

רשותות תקשורת - מטלה רשות על פרוטוקול QUIC

מגישיים:

ארד בן מנשה 207083353

עפרי ליפור 206391054

מצורף - הקלטות ווירשארק עם שמות בהתאם לחלק המבוצע ו-5 קבצי קוד:
sniffer.c,spoofing.c,attacker.c,makefile,gateway.c

חלק א' ה-sniffer

ב חלק א אנו יוצרים Sniffer ל חבילות tcp עם data כמו ב מטלה 2 - קלומר יש ל data מידע ש בראשו header ייחודי עם מידע נוסף על החבילה. אנו נרצה להוציא את המידע ולהדפיס את כל התוכן בחבילה ב הידון הקסדצימלי.

שאלות

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?

תשובה:

צרי root privileges כדי להריץ spoofer מכמה סיבות אפשריות:

1. המחשב רוצה לרוחה בעזרת הcryptosystem רשות לא רק חבילות למחשב האישי, אלא את כל הרשות ולשם כך הוא צריך להפעיל promiscuous mode שמאפשר לו לראות את כל המידע ברשת.

2. בשביל ליצור socket RAW כמו שלמדנו, צריך הרשות מיוחדת.
אנחנו בחלק א' גם צריכים את סעיף 1 וגם את 2, لكن אנו חייבים להריץ את התוכנית עם
הרשות הנקוונת.

הסביר על Sniffer

אנו פותחים RAW socket ומאזינים דרכו למידע לאחר שהגדכנו אותו מסוג promiscuous mode ומזהים חבילות tcp הקשורות לפורט 9999, ככלומר אם פורט 9999 הוא השולח או המתקבל. אנו בוחרים ב-9999 כיון שהוא הפורט של השירות שלנו מוגדרה.

אחרי שנכנסת ויצאת מהשרת אנחנו רוצים לבדוק את התוכן שלה - בהתאם ל-header שסופק לנו במתלה 2 בקובץ `host` על איך `data` של החבילות נראב ואיזה מידע מושבר

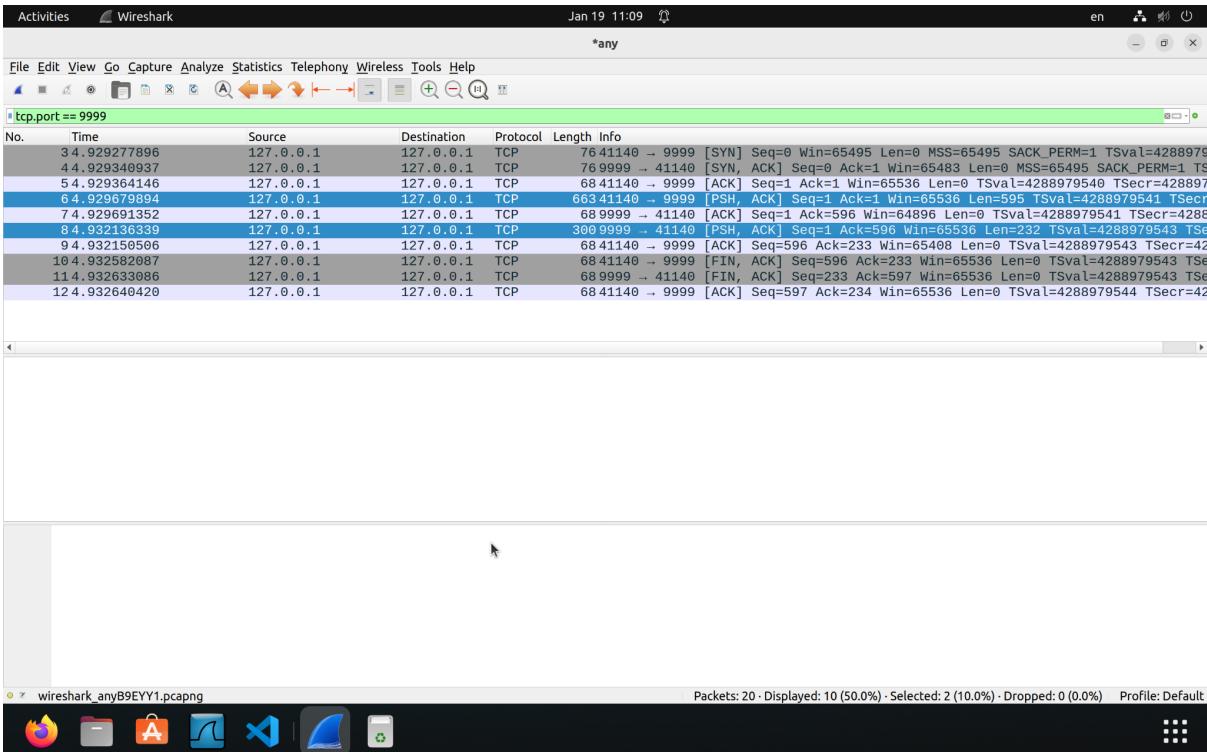
הרצת ה-Sniffer

לאחר יצירת התכנית בעזרת ה `makefile`, אנחנו מרים את התוכנית והיא מzdina כפ' שהסבנו לפ' כן. לאחר מכן אנחנו מפעילים את הסרבר במתלה 2 ואת הלוקה בזמן `wireshark` פועל והחbillת בקשה והתשובה מודפסת לקובץ טקסט עם כל הפרטים של header המיעוד של מטלה 2.

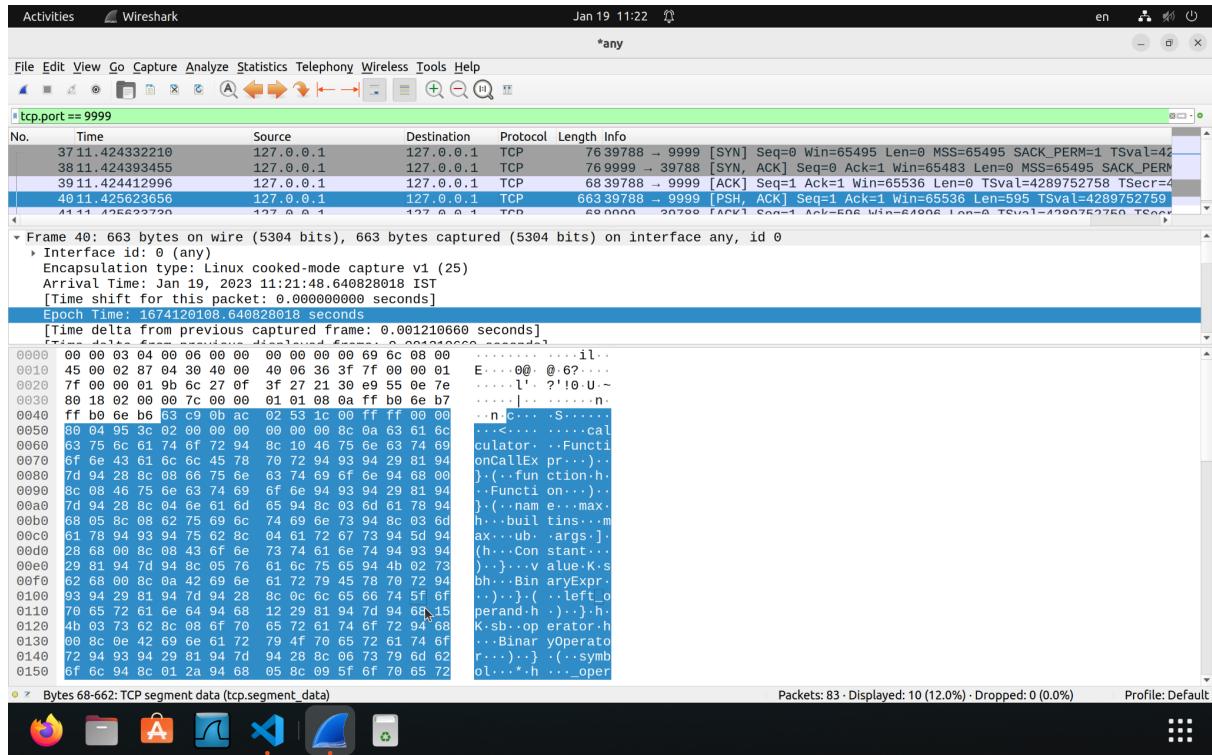
```
arad@arad:~/Desktop/Projects/Networking_22/Task_5$ sudo ./Sniffer  
[sudo] password for arad:  
started sniffing around...
```

```
arad@arad:~/Desktop/Projects/Networking_22/Task_2$ python  
3 server.py  
Listening on 127.0.0.1:9999  
Connection established with 127.0.0.1:41140  
{127.0.0.1:41140} Got request of length 595 bytes  
{127.0.0.1:41140} Sending response of length 232 bytes  
{127.0.0.1:41140} Connection closed  
  
● arad@arad:~/Desktop/Projects/Networking_22/Task_2$ python  
n3 client.py  
{127.0.0.1:9999} Connection established  
{127.0.0.1:9999} Sending request of length 595 bytes  
{127.0.0.1:9999} Got response of length 232 bytes  
Result: 42  
Steps:  
max(2, (3 * 4), log(E), (6 * 7), (9 / 8)) = max(2, 12, l  
og(2.718281828459045), (6 * 7), (9 / 8))  
= max(2, 12, 1  
.0, (6 * 7), (9 / 8))  
= max(2, 12, 1  
.0, 42, (9 / 8))  
= max(2, 12, 1  
.0, 42, 1.125)  
= 42  
{127.0.0.1:9999} Connection closed  
○ arad@arad:~/Desktop/Projects/Networking_22/Task_2$
```

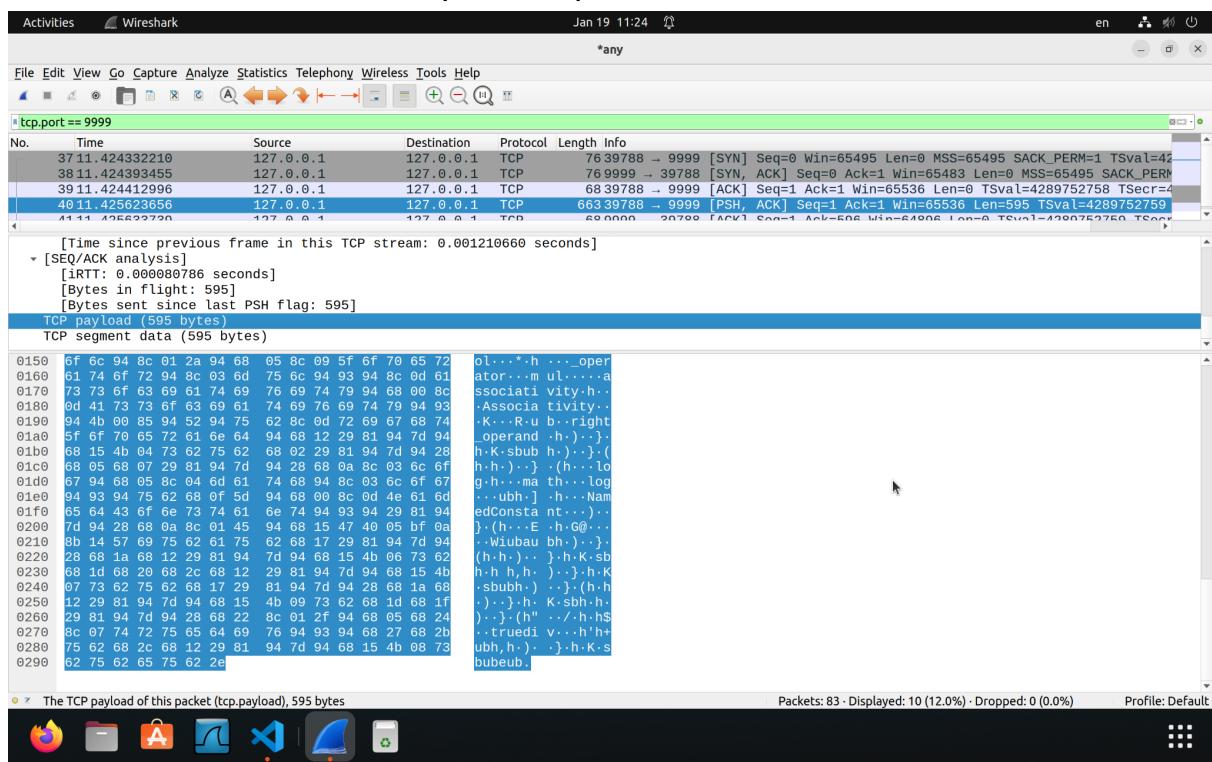
בתמונה הבאה אנו רואים את הרצאה של הקבצים הנ"ל. בתמונה הבאה אנו רואים את הרצאה של התכנית ב-wireshark (קובץ partA-1.pcapng).



כעת, נושאנו בין ה-headers לתוכנית שלנו ונו נראה שבגלו שיש לנו את header של ה-data אנו יכולים להוציא מידע נוסף wireshark לא יכול להוציא.



פה אנו רואים את הזמן שרשום ב- wireshark ואיך היא מתחילה.
בתמונה הבאה אנו רואים את גודל data של tcp ו את סוף החבילה.



הקובץ שאנו מקבלים לאחר הרצה:

```
Task_5 > 207083353_206391054.txt
1 REQUEST:
2 { source_ip: 127.0.0.1, dest_ip: 127.0.0.1, source_port: 9999, dest_port: 39788, timestamp: 1674120108,
3 total_length: 595, cache_flag:1, steps_flag:1, type_flag:1, status_code:0, cache_control:65535, data:
4 63 c9 0b ac 02 53 00 1c ff ff 00 00 80 04 95 3c 02 00 00 00 00 00 00 8c 0a 63 61 6c 63 75 6c
5 61 74 6f 72 94 8c 10 46 75 6e 63 74 69 6f 6e 43 61 6c 6c 45 78 70 72 94 93 94 29 81 94 7d 94 28
6 8c 08 66 75 6e 63 74 69 6f 6e 94 68 00 8c 08 46 75 6e 63 74 69 6f 6e 94 93 94 29 81 94 7d 94 28
7 8c 04 6e 61 6d 65 94 8c 03 6d 61 78 94 68 05 8c 08 62 75 69 6c 74 69 6e 73 94 8c 03 6d 61 78 94
8 93 94 75 62 8c 04 61 72 67 73 94 5d 94 28 68 00 8c 08 43 6f 6e 73 74 61 6e 74 94 93 94 29 81 94
9 7d 94 8c 05 76 61 6c 75 65 94 4b 02 73 62 68 00 8c 0a 42 69 6e 61 72 79 45 78 70 72 94 93 94 29
10 81 94 7d 94 28 8c 0c 6c 65 66 74 5f 6f 70 65 72 61 6e 64 94 68 12 29 81 94 7d 94 68 15 4b 03 73
11 62 8c 08 6f 70 65 72 61 74 6f 72 94 68 00 8c 0e 42 69 6e 61 72 79 4f 70 65 72 61 74 6f 72 94 93
12 94 29 81 94 7d 94 28 8c 06 73 79 6d 62 6f 6c 94 8c 01 2a 94 68 05 8c 09 5f 6f 70 65 72 61 74 6f
13 72 94 8c 03 6d 75 6c 94 93 94 8c 0d 61 73 73 6f 63 69 61 74 69 76 69 74 94 68 00 8c 0d 41 73
14 73 6f 63 69 61 74 69 76 69 74 79 94 93 94 4b 00 85 94 52 94 75 62 8c 0d 72 69 67 68 74 5f 6f 70
15 65 72 61 6e 64 94 68 12 29 81 94 7d 94 68 15 4b 04 73 62 75 62 68 02 29 81 94 7d 94 28 68 05 68
16 07 29 81 94 7d 94 28 68 0a 8c 03 6c 6f 67 94 68 05 8c 04 6d 61 74 68 94 8c 03 6c 6f 67 94 93 94
17 75 62 68 0f 5d 94 68 00 8c 0d 4e 61 6d 65 64 43 6f 6e 73 74 61 6e 74 94 93 94 29 81 94 7d 94 28
18 68 0a 8c 01 45 94 68 15 47 40 05 bf 0a 8b 14 57 69 75 62 61 75 62 68 17 29 81 94 7d 94 28 68 1a
19 68 12 29 81 94 7d 94 68 15 4b 06 73 62 68 1d 68 20 68 2c 68 12 29 81 94 7d 94 68 15 4b 07 73 62
20 75 62 68 17 29 81 94 7d 94 28 68 1a 68 12 29 81 94 7d 94 68 15 4b 09 73 62 68 1d 68 1f 29 81 94
21 7d 94 28 68 22 8c 01 2f 94 68 05 68 24 8c 07 74 72 75 65 64 69 76 94 93 94 68 27 68 2b 75 62 68
22 2c 68 12 29 81 94 }
23 |
24 RESPOND:
25 { source_ip: 127.0.0.1, dest_ip: 127.0.0.1, source_port: 39788, dest_port: 9999, timestamp: 1674120108,
26 total_length: 232, cache_flag:1, steps_flag:1, type_flag:0, status_code:200, cache_control:65535, data:
27 63 c9 0b ac 00 e8 c8 18 ff ff 00 00 80 04 95 d1 00 00 00 00 00 00 00 4b 2a 5d 94 28 8c 29 6d
28 61 78 28 32 2c 20 28 33 20 2a 20 34 29 2c 20 6c 6f 67 28 45 29 2c 20 28 36 20 2a 20 37 29 2c 20
29 28 39 20 2f 20 38 29 29 94 8c 34 6d 61 78 28 32 2c 20 31 32 2c 20 6c 6f 67 28 32 2e 37 31 38 32
30 38 31 38 32 38 34 35 39 30 34 35 29 2c 20 28 36 20 2a 20 37 29 2c 20 28 39 20 2f 20 38 29 29 94
31 8c 21 6d 61 78 28 32 2c 20 31 32 2c 20 31 2e 30 2c 20 28 36 20 2a 20 37 29 2c 20 28 39 20 2f 20
32 38 29 29 94 8c 1c 6d 61 78 28 32 2c 20 31 32 2c 20 31 2e 30 2c 20 34 32 2c 20 28 39 20 2f 20 38
33 29 29 94 8c 1a 6d 61 78 28 32 2c 20 31 32 2c 20 31 2e 30 2c 20 34 32 2c 20 31 2e }
```

קל לראות כי התחלה של ההדפסה והוסף תואמים בחבילת Request, וגם שלו הפרט
מידע תואמים ונוכנים לוירשארק ולסוג החביליה שנשלחה בתור data(למשל הסטוטו
וה-cache control באמצעות מציגים מידע תקין שתואם ל-ip ו-h-flag משתנה בהתאם
לסוג החביליה שנשלחה).

Spoofing - חלק ב' - שאלות

Question 1.

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

תשובה:

לא. כיוון שככל שאר החישובים בחבילה הם בהתאם לגודל החבילה המקורי, אם אנחנו מאפסים את הגודל אז כל שאר החישובים הם שגויים ויבצרו בעיות. למשל כאשר ניסינו השתמש בspoofed length, איפסנו את ה-packet `length` לו (ניתן לראות התמונה הבאה) הוירשארק לא זיהה את הפקטה - המחשבזרק את החבילה כי היא לא תקנית.

Question 2.

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

תשובה:

לא. כפי שלמדנו UDP הוא פרוטוקול לא אמין, המידע נשלח ואין דרכיים כמו ב-TCP לאמת שהתקבל המידע או לבדוק ERRORS בפקטה שנשלחה. לכן, אין חובה להשתמש `checksум`, אך ניתן להוסיף כדי לבטח טיפה שהחבילת שmag'עה היא כולה - וגם הביטוח הקטן זהה מאוד רגיש כי הוא סופר את האחדים בחלוקת ויכול לҚරות שאחד נכבה והשני נדליך - והוא `checksум` נכון ארך הפקטה פגומה.

הסביר על Spoofing

הקובץ Spoof.c, לאחר קימפול והרצה, שולח חבילה ICMP לכתובת הנבחרת, מהכתובת הנבחרת - יכולומר ניתן לשנות את המקור של החבילה ולשלוח אותה להלאה - יכולומר לזייף את המקור של החבילה. בתמונה הבאה ניתן לראות בקוד איפה אנו מושנים את המקור והכתובת מוגרב:

```
/*********************************************
 | Step 2: Fill in the IP header.
 *****/
struct ipheader *ip = (struct ipheader *)buffer;
ip->iph_ver = 4;
ip->iph_ihl = 5;
ip->iph_ttl = 20;
ip->iph_sourceip.s_addr = inet_addr("172.17.0.1");
ip->iph_destip.s_addr = inet_addr("8.8.8.8");
ip->iph_protocol = IPPROTO_ICMP;
ip->iph_len = htons(sizeof(struct ipheader) +
                     sizeof(struct icmpheader));
```

בדוגמה הנ"ל, כאשר נריץ את התכנית, חבילת ICMP רגילה תשלוח מהמחשב לשרתים של גוגל ונקבל תשובה(הרצתי פעמיים וקיבلتி כל פעם תשובה):

No.	Time	Source	Destination	Protocol	Length	Info
70	Jan 18 23:44	192.168.64.9	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 8)
80	Jan 18 23:44	8.8.8.8	192.168.64.9	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=118 (request in 7)
90	Jan 18 23:44	192.168.64.9	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 10)
100	Jan 18 23:44	8.8.8.8	192.168.64.9	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=118 (request in 9)

עכשו נשנה את כתובת המקור לכתובת מזויפת למשל 1.2.3.4, נשלח ונראה שהתקבל:

No.	Time	Source	Destination	Protocol	Length	Info
197	Jan 18 23:53	1.2.3.4	172.17.0.1	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 198)
198	Jan 18 23:53	172.17.0.1	1.2.3.4	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 197)
208	Jan 18 23:53	1.2.3.4	172.17.0.1	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 209)
209	Jan 18 23:53	172.17.0.1	1.2.3.4	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 208)

*הערה: שינוו את כתובת המקור לכתובת מזויפת ובחרנו שהיעד יהיה הכתובת המקומית של המחשב כדי שניראה את הבקשה והתשובה.

כעת, כדי לשלוח הודעות בפרוטוקולים אחרים עושים שינוי פשוט - כל שעליינו לעשות זה לשנות את קטע הקוד הבא:

```
// ****
// Spoof an ICMP echo request
// ****
int main()
{
    char buffer[1500];

    memset(buffer, 0, 1500);

    // Step 1: Fill in the ICMP header.
    // ****
    struct icmpheader *icmp = (struct icmpheader *) (buffer + sizeof(struct ipheader));
    icmp->icmp_type = 8; // ICMP Type: 8 is request, 0 is reply.

    // Calculate the checksum for integrity
    icmp->icmp_chksm = 0;
    icmp->icmp_chksm = (unsigned short)in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));

    // ****
    // Step 2: Fill in the IP header.
    // ****
    struct ipheader *ip = (struct ipheader *)buffer;
    ip->iph_ver = 4;
```

הסיבה לכך היא שבשלב זהה אנחנו רק יוצרים header icmp ומכוונים אותו עם ה-data לתחילת ה-buffer שאנו מוכנים לשלחן. אם היינו רצימ לשלוח udp/tcp או כל פרוטוקול אחר, היינו צריכים לבנות header של הפרטוקול הנבחר, להכניס את ה-data בהתאם ודומה.

חלק ג'

הוראות הרצה של חלק ג':

1. פתח את קבצי הדוקר מה-"[tirgut](#)".
2. פתח את מסוף אוביונטו בתוך הקובץ [Labsetup](#).
3. העבר את הקובץ [attacker.c](#) לתיקיה שנקראת "volumes" בתוך קובץ [Labsetup](#).
4. בטרמינל, הפעיל את הפקודות הבאות:
 5. **sudo docker-compose build**
 6. **sudo docker-compose up**
 7. פתחו שני טרמינלים נוספים, והפעילו את הפקודה:
docker ps
 8. הנ"ל יציג את המכוולות הפעולות ואת המזהים שלהן.
 9. כדי לפתחו את טרמינל התוקף או [hostA](#), הפעיל את הפקודות הבאות בטרמינל חדש:
 11. **docker exec <container id> /bin/bash**
 12. **docker exec -it <container id> /bin/bash**
 13. כדי להפעיל את פקודת **ping -n hostA**, הפעיל את הפקודה הבאה בתוך הטרמינל שלו:
ping <IP יעד>
 14. כדי להפעיל את קוד התוקף ברקע, עברו אל טרמינל התוקף והפעיל את הפקודות הבאות:
 16. **sudo gcc -o attacker attacker.c -lpcap**
 17. **./attacker**

סיכום:
הוראות לעיל מסופקות כדי להדריך את הקורא כיצד להגיד ולהפעיל את הסימולציה הכוללת את [hostA](#) ואת סקריפט התוקף. ההוראות מכסות את תהליך פתיחת הקבצים הדרושים, העברת סקריפט התוקף, בניה והרצה של הקונטיינרים באמצעות [docker-compose](#), פתיחת הטרמינל עבור [hostA](#) והתוקף, הפעלת פקודת **-ping** וסקריפט התוקף. ביצוע הוראות אלו בסדר הנכון יאפשר לך ריץ בהצלחה את הסימולציה ולכוד את הנתונים הדרושים.

a. First run – send a ping from Host A to Host B.

בריצה a אנחנו שולחים פינג מובנה (כמו שאושר בפורום בשל הדוקרים הקיימים שהביאו לנו) מה host ל b וigel של host הוא "alive" אז הוא אמור לשולח חזרה reply. ואנחנו מצפים שהattacker שלנו ישלח בנוספ reply.

הרצת הדוקרים

```
Leifer@DESKTOP-DS0857U:/mnt/c/Users/user/Documents/c/network/network_tasks/tasks5/LabSetup$ sudo docker-compose build
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
Leifer@DESKTOP-DS0857U:/mnt/c/Users/user/Documents/c/network/network_tasks/tasks5/LabSetup$ sudo docker-compose up
Starting seed-attacker ... done
Starting hostA-10.9.0.5 ... done
Starting hostB-10.9.0.6 ... done
Attaching to seed-attacker, hostB-10.9.0.6, hostA-10.9.0.5
hostB-10.9.0.6 | * Starting internet superserver inetd
hostA-10.9.0.5 | * Starting internet superserver inetd
```

הרצת seed-attacker והקנת פקודת הרצה

```
Leifer@DESKTOP-DS0857U:/mnt/c/Users/user/Documents/c/network/network_tasks/tasks5/LabSetup$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
def53cb6c21c handsonsecurity/seed-ubuntu:large "bash -c '/etc/init.d..." 2 hours ago Up 2 minutes hostB-10.9.0.6
2b83592a1247 handsonsecurity/seed-ubuntu:large "bash -c '/etc/init.d..." 2 hours ago Up 2 minutes hostA-10.9.0.5
10b0253dcdf handsonsecurity/seed-ubuntu:large "/bin/sh -c /bin/bash" 6 hours ago Up 2 minutes seed-attacker
Leifer@DESKTOP-DS0857U:/mnt/c/Users/user/Documents/c/network/network_tasks/tasks5/LabSetup$ docker exec 10 /bin/bash
root@DESKTOP-DS0857U:/# cd volumes/attacker
root@DESKTOP-DS0857U:/volumes/attacker# sudo gcc -o attacker attacker.c -lpcap
gcc: error: attacker.c: No such file or directory
root@DESKTOP-DS0857U:/volumes/attacker# cd ..
root@DESKTOP-DS0857U:/volumes/attacker# sudo ./attacker
```

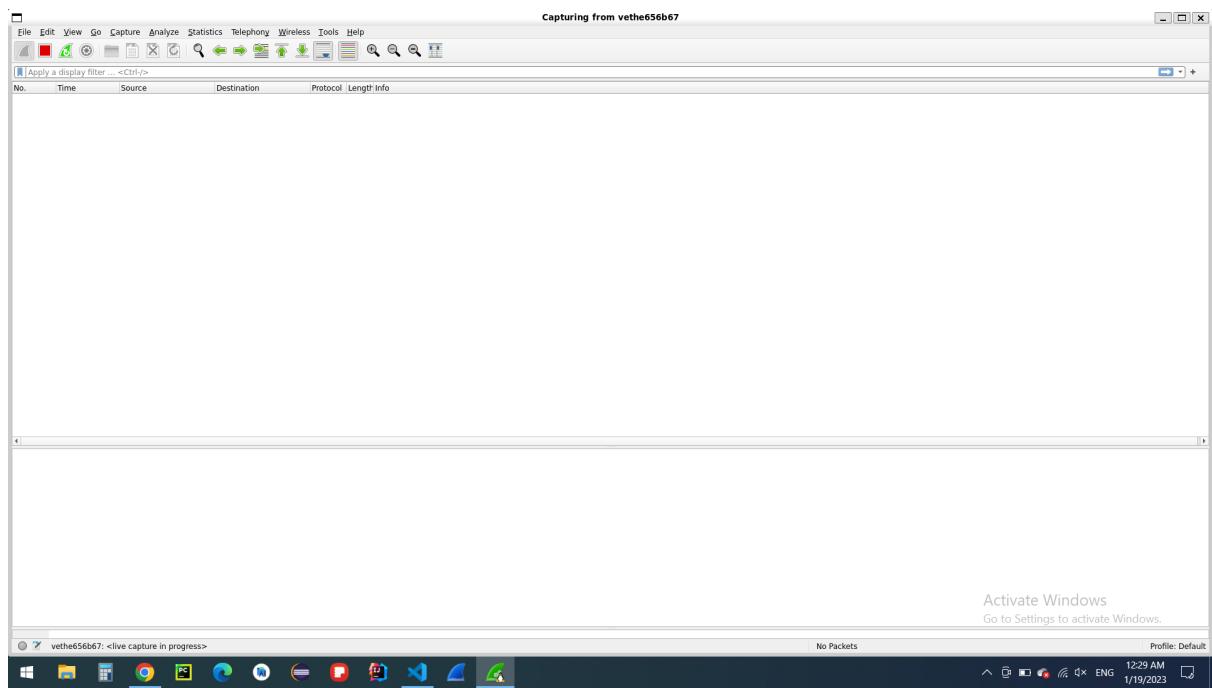
הרצה A hostB והכנת הרצת פינג ל hostA

The screenshot shows a Visual Studio Code interface with the 'TERMINAL' tab selected. The terminal window displays the following command-line session:

```
leifer@DESKTOP-DS0857U:/mnt/c/Users/user/Documents/c/network/network_tasks/tasks5/Labsetup$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
2bf53ch6c21c handsonsecurity/seed-ubuntu:large "bash -c '/etc/init..." 2 hours ago Up 15 seconds hostB-10.9.0.6
2b83592a1247 handsonsecurity/seed-ubuntu:large "bash -c '/etc/init..." 2 hours ago Up 15 seconds hostA-10.9.0.5
10b0253dcdf handsonsecurity/seed-ubuntu:large "/bin/sh -c /bin/bash" 6 hours ago Up 15 seconds seed-attacker
leifer@DESKTOP-DS0857U:/mnt/c/Users/user/Documents/c/network/network_tasks/tasks5/Labsetup$ docker exec -it 2b /bin/bash
root@2b83592a1247:/# sudo ping 10.9.0.6
```

The left sidebar shows a project structure under 'EXPLORER' with several files and folders related to network tasks and Docker setup. The status bar at the bottom indicates the current file is 'Makefile'.

הרצה הקלטת ווירשאך ברקע



הרצת attacker

The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal window displays the command 'root@DESKTOP-DS0867U:/volumes_attacker# ./attacker' followed by the message 'starting to sniffing...'. The file explorer on the left shows a project structure under 'NETWORK' with files like 'attacker.c', 'Gateway.c', 'Sniffer.c', and 'Sniffer.h'. A sidebar on the right lists various tools and services: powershell, partA, python client, python server, wireshark, hostA, docker Running, and attacker.

```
root@DESKTOP-DS0867U:/volumes_attacker# ./attacker
starting to sniffing...
```

הרצת פינג ל hostB

The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal window displays the command 'root@2b83592a1247:/# ping 10.9.0.6' followed by several ICMP echo requests and responses. It then shows '--- 10.9.0.6 ping statistics ---' and '5 packets transmitted, 5 received, 0% packet loss, time 4122ms'. The file explorer on the left shows the same project structure as the previous screenshot. A sidebar on the right lists the same set of tools and services.

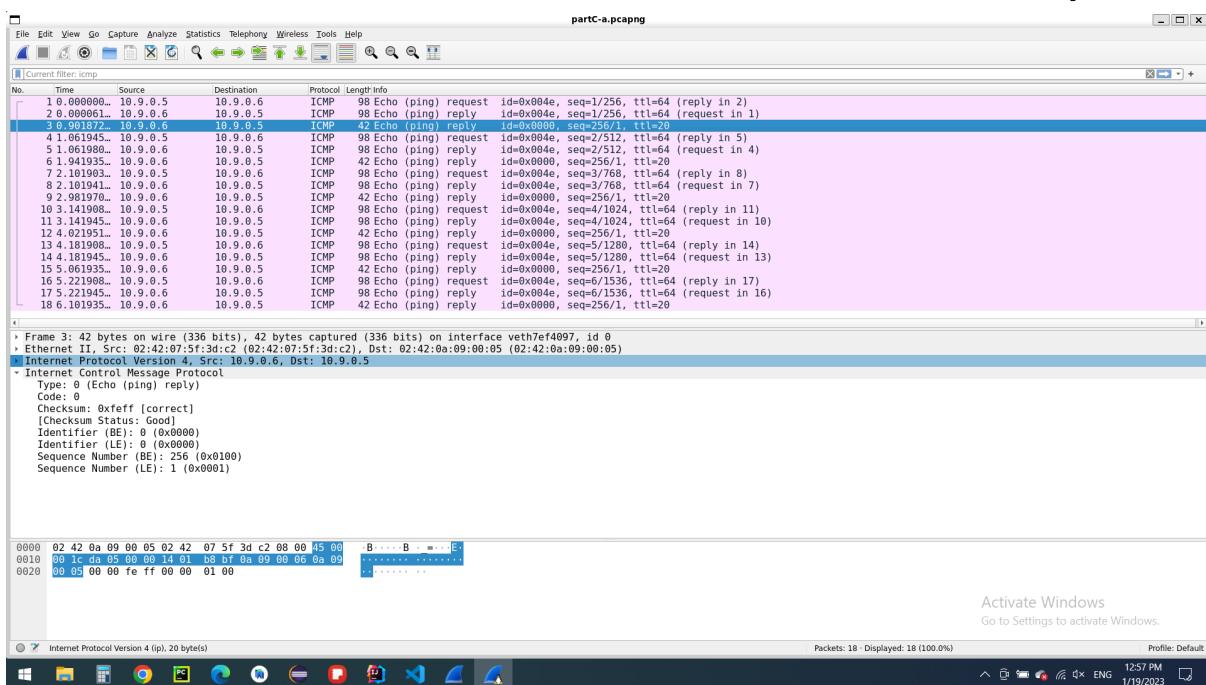
```
root@2b83592a1247:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.074 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.121 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.126 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.127 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.134 ms
--- 10.9.0.6 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4122ms
rtt min/avg/max/mdev = 0.074/0.116/0.134/0.021 ms
root@2b83592a1247:/#
```

הפלט בטרמינל של seed-attacker

```
root@DESKTOP-DS0857U:/volumes_attacker# ./attacker
starting to sniffing...
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
`~`
```

Activate Windows
Go to Settings to activate Windows.

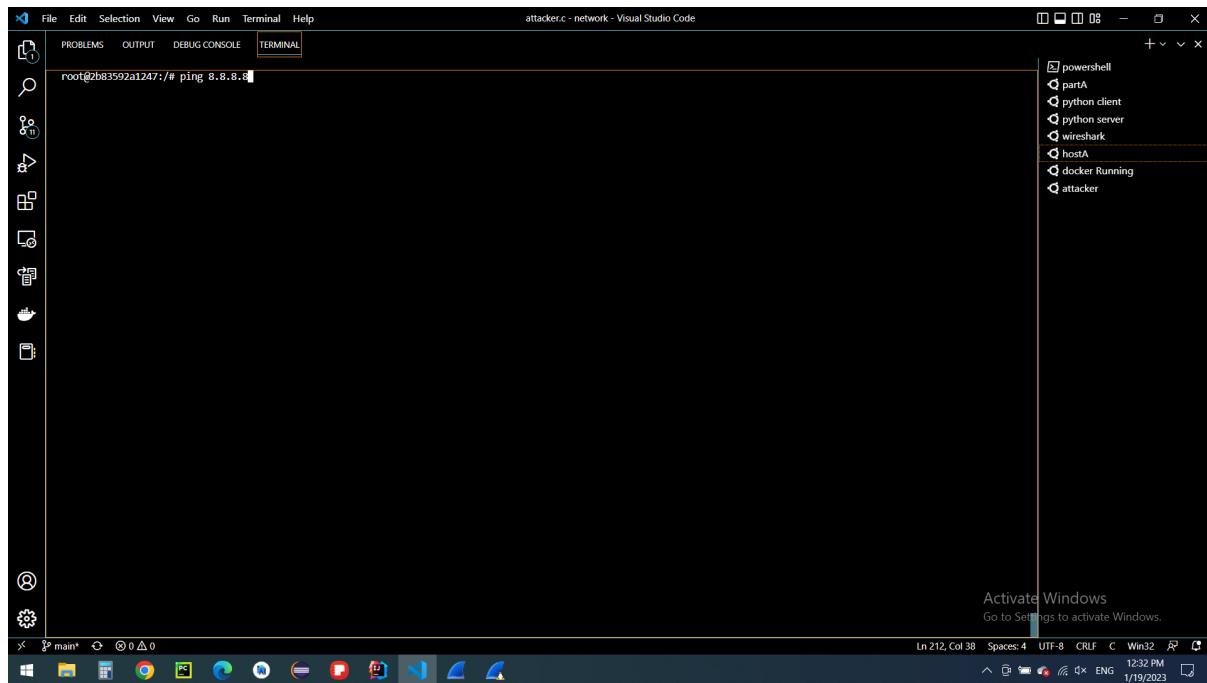
הקלטת וירשארק



כפי שניתן לראות, ה-attacker עבד כמצופה - בנוסוף לפינגים חזרה מ-`host`, אנחנו רואים ש-`host` עם כתובת 10.9.0.5 מקבל תשובה מכתובת 10.9.0.6 פעמיים - ואנו מזהים שהזאת אכן הודעה שנשלחה מה-attacker כי שמננו את ה-`host` להיות 0, וכן יש לנו חבילה נוספת שחזרה מ-`host`.

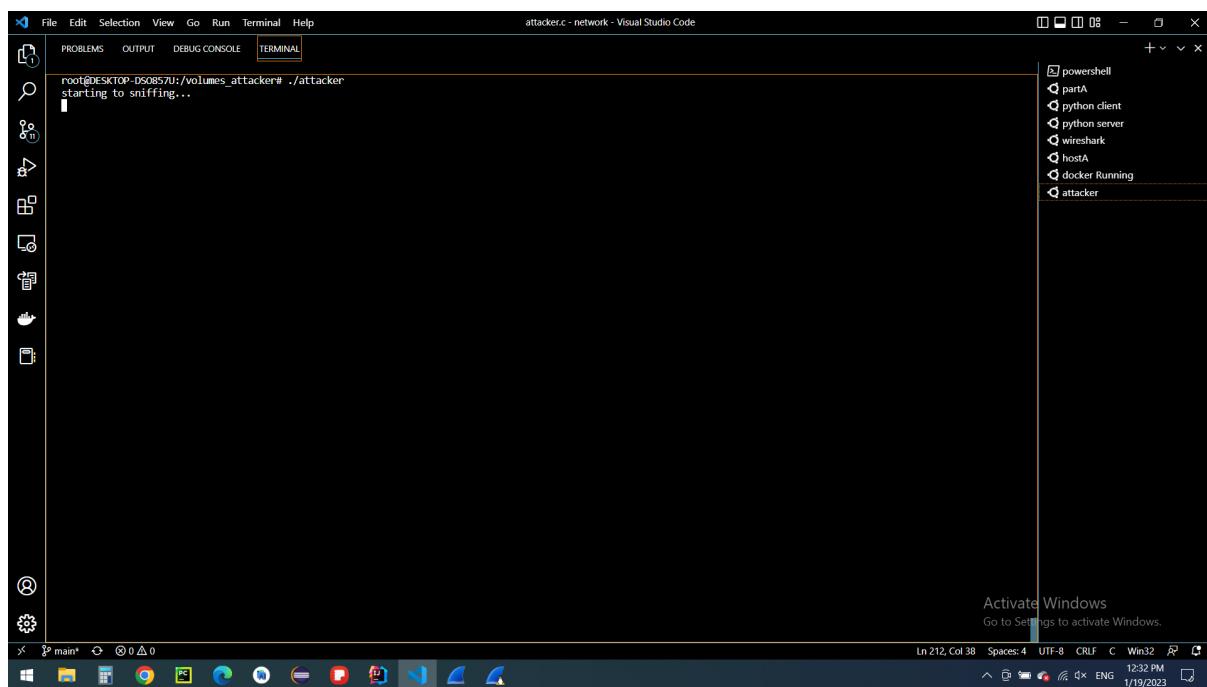
b. Second run – send a ping from Host A to a WAN IP (e.g., google DNS – 8.8.8.8)

הכנות הרצת ping לגוול (hostA רצים docker seed-attacker ו- hostA מסעיף קודם)



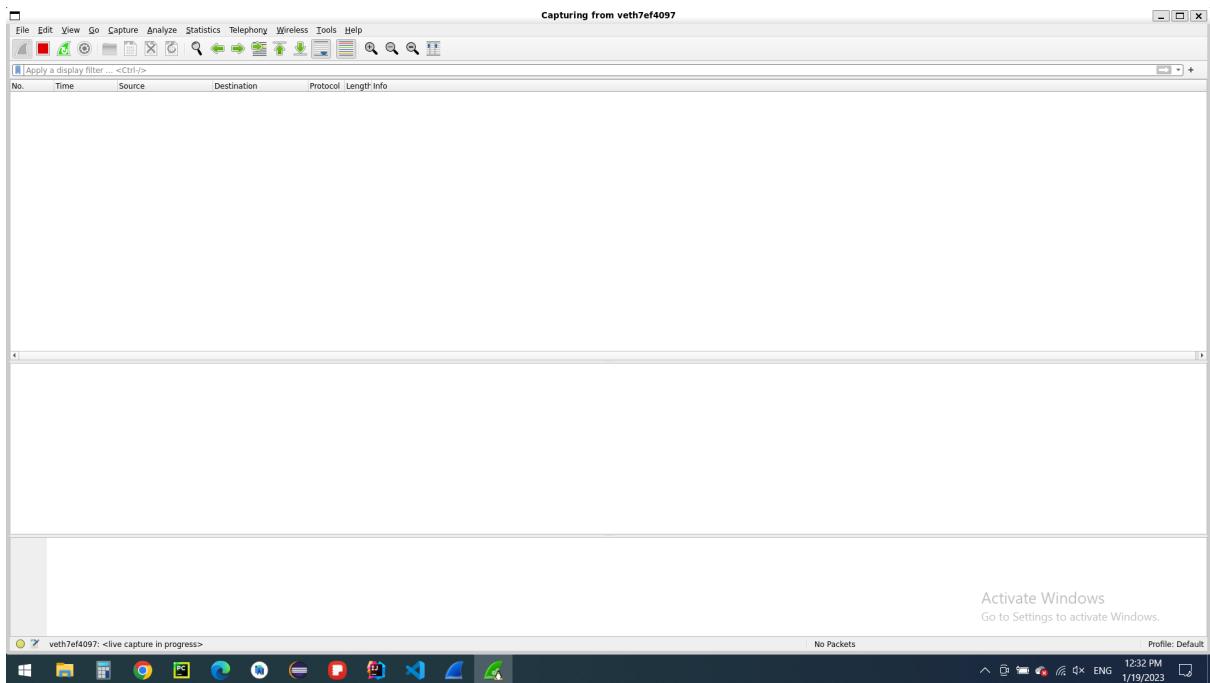
A screenshot of a Windows desktop environment showing a Visual Studio Code window. The terminal tab is active, displaying the command "root@2b83592a1247:/# ping 8.8.8.8". The status bar at the bottom shows "Ln 212, Col 38" and "12:32 PM 1/19/2023". The taskbar below includes icons for File Explorer, Task View, Start, and several pinned applications.

הרצת attacker

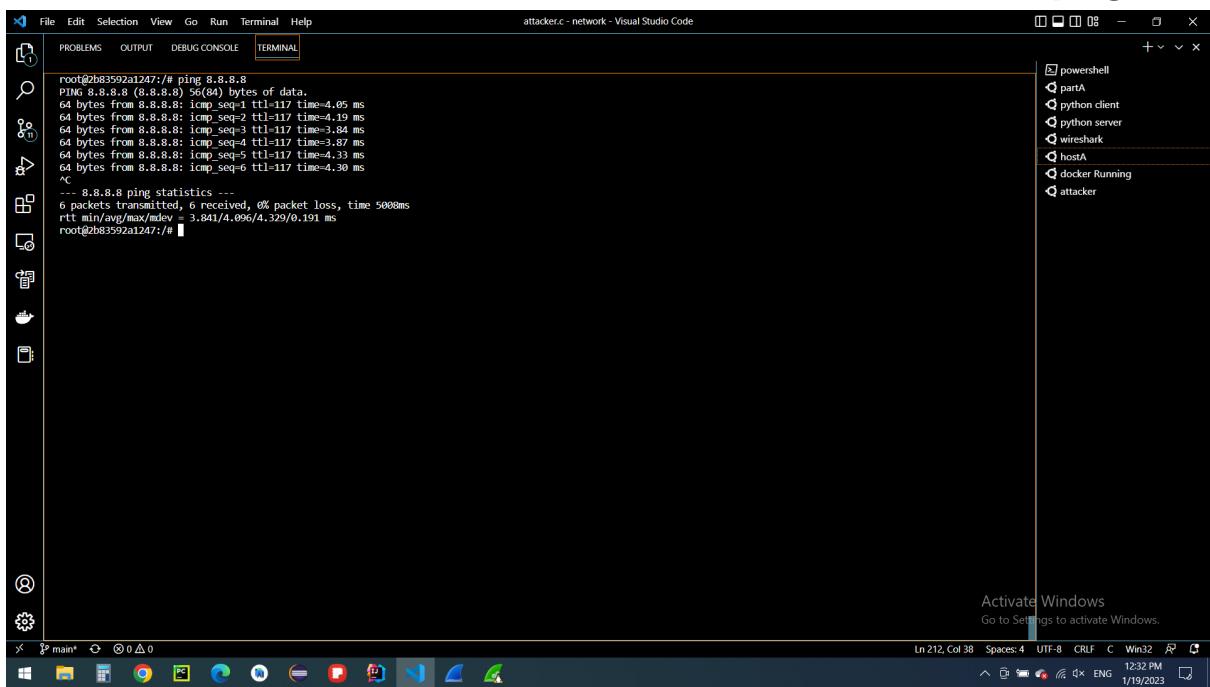


A screenshot of a Windows desktop environment showing a Visual Studio Code window. The terminal tab is active, displaying the command "root@DESKTOP-DS0857U:/volumes_attacker# ./attacker" followed by "starting to sniffing...". The status bar at the bottom shows "Ln 212, Col 38" and "12:32 PM 1/19/2023". The taskbar below includes icons for File Explorer, Task View, Start, and several pinned applications.

הקלת הווירשארק



הרצה ping לגוגל



קיבלה פלט מ-hostA: בשלב זה, אנו אוספים את הפלט מפקודת הפינג המובנית של מערכת אוביונטו. כפי שניתן לראות, הפלט מציג את סטטיסティקת הפינג, כגון מספר החבילות שנשלחו והתקבלו, קצב אובדן החבילות וזמן הילכה הלווי ושוב עבור החבילות. הפלט מציג גם את התגובהות הבודדות, כגון מספר icmp_seq ttl וזמן.

הפלט בטרמינל של seed-attacker

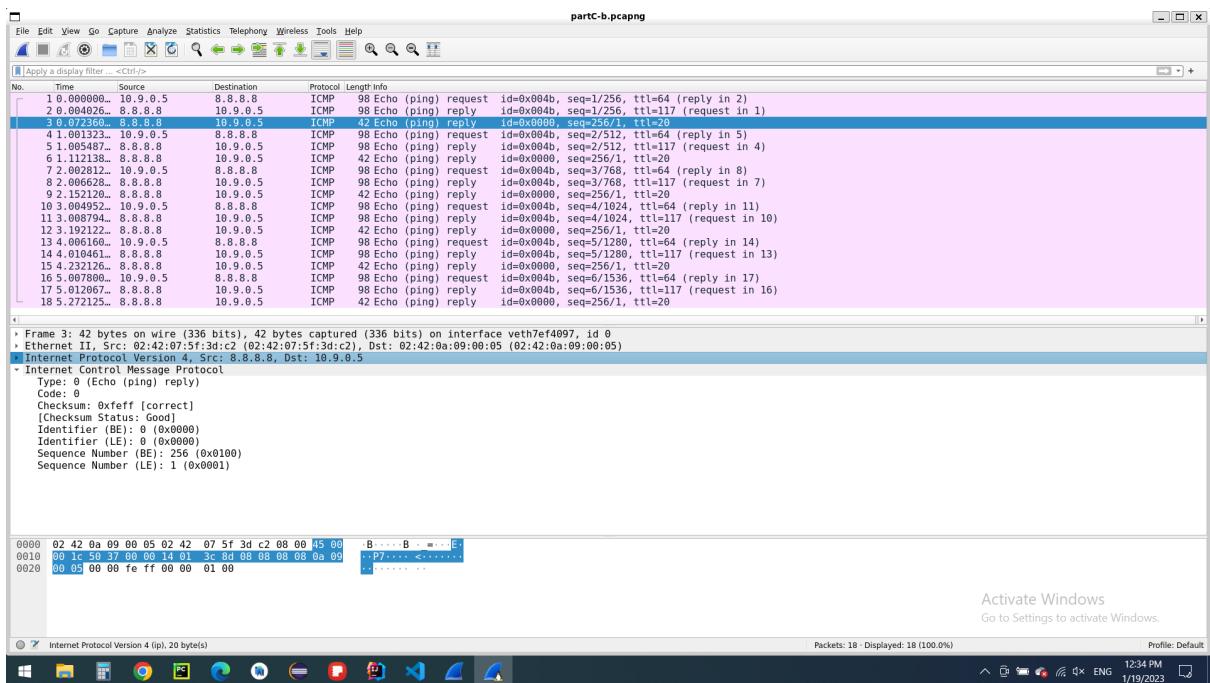
The screenshot shows a terminal window in Visual Studio Code with the title "attacker.c - network - Visual Studio Code". The terminal tab is selected. The output of the command "root@DESKTOP-DS0857U:/volumes_attacker# ./attacker" is displayed, showing the program's logic for sniffing ICMP requests and sending fake ICMP replays. The terminal interface includes a sidebar with icons for various tools like powershell, partA, python client, etc., and a status bar at the bottom showing file statistics and system information.

```
root@DESKTOP-DS0857U:/volumes_attacker# ./attacker
starting to sniffing...
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
^C
root@DESKTOP-DS0857U:/volumes_attacker#
```

הפעלת קוד התוקף: שלב זה כרוך בביצוע הסקריפט המכיל את הקוד ל-ICMP Packet Sniffer. רכיב הsnsniper של הקוד נועד לכלוד חבילות ICMP ECHO reply שנשלחות על ידי hostA שמרץ את סקריפט הפינג ברקע. כאשר נלכדת חבילה ICMP ECHO request, קוד התוקף שולח מיד חבילה תשובה מזויפת של ICMP ECHO למקור הבקשה, שהוא hostA במקרה זה.

קבלת פלט מהתוקף: לאחר הפעלת סקריפט התוקף, הפלט נאוסף ומונוטח. פלט זה מראה שהסקריפט פועל, מתחילה לרוחה, ושהוא מזזה בקשת ICMP, הוא שולח שידור חוזר של ICMP מזויף עבור הבקשה שזיהה.

הקלטת הוירשארק



שלב זה כולל ניתוח הנתונים שנלכדו על ידי Wireshark בשלבים הקודמים. הנתונים מציגים את הפרטים של חבילות-ICMP שהחלפו בין hostA IP: 10.9.0.5 לבין hostB IP: 8.8.8.8 אשר היעד הוא הדומין של גוגל.

מנתוני Wireshark, אנו יכולים לראות שhost-A שולח בקשות ping ICMP Echo, ומארח היעד שולח תשובות pong ICMP Echo. הנתונים מראים גם שקוד התוקף שולח תשובות מזויות של ICMP Echo, אותן ניתן לראות למשל בחבילה מס' 3. ניתן להזיהות את חבילות התשובות המזויות לפי ערכי הזיהוי השונים ומספר הרცף השונים בהשוואה לתשובות הילגיטימיות מארח היעד, עשינו כך בכונה על מנת לזהות את המזויות.

מטרת התוקף בתרחיש זה היא לירט ולתפעל את חבילות-ICMP שהחלפו בין hostA למארח היעד. על ידי שיליחת חבילת תשובה מזעית של ICMP Echo, התוקף עלול לששב את התקשרות בין מארח A למארח היעד, או להטעות מארח A להאמין שניתן להגעה אל מארח היעד כאשר אינו.

על ידי ניתוח נתוני Wireshark, אנו יכולים לראות את הפרטים של חבילות-ICMP שהחלפו ואת ההשפעה של פעולות התוקף על התקשרות בין hostA למארח היעד. זה מאפשר לנו להבין טוב יותר את התנהוגות של סקריפט התוקף ואת ההשפעות הפוטנציאליות שלו על התקשרות בראשת.

c. Third run – send a ping from Host A to a fake IP.

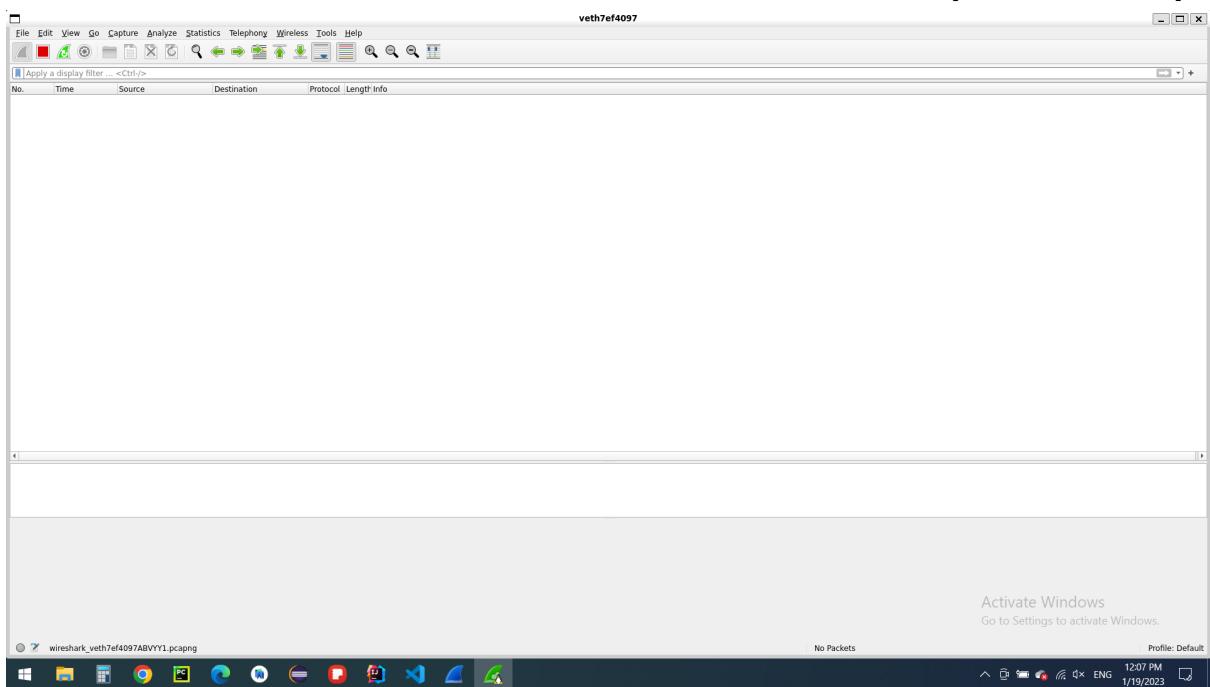
הכנות הרצת ל ip מזויף(A) docker seed-attacker ו hostA רצים מסעיף קודם

A screenshot of a Windows desktop environment showing a Visual Studio Code window. The terminal tab is active, displaying the command 'ping 2.2.2.2'. The status bar at the bottom shows the path 'main*' and the date/time '1/19/2023 11:50 AM'. The sidebar on the right lists several Docker containers: powershell, partA, python client, python server, wireshark, hostA, docker Running, and attacker. The 'hostA' container is currently selected. A message 'Activated Windows' with a link 'Go to Settings to activate Windows' is visible in the center of the screen.

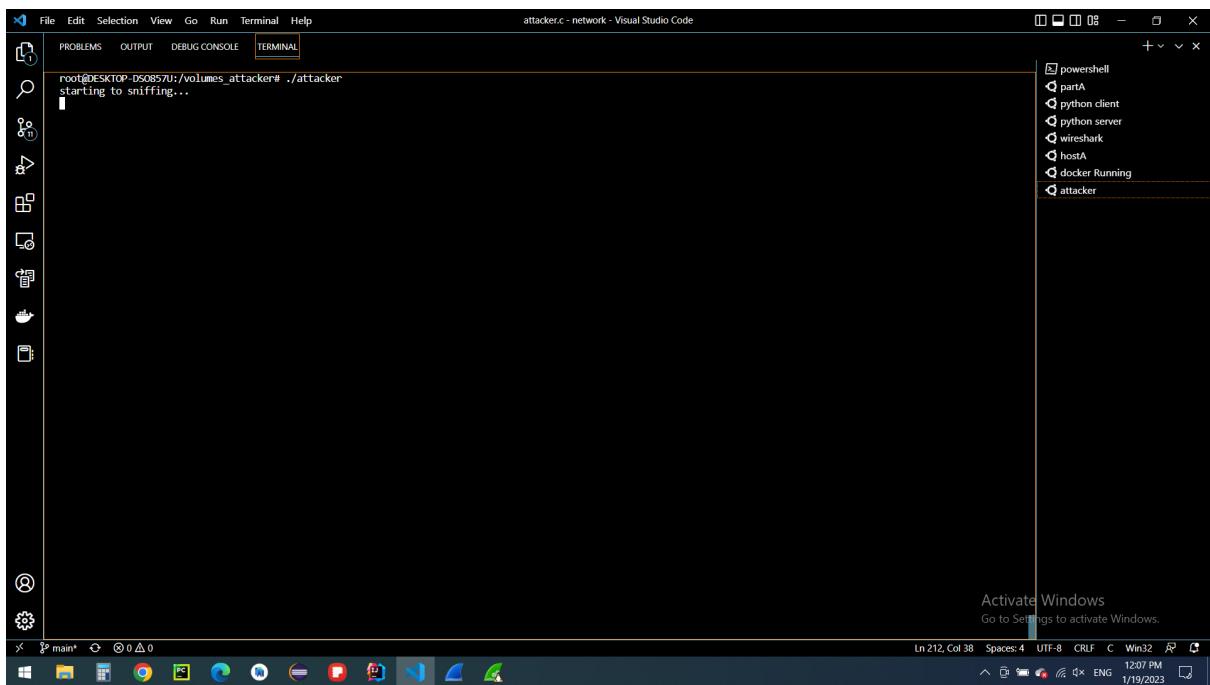
הכנות הרצת **attacker**

A screenshot of a Windows desktop environment showing a Visual Studio Code window. The terminal tab is active, displaying the command 'root@DESKTOP-DS0857U:/volumes/attacker# ./attacker'. The status bar at the bottom shows the path 'main*' and the date/time '1/19/2023 11:50 AM'. The sidebar on the right lists several Docker containers: powershell, partA, python client, python server, wireshark, hostA, docker Running, and attacker. The 'attacker' container is currently selected. A message 'Activate Windows' with a link 'Go to Settings to activate Windows' is visible in the center of the screen.

הקלטת הוירשארק



הרצת attacker



הפעלת קוד התוקף: שלב זה כרוך בביצוע הסקריפט המכיל את הקוד ל-ICMP Packet Sniffer. רכיב הסניפר של הקוד מwend למכוד חבילות ICMP ECHO reply שנשלחות על ידי host A שמרץ את סקריפט הפינג ברקע. כאשר נלכדת חבילה ICMP ECHO request, קוד התוקף שולח מיד חבילת תשובה מזויפת של ICMP ECHO למקור הבקשה, שהוא host A במקורה זה.

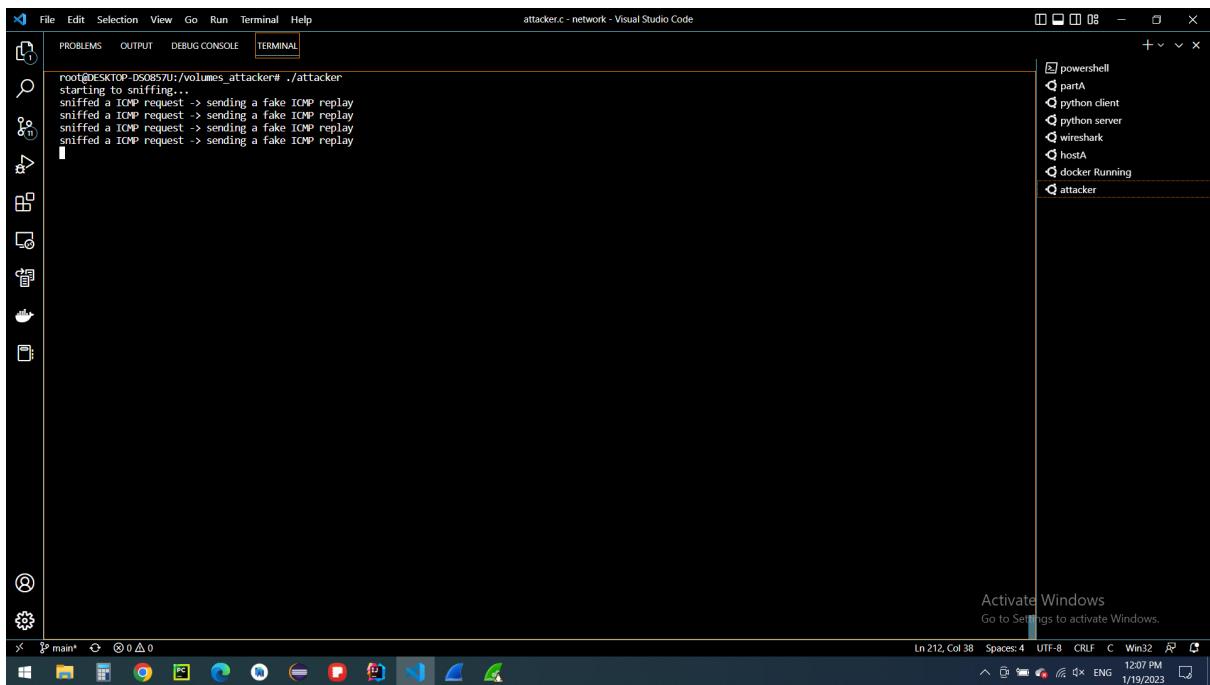
הרצה של ping ל host

The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal tab is selected, showing the command `ping 2.2.2.2` being run on a root shell. The output shows a ping to IP 2.2.2.2 with 56(84) bytes of data, 4 packets transmitted, 0 received, 100% packet loss, and a time of 3096ms. The status bar at the bottom indicates the terminal has 212 columns and 38 rows, and the encoding is UTF-8.

```
root@2b83592a1247:/# ping 2.2.2.2
PING 2.2.2.2 (2.2.2.2) 56(84) bytes of data.
<--- 2.2.2.2 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3096ms
```

בשלב זה, אנו משתמשים בפקודה ping מהתא host כדי לשЛОוח בקשות ICMP Echo לכתובת IP לא קיימת 2.2.2.2. אנו מצפים ללא תגובה מכותבת IP זו מכיוון שלא ניתן להגיע אליה. הפלט של פקודת ping מראה ש-4 חבילות שודרו אך אף אחת לא התקבלה, וכתוצאה לכך שיעור אובדן חבילות של 100%. זה מצבן שלא ניתן להגיע לマארח היעד וشبוקת הפינג נכשלה(2.2.2.2 הוא לא alive).

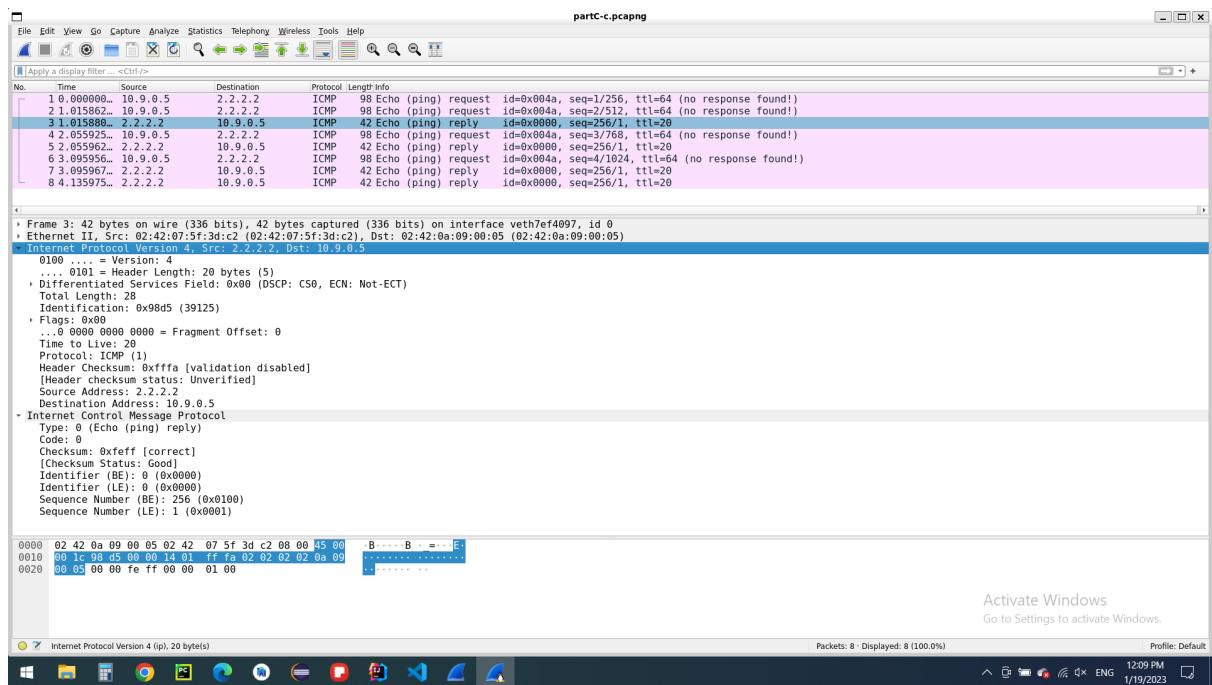
הפלט בטרמינל של seed-attacker



```
root@DESKTOP-D60857U:/volumes_attacker# ./attacker
starting to sniffing...
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
sniffed a ICMP request -> sending a fake ICMP replay
```

בשלב זה, אנו מרים את סקריפט התוקף ברקע בזמן שאנו שולחים את בקשות ה-ICMP Echo מ-hostA לכתובת IP המזוייפת. סקריפט התוקף מරחילה אחר חבילות בקשה של ICMP Echo וכאשר הוא מזהה אחת, הוא שולח חבילת תשובה מזויפת של ICMP Echo למקור הבקשה, שהוא hostA במקרה זה. הפלט מראה שהסקריפט פועל, מתחילה לרוחה ושזהו זיהה כמה חבילות בקשות ICMP, ואז הוא שולח שידור חוזר של ICMP מזויף עבור כל בקשה שזיהה.

הקלת הוירשארק



בשלב זה אנו מנתחים את הנתונים שנלכדו על ידי Wireshark בשלבים הקודמים. הנתונים מציגים את הפרטים של חבילות ה-ICMP שהוחלו בין IP hostA IP: 10.9.0.5 ו-IP המטרה היעד: 2.2.2.2 שהיא כתובה IP לא קיימת.

מנתוני Wireshark, אנו יכולים לראות ש-hostA שולח בקשות ping למאראח היעד, ומארח היעד אינו שולח תשובות ICMP Echo ping. במקרה זאת, קוד התוקף שולח תשובות מזיפות של ICMP Echo, אותן ניתן לראות בחבילה מספר 3. ניתן לזהות את מנוגת התשובות המזיפות לפי ערכי המזהה ומספר הרצף השונים בהשוואה לתשובות הלגיטימיות ממארח היעד, זאת עשינו בכונה על מנת לזהות אותם.

מטרת התוקף בתרחיש זה היא לירות ולתפעל את מנוגת ה-ICMP שהוחלו בין hostA למאראח היעד. על ידי שליחת חבילת תשובה מזיפית של ICMP Echo בזמן שלא ניתן להגיע למאראח היעד, התוקף יכול להטעתו מארח A להאמין שניין להגיע אל מאראח היעד כשלא ניתן להגיע אליו. זה יכול להזיק למנהל הרשות מכיוון שהוא יכול להוביל לאבחן שגוי של בעיות הרשות.

על ידי ניתוח נתונים Wireshark, אנו יכולים לראות את הפרטים של מנוגת ה-ICMP שהוחלו ואת ההשפעה של פעולות התוקף על התקשרות בין hostA למאראח היעד. זה מאפשר לנו להבין טוב יותר את ההתקנות של סקריפט התוקף ואת ההשפעות הפוטנציאליות שלו על התקשרות ברשות.

ח' ד' - Gateway

כפי שניתן לראות בקוד אנחנו מקבלים חבילות ומוסננים אותם לפי קו נבחר. לאחר מכן, כל חבילה שאנו מקבלים אנחנו מגירים עם סיכוי 50% האם לשלוח את החבילה כמו שהיא, לפורט $1+k$ כאשר k הוא הפורט ממנו הגיעה החבילה.

בתמונה הנ"ל אנו רואים מצד שמאל את ה **Gateway** בפעולה - מתחה לחבילות קדט שנשלחות ל 8.8.8.8 מג'רל מס' ובורר אם להעביר את החבילה לפורט 8001 או לא.

The figure shows a Wireshark interface with the following details:

- Activities**: Shows "Wireshark" as the active application.
- Date/Time**: Jan 18 23:09
- Search Bar**: *any
- Menu Bar**: File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help
- Toolbar Buttons**: New, Open, Save, Print, Capture, Stop, Filter, Find, Replace, Sort, Columns, Zoom, Help.
- Selected Filter**: udp.port == 8000 || udp.port ==8001
- Table Data** (Listed in the "Info" column):

No.	Time	Source	Destination	Protocol	Length	Info
42	4.2.757874934	127.0.0.1	127.0.0.1	UDP	58	54171 → 8000 Len=14
52	5.2.758097228	192.168.64.9	8.8.8.8	UDP	58	56698 → 8001 Len=14
124	12.4.889437839	127.0.0.1	127.0.0.1	UDP	58	55023 → 8000 Len=14
166	16.6.709679182	127.0.0.1	127.0.0.1	UDP	58	38063 → 8000 Len=14
176	17.6.709783350	192.168.64.9	8.8.8.8	UDP	58	56698 → 8001 Len=14
208	20.8.693966048	127.0.0.1	127.0.0.1	UDP	58	42661 → 8000 Len=14
218	21.8.694031798	192.168.64.9	8.8.8.8	UDP	58	56698 → 8001 Len=14
2810	28.10.747010391	127.0.0.1	127.0.0.1	UDP	58	50292 → 8000 Len=14
2910	29.10.747067099	192.168.64.9	8.8.8.8	UDP	58	56698 → 8001 Len=14
3312	33.12.762129372	127.0.0.1	127.0.0.1	UDP	58	37010 → 8000 Len=14
3415	34.15.308940215	127.0.0.1	127.0.0.1	UDP	58	60615 → 8000 Len=14

בווירשארק אנו רואים כי באמת החבילות מתקבלות ונשלחות הלאה, ויש פעמים שמתתקבל חבילה ולא מועברת שום חבילה הלאה. כלומר אנו יוצרים "Packet Loss" עם 50 אחוז איבוד פקטים.