Packet Sniffing and Spoofing

הנחיות להגשת המטלה:

את מטלה זו יש להגיש בזוגות כקובץ ZIP עם מספרי ת"ז של הסטודנטים/ות. לתיבת ההגשה במודל. הגשות באיחור יתאפשרו עד 4 ימים כאשר לכל יום איחור ירדו 5 נקודות. שימו לב, יש להגיש גם קובץ pdf המכיל צילומי מסך (בכל מקום שאתם עושים משהו תצלמו מסך ותסבירו איך הגעתם למסקנה) בכל מקום שאתם משדרים מידע צריך לצרף קובץ הקלטה. אין להגיש צילומי מסך של מרוכזים בקובץ אחד עם הסברים, קובץ ללא הסברים או צילומי מסך בלבד לא יבדקו

1) את המטלה יש להגיש עד התאריך המצוין בתיבת ההגשה

2) כל הקבצי המטלה (קוד, פלט תעבורה, הסבר) כולל הסברים שלכם והקלטות Wireshark דחוסים לקובץ ZIP ששמו הוא מס' ת.ז. של המגישים עם קו תחתון ביניהם ID_ID.

3) הגשה בזוגות זהו הסטנדרט וזו חובה. לאישורים חריגים שלחו מייל למרצה שלכם ותכתבו את אחראי המתרגלים, אלמוג שור.

4) מותר לכם להשתמש בכל החומר שנמצא במודל כולל קוד בתרגולים. חומרים אחרים אין אפשרות. כמובן שאפשר להיעזר באינטרנט להבנה של תהליכים וקוד אבל בשום פנים ואופן לא להעתיק קוד. כל שימוש בכלים כמו copilot or chatgpt יפסלו

5) אין איחורים ללא אישור מיוחד של רכז הקורס (עמית), איחור ללא אישור יגרור אפס אוטומטי.

6) הגשת העבודות תתבצע דרך מערכת ה Moodle של הקורס (לא דרך האימייל).

7) יש להקפיד על כללי עיצוב הקוד שנלמדו בתואר (נא להקפיד על פלט ברור, הערות קוד במידה ושמות משתנים בעלי משמעות). קוד רץ בלבד יכול לקבל לכל היותר ציון 60, שאר 40 הנקודות זה הסברים שלכם, ידע, קוד קריא וכו'.

8) ניתן להגיש תרגילים למערכת מספר בלתי מוגבל של פעמים כאשר כל הגשה דורסת את הקודמת.

9) העבודה הינה אישית של הזוג ואסור לקבל עזרה מאנשים מחוץ לאוניברסיטה או בתוכה לה. אנשים המתקשים ורוצים עזרה יכולים לפנות לצוות הקורס בשעות הקבלה או להעלות שאלה לאתר הקורס. עבדתם עם עוד זוג על בעיות מסוימות כתבו זאת בקובץ ה-pdf

10) אסור להעביר קטעי קוד בין סטודנטים, להעלות פתרונות או חלקי פתרונות לאתרים ברשת האינטרנט, פורומים או בקבוצות תקשורת שונות .

11) סטודנטים שיעתיקו פתרון, יקבלו 0 בכל המטלות בקורס ונעלה דיווח לוועדת המשמעת המוסדית.

12) קבצי הקוד המטלה נדרשים להתקמפל ולרוץ על מערכת הפעלה Ubuntu 22.04LTS

13) מסמך המתאר את המערכת והפתרון שלכם (המלצה, השתמשו במבנה המסמך ששלחנו לכם במודל)

For deep dive and to succeed on this Ex, it is recommending start with learning the materials in Appendix A and the link: https://www.tcpdump.org/pcap.html.

The Ex should contain these files:

Sniffer.c, Spoofer.c, Gateway.c, PDF of explanations, Wireshark pcap files.

Task A – 15%

Write your own sniffer for capturing packets.

Run your Ex 2 from Networks Course. Use your sniffer to sniff the TCP packets and write them out into a txt file named after your IDs. The format of each packet should be { source_ip: <input>, dest_ip: <input>, source_port: <input>, dest_port: <input>, timestamp: <input>, total_length: <input>, cache_flag: <input>, steps_flag: <input>, type_flag: <input>, status_code: <input>, cache_control: <input>, data: <input> }

The data output may be unreadable in ASCII form so write the output as hexadecimal.

In the paragraph following your explanations of this task, please answer the following question:

Question

- Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?

Write detailed research in your PDF about Your sniffer abilities and limitations

Task B – 25%

Write a spoofer **for spoofing packets**. Your spoofer should be able to spoof ICMP* packets.
The spoofer should fake the sender's IP and has a valid response. Your code should be able to spoof other protocols with small changes.

In the paragraph following your explanations of this task, please answer these questions:

Question 1.

Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

Question 2.

Using the raw socket programming, do you have to calculate the checksum for the IP header?

Write detailed research in your PDF about your spoofer abilities and limitations.

Task C – 50%

@Before -read Appendix B for composing the LAN of docker containers.

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. You need two machines on the same LAN. From machine A, you ping an IP X. This will generate an ICMP echo request packet. If X is alive, the ping program will receive an echo reply, and print out the response. Your sniff-and-then-spoof program runs on the attacker machine, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique.

Please follow those steps:

1. Compose the Docker Containers that are in the zip file from the Moodle (The TAs showed you how to do so in Tirgul 09)
2. Run your Ex_4 codes from Networks course in this new LAN. Use your sniffer from Task A to sniff the ICMP packets from the seed-attacker. Peel off the needed data for spoofing the packets and return a packet which made by you.
    a. First run – send a ping from Host A to Host B.
    b. Second run – send a ping from Host A to a WAN IP (e.g., google DNS – 8.8.8.8).
    c. Third run – send a ping from Host A to a fake IP.
3. **Write detailed research in your PDF about Task C.**

Task D - 10%

In this task, you will implement the Gateway.c file.

Gateway.c: Takes the name of a host on the command line and creates a datagram socket to that host (using port number P+1), it also creates another datagram socket where it can receive datagrams from any host on port number P; next, it enters an infinite loop in each iteration of which it receives a datagram from port P, then samples a random number using **((float)random())/((float)RAND_MAX)** - if the number obtained is greater than 0.5, the datagram received is forwarded onto the outgoing socket to port P+1, otherwise the datagram is discarded and the process goes back to waiting for another incoming datagram. Note that this gateway will simulate an unreliable network that loses datagrams with 50% probability.

Appendix A:

Task 2.1: Writing Packet Sniffing Program

Sniffer programs can be easily written using the *pcap* library. With *pcap*, the task of sniffers becomes invoking a simple sequence of procedures in the *pcap* library. At the end of the sequence, packets will be put in buffer for further processing as soon as they are captured. All the details of packet capturing are handled by the *pcap* library.

Task 2.1B: Writing Filters

Please write filter expressions for your sniffer program to capture each of the followings. You can find online manuals for pcap filters. In your lab reports, you need to include screenshots to show the results after applying each of these filters.

- Capture the ICMP packets between two specific hosts.
- Capture the TCP packets with a destination port number in the range from 10 to 100.

Task 2.1C: Sniffing Passwords.

Please show how you can use your sniffer program to capture the password when somebody is using telnet on the network that you are monitoring. You may need to modify your sniffer code to print out the data part of a captured TCP packet (telnet uses TCP). It is acceptable if you print out the entire data part, and then manually mark where the password (or part of it) is.

Task 2.2: Spoofing

When a normal user sends out a packet, operating systems usually do not allow the user to set all the fields in the protocol headers (such as TCP, UDP, and IP headers). OSes will set most of the fields, while only allowing users to set a few fields, such as the destination IP address, the destination port number, etc. However, if users have the root privilege, they can set any arbitrary field in the packet headers. This is called packet spoofing, and it can be done through raw sockets. Raw sockets give programmers the absolute control over the packet construction, allowing programmers to construct any arbitrary packet, including setting the header fields and the payload. Using raw sockets is quite straightforward; it involves four steps: (1) create a raw socket, (2) set socket option, (3) construct the packet, and (4) send out the packet through the raw socket. There are many online tutorials that can teach you how to use raw sockets in C programming.

Task 2.2A: Write a spoofing program.

Please write your own packet spoofing program in C. You need to provide evidence (e.g., Wireshark packet trace) to show that your program successfully sends out spoofed IP packets.

Task 2.2B: Spoof an ICMP Echo Request.

Spoof an ICMP echo request packet on behalf of another machine (i.e., using another machine's IP address as its source IP address). This packet should be sent to a remote machine on the Internet (the machine must be alive). You should turn on your Wireshark, so if your spoofing is successful, you can see the echo reply coming back from the remote machine.

5.1 Filling in Data in Raw Packets

When you send out a packet using raw sockets, you basically construct the packet inside a buffer, so when you need to send it out, you simply give the operating system the buffer and the size of the packet. Working directly on the buffer is not easy, so a common way is to typecast the buffer (or part of the buffer) into structures, such as IP header structure, so you can refer to the elements of the buffer using the fields of those structures. You can define the IP, ICMP, TCP, UDP and other header structures in your program. The following example show how you can construct an UDP packet:

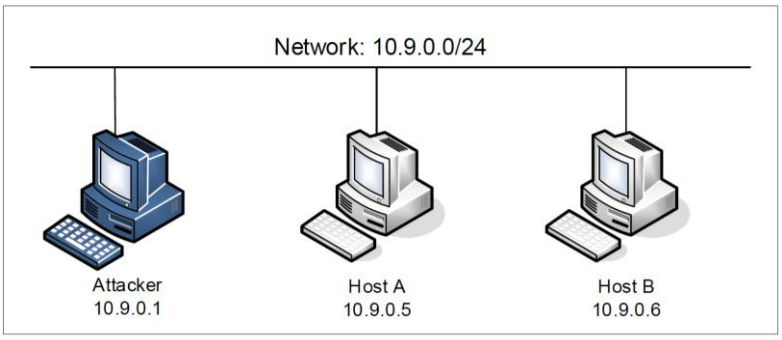
```
struct ipheader {
   type  field;
   .....
}
```

```
struct udpheader {
   type field;
   ......
}

// This buffer will be used to construct raw packet.
char buffer[1024];

// Typecasting the buffer to the IP header structure
struct ipheader *ip = (struct ipheader *) buffer;

// Typecasting the buffer to the UDP header structure
struct udpheader *udp = (struct udpheader *) (buffer
                            + sizeof(struct ipheader));

// Assign value to the IP and UDP header fields.
ip->field = ...;
udp->field = ...;
```

Appendix B:

2 Environment Setup using Container

In this Ex, we will use three machines that are connected to the same LAN. We will use three containers. We will do all the attacks on the attacker container, while using the other containers as the user machines. Diagram of the LAN:



2.1 Container Setup and Commands

Please download the Labsetup.zip file to your VM from the Moodle, unzip it, enter the Lab setup

folder, and use the docker-compose.yml file to set up the lab environment. Detailed explanation of the content in this file and all the involved Docker file can be found from the user manual.
In the following, we list some of the commonly used commands related to Docker and Compose.

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the "docker ps" command to find out the ID of the container, and then use "docker exec" to start a shell on that container.

/*Note: If a docker command requires a container ID, you do not need to
type the entire ID string. Typing the first few characters will
be sufficient, as long as they are unique among all the containers.
*/

2.2 About the Attacker Container

If you look at the Docker Compose file, you will see that the attacker container is configured differently from the other containers. Here are the differences:

- Shared folder. When we use the attacker container to launch attacks, we need to put the attacking code inside the attacker container. Code editing is more convenient inside the VM than in containers, because we can use our favourite editors. In order for the VM and container to share files, we have created a shared folder between the VM and the container using the Docker volumes. If you look at the Docker Compose file, you will find out that we have added the following entry to some of the containers. It indicates mounting the ./volumes folder on the host machine (i.e., the VM) to the /volumes folder inside the container. We will write our code in the ./volumes folder (on the VM), so they can be used inside the containers.

```
volumes:
        - ./volumes:/volumes
```

- Host mode. In this lab, the attacker needs to be able to sniff packets, but running sniffer programs inside a container has problems, because a container is effectively attached to a virtual switch, so it can only see its own traffic, and it is never going to see the packets among other containers. To solve this problem, we use the host mode for the attacker container. This allows the attacker container to see all the traffics. The following entry used on the attacker container:

```
network_mode: host
```

  When a container is in the host mode, it sees all the host's network interfaces, and it even has the same IP addresses as the host. Basically, it is put in the same network namespace as the host VM. However, the container is still a separate machine, because its other namespaces are still different from the host.