concepts and how the OS handles your packet to be ready to go over the network. You are also going to learn how to write your own sniffing or spoofing program using python. For those of you that haven’t taken a networking class this lab will naturally touch on networking concepts. We encourage you to look up concepts you don’t understand and ask questions.

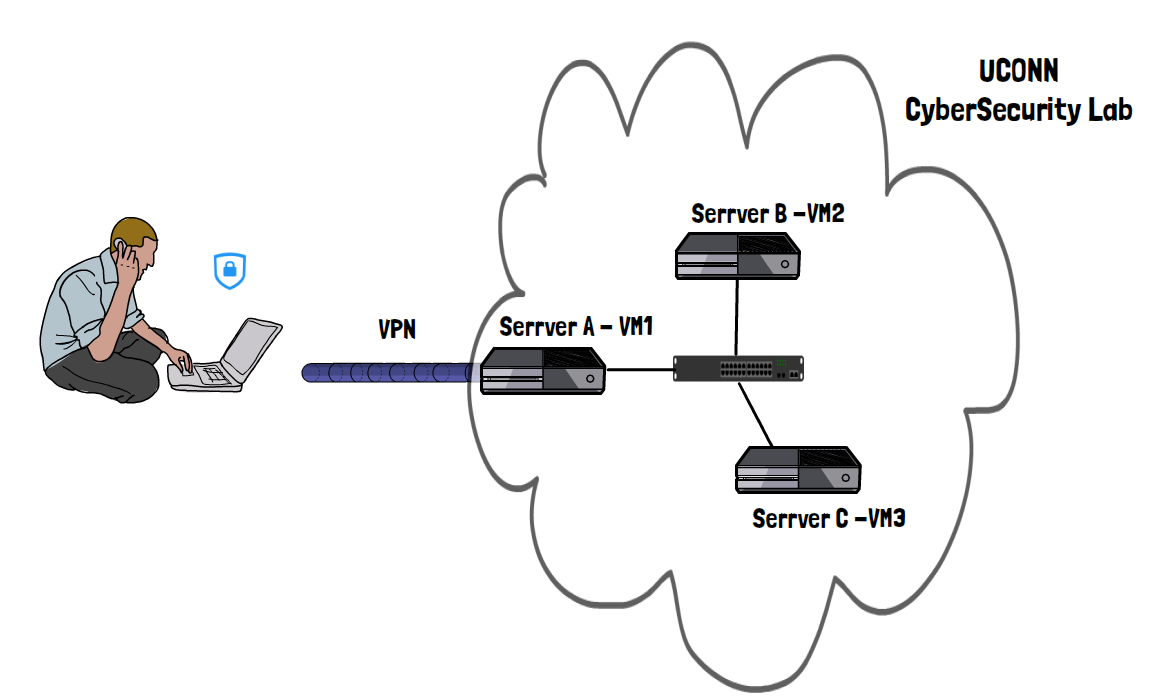


Fig.1: lab topology network

For this lab, we are providing the network configuration shown in fig 1. Note that these virtual machines are completely new, no changes that you made in previous labs persist to this environment. You have **2 VMs** assigned to your group. The main VM you have access to is **ServerA-VM1**, see fig.1. **ServerA-VM1** IP address is based on a formula that depends on both your section number and your group number. The overall IP is ***172.16.51.<20\*section number + group id>***. So, a student in group 4 and section 3 will use IP address **172.16.51.64.** Start by SSHing into your assigned virtual machine (as before) the username: ***cse*** and the password: ***cse3140,* change your password as soon as you login**.

Please add any script **Use screenshots whenever possible.**

**Question 1 (10 points):**

Sniffing and spoofing skills are important skills to learn. We will start by learning how to write a simple sniffing program using python and Scapy. [Scapy](https://scapy.readthedocs.io/en/latest/introduction.html) is a powerful module for packet manipulation. It can help packet parsing, sniffing, spoofing, sending and receiving. The following example can help you write your first sniffing program. You can have more advanced skills if you tried the same programs in C. However, we are interested in learning the fundamental skill, so we will follow an easy way of implementation.

We will have your group co-operate with some other group **(only for this question)**, **group A** is required to write a listener programmer ***“listenerA.py”*** that will be receiving a secret message from **group B**. Group B will be writing a sender program **“senderB.py”** that will send that secret message. Next we will switch roles. Group A will write the sender program **“senderA.py”** to send a message to the listener program ***“listenerB.py”*** that groupB should be using to listen to the secret message. Do not share the exchanged messages, but you can acknowledge receiving a message. Report the string you sent and received from the other group. A helpful sample for sending or receiving using python is shown below:

#---------------------------------------------------------------------------------

# Sender

import socket

UDP\_IP = "172.16.50.226"

UDP\_PORT = 3300

MESSAGE = b’secret\_message’

print("UDP target IP: %s" % UDP\_IP)

print("UDP target port: %s" % UDP\_PORT)

print("message: %s" % MESSAGE)

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) # UDP

sock.sendto(MESSAGE, (UDP\_IP, UDP\_PORT))

#---------------------------------------------------------------------------------

# Receiver

IP= "0.0.0.0"

Port = 3300

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind((IP,Port))

while True:

data, (ip,port) = sock.recvfrom(1024)

print("Sender:{} and port:{}".format(ip,port))

print("received message:{}".format(data))

**Question 2 (10 points):** **CONDUCTING RECONNAISSANCE**

Let’s extract some networking information about your running VM. SSH into your assigned ***VM-serverA***. A nice utility to learn is ***ifconfig***. Use the **ifconfig** to verify your IPv4 address.

***1)*** *What MAC* ***address*** *is assigned for this specific NIC?*

***2)*** *What is your interface* ***(name)****?*

CONDUCTING RECONNAISSANCE, Before launching an active attack, an attacker usually conducts a reconnaissance practice to acquire more information on the target network/hosts. **Let's take a look at some examples of network scans next.**The **ping** command is a way of determining if a host exists on the network. Most computers will respond to a ping (this is turned off in many security conscious organizations). The command is: ***ping <ip\_address>.*** Ping all of the machines on this subnet (write a script rather than doing this manually).

**What ranges are occupied?** **What ranges did you not receive a response from?**

Once the attacker identifies a host that is up and responding, the next step is to use a port scan to see whether the host has any open and listening. Usually, we scan the well-known ports that are in the range 1-1023; however, you can set the application to scan a wider range of ports.

You are now going to see what active services exist on the Server that you are SSHing to. You will be building a very simple port scanner (a very basic version of [https://nmap.org](https://nmap.org/)). You can use the command ***telnet*** to check what ports are open on your server A. The basic command is ***telnet <ip\_address> <port\_number>***. Check ports between **1-1023**. When you find an open port, record it below. [Look up what service that corresponds to](https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers). Record that below.

Note that ports are actually 16-bit numbers so they can extend to 65535. Extend your script to cover all possible ports. If you see ports above 1024 open, what do these likely correspond to? (Don’t record all of them.)

After discovering open ports, we may seek to take the next step and connect with the host using the open port. To mitigate this vulnerability, the network administrator should disable any ports that aren't required.

**Question 3 (10 points):** **SNIFFING NETWORK TRAFFIC (Wireshark)**

Open up Wireshark through the SSH terminal. (We recommend launching it in the background by using the ***&*** at the end of the command.) To have the window launch on your computer requires **X** windows forwarding. See instructions at <https://mobaxterm.mobatek.net/> if you receive an error about having no X11 server. Once Wireshark is open, start collecting all traffic.

***ping server 172.16.50.1, then analyze the captured traffic. Did you notice the ARP packets?***

***Why does the PC send out a broadcast ARP prior to sending the first ping request? What MAC addresses did you notice? screenshot your VM’s mac address and the mac address for the host (172.16.50.1)?***

You may need to clear your ARP cache in case you want to repeat this step. You can use **arp –n and arp –d commands** to display the arp cache and remove an entry in the cache**.**

There are filters for Wireshark that make understanding traffic much easier: <https://wiki.wireshark.org/DisplayFilters>. You may find it convenient to filter using the ***host <ip\_address>*** filter which requires that host to be involved in the communication.

**Question 4 (15 points):** Now let's analyze the traffic captured using Wireshark. There is a web server running on **Server A** (the one you SSH to). “For remote students, [Configure the SSH tunneling to be able to open the web page on your machine *(Check instructions on huskyCT)”]. there is a continuous server talking to your server A, namely server B (VM\_B). Server B is running an ftp server and has telnet protocol enabled. Reports the IP addresses for serverA, and serverB. Start running Wireshark then.*

* *open the web browser, and type the IP address of server A to see the web service home page*
* run **ftp <serverB>**, [username:**ftpcse,** password**:cse3140]**, download file named **FTP-TEST.**
* run **telnet <serverB>**, [username:**cse,** password**:cse3140]**,then close the connection
* run **ssh <serverB>**, then close the connection

At the completion of this run, list the following:

1. all port numbers observed
2. all IP addresses observed

Create a display filter to show only web traffic involving ***VM-serverA*** (the one you are using for the SSH). What is the display filter? What type of traffic is present from that machine?

Examine the traffic of a Telnet session to **VM-serverB**. You can telnet to any machine using the command **telnet <host\_IP>,** apply a filter that only displays Telnet-related traffic. *Right-click one of the Telnet lines in the Packet list section of Wireshark, and from the drop-down list, select Follow TCP Stream.* The Follow TCP Stream window displays the data for your Telnet session with the VM. ***What do you notice?***

Let’s now examine an SSH connection. Type **ssh <VM-serverB>**, to get a connection. Filter all SSH protocol connections with the remote serverC, ***what is the display filter?*** Right-click one of the SSHv2 lines in the Packet list section of Wireshark, and in the drop-down list, select the Follow TCP Stream option. Examine the Follow TCP Stream window of your SSH session. What do you notice, which do you prefer for a remote connection (SSH or TELNET)? Why?

List all the hosts that are involved in an SSH connection that you can capture using Wireshark, Isolate one source and destination involved in SSH. Filter just those packets. What information can you confidently learn about the two computers involved in the connection? What can you not decipher? What is your best guess as to why you can’t decipher everything?

**Question 5 (10 points): Poison ARP cache**

For now, you are probably wondering how these packets are sent over the internet. How does the TCP/IP protocol stack manage to know the location of the destination addresses? There are two types of addresses that are needed for any packet transfer; namely, physical address (MAC address) and logical address(IP address). So far, using the “ifconfig” command, we learned how to obtain the src IP and src MAC addresses. What about the destination MAC and the destination IP address for the VM I am willing to talk to?

For any communication, the destination IP address can be obtained using an addressing service namely, the [Domain Name System (DNS)](https://www.youtube.com/watch?v=mpQZVYPuDGU). DNS is the hierarchical and decentralized naming system used to identify computers, services, and other resources reachable through the Internet or other Internet Protocol networks. For the physical address, [ARP](https://www.youtube.com/watch?v=98CAjGW0lzk) protocol is used to obtain the physical address from a specific IP address. The IP addressing helps in end-to-end machine delivery. while the physical address decided the next step to take on the local network. In this question we will learn a famous attack that is using ARP messages, namely, “ARP cache poison attack”. The purpose of that attack is to poison the victims arp cache by injecting fake information about the physical address. For example, if the machine's B mac address is X, but you were able to update the victims arp cache to indicate that machine’s B mac address is Y. You can redirect the next step for the victim to go to Y instead of going to X.

ARP is a very simple lower layer protocol that does not have any protection like encryption or integrity checking. The ARP is also stateless, when it sends a request out if it forgets about that request, if it gets the reply it automatically updates the cache. Accordingly, the ARP cache information can easily get poisoned if an attacker manages to trick the victim's machine by updating it’s ARP cache. **You need to update the victim’s arp cache.** You can either send a spoofed ARP request or a spoofed ARP reply, even if a request was not triggered by the request.

Now let's consider **VM-serverA** and **VM-serverB**. We want to poison the cash of the target computer **VM-serverA**. Remember you have a sudo privilege only on **VM-serverB**. You will create an entry in **VM-serverA** that maps the mac address of Instructor’s VM (192.16.51.100) to your **VM-serverB** mac address. You can display the arp cache entries using the command “arp -n”. You will pretend to be the Instructor’s VM with the IP 192.16.50.100, and you will send a unicast fake request to the victim machine **VM-serverA**. When **VM-serverA** receives this request it will update the arp cache to match the fake mac address. you can use the following code:

from scapy.all import \*

VM\_target\_IP = “”

VM\_target\_MAC = “”

VICTIM\_IP = “”

FAKE\_MAC = “”

print(sending spoofed ARP REQUEST …”)

ether = Ether()

ether.dst = VM\_A\_MAC

ether.src = FAKE\_MAC

arp = ARP()

arp.psrc = VICTIM\_IP

arp.hwsrc = FAKE\_MAC

arp.pdst = VM\_A\_IP

arp.op = 1

fram = ether/arp

sendp(frame)

**Question 6 (10 points):** Now, as you have learned how to send and receive through the provide OS Socket API. You should be wondering. How can I sniff packets that are not intended for me? How is Wireshark able to present such details about the packet headers? A nice feature that an OS provides is the use of **raw sockets**. The provided raw socket allows applications to get the data directly without going through each layer of the TCP/IP stack. If you register yourself as an application to the raw socket, the OS will make a copy of each packet received to the existing raw sockets. ***There is a hidden machine that is continuously pinging each VM in the network.*** You need to write a simple sniffing program on **<server B>** to sniff only ping packets and report the source and destination IP and Ethernet addresses. You will find the following example helpful. **[serverC (username:cse & password:cse3140 )]**

from scapy.all import \*

def process\_packet(pkt):

# pkt.show()

if pkt.haslayer(IP):

ip = pkt[IP]

print("IP:{} --> {}".format(ip. src, ip.dst))

elif pkt.haslayer(TCP):

tcp = pkt[TCP]

print ("TCP ports:{} --> {}".format(tcp. sport, tcp.dport))

elif pkt.haslayer(UDP):

udp = pkt[UDP]

print ("UDP ports:{} --> {}".format(udp.sport,udp.dport))

else:

print("other protocol")

sniff (iface = 'enp0s3', filter = ‘tcp or udp’, prn = process\_packet, count =100)

**Question 7 (12 points):** [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) is the transport protocol used for applications that require reliability. TCP connections are initiated using a three-way handshake: **SYN**, **SYN-ACK**, and **ACK**. Before a client attempts to connect with a server, the server must first bind its application to a continuous listening port that is ready to receive connections. A client usually starts the connection establishment by sending a SYN packet. During the data exchange all packets sent from one side are ACKed by the other side to ensure reliable transmission. A connection termination phase uses either three-way or four-way handshake. ***What is the difference between three-way or four-way handshake used for terminating the connection?*** Try to find a three way handshake for connection establishment and another three way handshake connection termination in your captured TCP. You can use the FTP connection from the previous question. ***Which layer does this TCP belong to in the TCP/IP model?*** What actual data is sent in these three packets (ignoring TCP headers)? For one TCP connection, can you identify TCP 3-way handshake easily? Fill in the following information for the following connection establishment 3-way handshake **(Consider the TCP connection sending packets of length 1566 bytes)**. Why does TCP need to specify a window size (Win)?

|  | **Connection establishment** |  |
| --- | --- | --- |
|  |  |  |
| IP address: ……. |  | IP address: ……. |
| Port: ……… |  | Port: ……… |

For the FTP communication you previously tested, capture the connection termination and record the following.

|  | **Connection termination** |  |
| --- | --- | --- |
|  |  |  |
| IP address: ……. |  | IP address: ……. |
| Port: ……. |  | Port: ……… |

**Question 8 (5 points):** List all of the protocols you have seen in your capture, and specify the operating TCP/IP layer for these protocols? Based on Internet lookups, which of these protocols try to provide **security/encryption**?

**Question 9 (15 points):** This part requires you to reverse engineer a custom protocol that is running on the network. First you need to find this protocol. The server is listening to port 8800. The protocol runs periodically. **What display filter did you use to isolate the protocol?**  
Your goal for this question is to write a program that can connect to the server and eventually get a response that starts with: “Critical Inf.” There is a client running on the network that is responding properly to the server requests. You need to observe interactions between the running client and server to decide how the client is supposed to respond to each request. There are six stages in the interaction. The last four are randomized so don’t just copy paste from what you see the client on the network doing. Your job is to reverse engineer the protocol state machine that the server is running. This is representable using a simple finite automata: <https://en.wikipedia.org/wiki/Finite-state_machine>. For this question I encourage you to use the [Python Socket package](https://docs.python.org/2/library/socket.html). At some point you may need to change the source port that your client is using. This can be done with the command (for a socket named s before connecting): s.bind(('', <port\_num>)).

We have provided a simple client server example in your home directory (echoServer.py and helloClient.py) in your project2 folder. At some point during this question you’ll need to compute a cryptographic hash. You can do this using the hashlib package (import hashlib) and the following code:

**m = hashlib.sha256()**

**m.update(<thing to hash>)**

**hash = m.hexdigest()**

1. What do you need to do to get the server to respond to your first message?
2. Second message?
3. Third message? You may want to look at multiple interactions to figure out what’s happening
4. Fourth message?
5. Fifth message?
6. Sixth message? Did you successfully receive the critical info when you sent this message? How do you have to change your program?