- ping 8.8.8.8

כתובת ה IP הזו תקינה ומשוייכת לserver של גוגל.
במקרה כזה, ה user אמור לקבל הודעת echo reply
באופן תקין. נזכיר שה sniff_spoof שלנו עדיין פועל בצד של
התוקף. במקרה כזה, נוצרת הודעת echo reply מזויפת מצד התוקף.
ה user מקבל 2 הודעות reply זהות, עם אותם פרטים בדיוק לכל
הודעה (למשל אותו seq number אשר אמור להיות ייחודי עבור שליחת
ping מסוימת) במקרה כזה ה user יראה כפילים של הודעות
ה - echo reply.

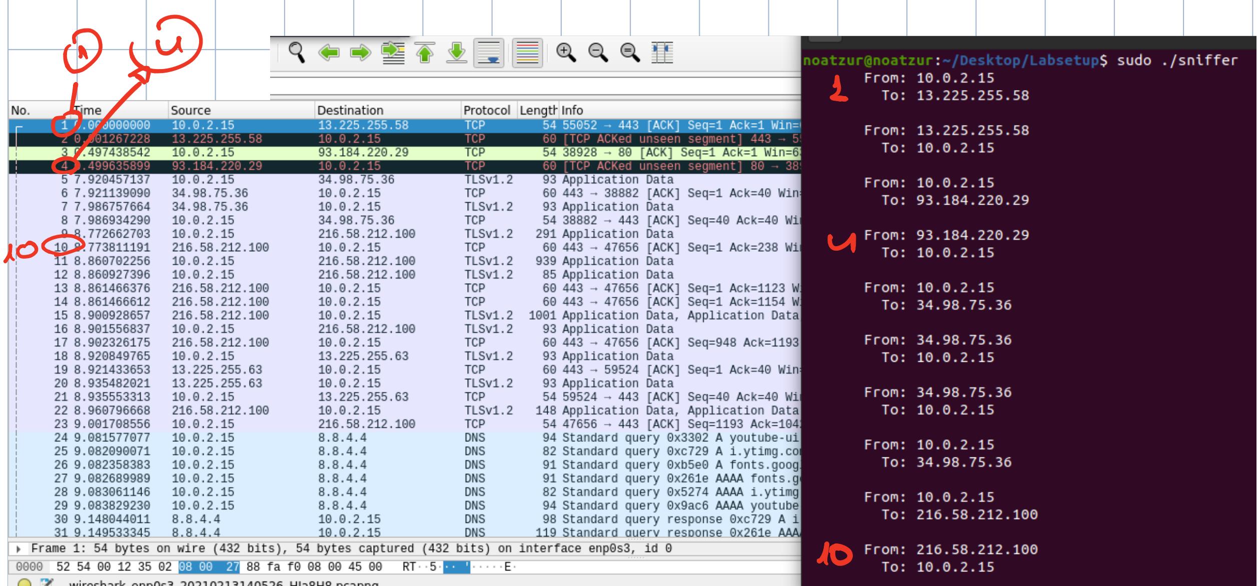
Duplicates:

```
[14]+  Stopped                 ping 8.8.8.8
root@9d56b5cb125d:/# 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=110 time=53.8 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=70.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=24.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=110 time=57.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=19.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=110 time=52.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=27.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=110 time=52.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=26.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=110 time=51.9 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=35.8 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=110 time=53.0 ms (DUP!)
^Z
[15]+  Stopped                 ping 8.8.8.8
root@9d56b5cb125d:/#
```

```
 44 3.413707868  8.8.8.8          10.9.0.5         ICMP   100 Echo (ping) reply    id=0x001d, seq=4/1024, ttl=110
 45 4.364994725  10.9.0.5         8.8.8.8          ICMP   100 Echo (ping) request  id=0x001d, seq=5/1280, ttl=64 (no respo
 46 4.364994725  10.9.0.5         8.8.8.8          ICMP   100 Echo (ping) request  id=0x001d, seq=5/1280, ttl=64 (reply ir
 47 4.365058001  10.0.2.15        8.8.8.8          ICMP   100 Echo (ping) request  id=0x001d, seq=5/1280, ttl=63 (reply ir
 48 4.391090147  8.8.8.8          10.9.0.5         ICMP   100 Echo (ping) reply    id=0x001d, seq=5/1280, ttl=64 (request
 49 4.391120115  8.8.8.8          10.9.0.5         ICMP   100 Echo (ping) reply    id=0x001d, seq=5/1280, ttl=64
 50 4.416788834  8.8.8.8          10.0.2.15        ICMP   100 Echo (ping) reply    id=0x001d, seq=5/1280, ttl=111 (request
 51 4.416837214  8.8.8.8          10.9.0.5         ICMP   100 Echo (ping) reply    id=0x001d, seq=5/1280, ttl=110
 52 4.416877169  8.8.8.8          10.9.0.5         ICMP   100 Echo (ping) reply    id=0x001d, seq=5/1280, ttl=110
 53 5.365314196  10.9.0.5         8.8.8.8          ICMP   100 Echo (ping) request  id=0x001d, seq=6/1536, ttl=64 (no respo
 54 5.365314196  10.9.0.5         8.8.8.8          ICMP   100 Echo (ping) request  id=0x001d, seq=6/1536, ttl=64 (reply ir
 55 5.365392894  10.0.2.15        8.8.8.8          ICMP   100 Echo (ping) request  id=0x001d, seq=6/1536, ttl=63 (reply ir
 56 5.396058229  02:42:0a:09:00:05                 ARP     44 Who has 10.9.0.1? Tell 10.9.0.5
 57 5.396058229  02:42:0a:09:00:05                 ARP     44 Who has 10.9.0.1? Tell 10.9.0.5
 58 5.396297626  02:42:e4:1f:95:5a                 ARP     44 10.9.0.1 is at 02:42:e4:1f:95:5a
 59 5.396318647  02:42:e4:1f:95:5a                 ARP     44 10.9.0.1 is at 02:42:e4:1f:95:5a
```

2.1A: הקוד בקובץ Sniff_921.C

בסעיף זה נבנה Sniffer בשפת C ללא פילטרים.



בתמונה מצולם ניתן לראות את ההרצה של קובץ Sniff_921.C ובמקביל הרצה של Wireshark. ניתן לראות שהפאקטות שהתקבלו ב-WS הוספו ע"י ה Sniffer והודפסו ה dst ו src איפי של כל אחד מהם לפי הסדר (למשל ① זו הפאקטה שהודפסה ראשונה בוורשארק וכך הלאה...)

Question 1:

① pcap_lookupdev(errbuf);

הפונ' מחפשת הכרטיס רשת הדיפולטיבי (ברירת מחדל) אשר דרכו נרצה
לראות את התעבורה (למשל פאקטות...) כמו כן, אם ישנם כמה כרטיסי
רשת כאלו, הפונ' מחזיר ראשון של כרטיסי הרשת.
כמו כן הפונ' מחזיר פוינטר ל-string שמייצג את שם כרטיס
הרשת (אם זו טעות של כרטיסי רשת אז לראשון).

② pcap-open_live(myDEV, BUFSIZ, 1, 1000, errbuf);

פותחת את כרטיס הרשת למצב "מפיסת" פאקטות.
הפונ' מקבלת את הפרמטרים הבאים:
(Device) myDEV - מקבלת את שם כרטיס הרשת שמצאנו קודם
שאותו נפעיל כך שיהיה במצב של מפיסת פאקטות
(snaplen) BUFSIZ - מגדיר את גודל המיר (buffer). המיר יכיל
פאקטות שנתפסו. (כמות הפאקטות שנשמרות במיר)
(promisc) - אם 1 אז מפעיל את המצב של
promiscuous - פרוץ.
0 ← המצב הפרוץ כבוי.
time out (to_ms) 1000 - (to_ms) - כמה זמן כל פאקטה נמצאת בבאפר

errbuf - פוינטר לבאפר שיחזיק שגיאות במידה ויצרקו,

③ pcap_loop(handle, -1, got_Packet, NULL) -

מהפונקציה הקודמת ②

הפונ׳ מקבלת פוינטר מסוג pcap_t שזה ה-session שפתחנו אל מול כרטיס הרשת הנבחר ועבור כל פאקטה שנתפסת קוראת לפונ׳ מסוימת שמטפלת בפאקטה, במקרה שלנו פונ׳ בשם got_packet שהגדרנו מעלה. הפונ׳ got_packet מבצעת את הפעולות הרצויות עם המידע שהתקבל.

-1 : מגדיר ל pcap_loop לרוץ בלולאה אינסופית עד שגורם חיצוני יעצור את הלולאה ותמשיך לתפוס פאקטות.

④ pcap_close (handle)-

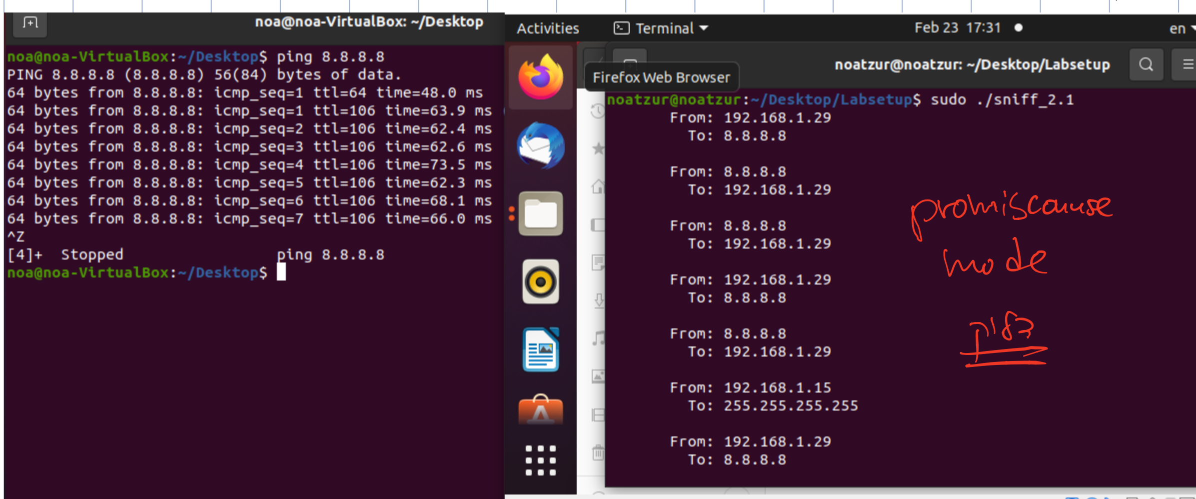
סוגרת את ה session שנפתח ע״י פונ׳ מס׳ ②. מוחקת את הקבצים והזכרון שהוקצה עבור הsession הזה.

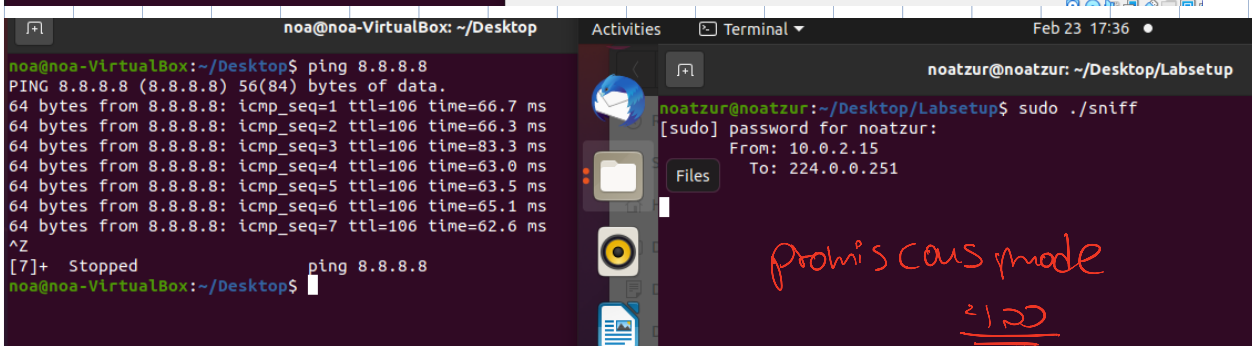
Question 2:

כדי לפתוח raw socket ובכך לאפשר לנו לקבל כל פאקטה שעוברת דרך כרטיס הרשת הנבחר, ולא דווקא פאקטות המיועדות אלינו, צריך הרשאת ברמת admin. אם כל משתמש היה יכול לפתוח raw socket ולהפעיל את ה- promiscouse mode, הוא היה יכול לבחון את התעבורה ברשת שעוברת דרך כרטיס הרשת ולאו דווקא מיועדת לכרטיס הספציפי.

ולכן העברת ה Sniffer שלנו למצב שבו הוא תופס פאקטות שלא מיועדות אליו דווקא, ידרוש root privilege. התוכנה תזרוק שגיאת segmentation fault אם לא נשתמש בהרשאת הroot. כנראה הפונ׳ שמגדירה גישה לכרטיס רשת דורשת root privilege. מגדירים זאת בפונ׳ pcap_open_live.

Question 3: כאשר promis mode דלוק, אנחנו בעצם מכניסים את
מכניס הsniffer שלנו למצב "פרוש" האומר לכרטיס הרשת - את מבצע סינון
של הפאקטות שנכנסות דרכך. כל פאקטה שעוברת - תתפוס אותה. כאשר המצב
לא דלוק, קורה הפעולה הרגילה שקורית מה"ז כאשר אנו משתמשים באינטרנט -
כרטיס הרשת מכניס אליו אך ורק פקטות המיועדות אליו, על כן, כדי שנבחן את
הsniffer שלנו כאשר המצב דלוק/כבוי, פתחנו מכונה נוספת (שלה Ip
שונה מהמכונה ה-1) וממנה הרצנו פינג ל8.8.8.8.
כאשר המצב היה דלוק, ניתן לראות שהפקטות נתפסו והודפסו למסך,
אך המצב היה כבוי, ניתן לראות שלא נתפסו פקטות ולא הודפס דבר:





* המכונה (שמיהן) שינו בהגדרות הnetwork ל- Allow vm בשדה של
promiscouse mode. רק כך הצלחנו לשנות בקוד מ-0 ל-1 ולהפך
ולראות את השינוי.

2.1B: filters הקוד בקובץ sniff_filter.c ו sniff_filter2.c

- sniff_filter.c - מסנן פאקטות מסוג ICMP מ-destination ו- source ספציפיים:

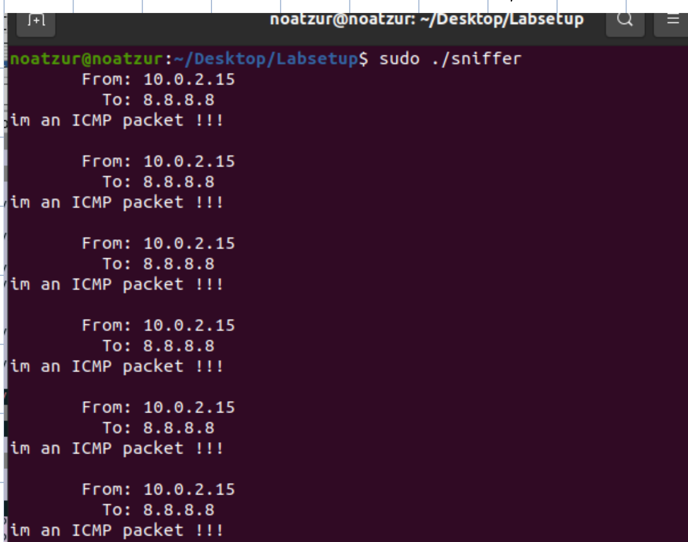
עשינו זאת ע"י הגדרת הבאפר filter-exp עם הביטוי

"Icmp and Src host 10.0.2.15 and dst host 8.8.8.8"

ע"י פונ' pcap_compile ממירים את ה String שלנו ושומרים את הקרסא המקומפלת של הפילטר במבנה מיוחד לכך (bpf_program).

ע"י הפונ' pcap_setfilter מפעילים את ה filter שלנו על ה session.

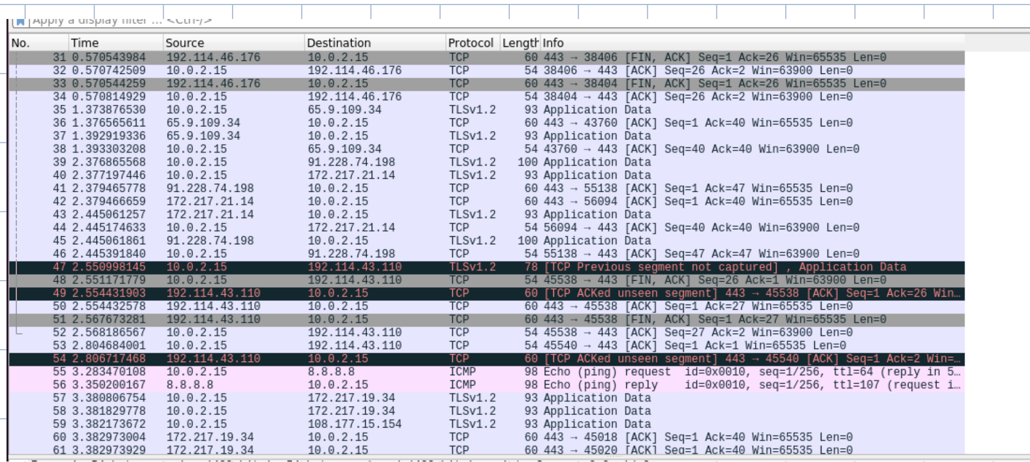
תמונה 1 - הדפסה של הפאקטים שנתפסו, ניתן לראות שבה רק את ה IP המצוין, ורק מסוג ICMP:



```
noatzur@noatzur:~/Desktop/Labsetup$ sudo ./sniffer
        From: 10.0.2.15
          To: 8.8.8.8
im an ICMP packet !!!

        From: 10.0.2.15
          To: 8.8.8.8
im an ICMP packet !!!

        From: 10.0.2.15
          To: 8.8.8.8
im an ICMP packet !!!

        From: 10.0.2.15
          To: 8.8.8.8
im an ICMP packet !!!

        From: 10.0.2.15
          To: 8.8.8.8
im an ICMP packet !!!

        From: 10.0.2.15
          To: 8.8.8.8
im an ICMP packet !!!
```

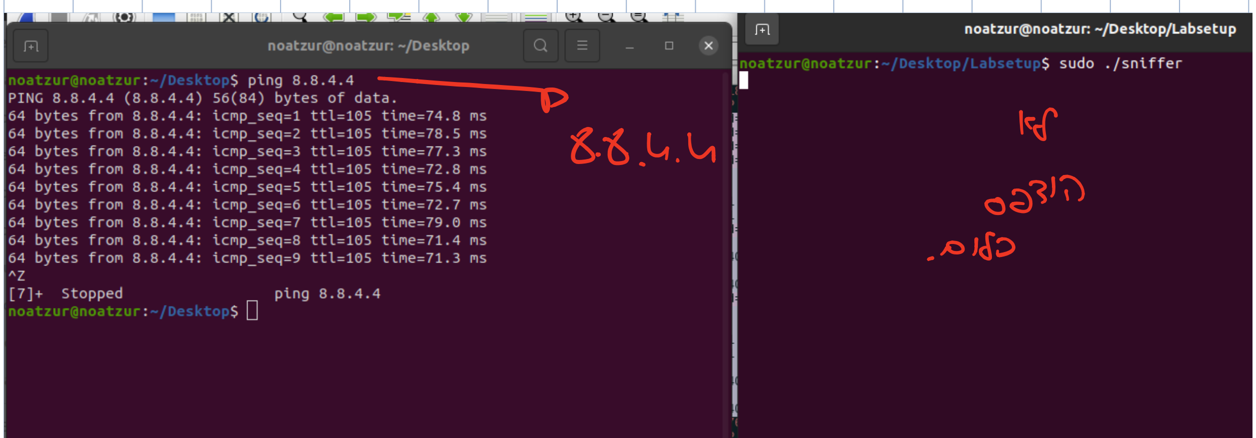
תמונה 2 מה wireshark שהופעל במקביל לפעולת ה sniffer

ניתן לראות שהתקבלו פאקטות מכל מיני סוגים אך נתפסו רק אלו עם ה IP המצוינים ומסוג ICMP:



No.	Time	Source	Destination	Protocol	Length	Info
31	0.570543984	192.114.46.176	10.0.2.15	TCP	60	443 → 38406 [FIN, ACK] Seq=1 Ack=26 Win=65535 Len=0
32	0.570742509	10.0.2.15	192.114.46.176	TCP	54	38406 → 443 [ACK] Seq=26 Ack=2 Win=63900 Len=0
33	0.570544259	192.114.46.176	10.0.2.15	TCP	60	443 → 38404 [FIN, ACK] Seq=1 Ack=26 Win=65535 Len=0
34	0.570814929	10.0.2.15	192.114.46.176	TCP	54	38404 → 443 [ACK] Seq=26 Ack=2 Win=63900 Len=0
35	1.373876530	10.0.2.15	65.9.109.34	TLSv1.2	93	Application Data
36	1.376565611	65.9.109.34	10.0.2.15	TCP	60	443 → 43760 [ACK] Seq=1 Ack=40 Win=65535 Len=0
37	1.392919336	65.9.109.34	10.0.2.15	TLSv1.2	93	Application Data
38	1.393303208	10.0.2.15	65.9.109.34	TCP	54	43760 → 443 [ACK] Seq=40 Ack=40 Win=63900 Len=0
39	2.376865568	10.0.2.15	91.228.74.198	TLSv1.2	100	Application Data
40	2.377197446	10.0.2.15	172.217.21.14	TLSv1.2	93	Application Data
41	2.379465778	91.228.74.198	10.0.2.15	TCP	60	443 → 55138 [ACK] Seq=1 Ack=47 Win=65535 Len=0
42	2.379466659	172.217.21.14	10.0.2.15	TCP	60	443 → 56094 [ACK] Seq=1 Ack=40 Win=65535 Len=0
43	2.445061257	172.217.21.14	10.0.2.15	TLSv1.2	93	Application Data
44	2.445174633	10.0.2.15	172.217.21.14	TCP	54	56094 → 443 [ACK] Seq=40 Ack=40 Win=63900 Len=0
45	2.445061861	91.228.74.198	10.0.2.15	TLSv1.2	100	Application Data
46	2.445391840	10.0.2.15	91.228.74.198	TCP	54	55138 → 443 [ACK] Seq=47 Ack=47 Win=63900 Len=0
47	2.550998145	10.0.2.15	192.114.43.110	TLSv1.2	78	[TCP Previous segment not captured] , Application Data
48	2.551171779	10.0.2.15	192.114.43.110	TCP	54	45538 → 443 [FIN, ACK] Seq=26 Ack=1 Win=63900 Len=0
49	2.554431903	192.114.43.110	10.0.2.15	TCP	60	[TCP ACKed unseen segment] 443 → 45538 [ACK] Seq=1 Ack=26 Win…
50	2.554432578	192.114.43.110	10.0.2.15	TCP	60	443 → 45538 [ACK] Seq=1 Ack=27 Win=65535 Len=0
51	2.567673281	192.114.43.110	10.0.2.15	TCP	60	443 → 45538 [FIN, ACK] Seq=1 Ack=27 Win=65535 Len=0
52	2.568186567	10.0.2.15	192.114.43.110	TCP	54	45538 → 443 [ACK] Seq=27 Ack=2 Win=63900 Len=0
53	2.804684001	10.0.2.15	192.114.43.110	TCP	54	45540 → 443 [ACK] Seq=1 Ack=1 Win=65535 Len=0
54	2.806717468	192.114.43.110	10.0.2.15	TCP	60	[TCP ACKed unseen segment] 443 → 45540 [ACK] Seq=1 Ack=2 Win=…
55	3.283470108	10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=0x0010, seq=1/256, ttl=64 (reply in 5…
56	3.350200167	8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0010, seq=1/256, ttl=107 (request i…
57	3.380806754	10.0.2.15	172.217.19.34	TLSv1.2	93	Application Data
58	3.381829778	10.0.2.15	172.217.19.34	TLSv1.2	93	Application Data
59	3.382173672	10.0.2.15	108.177.15.154	TLSv1.2	93	Application Data
60	3.382973004	172.217.19.34	10.0.2.15	TCP	60	443 → 45018 [ACK] Seq=1 Ack=40 Win=65535 Len=0
61	3.382973929	172.217.19.34	10.0.2.15	TCP	60	443 → 45020 [ACK] Seq=1 Ack=40 Win=65535 Len=0

תמונה 3 - עוד בדיקה שהבצעה לבדיקה הפילטר של הקוד:



```
noatzur@noatzur:~/Desktop$ ping 8.8.4.4
PING 8.8.4.4 (8.8.4.4) 56(84) bytes of data.
64 bytes from 8.8.4.4: icmp_seq=1 ttl=105 time=74.8 ms
64 bytes from 8.8.4.4: icmp_seq=2 ttl=105 time=78.5 ms
64 bytes from 8.8.4.4: icmp_seq=3 ttl=105 time=77.3 ms
64 bytes from 8.8.4.4: icmp_seq=4 ttl=105 time=72.8 ms
64 bytes from 8.8.4.4: icmp_seq=5 ttl=105 time=75.4 ms
64 bytes from 8.8.4.4: icmp_seq=6 ttl=105 time=72.7 ms
64 bytes from 8.8.4.4: icmp_seq=7 ttl=105 time=79.0 ms
64 bytes from 8.8.4.4: icmp_seq=8 ttl=105 time=71.4 ms
64 bytes from 8.8.4.4: icmp_seq=9 ttl=105 time=71.3 ms
^Z
[7]+  Stopped                 ping 8.8.4.4
noatzur@noatzur:~/Desktop$
```

```
noatzur@noatzur:~/Desktop/Labsetup$ sudo ./sniffer
```

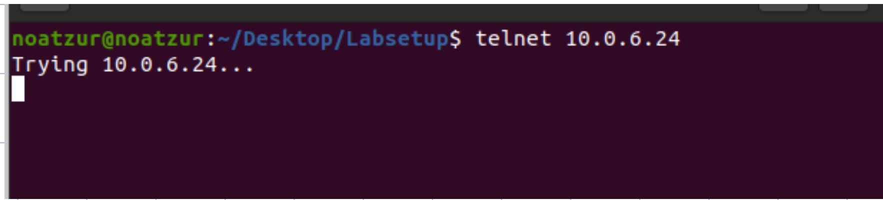

הפילטר הוא ל dst IP 8.8.8.8 ולא 8.8.4.4, ולכן ה Sniffer לא הדפיס כלום.

- פילטר TCP - sniff_filter2.c

פילטר_ מסוג TCP מטווח פורטים 10-100.

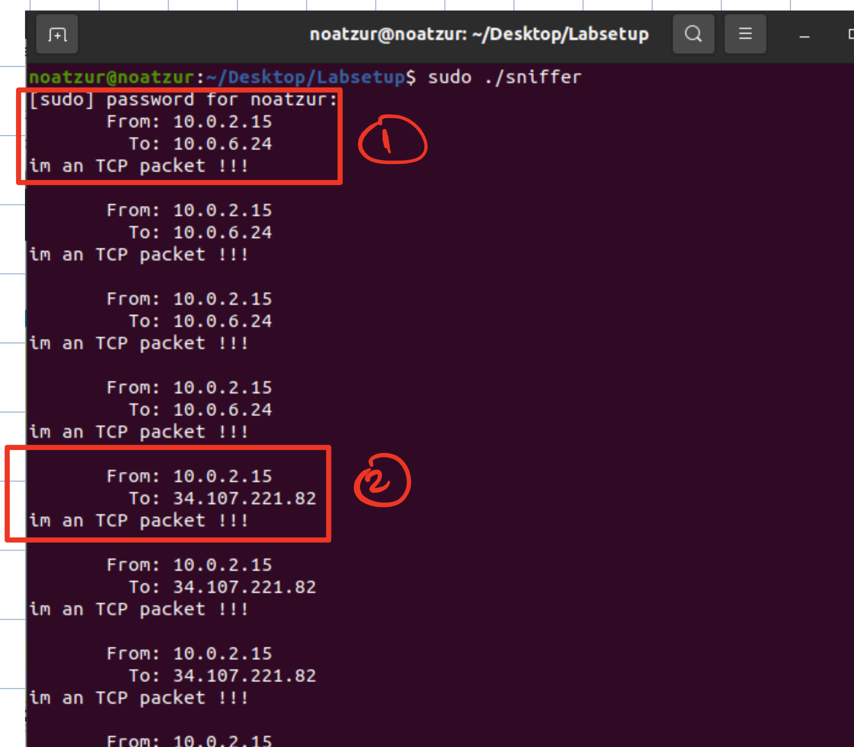
אותם שלבים בדיוק כמו בסעיף הקודם, רק שהפעם הביטוי המסנן הוא: "tcp and dst portrange 10-100"

על מנת לבדוק את הפילטר הנ"ל, מצאנו בויקיפדיה הסבר על פרוטוקול telnet - הפרוטוקול שמשתמש בחיבור לפורט 23 (שהוא בטווח המסונן), והוא יוצר חיבור TCP.



```
noatzur@noatzur:~/Desktop/Labsetup$ telnet 10.0.6.24
Trying 10.0.6.24...
```

כמו כן ניתן לראות בתמונה שנשלפו עוד פקטות TCP בטווח של 10-100:

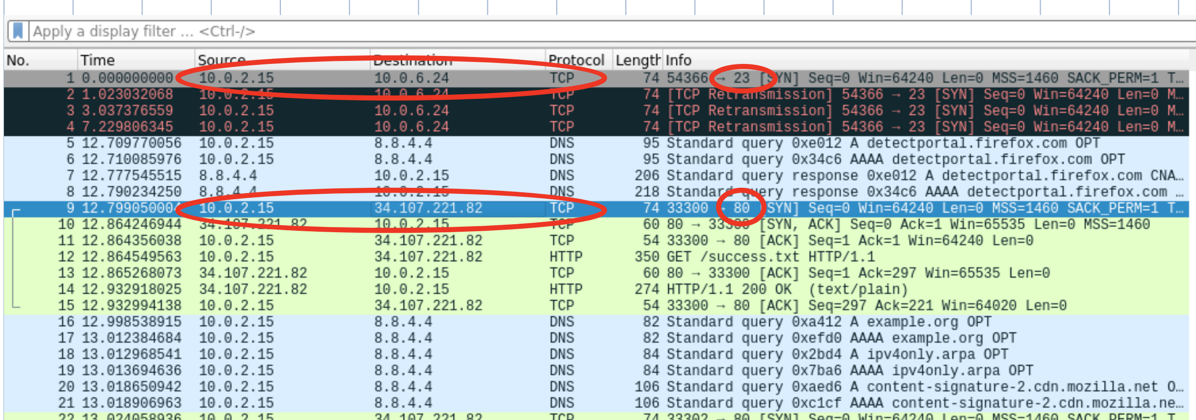


```
noatzur@noatzur: ~/Desktop/Labsetup
noatzur@noatzur:~/Desktop/Labsetup$ sudo ./sniffer
[sudo] password for noatzur:
        From: 10.0.2.15
          To: 10.0.6.24
im an TCP packet !!!

        From: 10.0.2.15
          To: 10.0.6.24
im an TCP packet !!!

        From: 10.0.2.15
          To: 10.0.6.24
im an TCP packet !!!

        From: 10.0.2.15
          To: 10.0.6.24
im an TCP packet !!!

        From: 10.0.2.15
          To: 34.107.221.82
im an TCP packet !!!

        From: 10.0.2.15
          To: 34.107.221.82
im an TCP packet !!!

        From: 10.0.2.15
          To: 34.107.221.82
im an TCP packet !!!

        From: 10.0.2.15
```


① ניתן לראות ב WS שהוא נשלח לdport 23

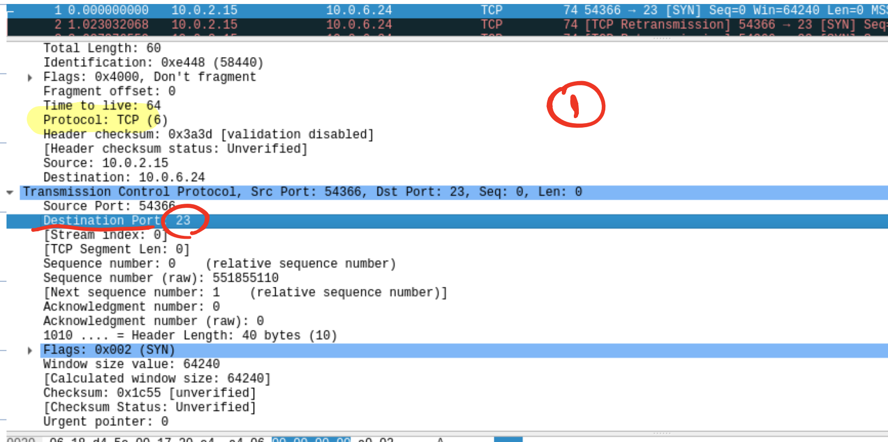
② ניתן לראות ב WS שהוא נשלח לdport 80

שניהם בטווח.

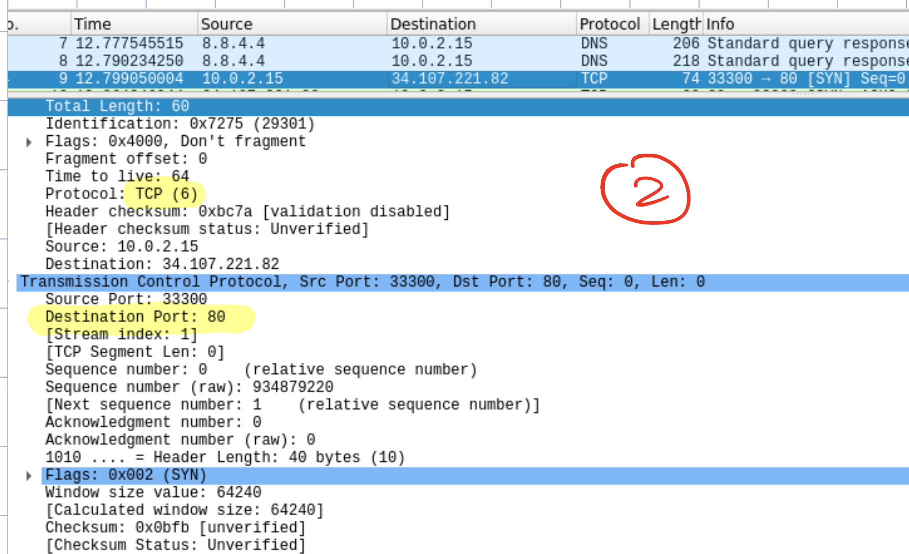


Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	10.0.6.24	TCP	74	54366 → 23 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
2	1.023032068	10.0.2.15	10.0.6.24	TCP	74	[TCP Retransmission] 54366 → 23 [SYN] Seq=0 Win=64240 Len=0 M…
3	3.037376559	10.0.2.15	10.0.6.24	TCP	74	[TCP Retransmission] 54366 → 23 [SYN] Seq=0 Win=64240 Len=0 M…
4	7.229806345	10.0.2.15	10.0.6.24	TCP	74	[TCP Retransmission] 54366 → 23 [SYN] Seq=0 Win=64240 Len=0 M…
5	12.709770056	10.0.2.15	8.8.4.4	DNS	95	Standard query 0xe012 A detectportal.firefox.com OPT
6	12.710085976	10.0.2.15	8.8.4.4	DNS	95	Standard query 0x34c6 AAAA detectportal.firefox.com OPT
7	12.777545515	8.8.4.4	10.0.2.15	DNS	206	Standard query response 0xe012 A detectportal.firefox.com CNA…
8	12.790234250	8.8.4.4	10.0.2.15	DNS	218	Standard query response 0x34c6 AAAA detectportal.firefox.com …
9	12.79905000[illegible]	10.0.2.15	34.107.221.82	TCP	74	33300 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…
10	12.864246944	34.107.221.82	10.0.2.15	TCP	60	80 → 33300 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
11	12.864356038	10.0.2.15	34.107.221.82	TCP	54	33300 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
12	12.864549563	10.0.2.15	34.107.221.82	HTTP	350	GET /success.txt HTTP/1.1
13	12.865268073	34.107.221.82	10.0.2.15	TCP	60	80 → 33300 [ACK] Seq=1 Ack=297 Win=65535 Len=0
14	12.932918025	34.107.221.82	10.0.2.15	HTTP	274	HTTP/1.1 200 OK (text/plain)
15	12.932994138	10.0.2.15	34.107.221.82	TCP	54	33300 → 80 [ACK] Seq=297 Ack=221 Win=64020 Len=0
16	12.998538915	10.0.2.15	8.8.4.4	DNS	82	Standard query 0xa412 A example.org OPT
17	13.012384684	10.0.2.15	8.8.4.4	DNS	82	Standard query 0xefd0 AAAA example.org OPT
18	13.012968541	10.0.2.15	8.8.4.4	DNS	84	Standard query 0x2bd4 A ipv4only.arpa OPT
19	13.013694636	10.0.2.15	8.8.4.4	DNS	84	Standard query 0x7ba6 AAAA ipv4only.arpa OPT
20	13.018650942	10.0.2.15	8.8.4.4	DNS	106	Standard query 0xaed6 A content-signature-2.cdn.mozilla.net O…
21	13.018906963	10.0.2.15	8.8.4.4	DNS	106	Standard query 0xc1cf AAAA content-signature-2.cdn.mozilla.ne…
22	13.024058936	10.0.2.15	34.107.221.82	TCP	74	33302 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T…



```
  1 0.000000000   10.0.2.15        10.0.6.24        TCP       74 54366 → 23 [SYN] Seq=0 Win=64240 Len=0 MSS
  2 1.023032068   10.0.2.15        10.0.6.24        TCP       74 [TCP Retransmission] 54366 → 23 [SYN] Seq=
    Total Length: 60
    Identification: 0xe448 (58440)
  ▸ Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0x3a3d [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 10.0.6.24
▾ Transmission Control Protocol, Src Port: 54366, Dst Port: 23, Seq: 0, Len: 0
    Source Port: 54366
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Sequence number (raw): 551855110
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  ▸ Flags: 0x002 (SYN)
    Window size value: 64240
    [Calculated window size: 64240]
    Checksum: 0x1c55 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
```

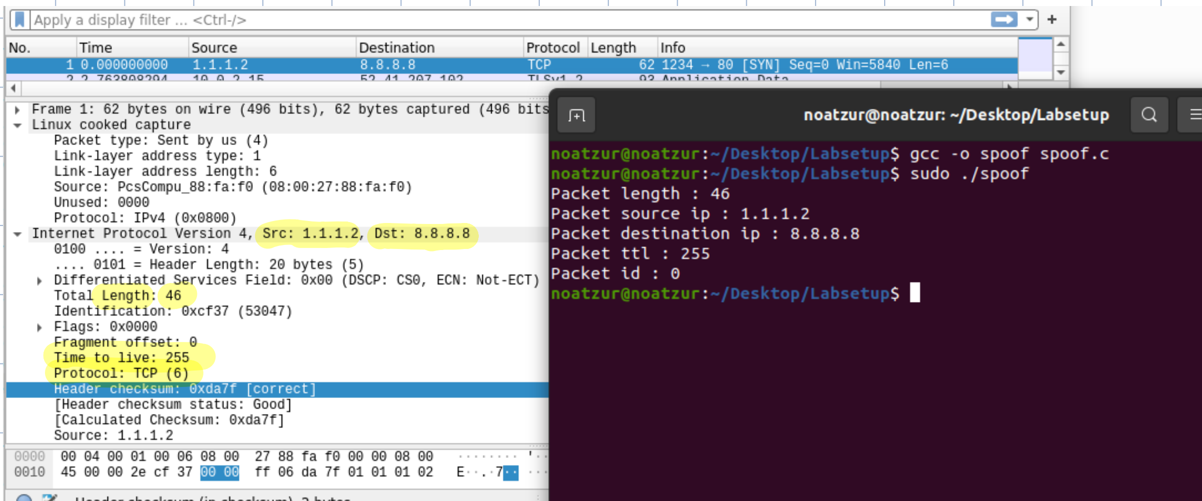



```
No.  Time           Source     Destination     Protocol Length Info
   7 12.777545515   8.8.4.4    10.0.2.15       DNS      206 Standard query response
   8 12.790234250   8.8.4.4    10.0.2.15       DNS      218 Standard query response
   9 12.799050004   10.0.2.15  34.107.221.82   TCP       74 33300 → 80 [SYN] Seq=0
    Total Length: 60
    Identification: 0x7275 (29301)
  ▸ Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xbc7a [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 34.107.221.82
  Transmission Control Protocol, Src Port: 33300, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 33300
    Destination Port: 80
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Sequence number (raw): 934879220
    [Next sequence number: 1    (relative sequence number)]
    Acknowledgment number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  ▸ Flags: 0x002 (SYN)
    Window size value: 64240
    [Calculated window size: 64240]
    Checksum: 0x0bfb [unverified]
    [Checksum Status: Unverified]
```


2.1c:

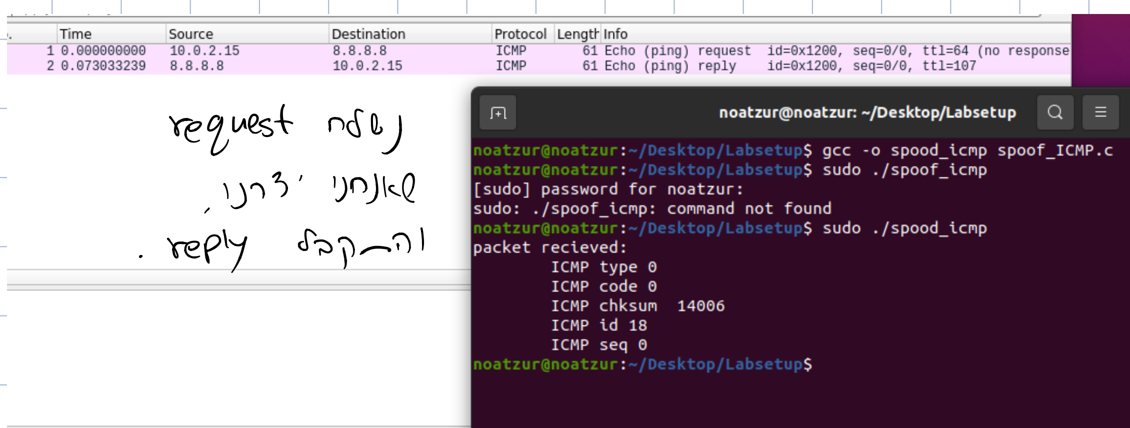
בסעיף זה התבקשנו להדפיס סיסמא של user אשר משתמש בפרוטוקול telnet, פרוטוקול זה משתמש ב port 23 ואינו מאובטח ולכן ניתן לראות את ה-data שהמשתמש מזין. בתמונה מטה הדפסתי למסך את כל ה-data שהתקבל והסיסמא מסומנת בצורה ידנית.

בנוסף צירפנו גם תמונה של WS שם ניתן לראות את הdata. על מנת לשלוח ע"י telnet



2.2B: Spoof ICMP request   קוד spoof_ICMP.c

בסעיף זה התבקשנו לבצע מתקפת spoof לייצור פקטה מסוג ICMP request (id = 8), השתמשנו בקוד שכתבנו בשאלה הקודמת כי הוא עונה על צרכים אלו בדיוק.



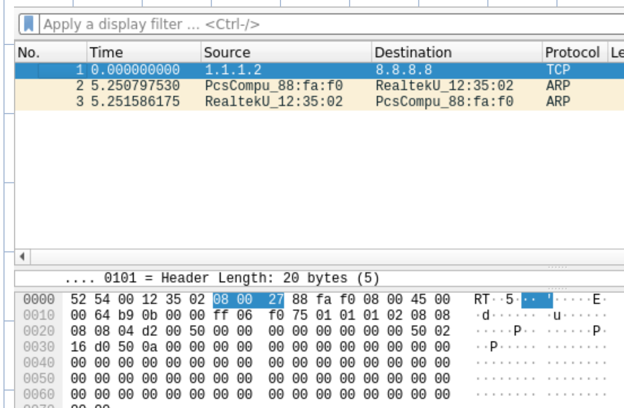
שלחנו request
שאותו יצרנו,
והתקבל reply.

בנוסף בסעיף זה התבקשנו לענות על 3 שאלות:

Question 4:

לאחר ניסוי קצר שעשינו בקובץ spoof.c, שבו שינינו את גודל ה ip header לגדלים כי לא משנה מה הגודל שנגדיר לו, כל עוד הערך גדול מ-20 כי זה הגודל המינימלי.

(וכל לשים גם ערכים גדולים, לדוגמא 100, הפקטה נשלחת באופן תקין ו"מרפד" את ה data ב-אפסים



ריפוד באפסים ← כאשר הגדרנו את הערך ל-100.

```
noatzur@noatzur:~/Desktop/Labsetup$ gcc -o spoof spoof.c
noatzur@noatzur:~/Desktop/Labsetup$ sudo ./spoof
[sudo] password for noatzur:
sendto failed: Invalid argument
noatzur@noatzur:~/Desktop/Labsetup$
```

השגיאה שנוצרה כאשר הגדרנו את הגודל להיות 10.

Question 5:

גם בסעיף זה בדקנו ניסוי קטן כדי לבחון אם מחקנו את השורה שמחשבת checksum עבור השדה המתאים ב Iph ושמנו רק 0, שליחת הפקטה עברה באופן תקין, וגם ws נתן לראות שה header checksum הוא correct וה status הוא good. מכאן הבנו שה raw socket מחשב עבורנו את ה checksum ולכן אין הכרח שאנו נעשה את החישוב.

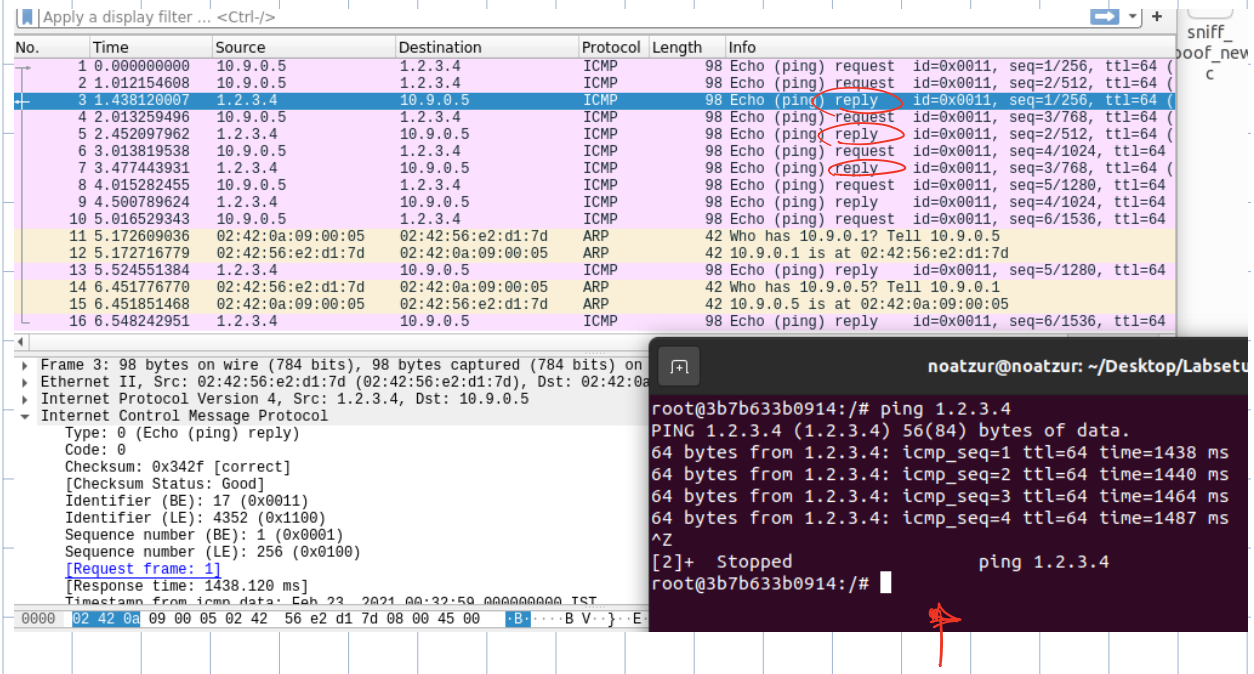
Question 6:

על מנת שנוכל לייצר פאקטה משלנו, לעצב ולשנות שדות ב-headers (נצטרך הרשאות מנהל), לא כל משתמש רגיל יוכל לעשות שינויים כאלה. כמו כן, גם כדי להסניף פאקטות שלא מיועדות לכרטיס הרשת הספציפי דרכו אנו עובדים, נצטרך root privilege. הפונ' של ספריית pcap דורשות זאת כמובן
על מנת לבצע את כל האמור למעלה, תכנית שננסה להריץ ללא הרשאה מתאימה לא תרוץ ותזרוק שגיאת permission שכן לא נוכל לייצר פאקטות ובכך לעשות את עיקר התוכנית - spoof.

2.3: Sniff & spoof

sniff_spoof_new.c

בסעיף זה התבקשנו "לחבר" את תכנית ה Sniff שכתבנו עם תכנית ה Spoof.
ב container של המשתמש הרגיל שלחנו ping ל IP שלא קיים 1.2.3.4 כאשר במקביל, בcontainer של ה attacker הרצנו את הSniff & spoof שלנו.
התכנית, ע"י פונ' ה Sniff שכתבנו, זיהתה את ה ping שנשלח מהמשתמש הרגיל (ICMP) וייצרה פאקטה ICMP reply מזויפת ונשלחה לאותו IP ממנו התקבל ה ping.
כך בעצם "יצרנו מצג שווא למשתמש הרגיל האומר שהוא קיבל תשובה מ IP שלא קיים, כאילו היה קיים...



ראה ש 1.2.3.4 אין קיים ומתקבל reply.