

פרוייקט הגשה – תקשורת ומחשוב

חלק 1:

1.1-a הסנפת פאקטות בשפת פייתון בעזרת scapy.

עם הרשאת רוט: ניתן לראות שהתוכנית עובדת ורצה כמו שצריך, כלומר קולטת את הפאקטות שעוברות בתעבורת הרשת.

```
seed@VM: ~$ 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=113 time=77.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=79.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=70.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=71.4 ms
^Z
[1]+  Stopped                  ping 8.8.8.8
seed@VM: ~$

[SEED Labs] enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
No. Time Source Destination Protocol Length Info
1 2021-02-24 10:4. 8.8.8.8 8.8.8.8 ICMP 98 Echo (ping) request id=0x0002, seq=0x0000
2 2021-02-24 10:4. 8.8.8.8 10.0.2.4 ICMP 98 Echo (ping) reply id=0x0002, seq=0x0000
3 2021-02-24 10:4. 8.8.8.8 8.8.8.8 ICMP 98 Echo (ping) request id=0x0002, seq=0x0000
4 2021-02-24 10:4. 8.8.8.8 10.0.2.4 ICMP 98 Echo (ping) reply id=0x0002, seq=0x0000
5 2021-02-24 10:4. 10.0.2.4 8.8.8.8 ICMP 98 Echo (ping) request id=0x0002, seq=0x0000
6 2021-02-24 10:4. 8.8.8.8 10.0.2.4 ICMP 98 Echo (ping) reply id=0x0002, seq=0x0000
7 2021-02-24 10:4. 10.0.2.4 8.8.8.8 ICMP 98 Echo (ping) request id=0x0002, seq=0x0000
8 2021-02-24 10:4. 8.8.8.8 10.0.2.4 ICMP 98 Echo (ping) reply id=0x0002, seq=0x0000

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_84:79:94 (08:00:27:84:79:94), Dst: RealtekU_12:35:00 (52:54:00:12:35:00)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 8.8.8.8
Internet Control Message Protocol

0000 52 54 00 12 35 00 08 00 27 84 79 94 08 00 45 00 RT: 5... 'y...E
0010 00 54 00 11 40 00 40 61 6e 84 8a 00 02 04 00 00 .T: @ @: n...
0020 08 08 08 00 a7 68 00 02 00 01 92 75 36 69 00 00 ....b...u6'
0030 00 00 c3 eb 05 00 00 00 00 00 10 11 12 13 14 15 .....
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 .....! "%$
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 &()'+,./012345
0060 36 37 67

Internet Control Message Protocol: Protocol Packets: 42 - Displayed: 8 (19.0%) Profile: Default

seed@VM: ~/Desktop$ sudo python3 sniffer.py
###[ Ethernet ]###
dst = 52:54:00:12:35:00
src = 08:00:27:84:79:94
type = IPv4
###[ IP ]###
version = 4
ihl = 5
tos = 0x0
len = 84
id = 45073
flags = DF
frag = 0
ttl = 64
proto = icmp
chksum = 0x6e84
src = 10.0.2.4
dst = 8.8.8.8
\options \
###[ ICMP ]###
type = echo-request
code = 0
chksum = 0xa768
id = 0x2
seq = 0x1
###[ Raw ]###
load = '\x92u6'\x00\x00\x00\x00\xc3\xeb\x05
\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
###[ Ethernet ]###
dst = 08:00:27:84:79:94
src = 52:54:00:12:35:00
type = IPv4
```

ללא הרשאת רוט: התוכנית קורסת, ניתן לדעת שזה יקרה כיוון שכאשר אין הרשאת רוט, זה אומר שאין גישה לכרטיס הרשת לפענח את התעבורה ולכן התוכנית תקרוס.

```
seed@VM: ~/Desktop$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 5, in <module>
    pkt = sniff(iface=['br-065667b75c49', 'docker0', 'enp0s3', 'lo'], 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 894, in _run
    sniff_sockets.update(
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 895, in <genexpr>
    (L2socket(type=ETH_P_ALL, iface=ifname, *arg, **karg),
  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, file
PermissionError: [Errno 1] Operation not permitted
[02/24/21]seed@VM: ~/Desktop$
```

לסיכום מה שניתן לראות הוא שכאשר יש הרשאת רוט ניתן לצפות בפאקטות העוברות ברשת בסביבה הביתית, וכאשר אין הרשאת רוט לא ניתן להסניף פאקטות ואז בעצם התוכנית קורסת. זה נובע כיוון שרק כאשר יש לנו הרשאה הכרטיס הרשת יכול להפוך למצב מוניטור.

-1.1 b

1. הסנפת פאקטות icmp בלבד בעזרת `.filter='icmp'`.

The screenshot displays a network sniffer interface with two main panes. The left pane shows the configuration of the sniffer, and the right pane shows the captured traffic.

Left Pane (Configuration):

```
[03/07/21]seed@VM: ~/.../1$ sudo python3 ./sniffer.py
sniffing packets
###[ Ethernet ]###
dst      = 52:54:00:12:35:00
src      = 08:00:27:fe:3e:a4
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 53881
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
checksum = 0x4c1c
src      = 10.0.2.4
dst      = 8.8.8.8
\options \
```

Right Pane (Traffic):

```
[03/07/21]seed@VM: ~/Desktop$ 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=115 time=46.3 ms
^Z
[2]+  Stopped                  ping 8.8.8.8
[03/07/21]seed@VM: ~/Desktop$
```

Bottom Pane (Packet Details):

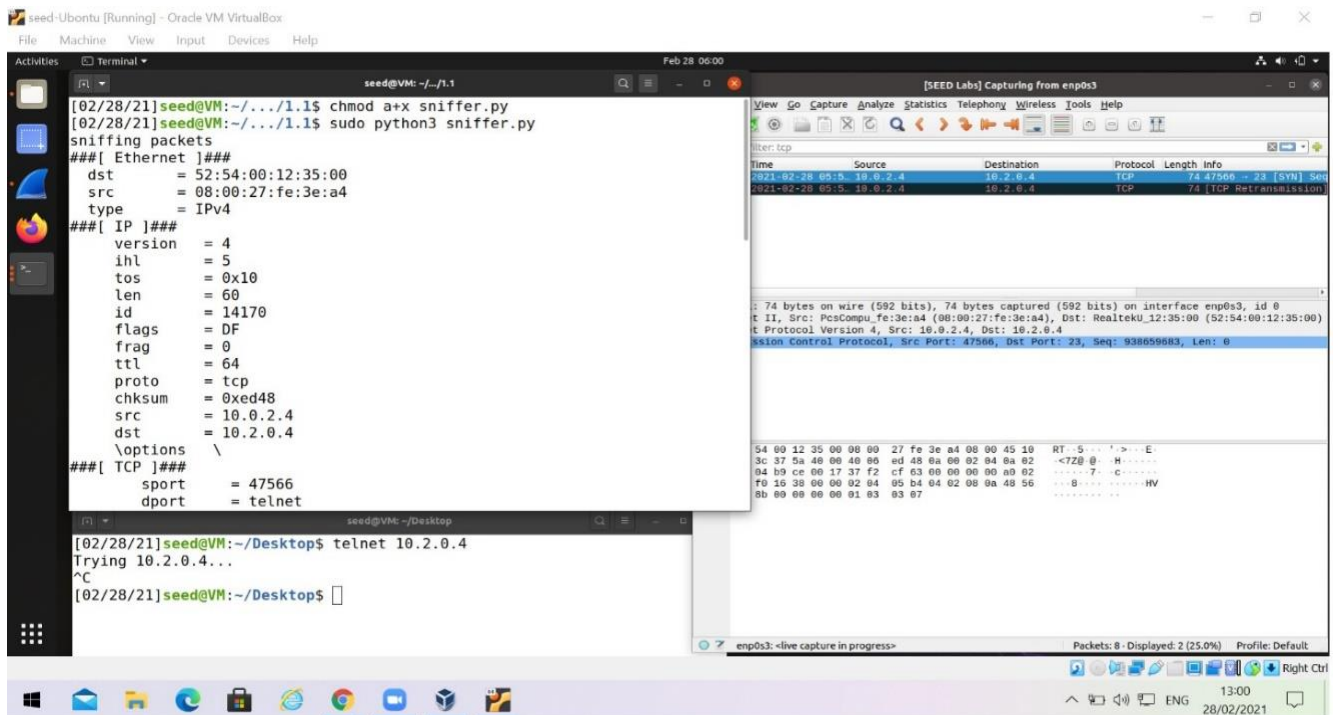
No.	Time	Source	Destination	Protocol	Length	Info
13	2021-03-07 03:43:58.813	10.0.2.4	8.8.8.8	ICMP	98	Echo (ping) request id=0x0004, seq=1/256, ttl=64 (reply in 14)
14	2021-03-07 03:43:58.859	8.8.8.8	10.0.2.4	ICMP	98	Echo (ping) reply id=0x0004, seq=1/256, ttl=115 (request in 13)

Frame 13: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_fe:3e:a4 (08:00:27:fe:3e:a4), Dst: RealtekU_12:35:00 (52:54:00:12:35:00)
Internet Protocol Version 4, Src: 10.0.2.4, Dst: 8.8.8.8
Internet Control Message Protocol

0000 52 54 00 12 35 00 08 00 27 fe 3e a4 08 00 45 00 RT..5...>...E.
0010 00 54 d2 79 40 00 40 01 4c 1c 0a 00 02 04 08 08 -T-y@-@- L-...
0020 08 08 08 00 9b cd 00 04 00 01 4e 92 44 60 00 00N.D`..
0030 00 00 fe 67 0c 00 00 00 00 00 10 11 12 13 14 15 ...g.....
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25!#\$%.

Internet Control Message Protocol: Protocol Packets: 14 · Displayed: 6 (42.9%) Profile: Default

2. הסנפת פאקטות מ-קו מסוים עם פורט 23(טלנט).



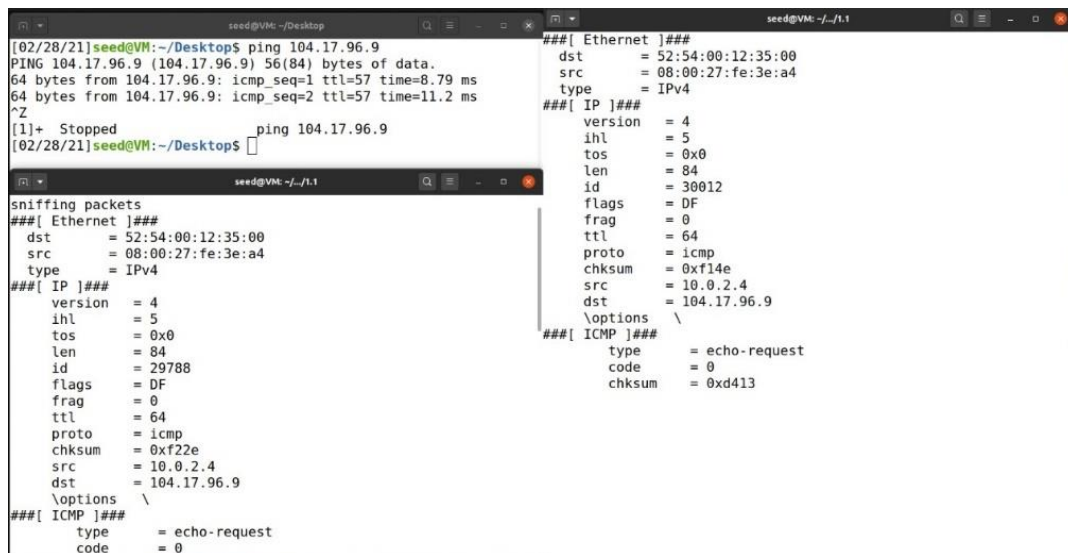
```
[02/28/21]seed@VM:~/1.1$ chmod a+x sniffer.py
[02/28/21]seed@VM:~/1.1$ sudo python3 sniffer.py
sniffing packets
##[ Ethernet ]##
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:fe:3e:a4
  type     = IPv4
##[ IP ]##
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 60
  id       = 14170
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0xed48
  src      = 10.0.2.4
  dst      = 10.2.0.4
  \options \
##[ TCP ]##
  sport    = 47566
  dport    = telnet

[02/28/21]seed@VM:~/Desktop$ telnet 10.2.0.4
Trying 10.2.0.4...
^C
[02/28/21]seed@VM:~/Desktop$
```

Wireshark capture details:

Time	Source	Destination	Protocol	Length	Info
0.000000	10.0.2.4	10.2.0.4	TCP	74	47566 → 23 [SYN] Seq=938659883
0.000000	10.0.2.4	10.2.0.4	TCP	74	TCP Retransmission

3. הסנפת פאקטות מסאבנט ספציפי, שהוא (104.17.96.0/8).



```
[02/28/21]seed@VM:~/Desktop$ ping 104.17.96.9
PING 104.17.96.9 (104.17.96.9) 56(84) bytes of data:
64 bytes from 104.17.96.9: icmp_seq=1 ttl=57 time=8.79 ms
64 bytes from 104.17.96.9: icmp_seq=2 ttl=57 time=11.2 ms
^Z
[1]+  Stopped                  ping 104.17.96.9
[02/28/21]seed@VM:~/Desktop$
```

Sniffing results:

```
##[ Ethernet ]##
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:fe:3e:a4
  type     = IPv4
##[ IP ]##
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 29788
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xf22e
  src      = 10.0.2.4
  dst      = 104.17.96.9
  \options \
##[ ICMP ]##
  type     = echo-request
  code     = 0
  chksum   = 0xd413
```

1.2

הסנפת פאקטת icmp request עם כתובת שרירותית, באמצעות רישם sudo python3 ואז הפעלת הפקודות הבאות:

```
>>> from scapy.all import *
```

```
>>> a = IP() ①
```

```
>>> a.dst = '10.0.2.3' ②
```

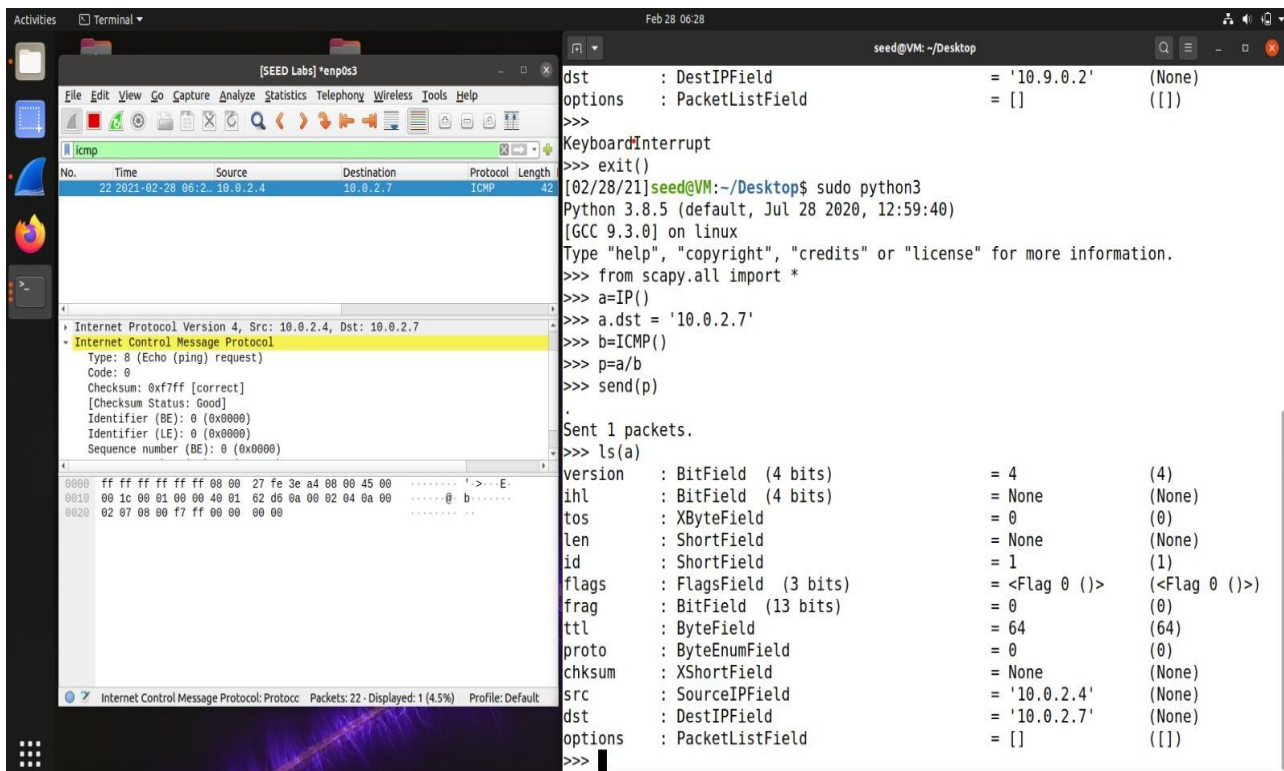
```
>>> b = ICMP() ③
```

```
>>> p = a/b ④
```

```
>>> send(p) ⑤
```

. Sent 1 packets

זה שולח פינג את הכתובת שאותה רשמנו a.dst = '10.0.2.3'



1.3 ttl: בתמונות ניתן לראות כי כאשר אנו משנים את ה-ttl של הפאקט ששולחים אז הפינג חזרה שאנחנו מקבלים הוא מגיע ממקום שונה, תחילה בדקנו עם טל 1 והראה שלא יצאנו מהרשת המקומית, אחר כך עם 2 הראה רשת כלשהי ואחר כך עם 3 הראה רשת אחרת. כלומר ttl קובע כמה נתבים הפקטה שאנו שולחים יכולה לעבור עד שנגמר ttl שלה. בתוכנית נריץ עבור 4 מקרים שונים ונראה כיצד היעד חזרה משתנה:

Ttl = 1 הגעה עד לנתב הקרוב אבל נתקעים שם, לא היעד שרצינו

The terminal window shows the following commands and output:

```
[02/28/21]seed@VM:~/1.3$ sudo python3 spoof.py
Traceback (most recent call last):
  File "spoof.py", line 1, in <module>
    a = IP()
NameError: name 'IP' is not defined
[02/28/21]seed@VM:~/1.3$ sudo python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> a=IP()
>>> a.dst = '1.2.3.4'
>>> a.ttl = 1
>>> b = ICMP()
>>> sent(a/b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sent' is not defined
>>> send(a/b)
.
Sent 1 packets.
>>>
```

The Wireshark capture shows two ICMP Echo (ping) requests from 10.0.2.1 to 1.2.3.4. Both requests are dropped by the destination (1.2.3.4) with a "Time-to-live exceeded" message.

Ttl = 2 ה תקדמות והגעה לנתב שונה אך שהוא איננו היעד האמיתי.

The terminal window shows the following commands and output:

```
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more info
rmation.
>>> from scapy.all import *
^[[A
>>> a=IP()
>>> a.dst = '1.2.3.4'
>>> a.ttl = 2
>>> b = ICMP()
>>> send(a/b)
.
Sent 1 packets.
>>>
```

The Wireshark capture shows two ICMP Echo (ping) requests from 10.0.2.1 to 1.2.3.4. The first request is dropped by the destination (1.2.3.4) with a "Time-to-live exceeded" message. The second request is received by the destination (1.2.3.4) and is an Echo (ping) request.

Ttl = 3 התקדמות והגעה לנתב שונה אך שהוא איננו היעד האמיתי.

The terminal window shows the execution of a Python script using Scapy to send a ping packet with a TTL of 3. The packet is sent from 212.179.37.1 to 1.2.3.4. The Wireshark capture shows the packet being received by the destination, which then responds with a 'Time-to-live exceeded' ICMP error message.

```
[02/28/21]seed@VM: ~/.../1.3$ sudo python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more info
>>> from scapy.all import *
>>> a=IP()
>>> a.ttl = 3
>>> b = ICMP()
>>> a.dst = '1.2.3.4'
>>> send(a/b)
.
Sent 1 packets.
>>>
```

Wireshark capture details:

- Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s3, id 0
- Ethernet II, Src: RealtekU_12:35:00 (52:54:00:12:35:00), Dst: PcsCompu_fe:3e:a4 (08:00:27:fe:3e:a4)
- Internet Protocol Version 4, Src: 212.179.37.1, Dst: 1.2.3.4
- Internet Control Message Protocol
- Type: 11 (Time-to-live exceeded)
- Code: 0 (Time to live exceeded in transit)
- Checksum: 0xf4ff [correct]
- [Checksum Status: Good]
- Unused: 00000000
- Internet Protocol Version 4, Src: 10.0.2.4, Dst: 1.2.3.4
- Internet Control Message Protocol
- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0xf7ff [unverified] [in ICMP error packet]
- [Checksum Status: Unverified]
- Identifier (BE): 0 (0x0000)
- Identifier (LE): 0 (0x0000)
- Sequence number (BE): 0 (0x0000)
- Sequence number (LE): 0 (0x0000)

Ttl = 15 הגעה ליעד הרצוי

The terminal window shows the execution of a Python script using Scapy to send a spoofed ping packet with a TTL of 15. The packet is sent from 10.0.2.4 to 8.8.8.8. The Wireshark capture shows the packet being received by the destination, which then responds with an 'Echo (ping) reply'.

```
[03/07/21]seed@VM: ~/.../1.3$ sudo python3 ./spoof.py
Sent 1 packets.
[03/07/21]seed@VM: ~/.../1.3$
```

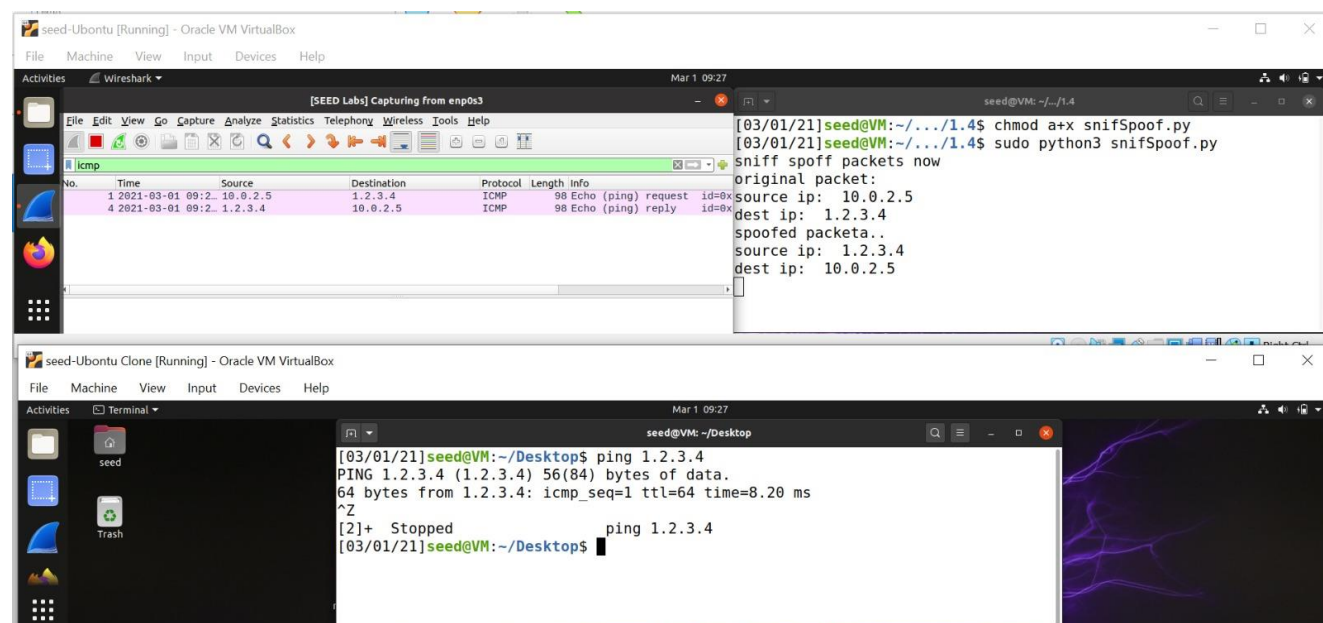
Wireshark capture details:

- Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface enp0s3, id 0
- Ethernet II, Src: PcsCompu_fe:3e:a4 (08:00:27:fe:3e:a4), Dst: RealtekU_12:35:00 (52:54:00:12:35:00)
- Internet Protocol Version 4, Src: 10.0.2.4, Dst: 192.168.1.1
- Internet Control Message Protocol
- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0xf7ff [unverified] [in ICMP error packet]
- [Checksum Status: Unverified]
- Identifier (BE): 0 (0x0000)
- Identifier (LE): 0 (0x0000)
- Sequence number (BE): 0 (0x0000)
- Sequence number (LE): 0 (0x0000)

1.4 .Sniffing and-then Spoofing

כאן בעצם יצרנו קוד אשר מחזיר לנו תשובה כלומר reply לפינג שאיננו קיים, כלומר לכל בקשה של icmp נקבל תשובה, בין אם המקור קיים או לא.

1. ping 1.2.3.4 # a non-existing host on the Internet : שליחת פינג אל 1.2.3.4 שהיא כתובת שלא קיימת אבל נקבל עליה תשובה בגלל התוכנית שבנינו.



2. ping 10.9.0.99 # a non-existing host on the LAN

בחלק זה כיוון שאין יעד כזה ברשת המקומית, נשלחת בקשה דרך פרוטוקול arp כדי לנסות למצוא האם מישהו מכיר את היעד או את הדרך אל היעד.

רשמנו בקוד filter='arp' ובתוך הפונקציה את השורה הבאה:

For ARP in pkt and pkt[ARP].op == 1

שהיא אמורה לגשת רק כאשר נשלחת פקטה דרך פרוטוקול ארפ ושאתה ננסה לעבוד ולהחזיר ארפ/פינג למקום שממנו נשלחה הבקשה הראשונית, אך לא הצלחנו. בנוסף ניסינו גם לעבור לראות איך המידע הפנימי של פקטת ה-arp בנוי (כפי שניתן לראות בתמונה למטה) ועובר ברשת ואולי איתו לעבד את המידע נכון ולשלוח חזרה פינג כמו שצריך, אך גם כאן סיימנו ללא הצלחה.

Writing Packet Sniffing Program 2.1

Understanding How a Sniffer Works– a 2.1

2.1.A

שאלה 1. `Open_live_pcap` מחזיר פאקטה שנקלטת מתעבורת הרשת ונקלטת על ידי המשתנה `handle`, הפונקציה מקבלת את הממשק של כרטיס הרשת, את גודל המקסימאלי למידע לפאקטה ובאפר המקבל מידע במקרא וקרטה שגיאה.

`Pcap_compile` מכניסה את המידע ממחרוזת הנקלטת בפונקציה אל האובייקט ובאמצעותו מגדירים את הפילטר, ובנוסף בודקת האם הפקטה בעלת ערכי רשת מסוג מסוים.

`pcap_setfilter` מריץ את האובייקט באמצעות הפילטר שהוכנס בקומפילציה.

`Pcap_loop` גורמת לקוד להמשיך לרוץ ולקלוט פאקטות שעוברות ברשת.

`pcap_close` סוגר את קליטת הפאקטות.

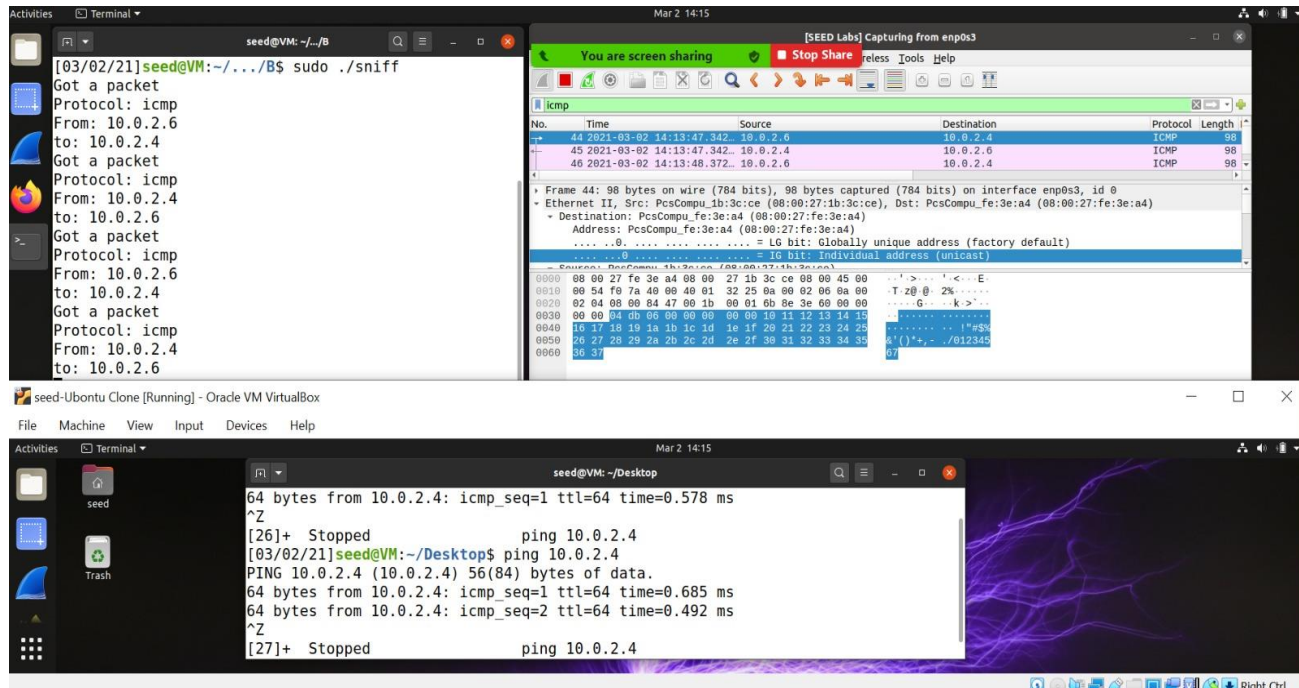
שאלה 2. נצטרך את הרשאות ה `root` כי ללא הרשאה זו המחשב אינו נותן גישה לשימוש בכרטיס הרשת להסנפה של מידע העובר בתעבורת הרשת. ללא הרשאות אלה התוכנית מקבלת שגיאת `segmentation fault` בזמן הריצה, מתקבלת הודעה שאין הרשאות שימוש בכרטיס הרשת.

שאלה 3. ההבדל במקרים כאשר `promiscuous mode` פועל הוא שיש תעבורה ברשת שאינה מיועדת אל המחשב יהיה זיהוי של המידע למרות שלא נשלח אל הכתובת שלנו, וכאשר `promiscuous mode` אינו פועל כרטיס הרשת לא יזהה את התעבורה אלא אם מיועדת אליו.

לדוגמא כאשר `promiscuous mode` מופעל במכונה אחת ונשלח ממוכנה אחרת פינג אל www.google.com נוכל לראות את הפינג שעבר דרך מכונה 1 וכאשר ה `promiscuous mode` אינו מופעל לא נוכל לראות את ההודעה שנשלחה ממכונה 1.

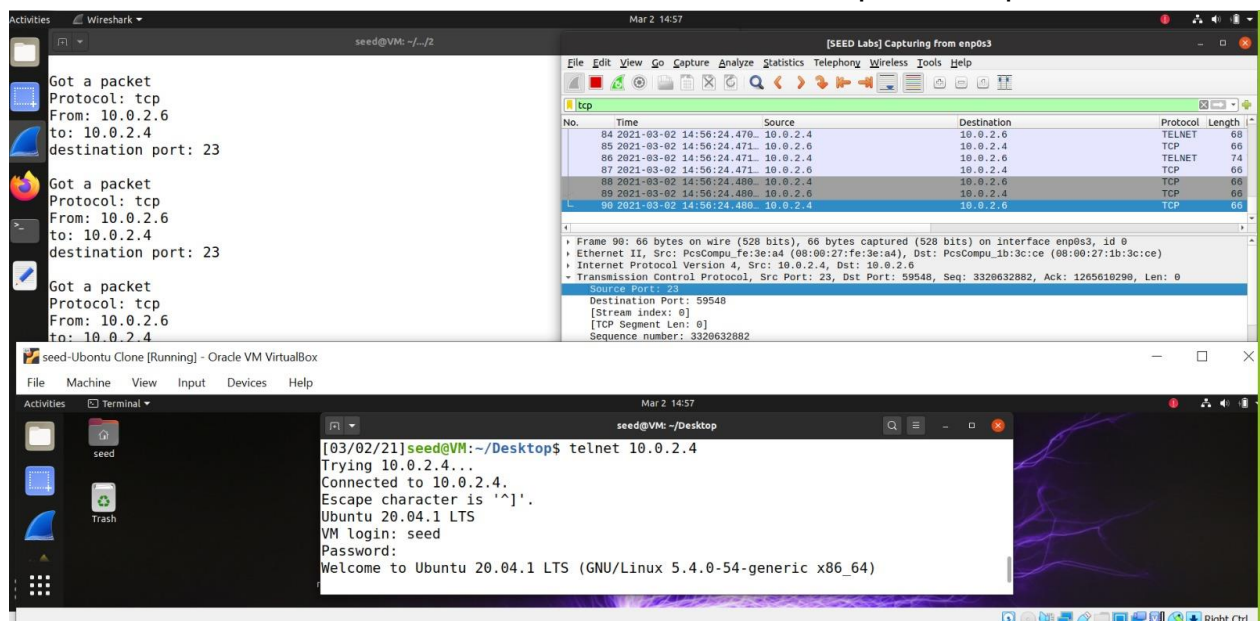
– B 2.1

- שאלה 1 - Capture the ICMP packets between two specific hosts
- שלחנו icmp מ 10.0.2.6 אל 10.0.2.4. כאשר בכתובת 10.0.2.4 רצה התוכנית שמזהה פאקטות



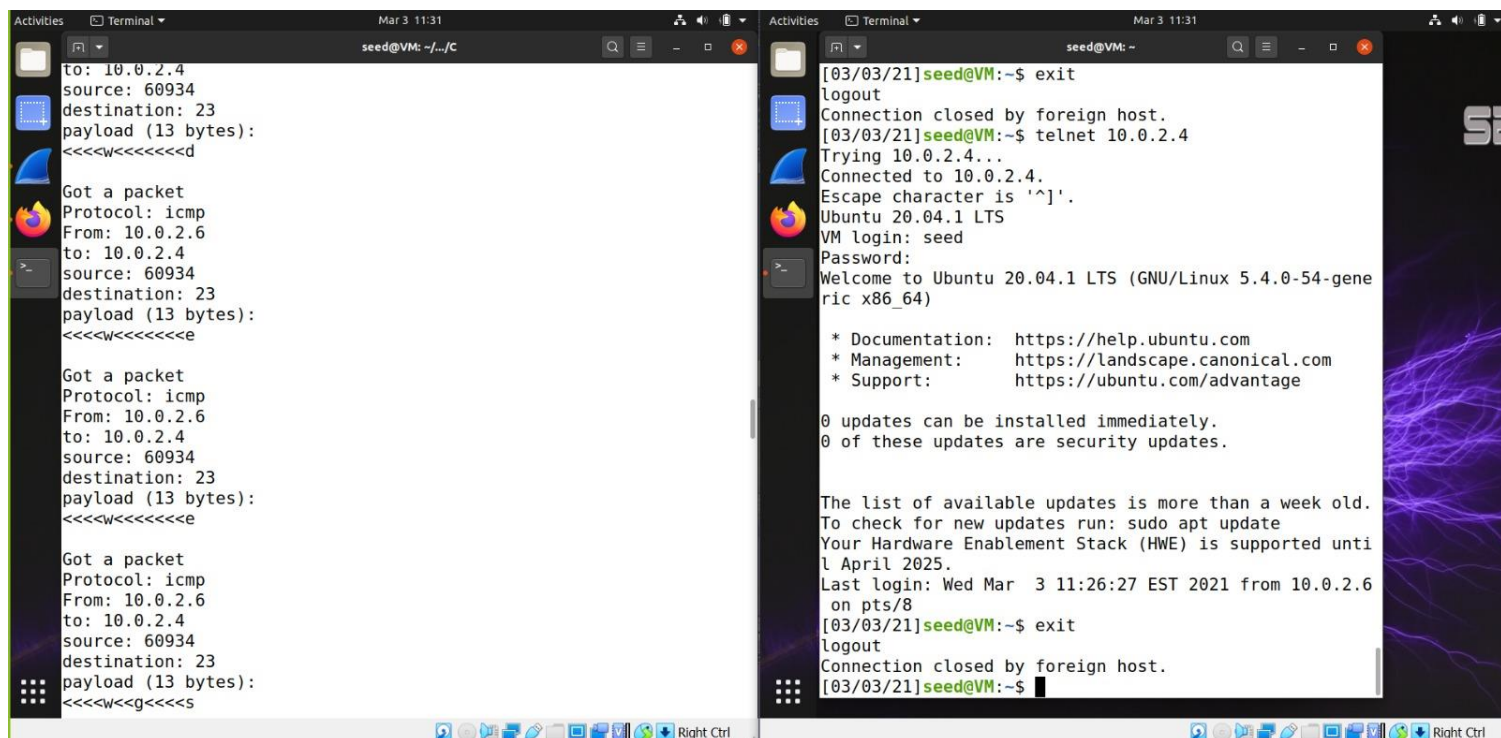
שאלה 2 - Capture the TCP packets with a destination port number in the range from 10 to 100.

עשינו זאת באמצעות העברת הפאקטה דרך פורט 23 (טלנט). והתוכנית מזהה רק פאקטות שעוברות דרך פורטים בין 10-100.



– c 2.1

Sniffing Passwords : בחלק זה העברנו פאקטה דרך פורט 23 (טלנט) שבו צריך לרשום משתמש וסיסמא, והמטרה בחלק זה היא לנסות לגלות מהי הסיסמא שאותה רושמים. הסיסמא נמצאת בסוף ה-payload של הפאקטה. כלומר ניתן לראות את התו בשורה האחרונה של כל חלק כאשר הסיסמא שאותה רשמנו היא dees.



The image displays two side-by-side terminal windows from a virtual machine named 'seed@VM:'. The left terminal shows a packet capture of an ICMP echo request (ping) from 10.0.2.6 to 10.0.2.4. The packet details include source (60934), destination (23), and a 13-byte payload represented by a series of less-than signs (<). The right terminal shows a telnet session to 10.0.2.4. The user 'seed' logs in and is prompted for a password. The terminal output shows the system is Ubuntu 20.04.1 LTS and displays system updates. The session ends with the user typing 'exit' and the connection being closed.

```
to: 10.0.2.4
source: 60934
destination: 23
payload (13 bytes):
<<<<w<<<<<<<<

Got a packet
Protocol: icmp
From: 10.0.2.6
to: 10.0.2.4
source: 60934
destination: 23
payload (13 bytes):
<<<<w<<<<<<<<

Got a packet
Protocol: icmp
From: 10.0.2.6
to: 10.0.2.4
source: 60934
destination: 23
payload (13 bytes):
<<<<w<<<<g<<<<s

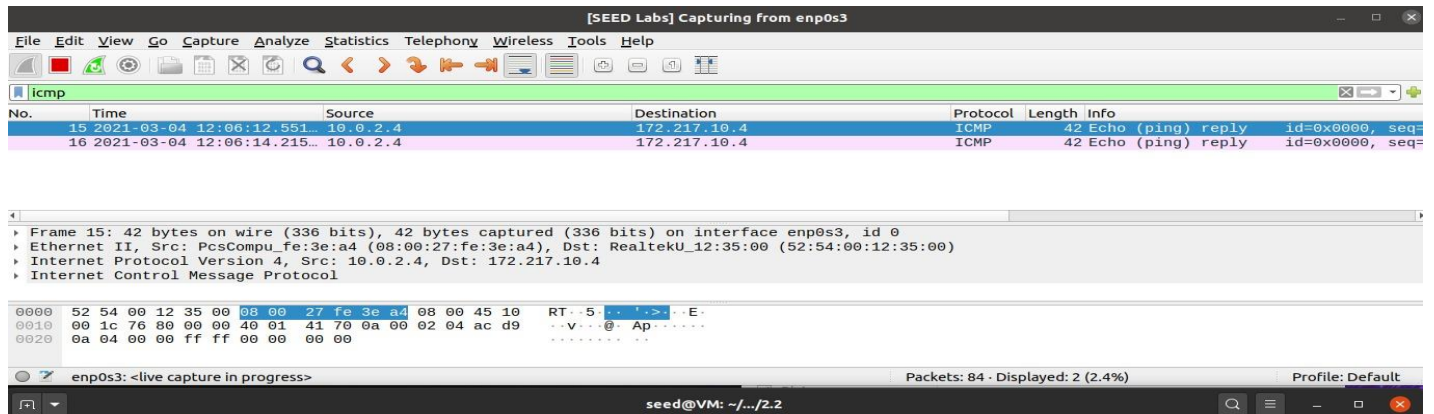
[03/03/21]seed@VM:~$ exit
logout
Connection closed by foreign host.
[03/03/21]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 updates can be installed immediately.
0 of these updates are security updates.

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Wed Mar  3 11:26:27 EST 2021 from 10.0.2.6 on pts/8
[03/03/21]seed@VM:~$ exit
logout
Connection closed by foreign host.
[03/03/21]seed@VM:~$
```

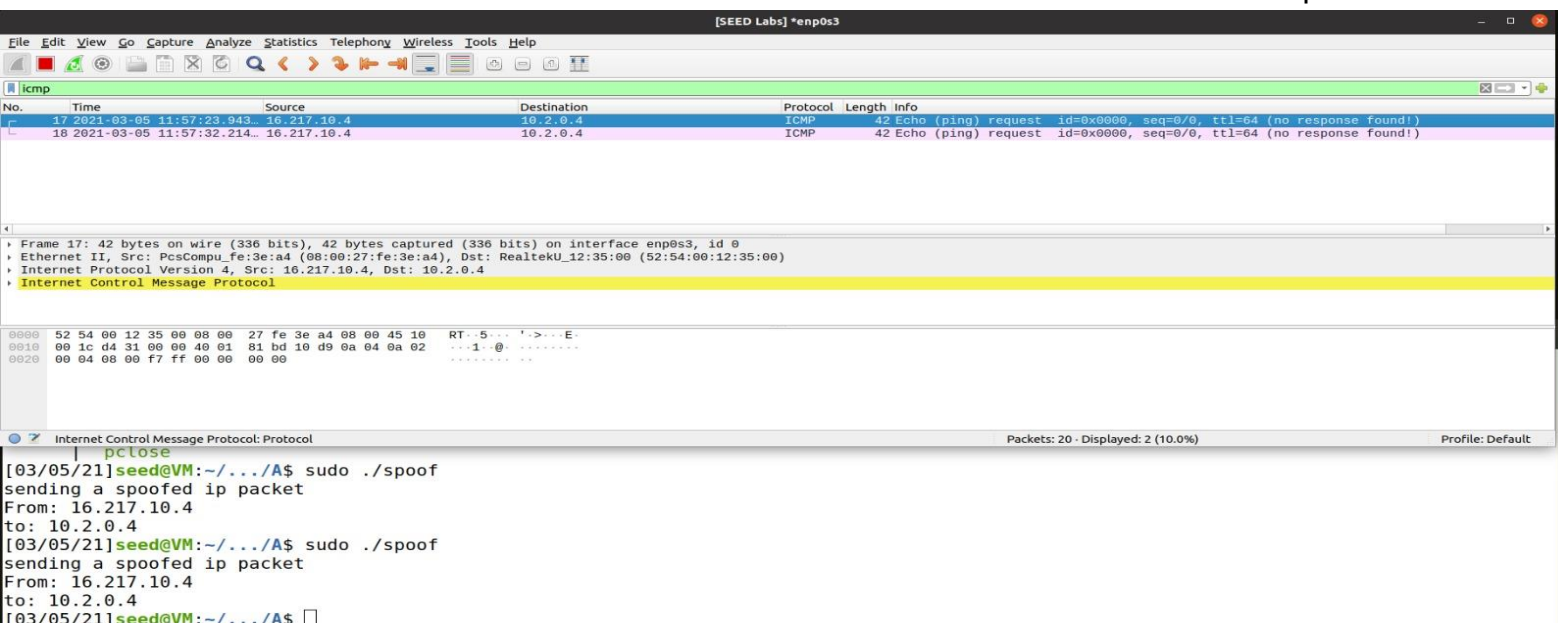
2.2 A – Write a spoofing program : בחלק זה המטרה היא השתלת פאקטות שקריות לתוך הרשת, כלומר שולחים פינג מסוג ריפליי למרות שמעולם לא קיבלנו פינג ריקווסט אל הכתובת הנוכחית. הרצנו את התוכנית פעמיים וראינו כי נשלחו 2 פאקטות "תשובה" מכתובת 10.0.2.4 אל כתובת 172.217.10.4. עשינו זאת באמצעות כתיבת קוד שיוצר פאקטה מסוג פינג "תשובה" שאותה שלחנו אל הכתובת הרצויה.



```
[03/04/21]seed@VM:~/.../2.2$ sudo ./spoof
sending a spoofed ip packet
From: 172.217.10.4
to: 10.0.2.4
[03/04/21]seed@VM:~/.../2.2$ sudo ./spoof
sending a spoofed ip packet
From: 172.217.10.4
to: 10.0.2.4
[03/04/21]seed@VM:~/.../2.2$
```

2.2 B –

בחלק זה התבקשנו ליצור שאלה מזויפת, שאותה שלחנו מכתובת אחרת אל הכתובת שלנו, והראנו את התעבורה דרך הווירשארק שבו ניתן לראות שעברו 2 פינג מסוג ריקווסט אל המחשב שלנו (10.0.2.4). כלומר שלחנו תעבורה שקרית ברשת. באמצעות יצירת פינג "בקשה" שאותה שלחנו אל עבר הכתובת הרצויה.



```
[03/05/21]seed@VM:~/.../A$ sudo ./spoof
sending a spoofed ip packet
From: 16.217.10.4
to: 10.2.0.4
[03/05/21]seed@VM:~/.../A$ sudo ./spoof
sending a spoofed ip packet
From: 16.217.10.4
to: 10.2.0.4
[03/05/21]seed@VM:~/.../A$
```


שאלה 4. - ניתן לבחור גודל אקראי העיקר שיהיה בגודל מינימאלי של אורך הפאקטה. כלומר לא קטן ממש מהגודל של סטראקט אייפי הדר, אצלנו יצא גודל מינימום 7168 כיוון שמשתמשים בפונקציית .htons.

שאלה 5. כן, יש חובה לחשב את ה-checksum כראוי, כיוון שזה אחראי לבדוק את האמינות של הפקטות שעוברות בתעבורת הרשת, כלומר לפעמים קורות תקלות כמו שליחה כפולה של ביטים או פספוס ואי שליחה של ביטים, כלומר שליחה של מידע לא זהה למידע שברצוננו להעביר, אז הצ'קסאם בודק בעצם שלא יקרו שגיאות שכאלה, באמצעות מנגנון שמזהה את אותן חריגות באמצעות חישוב הדאטה שנשלח והמספר ששמור באייפי הדר במקום של הצקסאם ומשווה אם זהים אז המידע עבר כראוי ואין חריגות והפקטה אמינה. לכן על המתכנת לחשב את הצקסאם כדי לבדוק שלא נוצרו חריגות בשליחת הפקטה דרך האינטרנט.

שאלה 6. - נצטרך את הרשאות ה-root כי ללא הרשאה זו אין גישה להריץ תוכניות המשתמשות בraw sockets. התוכנית תרוץ עד שנגיע לשורה בה רשום

```
int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)
```

או כל שימוש אחר בraw sockets. כי למשתמש רגיל אין גישה להשתמש במידע שקיים חלק זה.

חלק 2.3 - Sniff and then Spoof

בחלק זה אנו צריכים להפעיל גם את הסניפר שרשמנו בחלק 2.1 B ואחר כך להפעיל את הספופר שרשמנו בחלק 2.2 A. כלומר לזהות בתעבורת הרשת פקטה מסוג פינג ואחר כך להפעיל את הספופר שהוא יזייף לנו תשובה (reply) מהמקום שאליו נשלח הפינג. בויירשארק ניתן לראות ששלחנו פינג את כתובת 1.2.3.4 ממכונה 1, ממכונה 2 הפעלנו את התוכנית שלנו. התוכנית "החזירה" תשובה למכונה 1 מכתובת 1.2.3.4 למרות שאין כזו כתובת.

```
seed@VM: ~/Desktop
08/21]seed@VM:~/Desktop$ ping 1.2.3.4
1.2.3.4 (1.2.3.4) 56(84) bytes of data.

Stopped ping 1.2.3.4
08/21]seed@VM:~/Desktop$
```

```
sniff-and-spoof.c:116:2: warning: implicit declaration of function 'close'; did you mean 'pclose'? [-Wimplicit-function-declaration]
116 |     close(sock);
    |     ^~~~~
    |     pclose
[03/08/21]seed@VM:~/.../g$ sudo ./sniff-and-spoof
Got a packet
Protocol: icmp
From: 10.0.2.6
to: 1.2.3.4
sending a spoofed ip packet
From: 1.2.3.4
to: 10.0.2.6
```

Time	Source	Destination	Protocol	Length	Info
1	2021-03-08 09:06:23.321	10.0.2.6	ICMP	98	Echo (ping) request id=0x0001, seq=1/
4	2021-03-08 09:06:24.245...	1.2.3.4	ICMP	60	Echo (ping) reply id=0x0000, seq=0/

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0
Internet II, Src: PcsCompu_1b:3c:ce (08:00:27:1b:3c:ce), Dst: RealtekU_12:35:00 (52:54:00:12:35:00)
Internet Protocol Version 4, Src: 10.0.2.6, Dst: 1.2.3.4
Internet Control Message Protocol

Internet Control Message Protocol: Protocol Packets: 28 - Displayed: 2 (7.1%) Profile: Default