

## פרויקט ברשתות

הקמנו את כל המכונות הוירטואליות כמפורט לפי המדריכים באתר בעזרת docker ו virtualbox

### מטלת פייתון:

א.1. בסעיף זה לקחנו את הקוד שהביאו לנו במטלה על מנת להסניף את התעבורה ברשת.

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-b0536f20c0fc', filter='icmp', prn=print_pkt)
```

כמו שנסביר בהמשך כאשר אנחנו מריצים ללא הרשאות מנהל התוכנית תיכשל מכיוון שכדי להסניף אנחנו למצב הסנפה שדורש הרשאות מנהל ולכן זה נכשל.

כאשר מריצים בעזרת הרשאות מנהל נקבל את התוצאה הבאה :

```

###[ Ethernet ]###
  dst      = 02:42:0a:09:00:05
  src      = 02:42:e5:e0:58:91
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 60593
  flags    =
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x79e0
  src      = 10.9.0.1
  dst      = 10.9.0.5
  \options \
###[ ICMP ]###
  type     = echo-reply
  code     = 0
  chksum   = 0x4398
  id       = 0x108
  seq      = 0x2
###[ Raw ]###
  load     = '\x83\xa6:\x00\x00\x00\x00;\x84\x03\x00\x00
\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1
d\x1e\x1f !"#$%&\'()*+,-./01234567'

```

א.2. בסעיף הקודם הראנו שאנו מקבלים פקטות ICMP (PING)

לאחר מכן התבקשנו לתפוס רק פקטות TCP מIP מסוים ופורט יעד 23

```
pkt = sniff(iface='br-b0536f20c0fc', filter='tcp and src host 10.9.0.5 and dst port 23', prn=print_pkt)
```

בדקנו את זה בעזרת פקודת nc לפורט 23 וקיבלנו את התוצאה הבאה

```
###[ Ethernet ]###
  dst      = 02:42:e5:e0:58:91
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 52
  id       = 61448
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0x36a4
  src      = 10.9.0.5
  dst      = 10.9.0.1
  \options \
###[ TCP ]###
  sport    = 41540
  dport    = telnet
  seq      = 351793205
  ack      = 2980522090
  dataofs  = 8
  reserved = 0
  flags    = A
  window   = 502
  chksum   = 0x143e
  urgptr   = 0
  options  = [('NOP', None), ('NOP', None), ('Timestamp', (4
234156178, 3833419424))]
```

ולבסוף התבקשנו לעשות פילטור לפי סאבנט אז בחרנו את הפילטור הבא:

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='enp0s3', filter='dst net 128.230.0.0/16', prn=print_pkt)
```

ולאחר פקודת פינג למקום המתאים לפילטור שהגדרנו מקודם קיבלנו את התוצאה הבאה:

**PING 128.230.1.2 (128.230.1.2) 56(84) bytes of data.**

```
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:63:12:72
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 36035
  flags    = DF
  frag     = 0
  ttl      = 63
  proto    = icmp
  chksum   = 0x20f8
  src      = 10.0.2.6
  dst      = 128.230.1.2
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x9a9
  id       = 0x112
  seq      = 0x2
###[ Raw ]###
  load     = '\xaf\xa9:\x00\x00\x00\x00<\x08\x00\x00\x00
0\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x
1e\x1f !"#%&\'()*+,-./01234567'
```

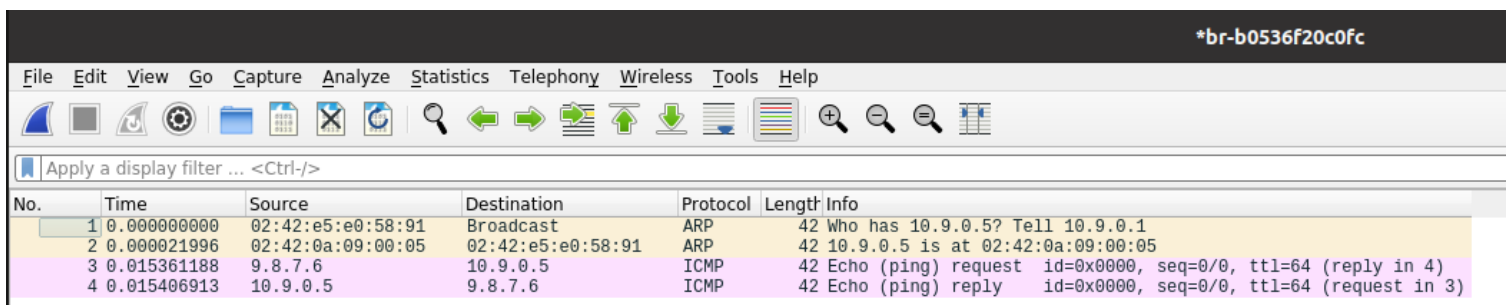
ב. בסעיף זה אנחנו מזייפים פקטות מסוג ICMP בעזרת סקאפי ונצרך תמונה מ wireshark שתראה שבאמת אנחנו גם מקבלים תשובה כלומר הזיוף עבד. הקוד שרשמנו בשביל לזייף:

```
#!/usr/bin/env python3
from scapy.all import *

ip = IP()
ip.dst = '10.9.0.5'
ip.src = '9.8.7.6'

icmp = ICMP()
packet = ip/icmp
send(packet)
```

תמונת הקלטת הרשת מwireshark שמראה שקיבלנו פינג חזרה.



The image shows a Wireshark network traffic capture. The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help), a toolbar with various icons, and a display filter bar showing 'Apply a display filter ... <Ctrl-/>'. The packet list table below shows four captured packets:

| No. | Time         | Source            | Destination       | Protocol | Length | Info  |
|-----|--------------|-------------------|-------------------|----------|--------|---|
| 1   | 0.0000000000 | 02:42:e5:e0:58:91 | Broadcast         | ARP      | 42     | Who has 10.9.0.5? Tell 10.9.0.1                             |
| 2   | 0.000021996  | 02:42:0a:09:00:05 | 02:42:e5:e0:58:91 | ARP      | 42     | 10.9.0.5 is at 02:42:0a:09:00:05                            |
| 3   | 0.015361188  | 9.8.7.6           | 10.9.0.5          | ICMP     | 42     | Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 4) |
| 4   | 0.015406913  | 10.9.0.5          | 9.8.7.6           | ICMP     | 42     | Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 3) |

ג.

בסעיף זה נממש את הכלי traceroute באמצעות סקאפי באופן ידני.

כל פעם נשלח פקטה עם ttl גבוה יותר עד שהפינג יעבוד וכך נדע כמה תחנות עברנו בדרך.

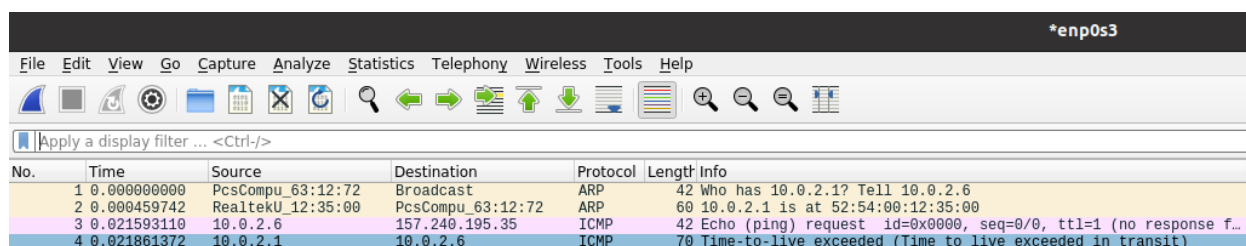
קוד:

```
#!/usr/bin/env python3
from scapy.all import *

ip = IP()
ip.dst = '157.240.195.35'
ip.ttl = 1

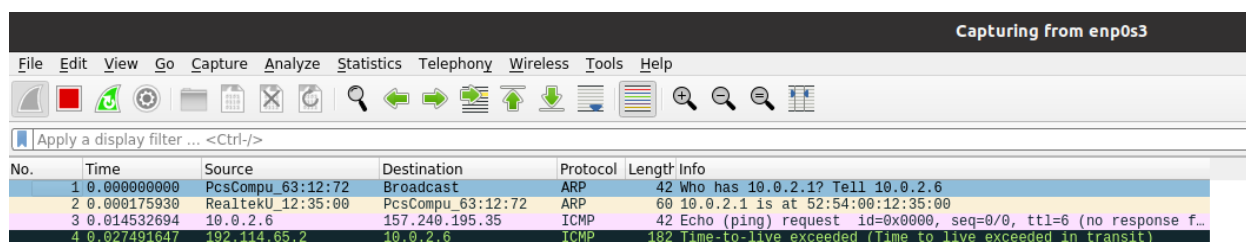
icmp = ICMP()
packet = ip/icmp
send(packet)
```

עבור ttl = 1



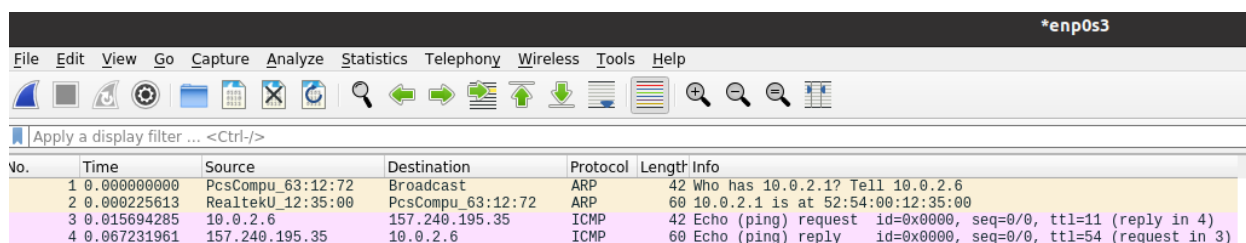
| No. | Time        | Source            | Destination       | Protocol | Length | Info  |
|-----|-------------|-------------------|-------------------|----------|--------|---|
| 1   | 0.000000000 | PcsCompu_63:12:72 | Broadcast         | ARP      | 42     | Who has 10.0.2.1? Tell 10.0.2.6                                 |
| 2   | 0.000459742 | RealtekU_12:35:00 | PcsCompu_63:12:72 | ARP      | 60     | 10.0.2.1 is at 52:54:00:12:35:00                                |
| 3   | 0.021593110 | 10.0.2.6          | 157.240.195.35    | ICMP     | 42     | Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response f... |
| 4   | 0.021861372 | 10.0.2.1          | 10.0.2.6          | ICMP     | 70     | Time-to-live exceeded (Time to live exceeded in transit)        |

עבור ttl = 6



| No. | Time        | Source            | Destination       | Protocol | Length | Info  |
|-----|-------------|-------------------|-------------------|----------|--------|---|
| 1   | 0.000000000 | PcsCompu_63:12:72 | Broadcast         | ARP      | 42     | Who has 10.0.2.1? Tell 10.0.2.6                                 |
| 2   | 0.000175930 | RealtekU_12:35:00 | PcsCompu_63:12:72 | ARP      | 60     | 10.0.2.1 is at 52:54:00:12:35:00                                |
| 3   | 0.014532694 | 10.0.2.6          | 157.240.195.35    | ICMP     | 42     | Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response f... |
| 4   | 0.027491647 | 192.114.65.2      | 10.0.2.6          | ICMP     | 182    | Time-to-live exceeded (Time to live exceeded in transit)        |

וכך עד שנגיע ל ttl=11 ונקבל תשובה:



| No. | Time        | Source            | Destination       | Protocol | Length | Info  |
|-----|-------------|-------------------|-------------------|----------|--------|---|
| 1   | 0.000000000 | PcsCompu_63:12:72 | Broadcast         | ARP      | 42     | Who has 10.0.2.1? Tell 10.0.2.6                             |
| 2   | 0.000225613 | RealtekU_12:35:00 | PcsCompu_63:12:72 | ARP      | 60     | 10.0.2.1 is at 52:54:00:12:35:00                            |
| 3   | 0.015694285 | 10.0.2.6          | 157.240.195.35    | ICMP     | 42     | Echo (ping) request id=0x0000, seq=0/0, ttl=11 (reply in 4) |
| 4   | 0.067231961 | 157.240.195.35    | 10.0.2.6          | ICMP     | 60     | Echo (ping) reply id=0x0000, seq=0/0, ttl=54 (request in 3) |

ולכן ניתן להסיק כי יש 11 תחנות בדרך

.ד

התבקשנו כעת להאזין לנתקף שלנו ועל כל בקשת פינג שיוצאת ממנו להחזיר לו מיד

```
#!/usr/bin/env python3
from scapy.all import *

ECHO_REQUEST = 8
ECHO_REPLY = 0

def spoof_packet(pkt):
    if ICMP in pkt and pkt[ICMP].type == ECHO_REQUEST:
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        icmp = ICMP(type=ECHO_REPLY, seq=pkt[ICMP].seq, id=pkt[ICMP].id)
        payload = pkt[Raw].load

        spoofedpkt = ip/icmp/payload
        send(spoofedpkt)
    pkt.show()

pkt = sniff(iface='br-b0536f20c0fc', filter='icmp and src host 10.9.0.5', prn=spoof_packet)
```

הסבר: אנחנו לוקחים כל פקטת ICMP מהנתקף מתאימים את כל השדות הרלוונטים והופכים מקור ויעד כדי שהוא יחשוב שקיבל תשובה אמיתית.

תוצאה של שליחת פינג לאייפי שלא קיים:

```
root@5f8661d49810:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
```

ולאחר הפעלת התוכנית זיוף:

```
root@5f8661d49810:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=53.2 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=25.9 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=13.6 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=21.2 ms
^C
--- 1.2.3.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 13.566/28.471/53.150/14.916 ms
```

עבור כתובת על אותו LAN נקבל את התוצאה הבאה:

```
root@5f8661d49810:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4099ms
pipe 4
```

ניתן לראות שהפקטה לא הגיעה. וזה כי הכתובת היא באותו LAN של המכונה שלנו ולכן פרוטוקול ARP בא לעזרתנו כדי לקבל את הMAC של המחשב בעל האיפיי שאנחנו מנסים לגשת אליו. כמובן שהוא לא קיים ולכן לא מקבלים תשובה חזרה והפקטה נזרקת ולא עוברת אצלו בתוכנית הסנפה ולכן לא מקבלת תשובה חזרה לפקטת הפינג שנשלחה

כעת רצינו לבדוק מה יקרה כאשר נשלח פינג למחשב שקיים:

```
root@5f8661d49810:/# 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=114 time=56.4 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=68.8 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=27.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=60.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=18.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=65.6 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 18.879/49.459/68.796/19.271 ms
```

כמובן שקיבלנו פעמיים reply עבור כל בקשת request שנשלחת (פעם אחת מהמחשב האמיתי ופעם אחת מהתוכנית שלנו)



## מטלה C:

### **א.1.**

פה אנו רוצים שוב להסניף לחבילות ולהדפיס את האייפי של המקור והיעד.

עשינו זאת באמצעות התוכנית הבאה (לקחנו שדות מאתר שהופנה אליו בעבודה ושמנו אצלנו בקוד כדי לגשת לשדות בתוך המבנים שמנהלים את הרשת) כל הקוד מצורף בסוף ופה נוסיף רק את הדברים החשובים:

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct sniff_ip *ip = (struct sniff_ip *) (packet + sizeof(struct sniff_ethernet));

    printf("Got a packet\n");
    printf("source IP: %s\n", inet_ntoa(ip->ip_src));
    printf("destination IP: %s\n", inet_ntoa(ip->ip_dst));
}
```

ומתקבלת התוצאה הבאה:

```
Got a packet
source IP: 10.9.0.5
destination IP: 10.9.0.1
Got a packet
source IP: 10.9.0.1
destination IP: 10.9.0.5
Got a packet
source IP: 10.9.0.5
destination IP: 10.9.0.1
```

## שאלות:

1. הפונקציה הראשונה שנקראת היא pcap\_open\_live זוהי פונקציה שפותחת את האינטרפייס למוד האזנה. לאחר מכן נקראת pcap\_compile שמכין את הפילטר שנכנס בpcap\_setfilter . ולבסוף קוראים ל pcap\_loop ובכך מתחילים להאזין לפקטות.
2. כמו שתיארנו בשאלה 1 כדי להיכנס למוד האזנה אנחנו צריכים הרשאות מנהל ובמידה ולא יהיה לנו התוכנית תקרוס בשורה שמנסה להעביר אותנו לשם, pcap\_open\_live.
3. כאשר מכבים את מוד ההאזנה אנחנו לא יכולים לקבל את הפקטות כי האינטרפייס לא במוד האזנה ולכן לא ניתן יהיה לתפוס פקטות

## 2.א

נוסיף פילטור שמתאים לסעיף:

```
"proto ICMP and (host 10.9.0.5 and 8.8.8.8)"
```

נצרף רשימת פרוטוקולים של IP שתשמש בהמשך :

| Hex  | Protocol Number | Keyword     | Protocol  | References/RFC                                      |
|------|-----------------|-------------|---|---|
| 0x00 | 0               | HOPOPT      | IPv6 Hop-by-Hop Option  | <a href="#">RFC 8200</a>                            |
| 0x01 | 1               | ICMP        | Internet Control Message Protocol   | <a href="#">RFC 792</a>                             |
| 0x02 | 2               | IGMP        | Internet Group Management Protocol  | <a href="#">RFC 1112</a>                            |
| 0x03 | 3               | GGP         | Gateway-to-Gateway Protocol   | <a href="#">RFC 823</a>                             |
| 0x04 | 4               | IP-in-IP    | IP in IP (encapsulation)  | <a href="#">RFC 2003</a>                            |
| 0x05 | 5               | ST          | Internet Stream Protocol  | <a href="#">RFC 1190</a> , <a href="#">RFC 1819</a> |
| 0x06 | 6               | TCP         | Transmission Control Protocol   | <a href="#">RFC 793</a>                             |
| 0x07 | 7               | CBT         | Core-based trees  | <a href="#">RFC 2189</a>                            |
| 0x08 | 8               | EGP         | Exterior Gateway Protocol   | <a href="#">RFC 888</a>                             |
| 0x09 | 9               | IGP         | Interior Gateway Protocol (any private interior gateway (used by Cisco for their IGRP)) |   |
| 0x0A | 10              | BBN-RCC-MON | BBN RCC Monitoring  |   |
| 0x0B | 11              | NVP-II      | Network Voice Protocol  | <a href="#">RFC 741</a>                             |
| 0x0C | 12              | PUP         | Xerox PUP   |   |
| 0x0D | 13              | ARGUS       | ARGUS   |   |
| 0x0E | 14              | EMCON       | EMCON   |   |
| 0x0F | 15              | XNET        | Cross Net Debugger  | <a href="#">IEN 158</a> <sup>[2]</sup>              |
| 0x10 | 16              | CHAOS       | Chaos   |   |
| 0x11 | 17              | UDP         | User Datagram Protocol  | <a href="#">RFC 768</a>                             |
| 0x12 | 18              | MUX         | Multiplexing  | <a href="#">IEN 90</a> <sup>[3]</sup>               |
| 0x13 | 19              | DCN-MEAS    | DCN Measurement Subsystems  |   |
| 0x14 | 20              | HMP         | Host Monitoring Protocol  | <a href="#">RFC 869</a>                             |

נדפיס את הפלט וכעת גם את הפרוטוקול. (מספר 1 זה ICMP):

```

root@5f8661d49810:/# 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=114 time=51.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=58.3 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 51.171/54.725/58.279/3.554 ms

```

```
Got a packet
source IP: 10.9.0.5
destination IP: 8.8.8.8
protocol: 1
Got a packet
source IP: 8.8.8.8
destination IP: 10.9.0.5
protocol: 1
```

כעת כמו בפייטון נרצה לפלטר לפי TCP ופורט יעד בין 10 ל100:

```
"proto TCP and dst portrange 10-100"
```

וכעת נבדוק את זה עם nc כמו בשאלה הקודמת ונקבל:

```
root@5f8661d49810:/# nc 8.8.8.8 50
```

```
Got a packet
source IP: 10.9.0.5
destination IP: 8.8.8.8
protocol: 6
Got a packet
source IP: 10.9.0.5
destination IP: 8.8.8.8
protocol: 6
```

ניתן לראות באמת שזה היה TCP לפי מספר הפרוטוקול והטבלה המצורפת.

### א.3

כעת נשתמש בכלי טלנט על מנת להסניף את הסיסמא (אנו מוסיפים הדפסה של data של הפקטה ושם נמצא את הסיסמא מופרדת בחבילות):

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct sniff_ip *ip = (struct sniff_ip *) (packet + sizeof(struct sniff_ethernet));
    char *data = (u_char *) packet + sizeof(struct sniff_ethernet) + sizeof(struct sniff_ip) + sizeof(struct sniff_tcp);
    int data_len = ntohs(ip->ip_len) - (sizeof(struct sniff_ip)) + sizeof(struct sniff_tcp);
    int i;

    printf("Got a packet\n");
    printf("source IP: %s\n", inet_ntoa(ip->ip_src));
    printf("destination IP: %s\n", inet_ntoa(ip->ip_dst));
    printf("protocol: %d\n", (ip->ip_p));

    if (data_len > 0) {
        for (i = 0; i < data_len; i++) {
            printf("%c", *data);
            data++;
        }
        printf("\n");
    }
}
```

ניתן לראות שמהתמונה הבאה קל לחלץ את הסיסמא שלנו כאשר מישהו מנסה להתחבר (dees):

Got a packet

source IP: 10.9.0.5

destination IP: 10.9.0.1

protocol: 6

00000d0h0:\000:CC Zh

Got a packet

source IP: 10.9.0.5

destination IP: 10.9.0.1

protocol: 6

00000C0i0:\0

CC Zh

Got a packet

source IP: 10.9.0.5

destination IP: 10.9.0.1

protocol: 6

000000e10:\N00CC Zh

Got a packet

source IP: 10.9.0.5

destination IP: 10.9.0.1

protocol: 6

00000s

## ב.1.

בסעיף זה נזייף פקטת TCP שהם מעל IP כמתבקש.

```
/* tcp */
typedef u_int tcp_seq;

struct sniff_tcp {
    u_short th_sport; /* source port */
    u_short th_dport; /* destination port */
    tcp_seq th_seq; /* sequence number */
    tcp_seq th_ack; /* acknowledgement number */
    u_char th_offx2; /* data offset, rsvd */
#define TH_OFF(th) (((th)->th_offx2 & 0xf0) >> 4)
    u_char th_flags;
#define TH_FIN 0x01
#define TH_SYN 0x02
#define TH_RST 0x04
#define TH_PUSH 0x08
#define TH_ACK 0x10
#define TH_URG 0x20
#define TH_ECE 0x40
#define TH_CWR 0x80
#define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
    u_short th_win; /* window */
    u_short th_sum; /* checksum */
    u_short th_urp; /* urgent pointer */
};
```

```

struct sniff_ip *ip = (struct sniff_ip *) buffer;
struct sniff_tcp *tcp = (struct sniff_tcp *) (buffer + sizeof(struct sniff_ip));
char *data = buffer + sizeof(struct sniff_ip) + sizeof(struct sniff_tcp);

sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if (sd < 0) {
    exit(-1);
}

sin.sin_family = AF_INET;
ip->ip_vhl = (4 << 4) | (20 >> 2);
ip->ip_tos = 0;
ip->ip_src.s_addr = inet_addr(SRC_IP);
ip->ip_dst.s_addr = inet_addr(DEST_IP);
ip->ip_id = htons(87654);
ip->ip_off = 0;
ip->ip_ttl = 50;
ip->ip_len = 50;
ip->ip_p = IPPROTO_TCP;
ip->ip_sum = 0;

tcp->th_sport = htons(8000);
tcp->th_dport = htons(9000);
tcp->th_seq = htonl(1);
tcp->th_ack = 0;
tcp->th_offx2 = 20;
tcp->th_flags = TH_FIN;
tcp->th_ack = 0;
tcp->th_win = htons(32767);
tcp->th_sum = 0;
tcp->th_urp = 0;

strcpy(data, "tcp message");

if (sendto(sd, buffer, ip->ip_len, 0, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    exit(-1);
}

```

הרצנו את הקוד וקיבלנו בwireshark את התוצאה הבאה

| *enp0s3  |             |          |             |          |   |
|--|-------------|----------|-------------|----------|---|
| File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help |             |          |             |          |   |
|  |             |          |             |          |   |
| Apply a display filter ... <Ctrl-/>  |             |          |             |          |   |
| No.  | Time        | Source   | Destination | Protocol | Length Info   |
| 1  | 0.000000000 | 10.9.0.1 | 8.8.8.8     | TCP      | 64 8000 → 9000 [FIN, Reserved] Seq=1 Win=32767, bogus TCP header... |



## ב.2

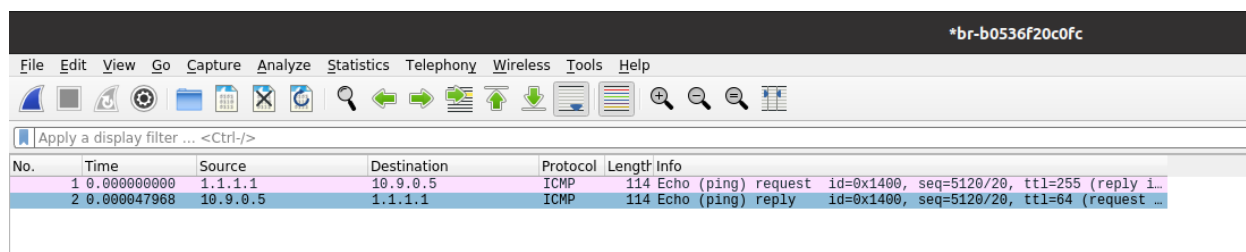
נעת נעשה דבר דומה עבור סעיף א רק שנשנה את כתובת המקור לכתובת אחרת

```
#define ECHO_REQUEST 8  
  
#define SRC_IP "1.1.1.1"  
#define DEST_IP "10.9.0.5"
```

```
struct sniff_icmp {  
    u_char type;  
    u_char code;  
    u_short checksum;  
    u_short id;  
    u_short seq;  
};
```

```
icmp->code = 0;  
icmp->type = ECHO_REQUEST;  
icmp->id = 20;  
icmp->seq = 20;  
icmp->checksum = 0x2bf9;
```

ניתן לראות את הזיוף בתמונה הבאה מwireshark



וניתן לראות שקיבלנו פקטת ICMP מסוג reply כמצופה שהזיוף עבד כשורה.

שאלות:

4. ניתן לשים כל ערך של גודל, ויתוקן במידת הצורך (כל עוד הגודל של חבילת הIP תקין)
5. ניתן לשים כל ערך של checksum ללא חישוב והפקטה תישלח (כנראה שטיפול בצד המקבל)
6. כמו שהסברנו בחלק של הפייתון אני צריכים הרשאות מנהל על מנת להיות במצב האזנה שדורש הרשאות מנהל.  
כאשר מנסים להריץ את התוכנית ללא זה התוכנית תקרוס ביצירת raw socket.

## ג.2

בסעיף זה נחבר את שתי הסעיפים הראשונים של ההסנפה והזיוף כמו שעשינו בחלק של הפיתוח על מנת להחזיר echo reply עבור כל request echo. שאנחנו רואים. לכן נשלב את שתי החלקים שעשינו עד כה.

תחילה קוראים בפונקציה main שמאתחלת את ההסנפות כמו שראינו והסברנו בסעיף הראשון. כמובן שנגדיר מה קורה בעת הגעת פקטה, וברגע שמגיעה פקטה ניצור פקטה חדשה ונחליף את המקור והיעד (וכל שאר השדות הרלוונטים) ונשלח אותה ברשת חזרה וכך בעצם יוצר מצב שבמקרה והנתקף שולח פינג למחשב שלא קיים הוא מקבל תשובה בכל זאת!

בפונקציית ההסנפה נבדוק שהגיע ping request ונקרא לפונקציית ping שתפורט בהמשך

```
// this function is called when packet is recieved
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct sniff_ip *ip = (struct sniff_ip *) (packet + sizeof(struct sniff_ethernet));
    struct sniff_icmp *icmp = (struct sniff_icmp *) (packet + sizeof(struct sniff_ethernet) + sizeof(struct sniff_ip));
    char *data = (u_char *) packet + sizeof(struct sniff_ethernet) + sizeof(struct sniff_ip) + sizeof(struct sniff_icmp);
    int data_len = ntohs(ip->ip_len) - (sizeof(struct sniff_ip)) - sizeof(struct sniff_icmp);

    if (ip->ip_p == 1) {
        if (icmp->type == 8) {
            printf("Got a packet\n");
            printf("source IP: %s\n", inet_ntoa(ip->ip_src));
            printf("destination IP: %s\n", inet_ntoa(ip->ip_dst));
            printf("protocol: %d\n", (ip->ip_p));

            ping(ip->ip_dst, ip->ip_src, data, data_len, icmp->id, icmp->seq);
        }
    }
}
```

לאחר סיום קבלת כל השדות הרלוונטים מהפקטה נקראת הפונקציה ping שמייצרת ping reply ושולחת אותו חזרה כדי לדמות את הפינג

פונקציה זו מייצרת את הפינג ושמה את כל השדות הנדרשים

```

// this function is called when packet is recieved
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct sniff_ip *ip = (struct sniff_ip *) (packet + sizeof(struct sniff_ethernet));
    struct sniff_icmp *icmp = (struct sniff_icmp *) (packet + sizeof(struct sniff_ethernet) + sizeof(struct sniff_ip));
    char *data = (u_char *) packet + sizeof(struct sniff_ethernet) + sizeof(struct sniff_ip) + sizeof(struct sniff_icmp);
    int data_len = ntohs(ip->ip_len) - (sizeof(struct sniff_ip)) - sizeof(struct sniff_icmp);

    if (ip->ip_p == 1) {
        if (icmp->type == 8) {
            printf("Got a packet\n");
            printf("source IP: %s\n", inet_ntoa(ip->ip_src));
            printf("destination IP: %s\n", inet_ntoa(ip->ip_dst));
            printf("protocol: %d\n", (ip->ip_p));

            ping(ip->ip_dst, ip->ip_src, data, data_len, icmp->id, icmp->seq);
        }
    }
}

```

נקמפל ונריץ ונשלח לכתובת שלא קיימת ונראה שמקבלים תשובה ולכן התוכנית עובדת !

```

root@5f8661d49810:/# ping 8.8.8.7
\PING 8.8.8.7 (8.8.8.7) 56(84) bytes of data.
80 bytes from 8.8.8.7: icmp_seq=1 ttl=255 time=440 ms
80 bytes from 8.8.8.7: icmp_seq=2 ttl=255 time=467 ms
80 bytes from 8.8.8.7: icmp_seq=3 ttl=255 time=490 ms
80 bytes from 8.8.8.7: icmp_seq=4 ttl=255 time=513 ms
^C
--- 8.8.8.7 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 440.012/477.524/512.948/27.082 ms

```

```

Got a packet
source IP: 10.9.0.5
destination IP: 8.8.8.7
protocol: 1
Got a packet
source IP: 10.9.0.5
destination IP: 8.8.8.7
protocol: 1
Got a packet
source IP: 10.9.0.5
destination IP: 8.8.8.7
protocol: 1
Got a packet
source IP: 10.9.0.5
destination IP: 8.8.8.7
protocol: 1

```

| *br-b0536f20c0fc   |             |          |             |          |        |                     |                                |                  |  |
|--|-------------|----------|-------------|----------|--------|---------------------|--------------------------------|------------------|--|
| File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help |             |          |             |          |        |                     |                                |                  |  |
| Apply a display filter ... <Ctrl-/>  |             |          |             |          |        |                     |                                |                  |  |
| No.  | Time        | Source   | Destination | Protocol | Length | Info                |                                |                  |  |
| 1  | 0.000000000 | 10.9.0.5 | 8.8.8.7     | ICMP     | 98     | Echo (ping) request | id=0x0019, seq=1/256, ttl=64   | (reply in 2)     |  |
| 2  | 0.715311017 | 8.8.8.7  | 10.9.0.5    | ICMP     | 114    | Echo (ping) reply   | id=0x0019, seq=1/256, ttl=255  | (request in 1)   |  |
| 3  | 0.999997538 | 10.9.0.5 | 8.8.8.7     | ICMP     | 98     | Echo (ping) request | id=0x0019, seq=2/512, ttl=64   | (reply in 4)     |  |
| 4  | 1.739667430 | 8.8.8.7  | 10.9.0.5    | ICMP     | 114    | Echo (ping) reply   | id=0x0019, seq=2/512, ttl=255  | (request in 3)   |  |
| 5  | 2.003982792 | 10.9.0.5 | 8.8.8.7     | ICMP     | 98     | Echo (ping) request | id=0x0019, seq=3/768, ttl=64   | (reply in 6)     |  |
| 6  | 2.763580957 | 8.8.8.7  | 10.9.0.5    | ICMP     | 114    | Echo (ping) reply   | id=0x0019, seq=3/768, ttl=255  | (request in 5)   |  |
| 7  | 3.004936534 | 10.9.0.5 | 8.8.8.7     | ICMP     | 98     | Echo (ping) request | id=0x0019, seq=4/1024, ttl=64  | (reply in ...)   |  |
| 8  | 3.787735434 | 8.8.8.7  | 10.9.0.5    | ICMP     | 114    | Echo (ping) reply   | id=0x0019, seq=4/1024, ttl=255 | (request in ...) |  |

כעת נראה את התופעה שראינו שכאשר שולחים למחשב קיים נקבל תשובה פעמיים פעם אחת מהמחשב האמיתי ופעם אחת מאיתנו

```

root@5f8661d49810:/# 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=53.2 ms
80 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=621 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=55.1 ms
80 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=641 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=53.9 ms
80 bytes from 8.8.8.8: icmp_seq=3 ttl=255 time=665 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=53.2 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, +3 duplicates, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 53.180/306.131/665.035/291.554 ms

```

| *br-b0536f20c0fc   |             |          |             |          |        |   |
|--|-------------|----------|-------------|----------|--------|---|
| File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help |             |          |             |          |        |   |
|  |             |          |             |          |        |   |
| Apply a display filter ... <Ctrl-/>  |             |          |             |          |        |   |
| No.  | Time        | Source   | Destination | Protocol | Length | Info  |
| 1  | 0.000000000 | 10.9.0.5 | 8.8.8.8     | ICMP     | 98     | Echo (ping) request id=0x001c, seq=1/256, ttl=64 (reply in 2)   |
| 2  | 0.053072794 | 8.8.8.8  | 10.9.0.5    | ICMP     | 98     | Echo (ping) reply id=0x001c, seq=1/256, ttl=113 (request 1...   |
| 3  | 0.621017025 | 8.8.8.8  | 10.9.0.5    | ICMP     | 114    | Echo (ping) reply id=0x001c, seq=1/256, ttl=255                 |
| 4  | 0.999747534 | 10.9.0.5 | 8.8.8.8     | ICMP     | 98     | Echo (ping) request id=0x001c, seq=2/512, ttl=64 (reply in 5)   |
| 5  | 1.054782247 | 8.8.8.8  | 10.9.0.5    | ICMP     | 98     | Echo (ping) reply id=0x001c, seq=2/512, ttl=113 (request 1...   |
| 6  | 1.641080237 | 8.8.8.8  | 10.9.0.5    | ICMP     | 114    | Echo (ping) reply id=0x001c, seq=2/512, ttl=255                 |
| 7  | 2.000112831 | 10.9.0.5 | 8.8.8.8     | ICMP     | 98     | Echo (ping) request id=0x001c, seq=3/768, ttl=64 (reply in 8)   |
| 8  | 2.053963418 | 8.8.8.8  | 10.9.0.5    | ICMP     | 98     | Echo (ping) reply id=0x001c, seq=3/768, ttl=113 (request 1...   |
| 9  | 2.665077374 | 8.8.8.8  | 10.9.0.5    | ICMP     | 114    | Echo (ping) reply id=0x001c, seq=3/768, ttl=255                 |
| 10   | 3.000678651 | 10.9.0.5 | 8.8.8.8     | ICMP     | 98     | Echo (ping) request id=0x001c, seq=4/1024, ttl=64 (reply in ... |
| 11   | 3.053805690 | 8.8.8.8  | 10.9.0.5    | ICMP     | 98     | Echo (ping) reply id=0x001c, seq=4/1024, ttl=113 (request ...   |
| 12   | 3.691094184 | 8.8.8.8  | 10.9.0.5    | ICMP     | 114    | Echo (ping) reply id=0x001c, seq=4/1024, ttl=255                |