

מטרה 5 רשתות תקשורת

מגימות

אבייה אורן 322273301

נתע כהן 325195774

תוכן עניינים:

taskA.....	2
taskB.....	7
taskC traceroute tool.....	10
taskD.....	12
מזהיף IP-ל A א. הפעלה ראשונה - שליחת פינגן hostA-10.9.0.5 ל hostB-10.9.0.6 :	13
ב. הפעלה שנייה - שליחת פינגן google DNS - 8.8.8.8 (למשל) IP-WAN ל A	14
ג. הפעלה שלישית - שליחת פינגן hostA-IP-ל A נשלח את הפינגן ל hostB-IP-ל B	15
ד. הפעלה רביעית - שליחת פינגן hostB-IP-ל B ל hostA-IP-ל A	16
taskE.....	18

taskA

We created a sniffer that can sniff packets of several kinds from the network.

Our sniffer uses scapy to sniff packets of many kinds: TCP, UDP, IP, ICMP, IGMP.

It writes down all the packet details on a txt file named “322273301_325195774.txt” (our ids).

To run the program you should write in the terminal:

sudo python3 sniffer.py

It is important to run the program with ‘sudo’ for root privilege, because it needs access to network interfaces, which are typically restricted to root/administrative users.

Without root privilege, the program will fail right at the start, when calling the function sniff() - when it tries to access the network interface.

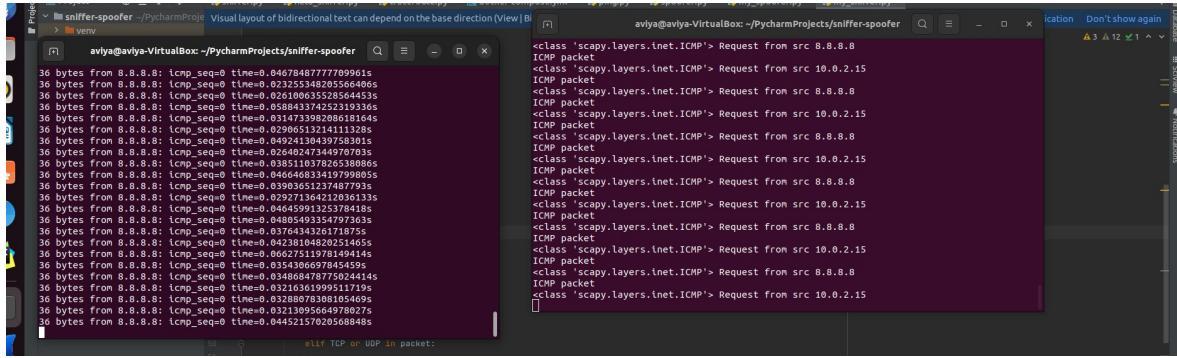
in order to change the kind of packets that the sniffer sniff, you should change it in the code, in the filter field when calling the sniff() function at the end of the code:

```
75 # Start the packet sniffer  
76 sniff(filter="icmp", prn=process_packet) # 2 עניינן
```

In order to check our sniffer we run it while running the last exercise with ping - to see if it catches the ICMP messages, and run it with Ex2 to show the TCP packets sniffing.

sniffing pings:

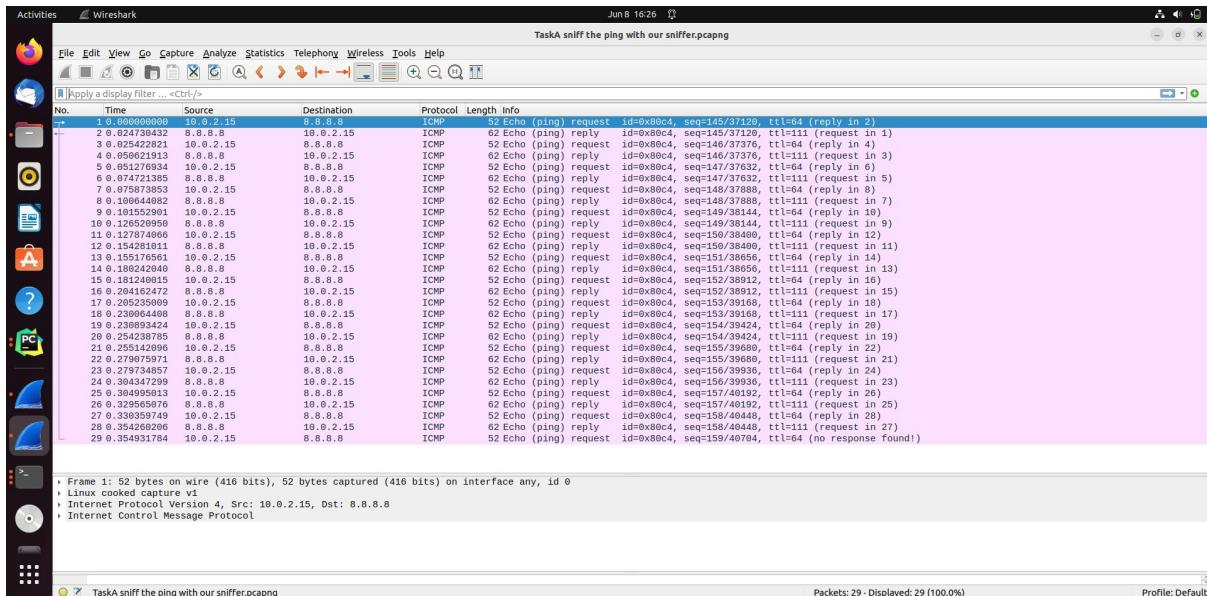
This is what the terminals look like:



The screenshot shows two terminal windows side-by-side. Both windows are titled 'aviva@aviva-VirtualBox: ~/PycharmProjects/sniffer-spoof'. The left window is labeled 'venv' and displays a continuous stream of ICMP packets from various sources (e.g., 8.8.8.8, 10.0.2.15) with sequence numbers ranging from 0 to over 1000. The right window also shows a similar stream of ICMP packets, mostly from 8.8.8.8, with sequence numbers between 0 and 1000. The terminal interface includes standard Linux command-line tools like 'ls', 'cd', and 'grep'.

```
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0467848777779961s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.023255348205566406s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.02610063552856453s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.05884337425231933s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0280651321411129s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0280651321411129s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0492413043975801s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.02640247344970703s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.038511037826538086s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.03056408334919705s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0398564877792179s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.029771364212936133s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.044659911325378418s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.04805493354777303s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0417646474641549s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0417646474641549s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.06627511978149414s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0354306697845459s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.03486847877502414s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0316038105469s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.03087388105469s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0321909564978027s
36 bytes from 8.8.8.8: icmp_seq=0 time=0.0445215702056884s
```

The appropriate wireshark:

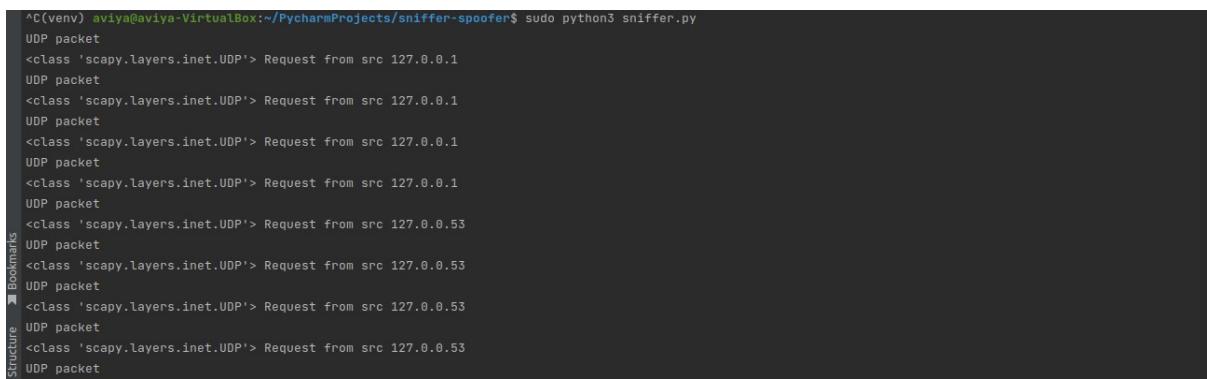


sniffing whatsapp , UDP/DNS packets:

הטרמינל של ה

sniffer

:



The screenshot shows a terminal window with the command 'sudo python3 sniffer.py' running. The output shows several UDP packets being captured. The first few are UDP requests from 127.0.0.1 to 127.0.0.1, likely DNS queries. Subsequent lines show various UDP traffic, including what appears to be WhatsApp protocol exchange between 127.0.0.1 and 127.0.0.53.

```
C(venv) aviva@aviva-VirtualBox:~/PycharmProjects/sniffer-spoof$ sudo python3 sniffer.py
UDP packet
<class 'scapy.layers.inet.UDP'> Request from src 127.0.0.1
UDP packet
<class 'scapy.layers.inet.UDP'> Request from src 127.0.0.53
```

הקלותה wireshark המתאימה:



sniffing Ex2:

ראשית הרכינו את הסניפר שיתחיל להסניף פקודות TCP:

```
Terminal: Local x Local (2) x Local (3) x + 
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofers$ sudo python3 sniffer.py
```

עכשו על מנת לבדוק אותו השתמשנו בקבצי מטלה 2 מהמודל, כנדרש.

הרכינו את proxy והserver:

```
Terminal: Local x Local (2) x + 
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofers$ cd EX2
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofers/EX2$ sudo python3 server.py
[sudo] password for aviya:
Listening on 127.0.0.1:9999

Terminal: Local x Local (2) x + 
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofers$ cd EX2
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofers/EX2$ sudo python3 proxy.py
[sudo] password for aviya:
Listening on 127.0.0.1:9998
```

ניתן לראות שנייהם השיבו שהם מאזינים בקוו שלטן - 127.0.0.1. השרת בפורט 9999 והפרקטי בפורט 9998.

ואז הרכינו את הקליינט, שמיד שולח בקשה:

```
Terminal: Local x Local (2) x Local (3) x Local (4) x + 
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofers$ cd EX2
EX2: command not found
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofers$ cd EX2
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofers/EX2$ sudo python3 client.py
[sudo] password for aviya:
```

ברטראמיאל של הקליננט התקבלה התשובה לבקשתה שלו:

```
Terminal: Local x Local (2) x Local (3) x Local (4) x + 
{127.0.0.1:9999} Connection established
{127.0.0.1:9999} Sending request of length 687 bytes
{127.0.0.1:9999} Got response of length 564 bytes
Result: -0.38748277824137206
Steps:
(sin(max(2, (3 * 4), 5, (6 * ((7 * 8) / 9)), (10 / 11))) / 12) * 13 = (sin(max(2, 12, 5, (6 * ((7 * 8) / 9)), (10 / 11))) / 12) * 13
= (sin(max(2, 12, 5, (6 * (56 / 9)), (10 / 11))) / 12) * 13
= (sin(max(2, 12, 5, (6 * 0.2222222222222222), (10 / 11))) / 12) * 13
= (sin(max(2, 12, 5, 37.33333333333336, (10 / 11))) / 12) * 13
= (sin(max(2, 12, 5, 37.33333333333336, 0.9090909090909091)) / 12) * 13
= (sin(37.33333333333336) / 12) * 13
= (-0.35767641068434347 / 12) * 13
= -0.02980653795702862 * 13
= -0.38748277824137206

{127.0.0.1:9999} Connection closed
[venv] aviyviya@aviyviya-VirtualBox:~/PycharmProjects/sniffer-spoofor/EX2$
```

ובטרמינל של הסניפר:

בסוף הסניפר מכנים את כל מה שהוא יכולת לתוך הקוביツ טיקסט ששמו תעודות הזהות שלנו והנה צילום מהתוך הקוביツ:

כאן רואים את
הדאטה של
הפאקטה ש
התרגיל
למשל.

מסקנת המקרה:

שימוש בעדכן מוכיח את עצמו כיעיל ביותר כלי לגישה להאזנה לרשות והעובדת אותו נוחה. בעזרת מסננים פשוטים של פרוטוקולים שונים אנחנו מודינים לרשות ומסניפים חבילות.CRZNG. נציין שמכיוון שרוב הרשות היום משתמשת בפרוטוקולים מאובטחים אז הדעתה בחבילות מוצפן. אך עדין המונע מידע נחשף בפנינו, כמו למשל לאיזה שירותים מחשב מסוים פונה ובאיזה תדרות. חשוב לציין שבשלב ההרצתה חייב לכתוב בהתחלה sudo על מנת לקבל הרשות אדמינ' כדי לאפשר כניסה למרחב הרשות.

קובצי הגשה:

sniffer.py

taskA sniff with sniffer.pcap

TaskA sniff the ping with our sniffer.pcap

taskA-sniffing task2.pcap

322273301_325195774.txt

*בקובץ הטקסט יש רק את ההרצתה עם מטלה 2 ולא את ההרצות האחרות.

taskB

OUR SPOOFER:

- aviyaoren@aviyaoren:~/Communication networks/EX5\$ sudo python3 spoofer.py
please choose and write one protocol: TCP / UDP / ICMP: TCP
spoof tcp

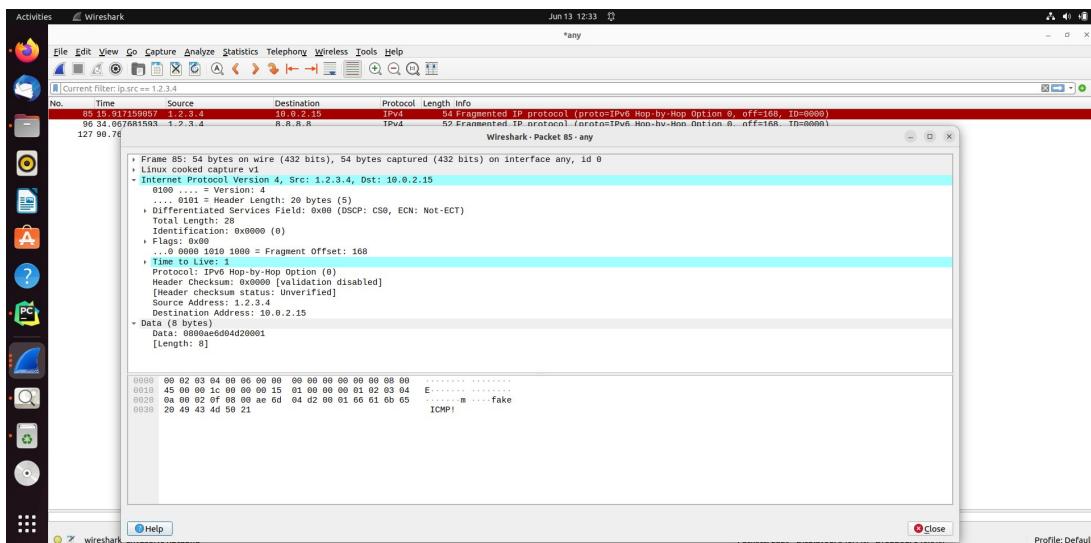
דרך הפעלה: המשתמש מתבקש לבחור פרוטוקול ויזיף פאקטה מתאימה ושלח אותה עם כתובת מזויפת.

לפנינו יש להגדיר בתוך הקוד את הכתובת אליה אנחנו מעוניינים לשלוח את הפאקטה (השתמשנו בlocalhost כברירת מחדל וגם ניסינו לשלוח ל 8.8.8.8) ואת הפורט שלה וכן את הכתובת המזויפת (השתמשנו ב 1.2.3.4) ואת הפורט המזויף שלה.
ואז spoofer שולח את הפאקטה המזויפת המתאימה. הטרמינל:

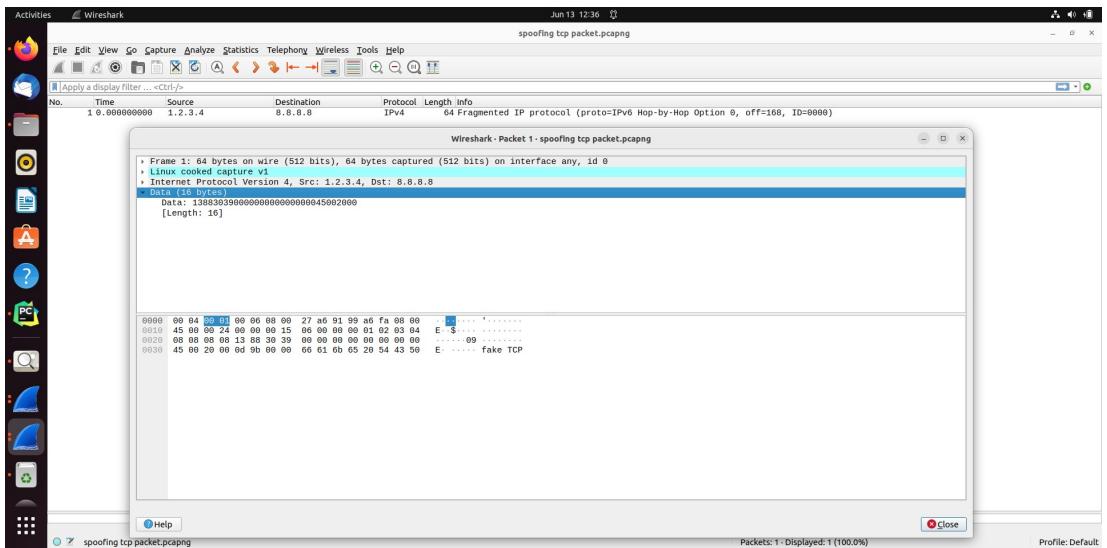
```
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofe$ sudo python3 spoofer.py  
please choose and write one protocol: TCP / UDP / ICMP: TCP  
spoof tcp  
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofe$ sudo python3 spoofer.py  
please choose and write one protocol: TCP / UDP / ICMP: UDP  
spoof udp  
(venv) aviya@aviya-VirtualBox:~/PycharmProjects/sniffer-spoofe$ sudo python3 spoofer.py  
please choose and write one protocol: TCP / UDP / ICMP: ICMP  
spoof icmp
```

(הפעלת spoofer נדרש לספק לו איזו פאקטה ברצונך לזייף (TCP / UDP / ICMP)

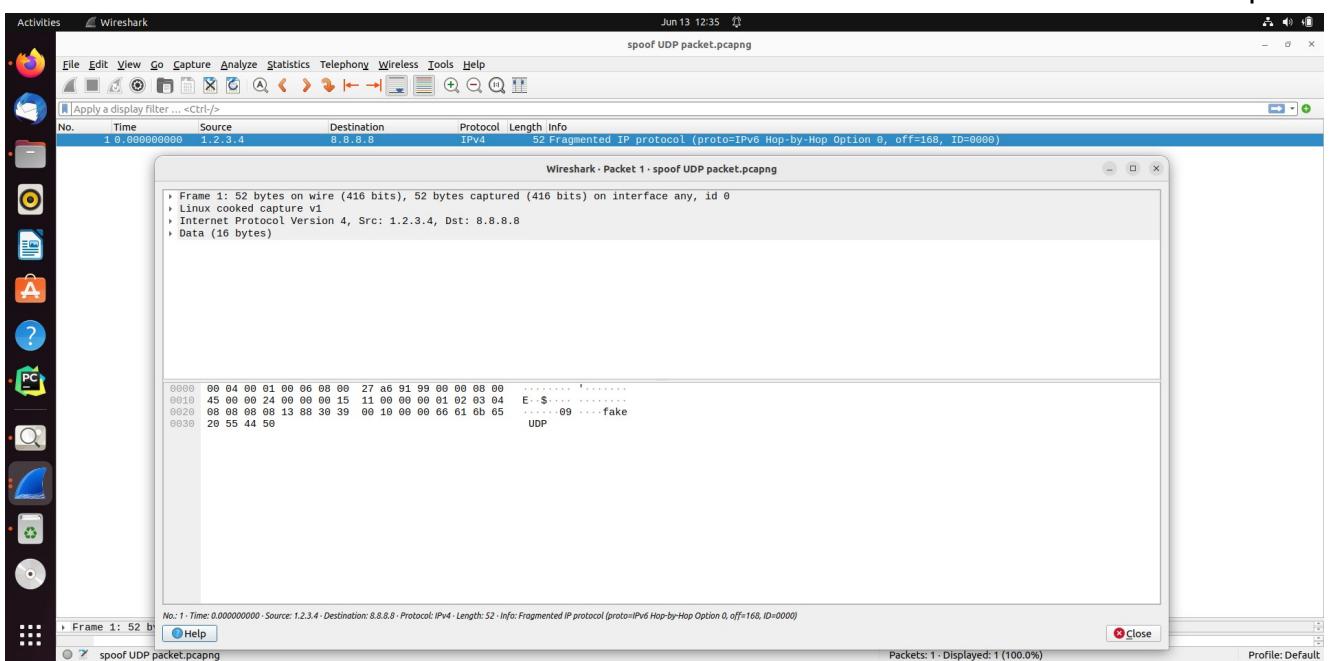
ניתן לראות בהקלות הווירשארק הבאות את שליחת הפאקטות הללו:
1. פאקטה ICMP מזויפת:



2. פאקטת TCP מזויפת



3. פאקטת UDP מזויפת



תשובות לשאלות שנשאלו:

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

כן, ניתן להגיד את אורך כוורתת ה-IP לערכים שרירותיים. אורך ה-IP משתנה לגודלו המקורי, לא משנה מה הגודל שנקבע על ידי המתכנת.

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

לא, זהו יתרון של raw socket , הירקן או מערכת הפעלה הבסיסית בונים את החבילה עם סכום הבדיקה באופן אוטומטי.

מסקנת המחקר:

שלחנו פאקטות ICMP עם כתובות מקור מזויפת. ולא קיבלנו תגובה אליהן כאשר שלחנו לשרת של גугл (8.8.8.8) ולא צփנו לשרת 10.9.0.15

- לאחר מחקר בגוגל, גילינו שיש כמה סיבות לכך שלא ענה אם שלחנו פאקטה מIP מזויף:
1. ניתוב - הנטבים באינטרנט עוקבים אחר טבלאות ניתוב כדי להעביר מנות. אם הנטבים מזהים שכותבת ה-IP המקורי של החבילה מזויפת או שאינה שייכת לטווח חוקי, הם עלולים להפסיק את הבדיקה או לא להעביר אותה הלאה.
בנוסף ניתן שם תישלח תגובה, היא תישלח לכותבת ה-IP המזויפת, שאינה קיימת או שאינה ניתנת להשגה.
2. סינון- מנגנים שchosmis תגובה כאשר הם מזהים כותבת IP חשודה או לא לגיטימית. סביר להניח שגוגל משתמש בהם גם המחשב שלנו.

קבצי הגשה:

spoof.py
spoof UDP packet.pcap
spoofing tcp packet.pcap
spoofing icmp packet.pcap

taskC traceroute tool

בחלק זה אנו משתמשים ב-scapy כדי להעריך את ה"מרחק" בנתבים, בין המחשב שלנו ליעד. הכל שעשוה את זה, וכך כתבנו traceroute משלנו.

איך עובד הקוד שלנו?

המשתמש מכניס את כתובת היעד IP הרצוי.

הפונקציה שלנו מגדרה תחילת ttl (time to live) ל 1.

ומגדירה מקסימום סיבובים (כדי להגביל את הריצה ולענות על טעות.)
מגדירה בנוסף את משתנה היעד בעזרת `socket.gethostbyname()`

כעת, אנו נכנסים לולאת `while`. בתוכה נוצרת פאקטה מסוג ICMP עם ttl עולה באחד עם כל סיבוב של הלולאה.

החbillה נשלחת ונו בוחנים את התגובה שמתתקבלת -

בגל שהttl קטן לא נקלט תגובה (הגענו לנット הראשון בדרך)

ונגדיל את ttl, ונשלח את החbillה שוב, ושוב נקלט timeout כשנגייע רק לנット הבא אחריו.
כך נמשכים הסיבובים עד שנגייע את היעד- את הנット הנוכחי.

בכל סיבוב אנו מדפיסות את המספר של ttl(שהוא גם מספר הסובבים שיידרשו לנו עד שנגייע ליעד)
וכן את כתובת הנット שאליו הגיעו.

*dagshim על הקוד אם יש, יתרונות/חסרונות:

מספר הסיבובים המקיים הוגדר ל 30 , אם יש יותר מ 30 נתבים בדרך הקוד שלנו לא יגיע אליו , אנחנו יוצאים מנקודות הנחה שדבר זהה הוא נדיר. בכל מקרה המספר הזה בר שינוי.

הבאנו כאן דוגמת הריצה עם הכתובת של גוגל כיעד כך נראה הקוד והפעלתו וכן הטרמינל לאחר הפעלה:

ניתן לראות שלאחר 19 נתבים בדרך, החbillה שלנו הגיעו אל היעד

The screenshot shows the PyCharm IDE interface. On the left, the code editor displays the `traceroute.py` script. The script uses the scapy library to perform a traceroute. It defines a `traceroute` function that takes a destination as an argument. Inside the function, it initializes `ttl` to 1, sets `max_hops` to 30, and gets the destination's IP via `socket.gethostbyname`. A `while` loop runs until `ttl` reaches `max_hops`. In each iteration, it creates an ICMP packet with increasing TTL, sends it to the destination, and receives a response. If there is no response, it prints the TTL. If the response type is 11 (ICMP Time Exceeded), it prints the TTL and the source IP. If the response type is 0 (ICMP Echo Reply), it prints the TTL and the source IP, indicating the destination was reached. Finally, it increments `ttl` by 1. An example usage at the bottom shows calling `traceroute(destination)` with "www.google.com".

On the right, the terminal window shows the command `sudo python3 traceroute.py` being run, followed by the password prompt [sudo] password for aviya. The terminal then lists the 19 intermediate hosts visited during the traceroute, ending with "19. 142.251.142.196 (Destination Reached)".

```
Jun 8 17:30:00 Jun 8 17:30:00
Run Tools VCS Window Help
traceroute.py  spoofer.py  sniffer.py
from scapy.all import *
|
1 usage
def traceroute(dest):
    ttl = 1
    max_hops = 30
    dst_ip = socket.gethostbyname(destination)

    while ttl <= max_hops:
        # Create an ICMP packet with increasing TTL
        packet = IP(dst=dst_ip, ttl=ttl) / ICMP()

        # Send the packet and receive the response
        reply = sr1(packet, verbose=False, timeout=2)

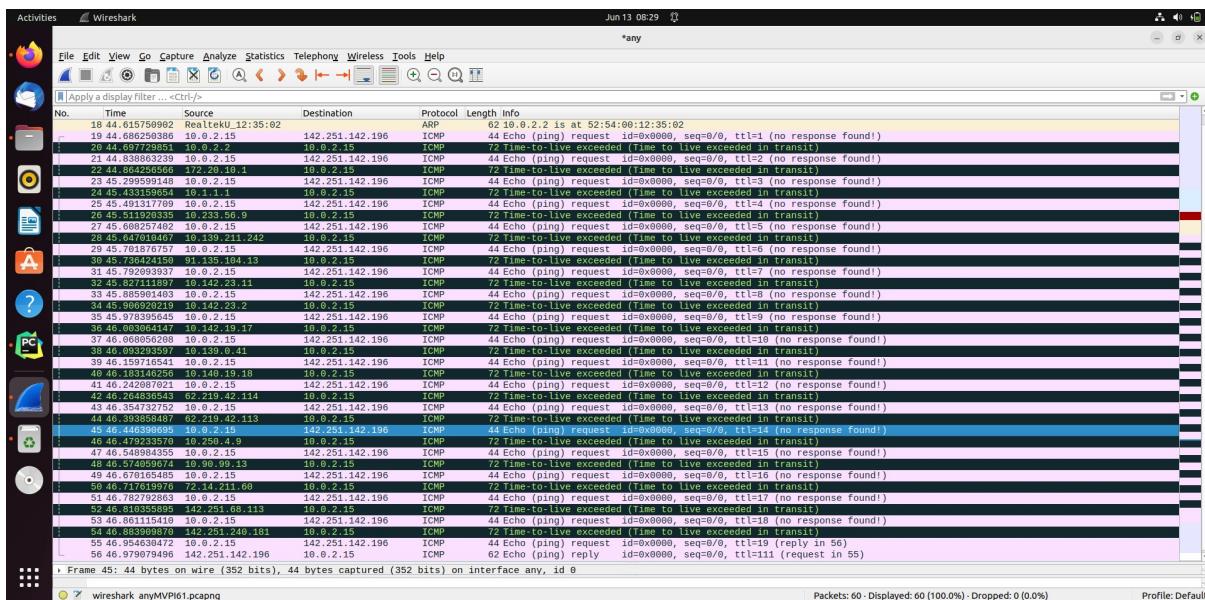
        if reply is None:
            # No response received within the timeout
            print(f"{ttl}. + + +")
        elif reply.type == 11:
            # ICMP Time Exceeded message received
            print(f"{ttl}. {reply.src}")
        elif reply.type == 0:
            # ICMP Echo Reply received (destination reached)
            print(f"{ttl}. {reply.src} (Destination Reached)")
            break

        ttl += 1

# Example usage
destination = "www.google.com" # Replace with the desired destination
traceroute(destination)
```

```
aviya@aviya-VirtualBox: ~/PycharmProjects/sniffer-spoofing $ sudo python3 traceroute.py
[sudo] password for aviya:
1. 10.0.2.2
2. 172.20.10.1
3. 10.1.1.1
4. 10.233.56.9
5. 10.139.211.242
6. 91.135.104.13
7. 91.135.104.2
8. 10.142.23.2
9. 10.142.19.17
10. 10.139.0.41
11. 10.140.19.18
12. 62.219.42.114
13. 62.219.42.113
14. 10.250.4.9
15. 10.90.99.13
16. 72.14.211.60
17. 142.251.231.41
18. 142.251.240.183
19. 142.251.142.196 (Destination Reached)
```

הנה צילום מסך של הקלטת wireshark המתאימה:



בקלטה ניתן לראות את שיחת הפאקטות (הping) בשורות הוורדות ולאחריה בכל פעם שלא הייתה תגובה מהיעד- שורה שחורה וכיთוב "no response found" בסופו של דבר עם הגעת החבילה אל היעד ישנו רצף של ping ו גם replay שלו בשתי שורות וירודות רצופות (המשךות בצלום).
הקלטת wireshark המתאימה מצורפת תחת השם ".traceroute with google".

מסקנת המחקר:

החבריות שנשלחות באינטרנט עוברות בין הרבה נתבים בדרך. כל נתב יודע לאן הוא מעביר (באופן קבוע) והחבורה עוברת כך נתב עד שהיא מגיעה אל היעד. כאשר מගבלים את זמן החיים של החבילה היא מספיקה כל פעם לעבור מספר מסוים של נתבים בדרך. וכך אנחנו יכולים להתקזות אחר כל הדרך שהיא עשתה עד הגעתה אל היעד.

קבצי הגשה:

traceroute.py

traceroute with google.pcap

taskD

- במשימה זו משלבים את הסניף והספופר שבנו כדי לישם תוכנית צוף. השתמשנו בדוקרים על מנת שייהו לנו 2 מכונות באוטו LAN.
- ממחשב A, אנחנו שולחים פינג לכתובת כלשהי. ועל המחשב השני שהוא המחשב התוקף מרים סניף וספופר כך שברגע שמסניפים בקשת ICMP מיד מוחזרים לשולח הבקשה תגובה מזויפת. וכך בעצם אנחנו מונטירים את המחשב מכל תקשורת.
- נזהה לראות בהקלחת הוירשארק 2 תשובות לכל פטוקן - אחת אמיתית ואחת מזויפת

*נשתמש בפומ' default בשבייל שליחת הפינה.

1. ליקחת המידע הדרוש (המקור והיעד) מהפקטה.
 2. זיוף פאקטת תגובה
 3. שליחת הפאקטה המזויפת לכטובות המקור.
 4. הדפסה של מקור הפאקטה המזויפת ויעדה

א. הפעלה ראשונה - שליחת פינג מ hostA-10.9.0.5 ל hostB-10.9.0.6

שלהנו פינג ל-10.9.0.6:

```
root@# 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=15.9 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.092 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.082 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.105 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.073 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.068 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.069 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.079 ms
```

והפעלו ברקע על הטרמינל של seed attacker את הseed-sniffer-spoofer שלהם.

```
^Croot@aviya-VirtualBox:/volumes# python3 sniffer_spoof.py
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 10.9.0.6 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 10.9.0.6 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
```

ניתן לראות שהוא מחזיר תగובות לפאקטות שהוא טופס

וניה הקלטת הווירשארק המתאימה:
ניתן לראות שיש 3 תשובות על כל request בגל שהסניף מסניף כל פאקטה שנשלחת הוא גם
מסניף את התגובה לפינג ומחייב גם עלייה תשובה...
אבל ל 10.9.0.5 ישן רק 2 תשובות שמוחזרות אחת מכתובת מזויפת- 1234 ואחת מהאמיתית.

ב. הפעלה שנייה - שליחת פינג מ A ל-IP WAN (למשל, -
(8.8.8.8)

נשלח את הפינג ל 8.8.8.8:

```
35 packets transmitted, 35 received, 0% packet loss,约耗时 0.0212ms  
rtt min/avg/max/mdev = 0.061/0.153/0.941/0.140 ms  
root@e3b724e84a5a:/# ping 8.8.8.8
```

צילום של ה-*sniffer-spoofers*

```
ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
^Croot@aviya-VirtualBox:/volumes# python3 sniffer_spoof.py

.
Sent 1 packets.

ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
.

Sent 1 packets.

ICMP echo reply from 1.2.3.4 to 8.8.8.8 was sent
.

Sent 1 packets.

ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
.

Sent 1 packets.

ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
.

Sent 1 packets.

ICMP echo reply from 1.2.3.4 to 8.8.8.8 was sent
.

Sent 1 packets.

ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
.

Sent 1 packets.

ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
.
```

צילום של הטרמינל:

```
aviya@aviya-VirtualBox: ~/PycharmProjects/sniffer-spoofing
```

```
64 bytes from 8.8.8.8: icmp_seq=45 ttl=110 time=44.2 ms
64 bytes from 8.8.8.8: icmp_seq=46 ttl=110 time=50.7 ms
64 bytes from 8.8.8.8: icmp_seq=47 ttl=110 time=45.8 ms
64 bytes from 8.8.8.8: icmp_seq=48 ttl=110 time=27.2 ms
64 bytes from 8.8.8.8: icmp_seq=49 ttl=110 time=44.2 ms
64 bytes from 8.8.8.8: icmp_seq=50 ttl=110 time=27.9 ms
64 bytes from 8.8.8.8: icmp_seq=51 ttl=110 time=28.3 ms
64 bytes from 8.8.8.8: icmp_seq=52 ttl=110 time=28.9 ms
64 bytes from 8.8.8.8: icmp_seq=53 ttl=110 time=27.7 ms
64 bytes from 8.8.8.8: icmp_seq=54 ttl=110 time=23.6 ms
64 bytes from 8.8.8.8: icmp_seq=55 ttl=110 time=25.4 ms
64 bytes from 8.8.8.8: icmp_seq=56 ttl=110 time=50.6 ms
64 bytes from 8.8.8.8: icmp_seq=57 ttl=110 time=28.9 ms
64 bytes from 8.8.8.8: icmp_seq=58 ttl=110 time=29.0 ms
64 bytes from 8.8.8.8: icmp_seq=59 ttl=110 time=40.6 ms
64 bytes from 8.8.8.8: icmp_seq=60 ttl=110 time=60.3 ms
64 bytes from 8.8.8.8: icmp_seq=61 ttl=110 time=69.2 ms
64 bytes from 8.8.8.8: icmp_seq=62 ttl=110 time=45.2 ms
64 bytes from 8.8.8.8: icmp_seq=63 ttl=110 time=64.1 ms
^C
--- 8.8.8.8 ping statistics ---
63 packets transmitted, 63 received, 0% packet loss, time 62828ms
```

צילום חלקו של הקלטת Wireshark

21 5.050973897 02:42:28.7d:ff:8f 02:42:6a:09:00:05 ARP 42 10.9.0.1 is at 02:42:28.7d:ff:8f
22 5.224843197 10.9.0.5 8.8.8.8 ICMP 98 Echo (ping) request id=0x0010, seq=6/1536, ttl=64 (reply in 24)
23 5.252585306 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
24 5.289344510 8.8.8.8 10.9.0.5 ICMP 98 Echo (ping) reply id=0x0010, seq=6/1536, ttl=64 (request in 22)
25 6.472296782 10.9.0.5 8.8.8.8 ICMP 98 Echo (ping) request id=0x0010, seq=7/1792, ttl=64 (reply in 26)
26 6.535348799 8.8.8.8 10.9.0.5 ICMP 98 Echo (ping) reply id=0x0010, seq=7/1792, ttl=64 (request in 25)
27 6.627397922 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
28 7.619394416 10.9.0.5 8.8.8.8 ICMP 98 Echo (ping) request id=0x0010, seq=8/2048, ttl=64 (reply in 29)
29 7.707520983 8.8.8.8 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
30 7.866549182 1.2.3.4 10.9.0.5 ICMP 98 Echo (ping) request id=0x0010, seq=9/2304, ttl=64 (request in 28)
31 8.644068225 10.9.0.5 8.8.8.8 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
32 8.665952964 1.2.3.4 10.9.0.5 ICMP 98 Echo (ping) reply id=0x0010, seq=9/2304, ttl=64 (reply in 33)
33 8.721281551 8.8.8.8 10.9.0.5 ICMP 98 Echo (ping) reply id=0x0010, seq=9/2304, ttl=64 (request in 31)
34 9.646076209 10.9.0.5 8.8.8.8 ICMP 98 Echo (ping) request id=0x0010, seq=10/2560, ttl=64 (reply in 35)
35 9.696912771 8.8.8.8 10.9.0.5 ICMP 98 Echo (ping) reply id=0x0010, seq=10/2560, ttl=64 (request in 34)
36 9.798062687 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
37 10.653401675 10.9.0.5 8.8.8.8 ICMP 98 Echo (ping) request id=0x0010, seq=11/2816, ttl=64 (reply in 38)
38 10.767555714 8.8.8.8 10.9.0.5 ICMP 98 Echo (ping) reply id=0x0010, seq=11/2816, ttl=64 (request in 37)
39 10.769337089 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64

ניתן לראות 2 תגובהות לכל שילחה מ-10.9.0.5 ל-8.8.8.8

תגובה ראשונה מזוייפת שלנו ותגובה שנייה מהשרת של 8.8.8.8

ג. הפעלה שלישית - שילוח פינג מ-A ל-B מזוייף.

שלחנו ping לכתובת 1.2.3.4 שלא קיימת.

```
root@42eea3b98b09:~# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
69 packets transmitted, 0 received, 100% packet loss, time 72665ms

root@42eea3b98b09:~#
```

ניתן לראות את הקלטת Wireshark המתאימה וכן עם יעד לא קיים כמובן שאין תגובה ל-Ping.
אבל ברגע למה שנראה בהמשר - באופן מעניין כאן למרות שהכתובות לא קיימת עדין נשלחת הودעה, השערה שלנו זהה קורה כאשר הכתובת **חיצונית** ואילו כאשר ננסה לשלוח לכתובת **פנימית** (=בתוך LAN שלנו) שלא קיימת אפילו לא יוצר ניסיון לשלוח הודעה כי המחשב יודע שהכתובות לא קיימת.

15 5.1699773128 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=6/1536, ttl=64 (no response found!)
16 5.1280800338 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
17 6.217486956 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=7/1792, ttl=64 (no response found!)
18 6.248101732 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
19 7.221846607 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=8/2048, ttl=64 (no response found!)
20 7.263761321 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
21 8.246030652 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=9/2304, ttl=64 (no response found!)
22 8.262844969 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
23 9.269978884 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=10/2560, ttl=64 (no response found!)
24 9.299635715 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
25 10.307121118 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=11/2816, ttl=64 (no response found!)
26 10.326578471 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
27 11.331881613 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=12/3072, ttl=64 (no response found!)
28 11.354762394 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
29 12.341822997 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=13/3328, ttl=64 (no response found!)
30 12.366617269 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
31 13.365428829 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=14/3584, ttl=64 (no response found!)
32 13.382551476 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
33 14.389481461 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=15/3840, ttl=64 (no response found!)
34 14.418728851 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
35 15.413264636 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=16/4096, ttl=64 (no response found!)
36 15.434449962 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
37 16.437684185 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=17/4352, ttl=64 (no response found!)
38 16.466639632 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
39 17.477779570 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=18/4608, ttl=64 (no response found!)
40 18.312710110 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
41 18.493447184 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=19/4864, ttl=64 (no response found!)
42 18.5568891357 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
43 19.509303083 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=20/5120, ttl=64 (no response found!)
44 19.526971013 1.2.3.4 10.9.0.5 ICMP 42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64
45 20.534792010 10.9.0.5 1.2.3.4 ICMP 98 Echo (ping) request id=0x0011, seq=21/5376, ttl=64 (no response found!)

Frame 14: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-a14b571ac5c4, id 0

בזמן שליחת הפינגים הרצינו את snifferspoof.py בפרק ע:

```
^[[A^[[A^Croot@aviya-VirtualBox:/volumes# python3 sniffer_spoof.py
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 1.2.3.4 was sent
.
Sent 1 packets.
ICMP echo reply from 1.2.3.4 to 10.9.0.5 was sent
```

ד. הפעלה רביעית - שליחת פינג מ A ל-IP מזוייף בראשת ה-LAN שלו

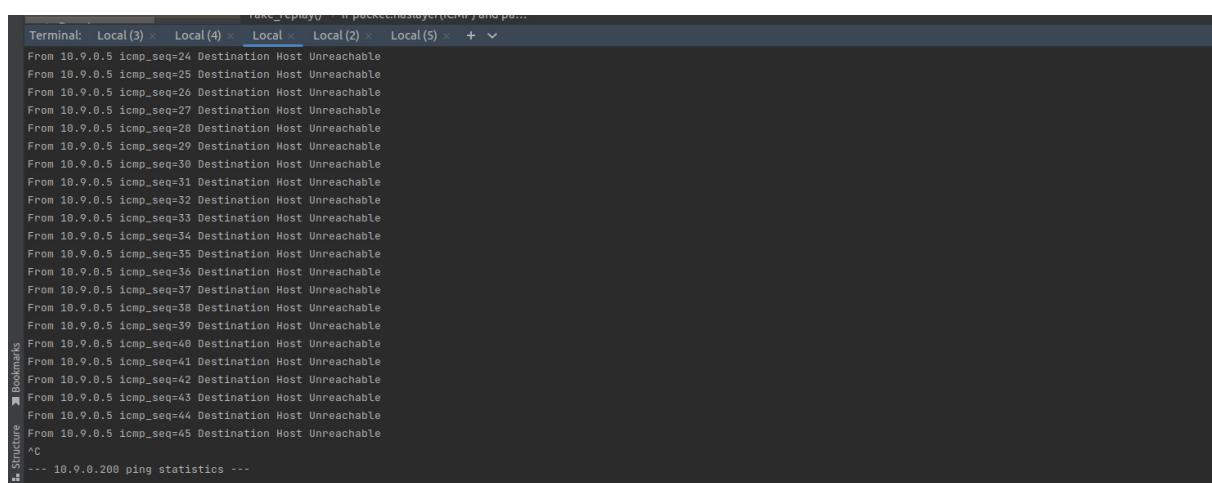
שלחנו את הפינג לIP מזויף בתוך LAN שלנו: "ping 10.9.0.200" = הרשת הפנימית של המחשב שלנו

הראנו את הפטוכניות:

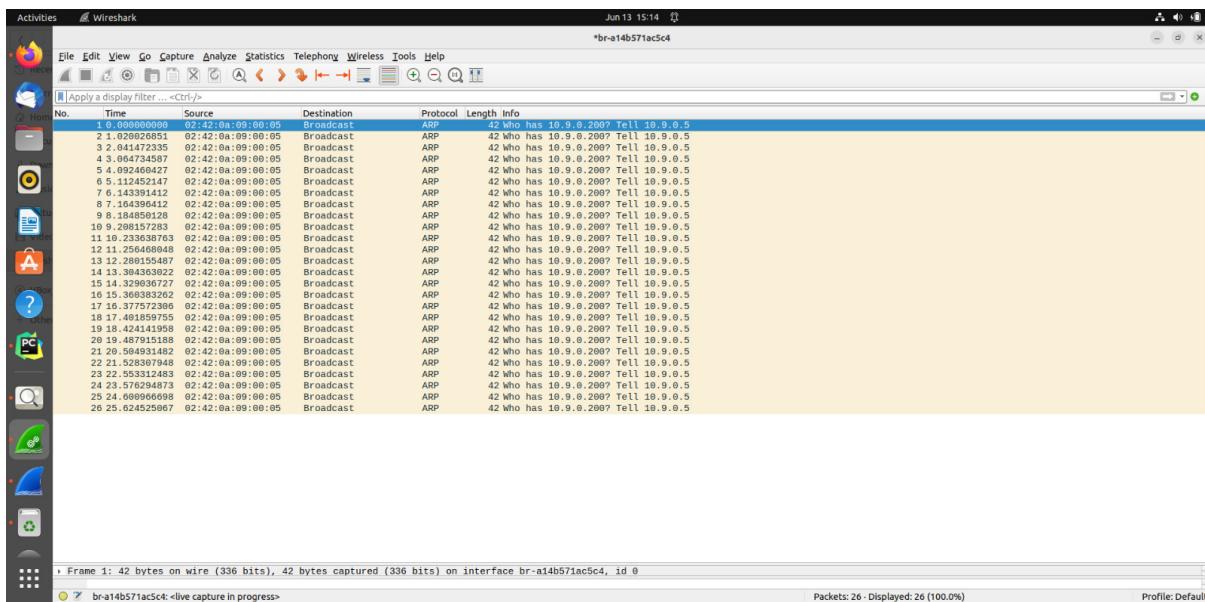
```
root@aviya-VirtualBox:/volumes# python3 sniffer_spoofer.py
```

בהקלטת wireshark המתואימה ניתן לראות שהכתובת הפנימית לא קיימת ולכן נראה את הכתובת Tell 10.9.0.6 "how has 10.9.0.20?" שבעצם אומר לנו שאין כתובת צו ולק התוכנית שלנו ישוישרת על 10.9.0.6 לא יכולה לשולח לczו כתובת.

ראיים עברו כל הפיניגים שההוויט unreachable



צילום של הוירשארק:



באינפו כתוב "who has 10.9.0.200? tell 10.9.0.5" כי משמעות הודעת הפינג שלנו היא חיפוש הכתובת (שלא קיימת).
הפינג נשלח בעזרת פרוטוקול ARP כי אנחנו מחפשים ברשת המקומית.

מסקנת המקרה:

ראינו

מהסעיפים המגוונים של מטלה זו למדנו שישתוּפ פעולה של סניפר וספופר יכול לשמש אותנו לדברים מעוניינים ואף לתקיפה מתוחכמת ברשת. שליחת חבילות מזויפות למחשב/שרת תוך מענה בלתי פועל לכל הودעה יוצאה יכול להעמיס עליו עד כדי קרישא.

קבצי הגשה:

ping from hostA to 8.8.8.8 with snif spoof.pcap
ping from hostA to hostB with snif spoof.pcap
ping to fake ip with snif spoof.pcap
ping to fake ip in our LAN with snif spoof.pcap
sniffer_spoof.py

taskE

בסוף זה נראה כיצד השתמש בסניף שבינו לccccccc סיסמות.
הסבר כללי:

הכוונה סיסמה בחיבור telnet בזמן שה嗅ינר מסניף פאקטות TCP (הרעיון הוא ש telnet הוא דוגמא לפורוטוקול לא מאובטח או מוצפן המבוסס טקסט ולא כמו פרוטוקול https כמו רוב האתרים בראשת בימינו, אפשר לתפוא שם בקלות את הטקסט שעבר - בין אם סיסמה או טקסט אחר).

יצרנו קובץ password sniffing שבו אנחנו מסניפות פאקטות telnet מפורוטוקול 23.
איך הקוד עובד?

מסניפים פאקטות tcp שעוברות בתוך הiface המתאים.

הפעולה שמתבצעת בעת כל הסנפת פאקטה היא:

בדיקה האם הפאקטה באמת לפורוטוקול 23 וגם מסוג TCP.

ואז חילוץ המידע שעבר בה באמצעות המתוודה: payload.

אנחנו מדפיסות את המידע לטרמינל ובנוסף כותבות אותו בקובץ הטקסט המצורף(telnet_packets.txt)

אוף הפעולה שלנו:

. מתוך טרמינל hostA נסיוון ברקע את הסנפת הסיסמה (sniffing_password).

ואז שלחנו נסיוון לחיבור hostB telnet לhostA.

```
root@aviya-VirtualBox:/volumes# python3 sniffing_password.py
```

טרמינל hostA

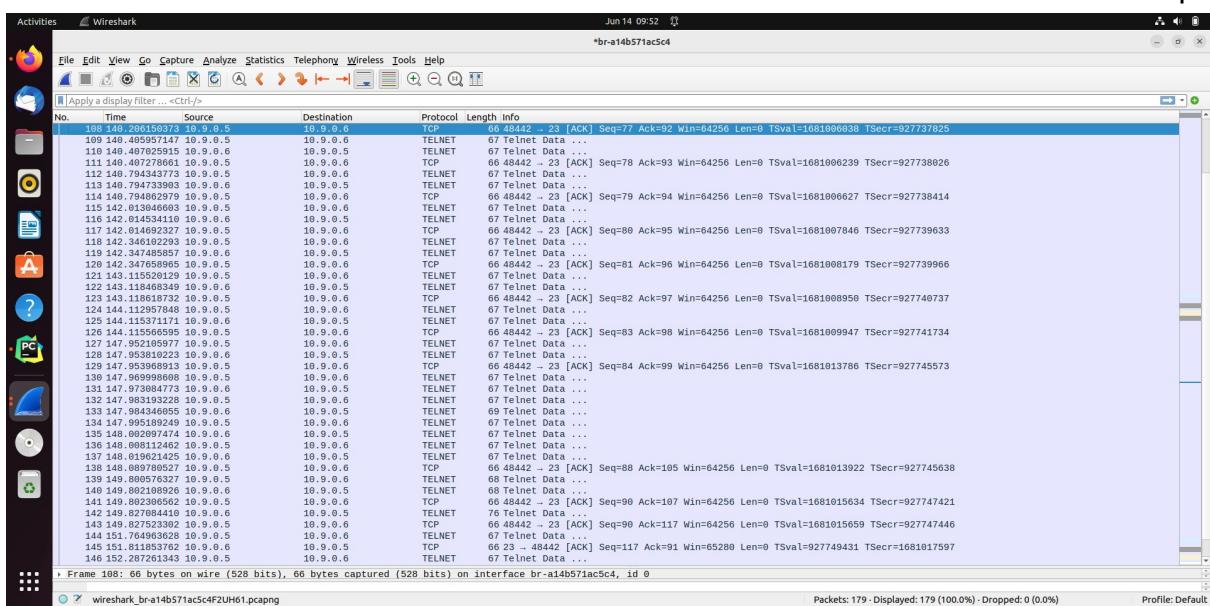
```
root@42eea3b98b09:/# telnet 10.9.0.6 23
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^['.
Ubuntu 20.04.1 LTS
e3b724e84a5a login: aviyaAndNeta
Password:

Login incorrect
```

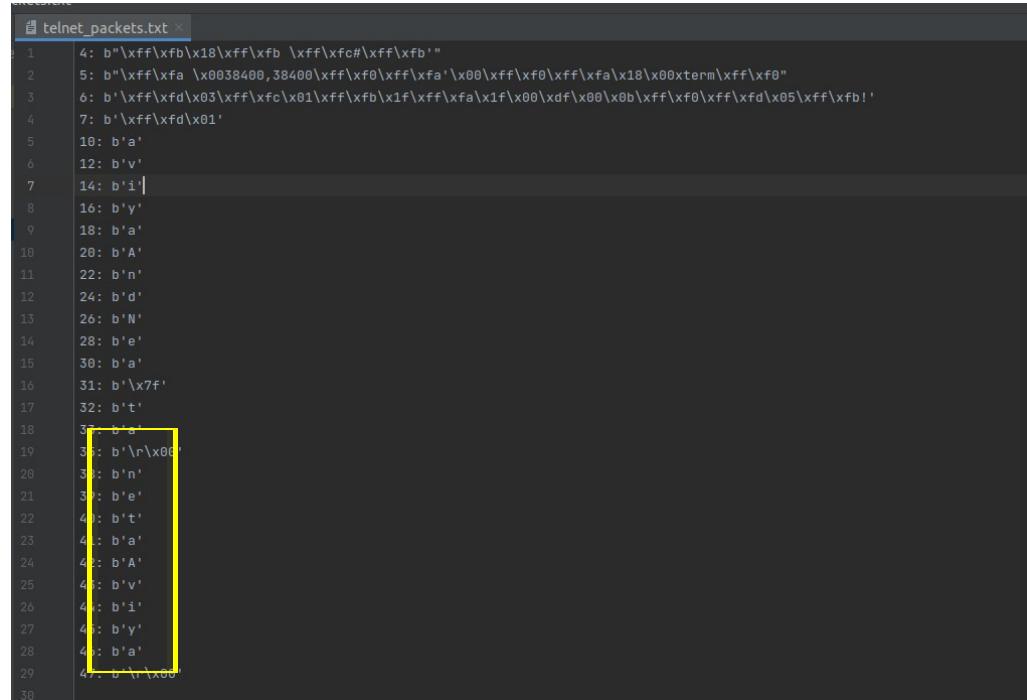
ואלו הינו התוצאות בseedattacker

```
[root@aviya-VirtualBox:/volumes# python3 sniffing_password.py
Telnet packet captured: b"\xff\xfb\x18\xff\xfb \xff\xfc#\xff\xfb"
Telnet packet captured: b"\xff\xfa \x0038400,38400\xff\xf0\xff\xfa'\x00\xff\xf0\xff\xfa\x18\x00xterm\xff\xf0"
Telnet packet captured: b'\xff\xfd\x03\xff\xfc\x01\xff\xfb\x1f\xff\xfa\x1f\x00\xdf\x00\x0b\xff\xf0\xff\xfd\x05\xff\xfb'
Telnet packet captured: b'\xff\xfd\x01'
Telnet packet captured: b'a'
Telnet packet captured: b'v'
Telnet packet captured: b'i'
Telnet packet captured: b'y'
Telnet packet captured: b'a'
Telnet packet captured: b'A'
Telnet packet captured: b'n'
Telnet packet captured: b'd'
Telnet packet captured: b'N'
Telnet packet captured: b'e'
Telnet packet captured: b'a'
Telnet packet captured: b'\x7f'
Telnet packet captured: b't'
Telnet packet captured: b'a'
Telnet packet captured: b'r\x00'
Telnet packet captured: b'n'
Telnet packet captured: b'e'
Telnet packet captured: b't'
Telnet packet captured: b'a'
Telnet packet captured: b'A'
```

הקלות wireshark המתאימה:



הסיסמה שלנו הייתה `aViyaAveta`
ואכן בקובץ הטקסט ניתן לראות בקלות את הסיסמה אותן אחר אותן:



```
telnet_packets.txt
1: b'\xff\xfb\x18\xff\xfb \xff\xfc#\xff\xfb'
2: b"\xff\xfa \x0038400,38400\xff\xf0\xff\xfa'\x00\xff\xf0\xff\xfa\x18\x00xterm\xff\xf0"
3: b'\xff\xfd\x03\xff\xfc\x01\xff\xfb\x1f\xff\xfa\x00\xdf\x00\x0b\xff\xf0\xfd\x05\xff\xfb!'
4: b'\xff\xfd\x01'
5: 10: b'a'
6: 12: b'v'
7: 14: b'i'|  
8: 16: b'y'
9: 18: b'a'
10: 20: b'A'
11: 22: b'n'
12: 24: b'd'
13: 26: b'N'
14: 28: b'e'
15: 30: b'a'
16: 31: b'\x7f'
17: 32: b't'
18: 37: b'a'  
37: b'\r\x00'  
38: b'n'  
39: b'e'  
40: b't'  
41: b'a'  
42: b'A'  
43: b'v'  
44: b'i'  
45: b'y'  
46: b'a'  
47: b'\r\x00'
```

מסקנת המחבר:

בcheinור telnet, מכיוון שהוא לא מאובטח, ניתן להסניף פקודות כך שהדעתה שלהם גלויה לנו, כולל סיסמות. וכן ניתן להסניף ולזיהות את הסיסמה בפקטה הרלוונטית, בעזרת הסופה מסוננת לפרטוקול TCP, חילוץ הדעתה מהפקות, ואיתור הפקטה של שליחת הסיסמה.

קובצי הגשה –
`,sniffing_passwords.py`
`,telnet_packets.txt`
וקובץ `sniffing telnet password.pcap`

bibliograph:

<https://www.geeksforgeeks.org/>

<https://rootinstall.com/>

Ex2, Ex4