

פרויקט סיום- תקשורת ומחשוב

מגישים: עמית חג'ג' - 205837727

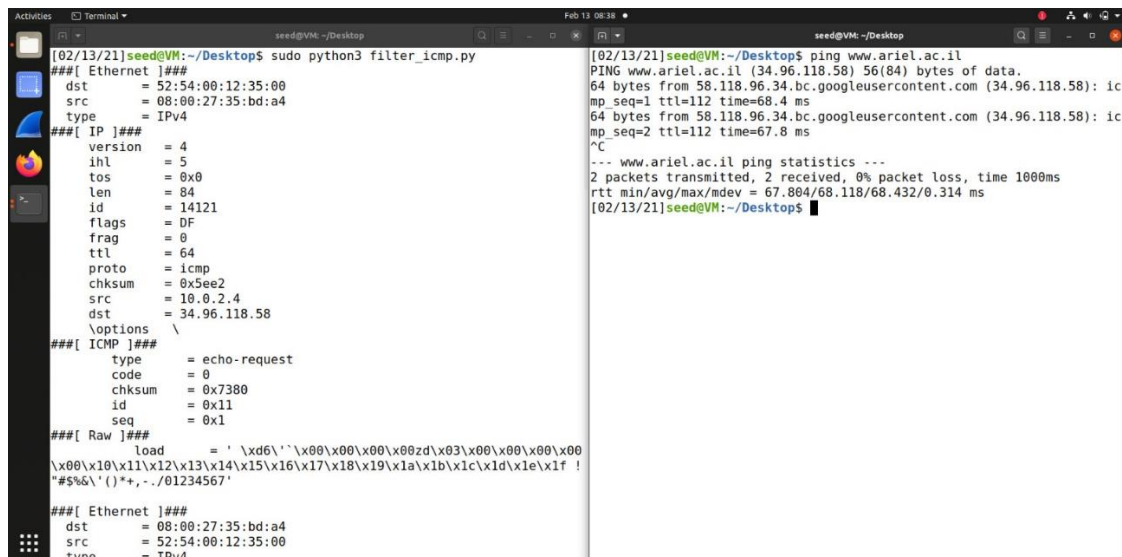
כפיר אטינגר - 311611677

כל הקוד איתו עבדנו נמצא ב- https://github.com/kfiree/Communication_final

חלק 1- Python

משימה 1.1A:

במשימה זו אנחנו נדרשים להפעיל את התכנית עם root privilege. ניתן לראות בתמונה שהתכנית עובדת כראוי ופאקטות נתפסות. כמו כן, ניתן לראות את שכבות הפאקטה. (IP, ICMP, ETH, RAW data)



```
[02/13/21]seed@VM:~/Desktop$ sudo python3 filter_icmp.py
##[ Ethernet ]##
dst      = 52:54:00:12:35:00
src      = 08:00:27:35:bd:a4
type     = IPv4
##[ IP ]##
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 14121
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x5ee2
src      = 10.0.2.4
dst      = 34.96.118.58
options  \
##[ ICMP ]##
type     = echo-request
code     = 0
chksum   = 0x7380
id       = 0x11
seq      = 0x1
##[ Raw ]##
load     = '\xd6\'''\x00\x00\x00\x00zd\x03\x00\x00\x00\x00
\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !
"$%&\'()*+,-./01234567'
##[ Ethernet ]##
dst      = 08:00:27:35:bd:a4
src      = 52:54:00:12:35:00
type     = IPv4
```

כעת אנחנו מריצים ללא root privilege. (הסבר בתמונה)

```
seed@VM: ~/Desktop
[02/13/21] seed@VM:~/Desktop$ python3 filter_icmp.py
Traceback (most recent call last):
  File "filter_icmp.py", line 7, in <module>
    pkt = sniff(filter="icmp", prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 906, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type)) # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[02/13/21] seed@VM:~/Desktop$
```

בגלל שהרצנו
ללא הרשאות
קיבלנו
permission
error

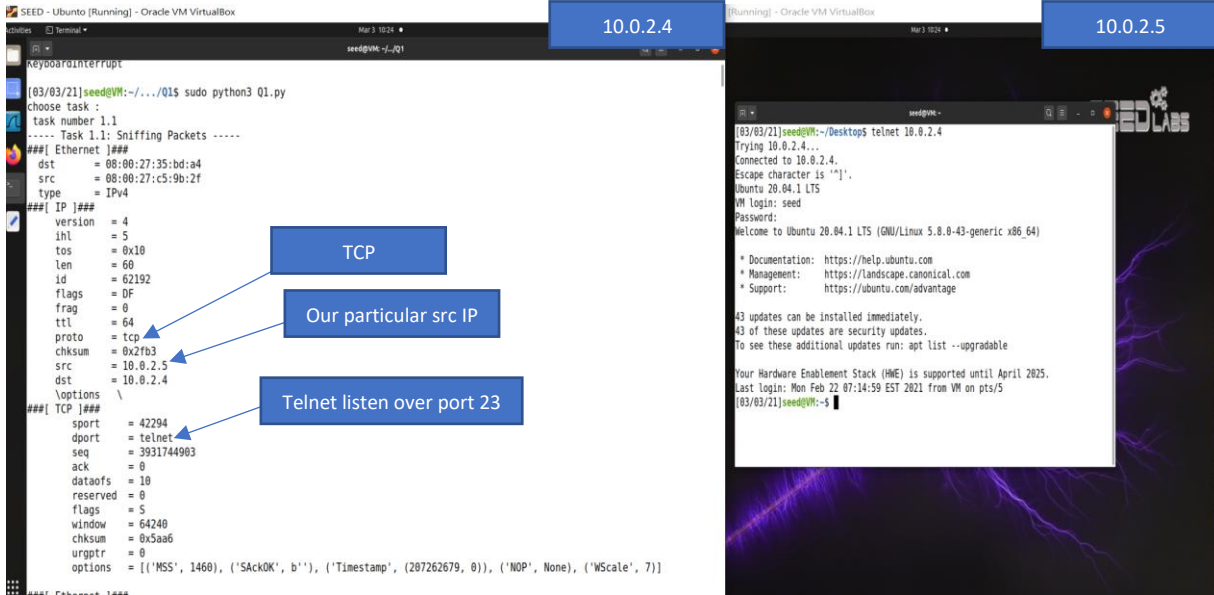
משימה 1.1B:

כאן אנחנו נדרשים להריץ את התכנית אבל בכל פעם להגדיר פילטר אחר.
פילטר 1-ICMP.

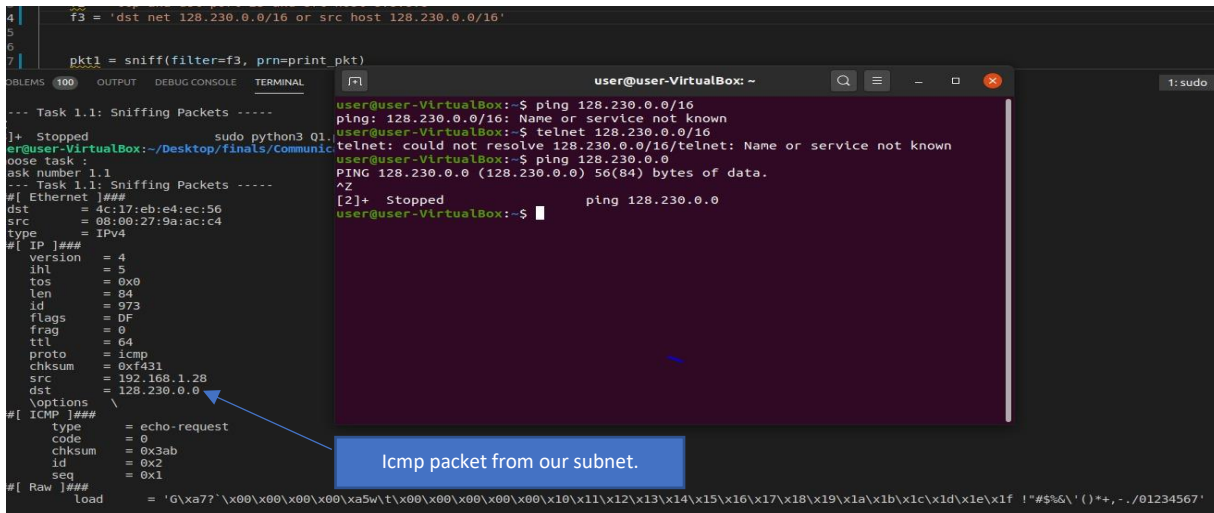
```
user@user-VirtualBox:~/Desktop/finals/Communication
[sudo] password for user:
choose task :
task number 1.1
Task 1.1: Sniffing Packets -----
###[ Ethernet ]###
dst      = 4c:17:eb:e4:ec:56
src      = 08:00:27:9a:ac:c4
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 41794
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xc592
src      = 192.168.1.28
dst      = 8.8.8.8
\options
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0x1dfa
id       = 0x1
seq      = 0x1
###[ Raw ]###
Load     = '\xd5\xa4?'\x00\x00\x00\x00\x02,\x04\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'

user@user-VirtualBox:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=112 time=338 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=112 time=52.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=112 time=99.6 ms
^Z
[1]+  Stopped                  ping 8.8.8.8
user@user-VirtualBox:~$
```

פילטר 2- IP & dest port 23

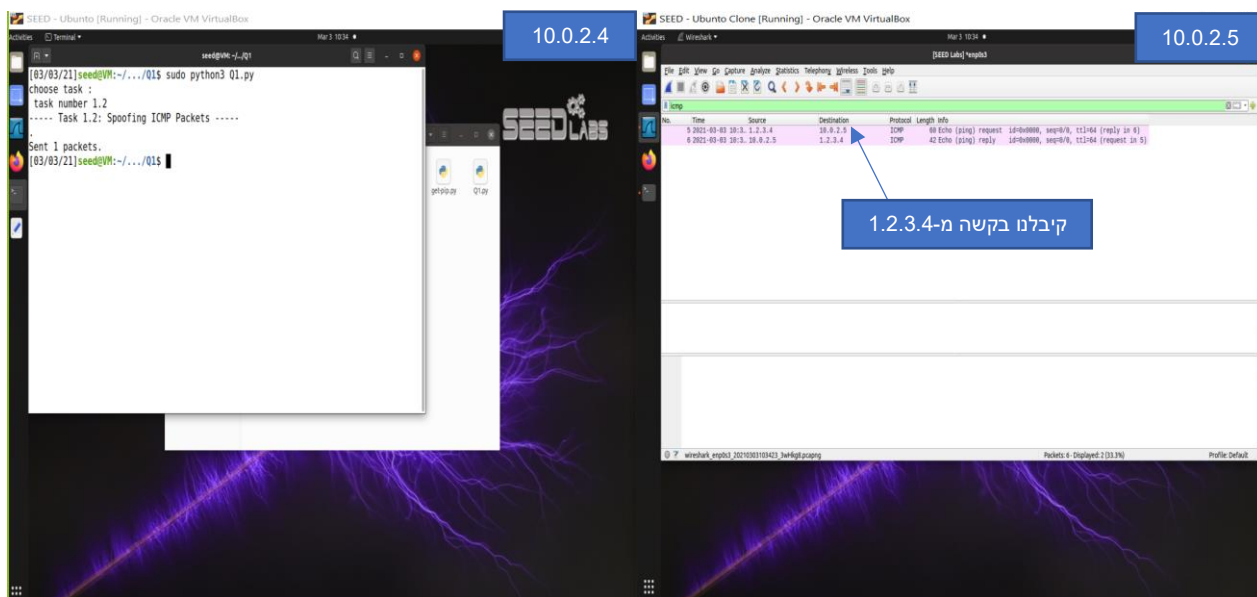


פילטר 3- מ-subnet ספציפי.



משימה 1.2:

במשימה הזאת אנחנו נדרשים לעשות ספוף לחבילה שנשלחת אל מכונה שנייה על הרשת מכתובת IP כלשהי. (שלחנו ממכונה אחת) (שמאלית) למכונה השנייה (ימנית) מכתבות פיקטיבית)



משימה 1.3:

במשימה זאת אנחנו התבקשנו לממש סוג של traceroute. התוכנית צריכה לספור את מספר הראוטרים מכתובת המקור עד לכתובת היעד. ביצענו זאת ע"י שליחת פאקטה מסוג ICMP (echo-request) עם ttl=1. בכל איטרציה הגדלנו את ttl באחד עד שהגענו ליעד. כל ראوتر בדרך מקטין את ttl באחד וכאשר ראوتر מקבל פאקטה, אם ttl=1 הוא מחזיר הודעת ICMP מסוג 11 (time exceeded) במקום להמשיך לראוטר הבא. ואם הראוטר הוא כתובת היעד אז הוא יחזיר למקור ICMP מסוג 0 (echo reply). בתמונה ניתן לראות שלאחר 14 ראוטרים הגענו אל היעד. אפשר לראות שלא קיבלנו תשובה מכל ראוטר בדרך ולכן הוספנו בקוד timeout במידה ולא הגיעה תשובה בזמן.

הראוטר הראשון במסלול של הפקטה

```
=====
dropped by 82.102.132.77
passed through 8 routers, reply type = 11
=====
dropped by 82.102.132.78
passed through 9 routers, reply type = 11
=====
dropped by 80.179.166.17
passed through 10 routers, reply type = 11
=====
dropped by 72.14.212.234
passed through 11 routers, reply type = 11
=====
dropped by 108.170.252.65
passed through 12 routers, reply type = 11
=====
dropped by 172.253.73.155
passed through 13 routers, reply type = 11
=====
we have reached our destination, and it only took 14 routes
```

1

```
[03/02/21]seed@VM:~/.../Q1$ sudo python3 q1.py
choose task :
task number 1.3
=====
dropped by 10.0.2.1
passed through 0 routers, reply type = 11
=====
dropped by 192.168.68.1
passed through 1 routers, reply type = 11
=====
dropped by 192.168.0.1
passed through 2 routers, reply type = 11
=====
dropped by 10.170.1.1
passed through 3 routers, reply type = 11
=====
error. no reply received
=====
dropped by 172.17.3.117
passed through 5 routers, reply type = 11
=====
dropped by 172.17.3.54
passed through 6 routers, reply type = 11
=====
dropped by 212.199.139.85
passed through 7 routers, reply type = 11
=====
error. no reply received
=====
dropped by 82.102.132.77
passed through 8 routers, reply type = 11
=====
```

2

הפקטה הגיעה אל היעד אחרי 14 ראוטרים

משימה 1.4:

במשימה זאת אנחנו מבצעים סינף ואז ספוף בשלושה מקרים שונים.
מקרה 1- כתובת לא אמיתית ברשת.

בקשות לכתובת לא אמיתית

הודעות reply שאנחנו יצרנו

[03/02/21]seed@VM:~\$ ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.

מקרה 2- כתובת לא אמיתית על ה-LAN.

Wireshark capture showing ICMP Echo (ping) requests and replies. The capture is filtered on 'icmp'. The packet list shows several requests and replies between 10.9.0.99 and 10.9.0.99. The packet details pane shows the ICMP Echo (ping) request details, including the Identifier (BE): 3 (0x0003), Identifier (LE): 768 (0x0300), Sequence number (BE): 11 (0x000b), and Sequence number (LE): 2816 (0x0b00).

Annotations:

- בקשות לכתובת לא קיימת על LAN (Requests for non-existent address on LAN)
- הודעת reply שאנחנו שלחנו. (Reply message we sent.)

Terminal output:

```
[03/02/21]seed@VM:~$ ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
```

מקרה 3- כתובת אמיתית ברשת.

Wireshark capture showing ICMP Echo (ping) requests and replies. The capture is filtered on 'icmp'. The packet list shows several requests and replies between 10.9.0.99 and 8.8.8.8. The packet details pane shows the ICMP Echo (ping) request details, including the Identifier (BE): 5 (0x0005), Identifier (LE): 1280 (0x0500), Sequence number (BE): 1 (0x0001), and Sequence number (LE): 256 (0x0100).

Annotations:

- בקשה מכתובת אמיתית באינטרנט (Request from real IP on Internet)
- מכיוון שמדובר בכתובת אמיתית, קיבלנו 2 הודעות reply. אחת שאנחנו שלחנו ואחת מהכתובת עצמה. (Since it's a real IP, we received 2 reply messages. One we sent and one from the IP itself.)

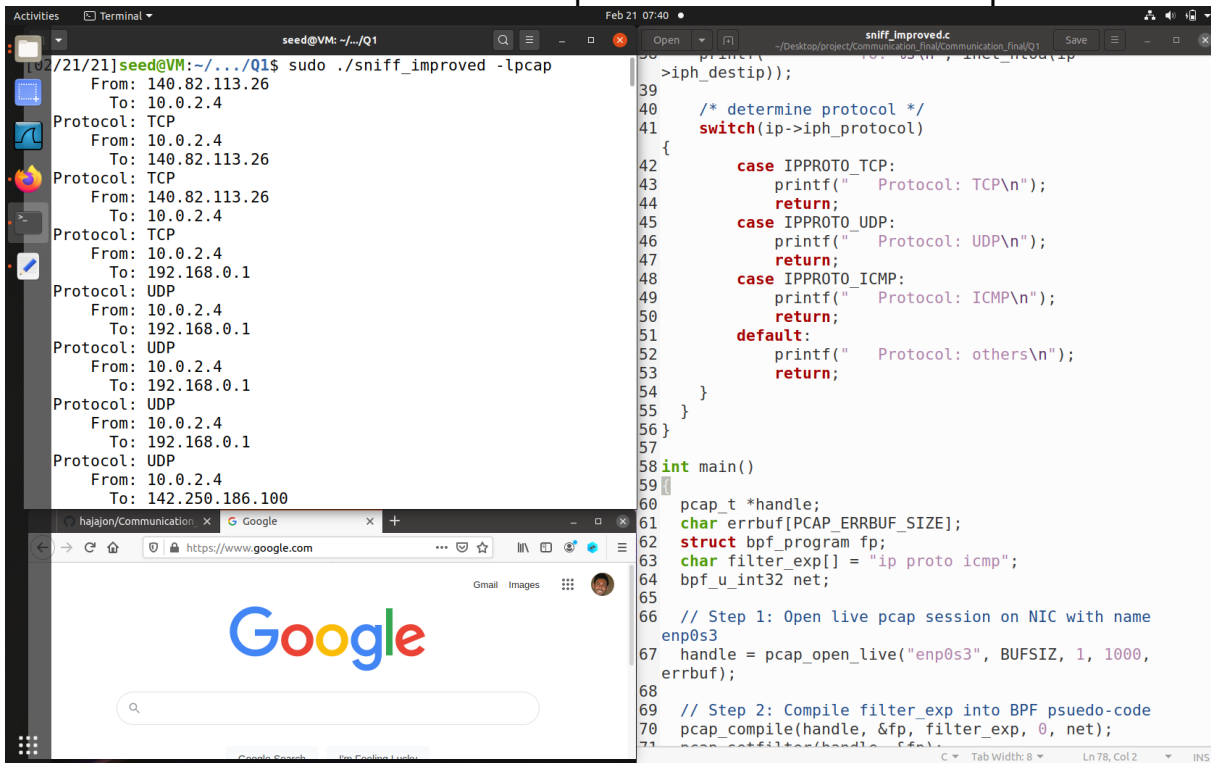
Terminal output:

```
[03/02/21]seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
```

חלק 2- C

משימה 2.1A:

בתמונה למטה ניתן לראות את התכנית שתופסת פאקטות מכל סוג.



The screenshot shows a Linux desktop environment. On the left, a terminal window displays the output of the command `sudo ./sniff_improved -lpcap`. The output lists several network packets with their source and destination IP addresses and protocols. On the right, a code editor shows the source code of the `sniff_improved.c` program. The code includes a `switch` statement to determine the protocol of each packet (TCP, UDP, ICMP, or others) and a `main` function that sets up a live pcap session on the `enp0s3` interface and compiles a BPF filter to capture all traffic.

```
seed@VM: ~/.../Q1$ sudo ./sniff_improved -lpcap
From: 140.82.113.26
To: 10.0.2.4
Protocol: TCP
From: 10.0.2.4
To: 140.82.113.26
Protocol: TCP
From: 140.82.113.26
To: 10.0.2.4
Protocol: TCP
From: 10.0.2.4
To: 192.168.0.1
Protocol: UDP
From: 10.0.2.4
To: 192.168.0.1
Protocol: UDP
From: 10.0.2.4
To: 192.168.0.1
Protocol: UDP
From: 10.0.2.4
To: 192.168.0.1
Protocol: UDP
From: 10.0.2.4
To: 142.250.186.100

/* determine protocol */
switch(ip->iph_protocol)
{
    case IPPROTO_TCP:
        printf(" Protocol: TCP\n");
        return;
    case IPPROTO_UDP:
        printf(" Protocol: UDP\n");
        return;
    case IPPROTO_ICMP:
        printf(" Protocol: ICMP\n");
        return;
    default:
        printf(" Protocol: others\n");
        return;
}

int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "ip proto icmp";
    bpf_u_int32 net;

    // Step 1: Open live pcap session on NIC with name
    enp0s3
    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000,
    errbuf);

    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);
}
```

שאלה 2.1.1:

בקוד המצורף, התהליך מתחיל ב**main** כאשר אנחנו פותחים סשן `pcap` חדש עבור הממשק שבחרנו. (שדה `рик`, יקלוט מכל הממשקים שעל כרטיס הרשת) לאחר מכן אנחנו הופכים את המחרוזת `filter_exp` לקובץ שיהווה פילטר עבור `pcap` בשורה שאחריה. (מכניסים קובץ זה אל `fp`) בשלב האחרון אנחנו עוברים בלולאה על הפאקטות שעוברות בממשק שהגדרנו ואז תופסים אותם לפי הפילטר שהגדרנו גם כן. כל פאקטה שנתפסה הולכות לפונקציה ייעודית לטיפול. אותה פונקציה במקרה שלנו היא הפונקציה בתחילת הקוד שבעצם מדפיסה למסך, איזה סוג פאקטה זאת, כתובת מקור וכתובת יעד.

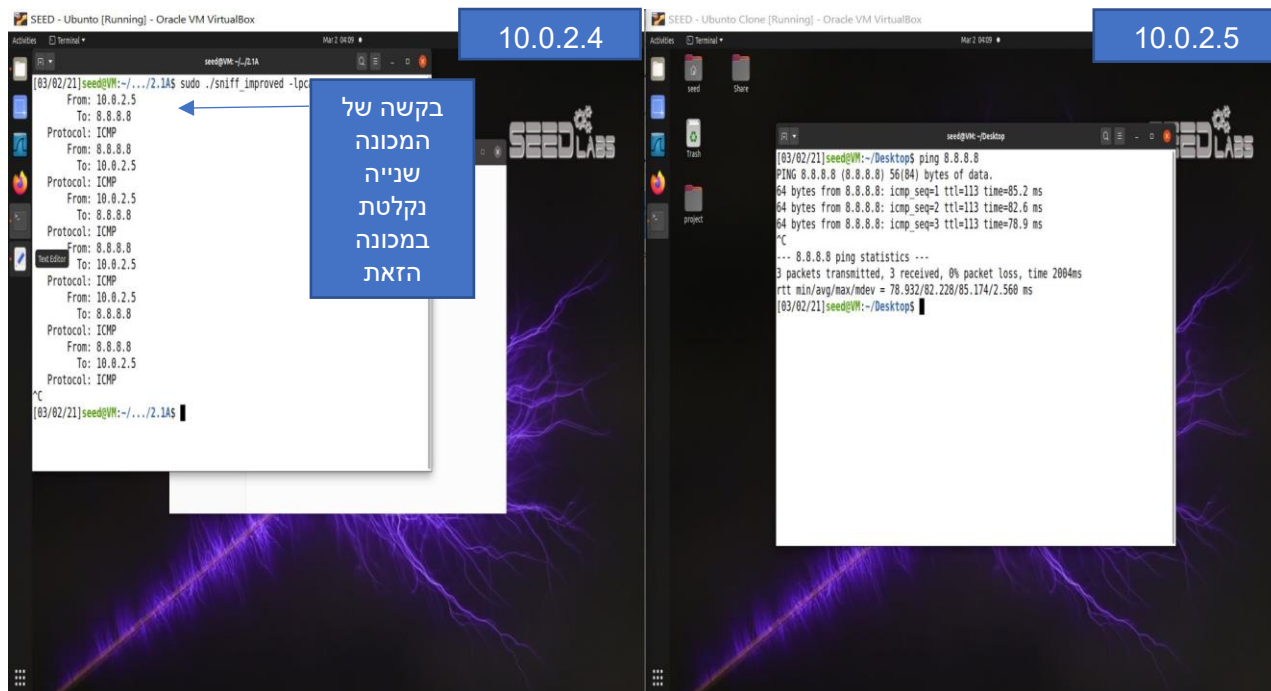
שאלה 2.1.2:

אנחנו מוכרחים `root privileges` מכיוון שאנחנו רוצים לשנות `promiscuous mode` של כרטיס הרשת. הרעיון מאחורי זה הוא שלא כל משתמש יוכל לגעת בהגדרות אלו ולהפוך את המחשב שלנו לפגיע. לגבי השאלה השנייה, השלב בו אנחנו נתקעים הוא של ההרצה עצמה של התכנית. (לאחר הקימפול)

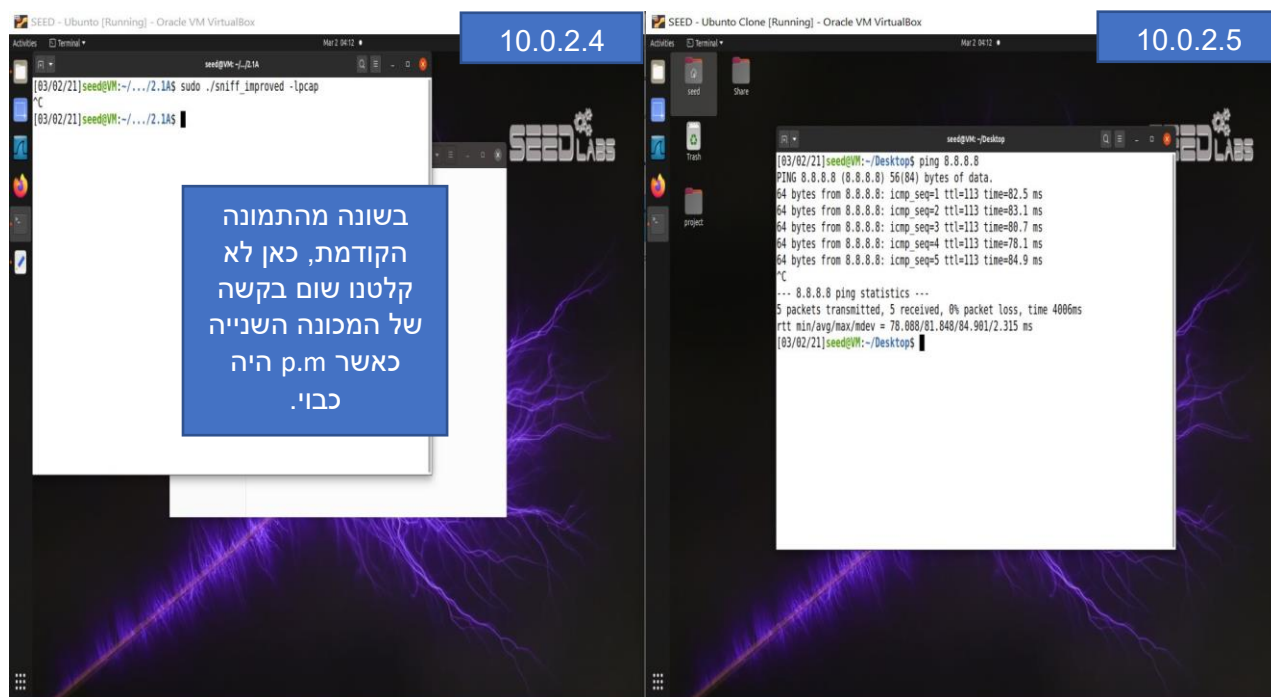
שאלה 2.1.3:

כאשר מפעילים `m.p` אנחנו מרחרחים כל פאקטה שעוברת בכרטיס הרשת. אבל, כאשר `m.p` כבוי אנחנו יכולים לרחרח רק פאקטות שמועדות אלינו או יוצאות מאיתנו.

תמונה ראשונה: מצב `m.p` היה דלוק.



תמונה שנייה: מצב מ.ח היה כבוי.

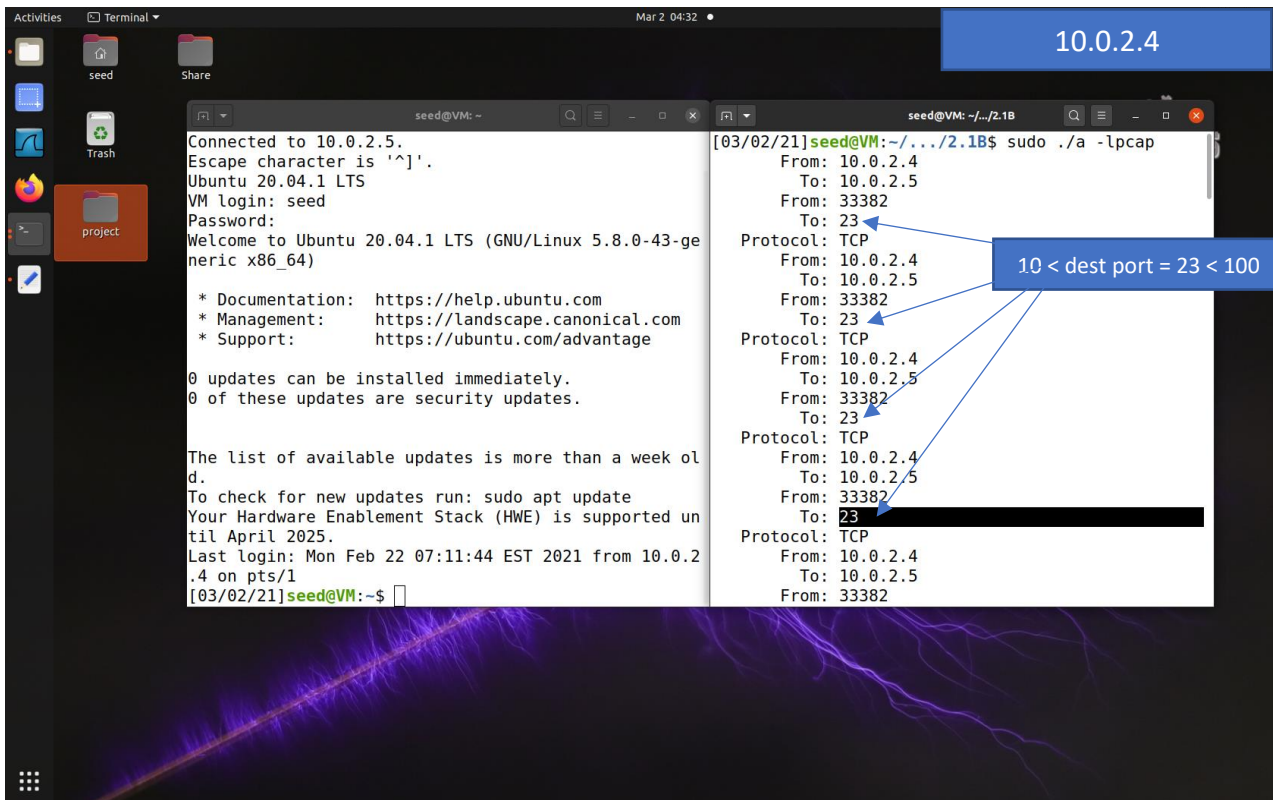


משימה 2.1B:

בתמונה כאן אנחנו תופסים רק חבילות מסוג ICMP.

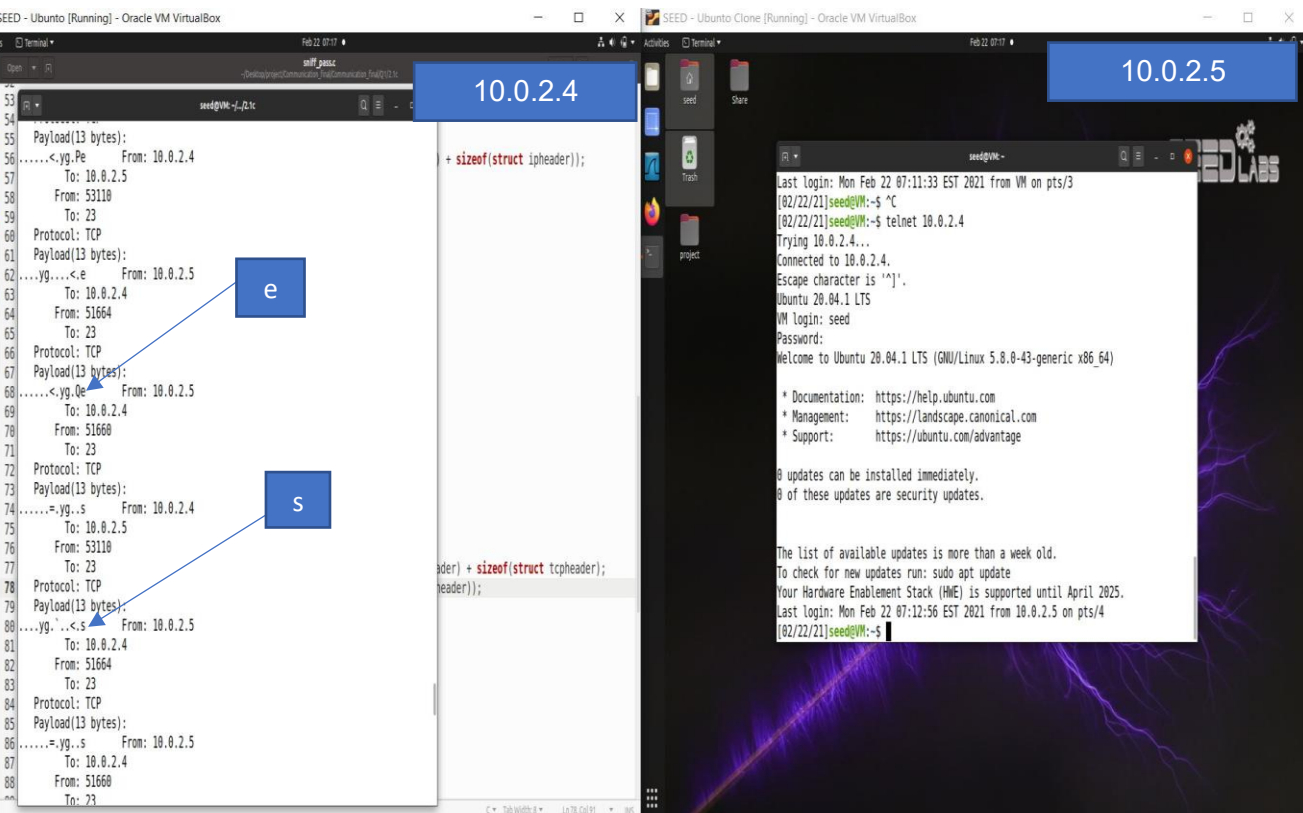
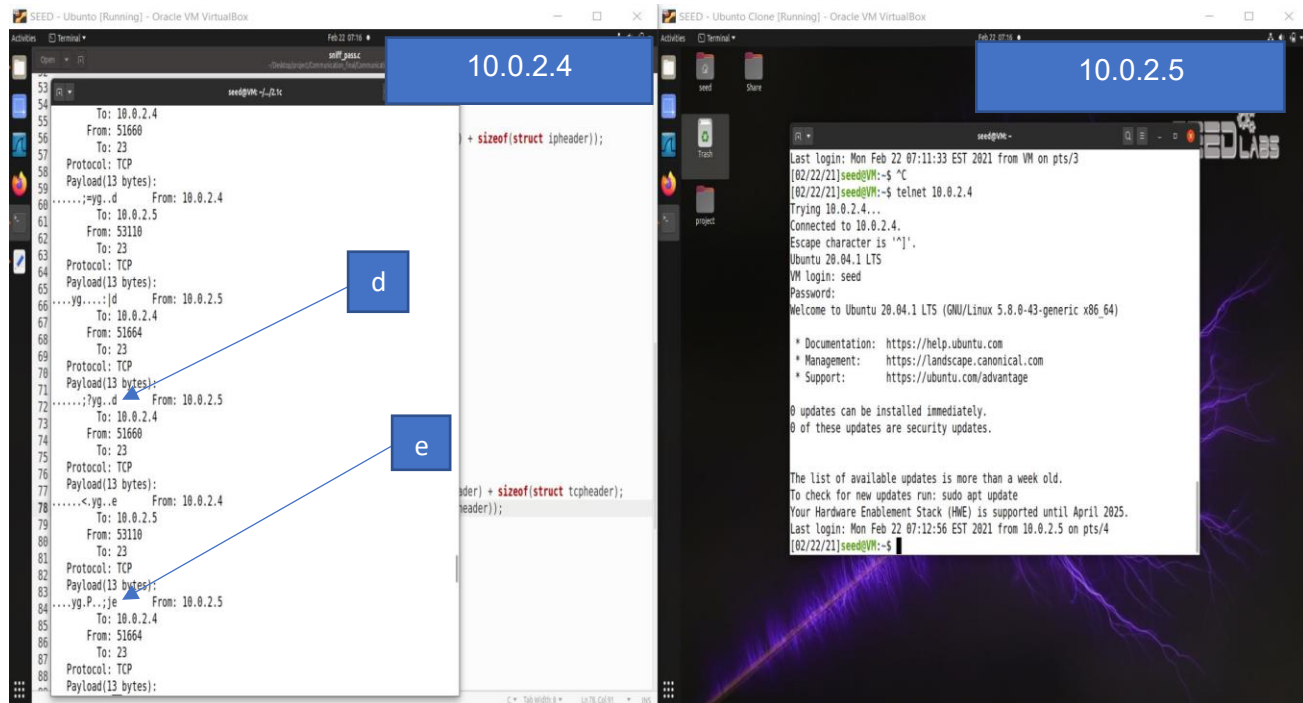
The image displays two terminal windows from a virtual machine environment, likely Oracle VM VirtualBox, running Ubuntu 20.04 LTS. The left window, titled 'SEED - Ubuntu [Running] - Oracle VM VirtualBox', shows a user named 'seed' at the prompt 'seed@VM: ~\$'. The user has executed the command 'sudo gcc sniff_filters.c -lpcap' and is running a series of 'tcpdump' commands to capture network traffic on interface 'eth0'. The output shows several ICMP echo requests (ping) from 10.0.2.5 to 10.0.2.4. A blue box with the text '10.0.2.4' is overlaid on the right side of this terminal. The right window, titled 'SEED - Ubuntu [Running] - Oracle VM VirtualBox', shows the same user at the prompt 'seed@VM: ~\$'. The user has executed 'ping -c 1 10.0.2.5' and is now running 'ping -c 1 10.0.2.4'. The output shows the ping statistics for 10.0.2.4, indicating 6 packets transmitted, 6 received, 0% packet loss, and a time of 5119ms. A blue box with the text '10.0.2.5' is overlaid on the right side of this terminal. Both windows show the standard Ubuntu login banner and system information.

בתמונה כאן אנחנו תופסים חבילות מסוג TCP ובטווח פורטים מסוים (10-100).



משימה 2.1C:

כאן אנחנו נרצה לתפוס פאקטה ולשלוח ממנה את הסיסמא שנשלחה. המידע ששלחנו עבר באמצעות בקשת telnet שהיא מסוג TCP ומכילה בתוכה את הסיסמא למכונה השולחת את הבקשה.



ניתן לראות בחצים שהצלחנו להוציא מכל חבילה את הדאטה שלה ובכל חבילה היה קיים חלק מהסיסמא כולה אותה חיפשנו.

משימה 2.2A:

כאן אנחנו כתבנו תוכנית שתעשה ספופינג ותשלח חבילה מסוג ICMP מכתובת IP כלשהי אל המכונה שלנו.

The screenshot shows a terminal window on the left and a Wireshark packet capture window on the right. The terminal window displays the command `sudo ./spoofer -lpcap` being executed in a shell with the prompt `seed@VM: ~/.../2.2A`. The Wireshark window shows a single packet capture with the following details:

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-02-22 08:33.127.0.0.53	127.0.0.53	10.0.2.4	ICMP	7184	Echo (ping) request

Two blue callout boxes with arrows point to the packet details:

- A box labeled "כתובת כלשהי" (Some address) points to the Source field (127.0.0.53).
- A box labeled "כתובת שלי" (My address) points to the Destination field (10.0.2.4).

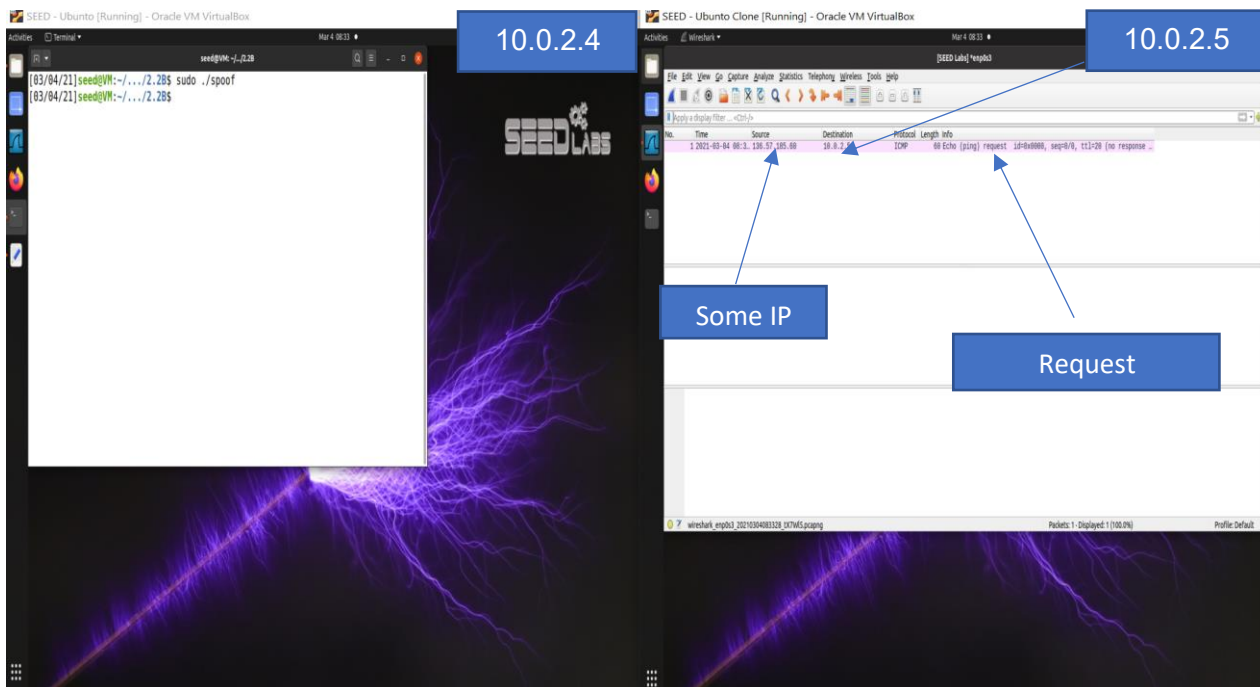
The packet details pane shows the following structure:

- Frame 1: 7184 bytes on wire (57472 bits) captured (57472 bits) on interface
- Linux cooked capture
- Internet Protocol Version 4, Src: 127.0.0.53, Dst: 10.0.2.4
- Internet Control Message Protocol

The packet bytes pane shows the raw data in hexadecimal and ASCII format.

משימה 2.2B:

במשימה הזאת אנחנו הפעלנו את המכונה הראשית (10.0.2.4) וממנה עשינו ספויפינג למכונה השנייה (10.0.2.5) מכתובת כלשהי.



שאלה 2.2.4:

לאחר שניסינו לשנות לגודל שרירותי כלשהו, אנו יכולים להסיק שאי אפשר מכיוון שהקוד רץ אבל שלח שגיאה כל פעם.

שאלה 2.2.5:

לא חייב, המערכת הפעלה דואגת לחשב לבד. בקוד שלנו ניתן לראות שאין חישוב של check sum.

שאלה 2.2.6:

אנו חייבים הרשאות root כי בתרגיל הזה אנחנו שלחנו פאקטות שאנחנו הרכבנו בsocket שפתחנו במיוחד. גישה כזאת לרכיבים של מערכת ההפעלה מצריכה כמובן, הרשאות גבוהות יותר.

משימה 2.3:

במשימה הזאת אנחנו עושים מאזינים ממכונה אחת (10.0.2.4) למכונה השנייה (10.0.2.5) ששולחת פינג לכתובת לא קיימת, ועבור כל חבילה שנתפוס אנחנו עושים ספווינג ושולחים הודעת reply למכונה השנייה מהכתובת שלא קיימת מהמכונה ראשונה.

The screenshot displays two virtual machines running Ubuntu. The left VM (10.0.2.4) is running a netcat listener on port 13. The right VM (10.0.2.5) is running a ping command to 1.2.3.4. The terminal output on the right shows the ping command and the resulting statistics: 4 packets transmitted, 0 received, 100% packet loss, time 3095ms. A blue box with the text "כתובת לא קיימת" (Invalid address) points to the ping command. Below the terminal, the Wireshark packet capture is shown, displaying 4 ICMP Echo (ping) request messages from 10.0.2.5 to 1.2.3.4. A blue box with the text "4 requests messages" points to these messages. Another blue box with the text "Reply messages for each request." points to the corresponding ICMP Echo (ping) reply messages in the capture. The bottom of the Wireshark window shows the packet details for the first request, including the Ethernet II, Internet Protocol Version 4, and Internet Control Message Protocol (ICMP) fields.