

PACKETS SNIFFING & SPOOFING

ליאור וינמן יועד תמר
213081763 213451818

20 בינואר 2023

תוכן עניינים

3	1	פרק א' - תיאור המערכת
3	1.1	הרצת הפרויקט
3	1.2	קבצי עזר
4	1.3	הקובץ היוצר - <i>Makefile</i>
5	1.4	מימוש רחרחן חבילות - <i>Sniffer.c</i>
10	1.5	מימוש זייפן חבילות - <i>Spoof.c</i>
12	1.6	מימוש זייפן תגובות - <i>Sniff_and_spoof.c</i>
15	1.7	מימוש שער - <i>Gateway.c</i>
17	2	פרק ב' - תעבורה ופלט
17	2.1	רחרחן
17	2.1.1	פלט בטרמינל
19	2.1.2	תעבורת <i>Wireshark</i>
20	2.1.3	פלט כקובץ
21	2.2	זייפן חבילות
21	2.2.1	פלט בטרמינל
21	2.2.2	תעבורת <i>Wireshark</i>
21	2.3	זייפן תגובות
21	2.3.1	חלק א
22	2.3.2	חלק ב
23	2.3.3	חלק ג
24	2.4	שער
24	2.4.1	פלט
24	2.4.2	תעבורה
25	3	פרק ג' - מענה על השאלות
25	3.1	שאלה 1
25	3.2	שאלה 2
25	3.3	שאלה 3
26	4	פרק ד' - יכולות, הגבלות ומחקר נוסף
26	4.1	רחרחן
26	4.1.1	יכולות
26	4.1.2	מגבלות
26	4.2	זייפן חבילות
26	4.2.1	יכולות
26	4.2.2	מגבלות

26	שינויים קטנים נוספים	4.2.3	
27	תגובות	4.3	
27	הסבר מפורט	4.3.1	
27	אופן המימוש	4.3.2	
27	יכולות	4.3.3	
27	מגבלות	4.3.4	
28	ביבליוגרפיה	5	

1 פרק א' - תיאור המערכת

1.1 הרצת הפרויקט

בפרויקט זה ישנו מספר רב של קבצים. קבצי המטלה הניתנים להרצה הם - *Sniffer.c* (A), *Spoof.c* (B), *Sniff_and_spoof.c* (C), *Gateway.c* (D). שאר הקבצים הם קבצים פנימיים, קבצי עזר שגם ניתנים להרצה אך הם לא עיקר ודרישת המטלה. על מנת להריץ את הפרויקט, יש ראשית להוריד את כל הקבצים לתוך תיקיה משותפת במחשב, נציין שיש להשתמש במערכת ההפעלה *Linux Ubuntu LTS* בגרסה העדכנית ביותר 22.04.

לאחר הורדת כלל הקבצים, נפתח חלון טרמינל על התקיה שבה נמצאים הקבצים - נכתוב הפקודה "make all" - הדבר יגרום ליצירת קבצי ההרצה של התוכניות שכתבנו. (מומלץ לכתוב גם את הפקודה "clear" כדי לנקות את הפלט הסטנדרטי). כעת נסביר כיצד להריץ כל קובץ הנדרש במטלה: (כיוון שמטלות אלו כבר עברו - אנו מניחים כי הקורא יודע כיצד להריץ את קבצי מטלות 2 ו-4!)

אם ברצוננו להריץ את החרחון *Sniffer.c* - נרשום `sudo ./Sniffer`, ולאחר מכן נריץ את קבצי מטלה 2. (תזכורת: "python3 server.py" → "python3 proxy.py" → "python3 client.py")

אם ברצוננו להריץ את זייפן החבילות *Spoof.c* - נרשום `sudo ./Spoof`.

אם ברצוננו להריץ את זייפן התגובות *Sniff_and_Spoof.c* - עלינו להריץ את ה-docker על ידי:

1. `docker - compose build`

2. `docker ps - a` - כעת נפתח שלשה טרמינלים ובכל אחד מהם נריץ את אחת השורות הבאות:

3. `sudo docker exec - it < ATTACKER_ID > /bin/bash`

4. `sudo docker exec - it < hostA_ID > /bin/bash`

5. `sudo docker exec - it < hostB_ID > /bin/bash`

וכעת בכל מחשב אנו יכולים לבצע את הפעולות הנדרשות לנו.

אם ברצוננו להריץ את השער *Gateway.c* נרשום - `./Gateway < ip_address >`, ובטרמינל נפרד נרשום - `echo - n example | nc - 4u - w1 10.0.2.15 9090`.

1.2 קבצי עזר

לפרויקט זה מצורפים מספר קבצי עזר אשר מטרתם להשלים את קבצי המטלה שנדרשנו לכתוב.

1. קבצי מטלה 2 - *calculator.py*, *api.py*, *client.py*, *proxy.py*, *server.py* - שימושים הוא בסעיף הראשון שכן אנו מסניפים את החבילות הספציפיות האלו שנוצרות על ידי המטלה הזו.

2. קובץ ממטלה 4 - *ping.c* - שימושו הוא בסעיף השלישי ששם אנו מבצעים שליחות פינג לכתובת לא פעילה ומחזירים לה תגובות פונג כך שנראה שהכתובת פעילה. (נמצאים בתקיה בשם "ex2").

3. הקלטות וירשק - הקלטות של החבילות והתעבורה של כל אחד מהסעיפים (נמצאים בתיקיה "ex5_rec" ואז שם יש תתי תיקיות עם ההקלטות הרלוונטיות לכל סעיף).

1.3 הקובץ היוצר - Makefile

הקובץ הנ"ל הוא הקובץ שממיר את קבצי הקוד (הכתובים בשפת C) ומייצר מהם קבצי הרצה על מנת שנוכל להריץ בפועל את התוכניות שאנו כותבים. נעבור על הקוד:

```
CC = gcc # compiler
FLAGS = -Wall -g # compilation flags
TARGETS = Sniffer Spoofer Sniff_and_spoof Gateway Ping # exe targets

.PHONY: all clean

all: $(TARGETS)

Sniffer: Sniffer.o
    $(CC) $(FLAGS) -o Sniffer Sniffer.o -lpcap
Sniffer.o: Sniffer.c
    $(CC) $(FLAGS) -c Sniffer.c
Spoofer: Spoofer.o
    $(CC) $(FLAGS) -o Spoofer Spoofer.o
Spoofer.o: Spoofer.c
    $(CC) $(FLAGS) -c Spoofer.c
Sniff_and_spoof: sniff_and_spoof.o
    $(CC) $(FLAGS) -o Sniff_and_spoof sniff_and_spoof.o -lpcap
sniff_and_spoof.o: sniff_and_spoof.c
    $(CC) $(FLAGS) -c sniff_and_spoof.c
Ping: ping.o
    $(CC) $(FLAGS) -o Ping ping.o
ping.o: ping.c
    $(CC) $(FLAGS) -c ping.c
Gateway: gateway.o
    $(CC) $(FLAGS) -o Gateway gateway.o
gateway.o: gateway.c
    $(CC) $(FLAGS) -c gateway.c
clean:
    rm -f *.o *.h.gch $(TARGETS)
```

כאן אנו לכל קובץ, מייצרים מקבצי הקוד קבצים של אובייקטים בינאריים (קומפילציה) ולאחר מכן אנו מייצרים מהקבצים הבינאריים קבצי הרצה (לינקוג').

1.4 מימוש רחרחן חבילות - Sniffer.c

כאן נעבור על הקוד של רחרחן החבילות שכתבנו:

```
#include <stdio.h> // standard
#include <stdlib.h> // exit
#include <pcap.h> // packet capturing
#include <netinet/ip.h> // ip header
#include <netinet/tcp.h> // tcp header
#include <net/ethernet.h> // ethernet header
#include <errno.h> // last error number
#include <string.h> // memset

/* just for comfort */
#define PROMISCUE_MODE 1
#define NON_PROMISCUE_MODE 0
#define DONT_STOP_CAPTURE -1

/* how many packets has been received*/
long packet_num = 0;

/* application header */
struct apphdr
{
    uint32_t unixtime;
    uint16_t length;

    union
    {
        uint16_t flags;
        uint16_t _:3, cache_flag:1, steps_flag:1, type_flag:1, status_code:10;
    };

    uint16_t cache_control;
    uint16_t __;
};
```

אלו הם החלקים המעובדים מראש - ספריות שהשתמשנו בהן, קבועים שהגדרנו לצורך נוחות, משתנה גלובלי שסופר כמה חבילות הגיעו אלינו סה"כ, מבנה שהגדרנו (לפי התמונה של דגם החבילה במסלה 2) המתאר את שכבת האפליקציה של החבילות שאנו מסניפים.

```
/* printing each field in ethernet header */
void print_ethernet_header(const struct ether_header *eth)
{
    printf("----- ETHERNET II -----\n");
    printf("Source MAC address: %02x:%02x:%02x:%02x:%02x:%02x\n",
        eth->ether_shost[0], eth->ether_shost[1], eth->ether_shost[2], eth->ether_shost[3], eth->ether_shost[4], eth->ether_shost[5]);
    printf("Destination MAC address: %02x:%02x:%02x:%02x:%02x:%02x\n",
        eth->ether_dhost[0], eth->ether_dhost[1], eth->ether_dhost[2], eth->ether_dhost[3], eth->ether_dhost[4], eth->ether_dhost[5]);
    printf("Type: %s\n", ntohs(eth->ether_type) == 0x0800 ? "IPv4 (0x0800)" : "UNKNOWN");
}
```

כאן אנו הגדרנו פונקציה שתדפיס למשתמש את שכבת האתרנט של החבילה שהסנפנו.

```

/* printing each field in ip header */
void print_ip_header(const struct iphdr *ip, FILE *packets)
{
    printf("----- Internet Protocol ----- \n");
    printf("Version: %u\n", ip->version);
    printf("Header Length: %u bytes (%u)\n", 4 * ip->ihl, ip->ihl);
    printf("Total Length: %u\n", ntohs(ip->tot_len));
    printf("Identification: 0x%04x (%u)\n", ntohs(ip->id), ntohs(ip->id));
    printf("Fragment Offset: %u\n", ntohs(ip->frag_off) & 0x1fff);
    printf("Time to Live: %u\n", ip->ttl);
    printf("Protocol: %s (%u)\n", (ip->protocol == IPPROTO_TCP ? "TCP" : "UNKNOWN"), ip->protocol);
    printf("Header Checksum: 0x%04x\n", ntohs(ip->check));
    printf("Source Address: %s\n", inet_ntoa(*(struct in_addr*)&ip->saddr));
    printf("Destination Address: %s\n", inet_ntoa(*(struct in_addr*)&ip->daddr));

    fprintf(packets, "{ source_ip: %s, dest_ip: %s, ", inet_ntoa(*(struct in_addr*)&ip->saddr), inet_ntoa(*(struct in_addr*)&ip->daddr));
}

```

כאן אנו הגדרנו פונקציה שתדפיס למשתמש את שכבת האינטרנט-פרוטוקול של החבילה שהסנפנו וגם תייצא לקובץ את השדות הרלוונטיים.

```

/* printing each field in tcp header */
int print_tcp_header(const struct tcphdr *tcp, unsigned int ip_size, FILE *packets)
{
    printf("----- Transmission Control Protocol ----- \n");
    printf("Source Port: %u\n", ntohs(tcp->source));
    printf("Destination Port: %u\n", ntohs(tcp->dest));
    printf("TCP Segment Len: %u\n", htons(tcp->doff) * 4);
    printf("Sequence Number (raw): %u\n", ntohl(tcp->seq));
    printf("Acknowledgment Number: %u\n", ntohl(tcp->ack_seq));
    printf("Header Length: %u bytes\n", tcp->doff * 4);
    printf("Flags: FIN = %u, SYN = %u, RST = %u, PSH = %u, ACK = %u, URG = %u\n",
           tcp->fin, tcp->syn, tcp->rst, tcp->psh, tcp->ack, tcp->urg);
    printf("Window: %u\n", ntohs(tcp->window));
    printf("Checksum: 0x%04x\n", ntohs(tcp->check));
    printf("Urgent Pointer: %u\n", ntohs(tcp->urg_ptr));

    fprintf(packets, " source_port: %u, dest_port: %u, ", ntohs(tcp->source), ntohs(tcp->dest));

    if(tcp->psh)
    {
        return 1;
    }
    return 0;
}

```

כאן אנו הגדרנו פונקציה שתדפיס למשתמש את שכבת הטיסיפי של החבילה שהסנפנו וגם תייצא לקובץ את השדות הרלוונטיים.

```

/* printing each field in ethernet header */
void print_app_header(const struct pcap_pkthdr *header, const u_char *pkt_bytes, struct apphdr *app, FILE *packets)
{
    app->flags = ntohs(app->flags);
    uint16_t cache_flag = ((app->flags) >> 12) & 1;
    uint16_t steps_flag = ((app->flags) >> 11) & 1;
    uint16_t type_flag = ((app->flags) >> 10) & 1;
    uint16_t status_code = app->status_code;
    uint16_t cache_control = ntohs(app->cache_control);
    printf("----- Application Header -----\n");
    printf("timestamp: %ld (seconds)\n", (header->ts.tv_sec + (header->ts.tv_usec / 1000000)));
    printf("total length: %d\n", header->len);
    printf("cache flag: %hu\n", cache_flag);
    printf("steps flag: %hu\n", steps_flag);
    printf("type flag: %hu\n", type_flag);
    printf("status code: %hu\n", status_code);
    printf("cache control: %hu\n", cache_control);

    fprintf(packets, " timestamp: %ld, total length: %d, cache flag: %hu, steps flag: %hu, type flag: %hu, status code: %hu, cache control: %hu"
        , (header->ts.tv_sec + (header->ts.tv_usec / 1000000)), header->len, cache_flag, steps_flag, type_flag, status_code, cache_control);
}

```

כאן אנו הגדרנו פונקציה שתדפיס למשתמש את שכבת האפליקציה של החבילה שהסנפנו וגם תייצא לקובץ את השדות הרלוונטיים.

```

void print_payload(const struct pcap_pkthdr *header, const u_char *pkt_bytes, struct apphdr *app, struct iphdr *ip, struct tcphdr *tcp, FILE *packets)
{
    uint16_t app_length = ntohs(app->length);
    const u_char *payload = (pkt_bytes + sizeof(struct ether_header) + ip->ihl * 4 + tcp->doff * 4) + 12;
    printf("----- Payload ----- \n");
    fprintf(packets, " , \ndata:");
    for(int i = 0; i < app_length; ++i)
    {
        if(!(i & 15))
        {
            printf("\n%04X: ", i);
            fprintf(packets, "\n%04X: ", i);
        }
        printf("%02X ", ((unsigned char*)payload)[i]);
        fprintf(packets, "%02X ", ((unsigned char*)payload)[i]);
    }
    printf("\n");
    fprintf(packets, " }\n\n");
}

```

כאן אנו הגדרנו פונקציה שתדפיס למשתמש את המידע עצמו שמועבר בחבילה וגם תייצא לקובץ את המידע.

```

void packet_data(u_char *user, const struct pcap_pkthdr *header, const u_char *pkt_bytes)
{
    struct ether_header *eth;
    struct iphdr *ip;
    struct tcphdr *tcp;
    struct apphdr *app;

    eth = (struct ether_header*)pkt_bytes;
    ip = (struct iphdr*)(pkt_bytes + sizeof(struct ether_header));
    tcp = (struct tcphdr*)(pkt_bytes + sizeof(struct ether_header) + sizeof(struct iphdr));
    app = (struct apphdr*)(pkt_bytes + sizeof(struct ether_header) + ip->ihl * 4 + tcp->doff * 4);

    int x = 0;

    FILE *packets = fopen("213081763_213451818.txt", "a");

    //packets printing to user (console)
    printf("Packet Number - %ld\n", ++packet_num);
    print_ethernet_header(eth);
    print_ip_header(ip, packets);
    x = print_tcp_header(tcp, ip->tot_len, packets);
    print_app_header(header, pkt_bytes, app, packets);
    if(x)
    {
        print_payload(header, pkt_bytes, app, ip, tcp, packets);
    }
    else
    {
        fprintf(packets, " data: NOT-A-PSH }\n\n");
        printf("----- Payload -----\\nNOT-A-PSH\\n");
    }
    printf("\\n");

    fclose(packets);
}

```

כאן הגדרנו פונקציה אשר תיקרא בכל פעם אשר אנו מזהים תבילה חדשה, הפונקציה תגדיר את כל שכבות התעבורה של החבילה, תדפיס למשתמש את כל השדות של החבילה וכן תייצא לקובץ את כל החבילות.

```

/* finding all network devices */
// all_devs[name] = {enp0s3, any, lo, bluetooth-monitor, nflog, nqueue, dbus-system, dbus-session}
if(pcap_findalldevs(&all_devs, err) < 0)
{
    fprintf(stderr, "pcap_findalldevs() failed with error code %d.\\nerror message: %s.\\n", errno, err);
    exit(EXIT_FAILURE);
}

while(strcmp(all_devs->name, "lo") != 0)
{
    all_devs = all_devs->next;
}

device = all_devs->name;

```

כעת נתחיל להציג את הפונקציה הראשית. כאן אנו מוצאים את כל מכשירי הרשת שהמחשב שלנו מזהה - ברצוננו להסניף חבילות מהתעבורה המקומית לכן נחפש אותה.


```
// opening the device for packet capturing
handle = pcap_open_live(device , IP_MAXPACKET, PROMISCUSE_MODE, 1000, err);
if(handle == NULL)
{
    fprintf(stderr, "pcap_open_live() failed with error code %d.\nerror message: %s.\n", errno, err);
    exit(EXIT_FAILURE);
}
```

כאן אנו פותחים את המכשיר הרצוי להסנפת חבילות.

```
// applying sniffing filter
filter = "tcp";
if (pcap_compile(handle, &fp, filter, 0, net) < 0)
{
    fprintf(stderr, "pcap_compile() failed with error code: %d.\nerror message: %s.\n", errno, pcap_geterr(handle));
    exit(EXIT_FAILURE);
}
if (pcap_setfilter(handle, &fp) < 0)
{
    fprintf(stderr, "pcap_setfilter() failed with error code: %d.\nerror message: %s.\n", errno, pcap_geterr(handle));
    exit(EXIT_FAILURE);
}
```

כאן אנו מגדירים סינון לחבילות שהסניפר יציג לנו, כיוון שאנחנו מעוניינים רק בחבילות הספיציפיות של מטלה 2.

```
// what
printf("Sniffing from: \"%s\", with filters: \"%s\"...\n", device, filter);

// packet processing
pcap_loop(handle, DONT_STOP_CAPTURE, packet_data, NULL);

// closing sniffer handler
pcap_close(handle);

// freeing all
pcap_freealldevs(all_devs);

// exit without errors
exit(EXIT_SUCCESS);
```

כאן אנו מציגים למשתמש על איזה מכשיר הופעלה הסנפה ועם אילו סינונים, לאחר מכן מתחילה הסנפת חבילות פעילה. שבסופה אנו סוגרים את המשאבים ויוצאים מהתוכנית.

1.5 מימוש זייפן חבילות - *Spoof.c*

כעת נעבור על הקוד של זייפן החבילות.

```
unsigned short checksum(void *b, int len)
{
    unsigned short *buf = b;
    unsigned int sum=0;
    unsigned short result = 0;

    for ( sum = 0; len > 1; len -= 2 )
        sum += *buf++;
    if ( len == 1 )
        sum += *(unsigned char*)buf;

    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum;
    return result;
}
```

כאן כתבנו פונקציה אשר מחשבת את שדה ה-*checksum* של החבילה הנשלחת.

```
void send_raw_ip_packet(struct iphdr* ip)
{
    int enable = 1;

    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if(sock <= 0)
    {
        perror("socket() failed");
        exit(1);
    }

    if(setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable)))
    {
        perror("setsockopt() failed");
        close(sock);
        exit(1);
    }

    struct sockaddr_in addr;
    memset(&addr, 0, sizeof(addr));

    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = ip->daddr;
```

כאן כתבנו פונקציה אשר פותחת שקע תקשורת ומגדירה את המבנה של התקשורת, נשים לב כי אנו משנים את כתובת המקור של חבילת התגובה לכתובת היעד של חבילת הבקשה כך שכאשר תישלח חבילה לכתובת פיקטיבית, זה יראה כאילו שהכתובת פעילה.

```

size_t s = sendto(sock, ip, ntohs(ip->tot_len), 0, (struct sockaddr*)&addr, sizeof(addr));

if(s < 0)
{
    perror("sendto() failed");
    close(sock);
    exit(1);
}
else if(s == 0)
{
    fprintf(stderr, "send 0 bytes!\n");
    close(sock);
    exit(1);
}

close(sock);
}

```

כאן אנו בפועל שולחים את תבילת התגובה ולאחר מכן סוגרים את השקע.

```

int main(int argc, char *argv[])
{
    char buffer[IP_MAXPACKET] = {'\0'};

    struct icmp *icmp = (struct icmp*)(buffer + sizeof(struct iphdr));
    icmp->type = 8;
    icmp->checksum = 0;
    icmp->checksum = checksum((unsigned short*)icmp, sizeof(struct icmp));

    struct iphdr *ip = (struct iphdr*)buffer;
    ip->version = 4;
    ip->ihl = 5;
    ip->ttl = 20;
    ip->saddr = inet_addr("1.2.3.4");
    ip->daddr = inet_addr("10.0.2.15");
    ip->protocol = IPPROTO_ICMP;
    ip->tot_len = htons(sizeof(struct iphdr) + sizeof(struct icmp));

    send_raw_ip_packet(ip);

    exit(0);
}

```

זוהי הפונקציה הראשית, כאן אנו מגדירים ידנית כל שדה בשכבת האינטרנט פרוטוקול, לצורך העניין - אנו שולחים מהכתובת של המחשב (נתב) אל כתובת שידועה כמזויפת שהיא - 1.2.3.4

1.6 מימוש זייפן תגובות - *Sniff_and_spoof.c*

כאן נעבור על הקוד של זייפן תגובות הפינג שכתבתנו:

```
#include <stdio.h> // standard
#include <stdlib.h> // exit
#include <pcap.h> // packet capturing
#include <netinet/ip.h> // ip header
#include <netinet/tcp.h> // tcp header
#include <net/ethernet.h> // ethernet header
#include <errno.h> // last error number
#include <string.h> // memset
#include <netinet/ip_icmp.h>
#include <bits/types.h>
#include <unistd.h>

/* just for comfort */
#define PROMISCUSE_MODE 1
#define NON_PROMISCUSE_MODE 0
#define DONT_STOP_CAPTURE -1

/* how many packets has been received*/
long packet_num = 0;
```

אלו החלקים המעובדים מראש - אותה משמעות כמו ברחרחן.

```
unsigned short checksum(void *b, int len)
{
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result = 0;

    for (sum = 0; len > 1; len -= 2)
        sum += *buf++;
    if (len == 1)
        sum += *(unsigned char *)buf;

    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum;
    return result;
}
```

כאן אנו כתבנו פונקציה אשר מחשבת את שדה ה-*checksum* בחבילה שנשלח.

```

void packet_data(u_char *user, const struct pcap_pkthdr *header, const u_char *pkt_bytes)
{
    struct iphdr *ip_req;

    ip_req = (struct iphdr*)(pkt_bytes + sizeof(struct ether_header));

    char replay_pack[IP_MAXPACKET] = {'\0'};

    struct icmphdr *icmp = (struct icmphdr *)(replay_pack + sizeof(struct iphdr));

    icmp->type = 0;
    icmp->checksum = 0;
    icmp->checksum = checksum((unsigned short *)icmp, sizeof(struct icmphdr));

    struct iphdr *ip = (struct iphdr *)replay_pack;
    ip->version = 4;
    ip->ihl = 5;
    ip->ttl = 20;
    ip->saddr = ip_req->daddr;
    ip->daddr = ip_req->saddr;
    ip->protocol = IPPROTO_ICMP;
    ip->tot_len = htons(sizeof(struct iphdr) + sizeof(struct icmphdr));
}

```

כאן אנו כתבנו פונקציה אשר תיקרא לכל תבילה שנשניף, כאן אנו מגדירים את השכבות של התבילה שאנו רוצים לשלוח כתגובה. מכניסים לחבילת התגובה מידע מזויף שנקבל מהחבילה שהתקבלה.

```

int sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
if (sock <= 0)
{
    perror("socket() failed");
    exit(1);
}

if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable)))
{
    perror("setsockopt() failed");
    close(sock);
    exit(1);
}

struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = ip->daddr;

size_t s = sendto(sock, ip, ntohs(ip->tot_len), 0, (struct sockaddr *)&addr, sizeof(addr));

if (s < 0)
{
    perror("sendto() failed");
    close(sock);
    exit(1);
}
else if (s == 0)
{
    fprintf(stderr, "send 0 bytes!\n");
    close(sock);
    exit(1);
}

printf("Replay has been send to: %s\n", inet_ntoa*((struct in_addr*)&ip_req->saddr));
close(sock);

```

כאן אנו פותחים שקע תקשורת ושולחים תגובה מזויפת.

בפונקציה הראשית כל התהליך של ההסנפה זהה פרט לפילטר שהגדרנו, נציג את ההבדל היחיד:

```

// applying sniffing filter
filter = "icmp";
if (pcap_compile(handle, &fp, filter, 0, net) < 0)
{
    fprintf(stderr, "pcap_compile() failed with error code: %d.\nerror message: %s.\n", errno, pcap_geterr(handle));
    exit(EXIT_FAILURE);
}
if (pcap_setfilter(handle, &fp) < 0)
{
    fprintf(stderr, "pcap_setfilter() failed with error code: %d.\nerror message: %s.\n", errno, pcap_geterr(handle));
    exit(EXIT_FAILURE);
}

```

ההבדל הוא שהפעם מופעל סינון לחבילות של הודעת אינטרנט (ICMP).

1.7 מימוש שער - Gateway.c

כאן נעבור על הקוד של השער:

```
void Send_data(int sockfd, struct sockaddr_in cliaddr, socklen_t len)
{
    ssize_t n;
    float rnd;

    rnd = ((float)random()) / ((float)RAND_MAX);
    printf("Random number: %f\n", rnd);

    if (rnd > 0.5)
    {
        // Send data to the socket
        char *msg = "Hello!";
        n = sendto(sockfd, msg, strlen(msg) + 1, 0, (struct sockaddr *)&cliaddr, len);
        if (n < 0)
        {
            perror("Error sending data");
            close(sockfd);
        }
        else
        {
            printf("send...\n");
        }
    }
}
```

כאן כתבנו פונקציה אשר מגרילה מספר בין 0 ל-1 ואם המספר גדול מחצי שולחת הודעה.

```
int main(int argc, char *argv[])
{
    if (argc != 2) // checking that the user has specified an IP address
    {
        printf("usage: ./Gateway <ip>\n");
        exit(EXIT_FAILURE);
    }

    int sockfd;
    char buffer[MAX_LEN];
    struct sockaddr_in servaddr, cliaddr;
    socklen_t len;
    ssize_t n;

    // Create the socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0)
    {
        perror("Error creating socket");
        exit(errno);
    }

    // Bind the socket to a specific port
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    if (bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
    {
        perror("Error binding socket");
        close(sockfd);
        exit(errno);
    }
}
```

כאן פתחנו שקע לתקשורת.

```

len = sizeof(cliaddr);
while (1)
{
    n = recvfrom(sockfd, buffer, MAX_LEN, 0, (struct sockaddr *)&cliaddr, &len);
    if (n < 0)
    {
        perror("Error receiving data");
        close(sockfd);
        exit(errno);
    }
    else
    {
        buffer[n] = '\0';
        printf("Received %ld bytes from %s:%d\n", n, inet_ntoa(cliaddr.sin_addr), ntohs(cliaddr.sin_port));
        if(inet_aton(argv[1], &cliaddr.sin_addr) < 0)
        {
            perror("inet_aton() failed");
            close(sockfd);
            exit(errno);
        }
        cliaddr.sin_port = htons(PORT + 1);
        Send_data(sockfd, cliaddr, len);
        cliaddr.sin_port = htons(PORT);
    }
}

// Close the socket
close(sockfd);
return 0;

```

כאן אנו מאזינים לחיבורים נכנסים ומקבלים הודעות נכנסות.

2 פרק ב' - תעבורה ופלט

2.1 רחרחן

כעת נציג את הפלט של הסניפר שכתבנו (פלט בטרמינל ופלט כקובץ) וכמו כן נשווה עם תעבורת ווירשארק כדי לוודא אמינות.

2.1.1 פלט בטרמינל

כאן נציג את פלט לדוגמה המתקבל לאחר הרצת התוכנית.

```
Packet Number - 1
----- ETHERNET II -----
Source MAC address: 00:00:00:00:00:00
Destination MAC address: 00:00:00:00:00:00
Type: IPv4 (0x0800)
----- Internet Protocol -----
Version: 4
Header Length: 20 bytes (5)
Total Length: 60
Identification: 0x97c8 (38856)
Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0xa4f1
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
----- Transmission Control Protocol -----
Source Port: 34316
Destination Port: 9999
TCP Segment Len: 10240
Sequence Number (raw): 1695345233
Acknowledgment Number: 0
Header Length: 40 bytes
Flags: FIN = 0, SYN = 1, RST = 0, PSH = 0, ACK = 0, URG = 0
Window: 65495
Checksum: 0xfe30
Urgent Pointer: 0
----- Application Header -----
timestamp: 1674223452 (seconds)
total length: 74
cache flag: 0
steps flag: 1
type flag: 0
status code: 611
cache control: 38384
----- Payload -----
NOT-A-PSH
```

כאן אנו רואים חבילה, (לצורך העניין זוהי החבילה הראשונה בלחצית הידיים המשולשת של פתיחת הקשר), לכל חבילה אנו מדפיסים את כל השדות שיש לנו גישה אליהם משכבות התעבורה.

```

Packet Number - 4
----- ETHERNET II -----
Source MAC address: 00:00:00:00:00:00
Destination MAC address: 00:00:00:00:00:00
Type: IPv4 (0x0800)
----- Internet Protocol -----
Version: 4
Header Length: 20 bytes (5)
Total Length: 739
Identification: 0x97ca (38858)
Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0xa248
Source Address: 127.0.0.1
Destination Address: 127.0.0.1
----- Transmission Control Protocol -----
Source Port: 34316
Destination Port: 9999
TCP Segment Len: 8192
Sequence Number (raw): 1695345234
Acknowledgment Number: 1245394162
Header Length: 32 bytes
Flags: FIN = 0, SYN = 0, RST = 0, PSH = 1, ACK = 1, URG = 0
Window: 512
Checksum: 0x00d8
Urgent Pointer: 0
----- Application Header -----
timestamp: 1674223452 (seconds)
total length: 753
cache flag: 1
steps flag: 1
type flag: 1
status code: 0
cache control: 65535

```

```

----- Payload -----

0000: 80 04 95 98 02 00 00 00 00 00 00 8C 0A 63 61 6C
0010: 63 75 6C 61 74 6F 72 94 8C 0A 42 69 6E 61 72 79
0020: 45 78 70 72 94 93 94 29 81 94 7D 94 28 8C 0C 6C
0030: 65 66 74 5F 6F 70 65 72 61 6E 64 94 68 02 29 81
0040: 94 7D 94 28 68 05 68 00 8C 10 46 75 6E 63 74 69
0050: 6F 6E 43 61 6C 6C 45 78 70 72 94 93 94 29 81 94
0060: 7D 94 28 8C 08 66 75 6E 63 74 69 6F 6E 94 68 00
0070: 8C 08 46 75 6E 63 74 69 6F 6E 94 93 94 29 81 94
0080: 7D 94 28 8C 04 6E 61 6D 65 94 8C 03 73 69 6E 94
0090: 68 0C 8C 04 6D 61 74 68 94 8C 03 73 69 6E 94 93
00A0: 94 75 62 8C 04 61 72 67 73 94 5D 94 68 09 29 81
00B0: 94 7D 94 28 68 0C 68 0E 29 81 94 7D 94 28 68 11
00C0: 8C 03 6D 61 78 94 68 0C 8C 08 62 75 69 6C 74 69
00D0: 6E 73 94 8C 03 6D 61 78 94 93 94 75 62 68 16 5D
00E0: 94 28 68 00 8C 08 43 6F 6E 73 74 61 6E 74 94 93
00F0: 94 29 81 94 7D 94 8C 05 76 61 6C 75 65 94 4B 02
0100: 73 62 68 02 29 81 94 7D 94 28 68 05 68 22 29 81
0110: 94 7D 94 68 25 4B 03 73 62 8C 08 6F 70 65 72 61
0120: 74 6F 72 94 68 00 8C 0E 42 69 6E 61 72 79 4F 70
0130: 65 72 61 74 6F 72 94 93 94 29 81 94 7D 94 28 8C
0140: 06 73 79 6D 62 6F 6C 94 8C 01 2A 94 68 0C 8C 09
0150: 5F 6F 70 65 72 61 74 6F 72 94 8C 03 6D 75 6C 94
0160: 93 94 8C 0D 61 73 73 6F 63 69 61 74 69 76 69 74
0170: 79 94 68 00 8C 0D 41 73 73 6F 63 69 61 74 69 76

```

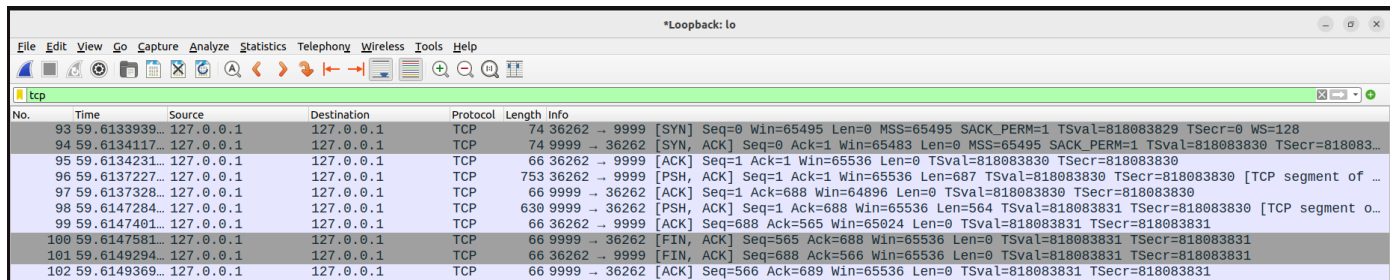
כאן אנו רואים חבילה נוספת, הפעם חבילה זו כן מכילה מידע. נשים לב בנוסף שזוהי גם חבילה מספר 4 (שהיא עם מידע אחרי שלוש חבילות של פתיחת הקשר).

```
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$ sudo ./Sniffer
Sniffing from: "lo", with filters: "tcp"...
```

כמובן שגם בתחילת התוכנית אנו מדפיסים למשתמש מאיזה מכשיר מתבצעת ההסנפה ועם אילו מסננים

2.1.2 תעבורת Wireshark

כעת נציג את התעבורה הכוללת לאחר הרצה יחידה של מטלה 2 שהתקבלה בוויירשארק. כאן אנו רואים 01



No.	Time	Source	Destination	Protocol	Length	Info
93	59.6133939...	127.0.0.1	127.0.0.1	TCP	74	36262 → 9999 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=818083829 TSecr=0 WS=128
94	59.6134117...	127.0.0.1	127.0.0.1	TCP	74	9999 → 36262 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=818083830 TSecr=818083...
95	59.6134231...	127.0.0.1	127.0.0.1	TCP	66	36262 → 9999 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=818083830 TSecr=818083830
96	59.6137227...	127.0.0.1	127.0.0.1	TCP	753	36262 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=687 TSval=818083830 TSecr=818083830 [TCP segment of ...
97	59.6137328...	127.0.0.1	127.0.0.1	TCP	66	9999 → 36262 [ACK] Seq=1 Ack=688 Win=64896 Len=0 TSval=818083830 TSecr=818083830
98	59.6147284...	127.0.0.1	127.0.0.1	TCP	630	9999 → 36262 [PSH, ACK] Seq=1 Ack=688 Win=65536 Len=564 TSval=818083831 TSecr=818083830 [TCP segment o...
99	59.6147401...	127.0.0.1	127.0.0.1	TCP	66	36262 → 9999 [ACK] Seq=688 Ack=565 Win=65024 Len=0 TSval=818083831 TSecr=818083831
100	59.6147581...	127.0.0.1	127.0.0.1	TCP	66	9999 → 36262 [FIN, ACK] Seq=565 Ack=688 Win=65536 Len=0 TSval=818083831 TSecr=818083831
101	59.6149294...	127.0.0.1	127.0.0.1	TCP	66	36262 → 9999 [FIN, ACK] Seq=688 Ack=566 Win=65536 Len=0 TSval=818083831 TSecr=818083831
102	59.6149369...	127.0.0.1	127.0.0.1	TCP	66	9999 → 36262 [ACK] Seq=566 Ack=689 Win=65536 Len=0 TSval=818083831 TSecr=818083831

חבילות, שלוש הראשונות הן פתיחת קשר, שלוש האחרונות הן סגירת קשר יש שתי חבילות שבהן נשלח מידע ועוד שתי חבילות אישור על קבלת המידע.

2.1.3 פלט כקובץ

בסוף ריצת התוכנית, אנו מייצאים לקובץ את כל המידע שנדרש על החבילות שהתקבלו, נציג מידע לדוגמה מהקובץ שהתקבל (החבילות שהצגנו קודם לכן).

```
{ source_ip: 127.0.0.1, dest_ip: 127.0.0.1, source_port: 34316, dest_port: 9999, timestamp: 1674223452, total_length: 74, cache_flag: 0, steps_flag: 1, type_flag: 0, status_code: 611, cache_control: 38384 data: NOT-A-PSH }
{ source_ip: 127.0.0.1, dest_ip: 127.0.0.1, source_port: 9999, dest_port: 34316, timestamp: 1674223452, total_length: 74, cache_flag: 0, steps_flag: 1, type_flag: 0, status_code: 611, cache_control: 63798 data: NOT-A-PSH }
{ source_ip: 127.0.0.1, dest_ip: 127.0.0.1, source_port: 34316, dest_port: 9999, timestamp: 1674223452, total_length: 66, cache_flag: 0, steps_flag: 1, type_flag: 0, status_code: 611, cache_control: 53883 data: NOT-A-PSH }
{ source_ip: 127.0.0.1, dest_ip: 127.0.0.1, source_port: 34316, dest_port: 9999, timestamp: 1674223452, total_length: 753, cache_flag: 1, steps_flag: 1, type_flag: 1, status_code: 0, cache_control: 65535, data:
0000: 80 04 95 98 02 00 00 00 00 00 00 8C 0A 63 61 6C
0010: 63 75 6C 61 74 6F 72 94 8C 0A 42 69 6E 61 72 79
0020: 45 78 70 72 94 93 94 29 81 94 7D 94 28 8C 0C 6C
0030: 65 66 74 5F 6F 70 65 72 61 6E 64 94 68 02 29 81
0040: 94 7D 94 28 68 05 68 00 8C 10 46 75 6E 63 74 69
0050: 6F 6E 43 61 6C 6C 45 78 70 72 94 93 94 29 81 94
0060: 7D 94 28 8C 08 66 75 6E 63 74 69 6F 6E 94 68 00
0070: 8C 08 46 75 6E 63 74 69 6F 6E 94 93 94 29 81 94
0080: 7D 94 28 8C 04 6E 61 6D 65 94 8C 03 73 69 6E 94
0090: 68 0C 8C 04 6D 61 74 68 94 8C 03 73 69 6E 94 93
00A0: 94 75 62 8C 04 61 72 67 73 94 5D 94 68 09 29 81
00B0: 94 7D 94 28 68 0C 68 0E 29 81 94 7D 94 28 68 11
00C0: 8C 03 6D 61 78 94 68 0C 8C 08 62 75 69 6C 74 69
00D0: 6E 73 94 8C 03 6D 61 78 94 93 94 75 62 68 16 5D
00E0: 94 28 68 00 8C 08 43 6F 6E 73 74 61 6E 74 94 93
00F0: 94 29 81 94 7D 94 8C 05 76 61 6C 75 65 94 4B 02
0100: 73 62 68 02 29 81 94 7D 94 28 68 05 68 22 29 81
0110: 94 7D 94 68 25 4B 03 73 62 8C 08 6F 70 65 72 61
0120: 74 6F 72 94 68 00 8C 0E 42 69 6E 61 72 79 4F 70
```

```
, timestamp: 1674223452, total_length: 74, cache_flag: 0, steps_flag: 1, type_flag: 0, status_code: 611, cache_control: 38384 data: NOT-A-PSH }
, timestamp: 1674223452, total_length: 74, cache_flag: 0, steps_flag: 1, type_flag: 0, status_code: 611, cache_control: 63798 data: NOT-A-PSH }
, timestamp: 1674223452, total_length: 66, cache_flag: 0, steps_flag: 1, type_flag: 0, status_code: 611, cache_control: 53883 data: NOT-A-PSH }
, timestamp: 1674223452, total_length: 753, cache_flag: 1, steps_flag: 1, type_flag: 1, status_code: 0, cache_control: 65535,
```

כאן אנו רואים ייצוא לדוגמה של ארבעת החבילות הראשונות בפורמט שהתבקש. מייצאים את השדות הרלוונטיים בלבד (לעומת ההדפסה למשתמש שהיא יותר נרחבת).

2.2 זייפן חבילות

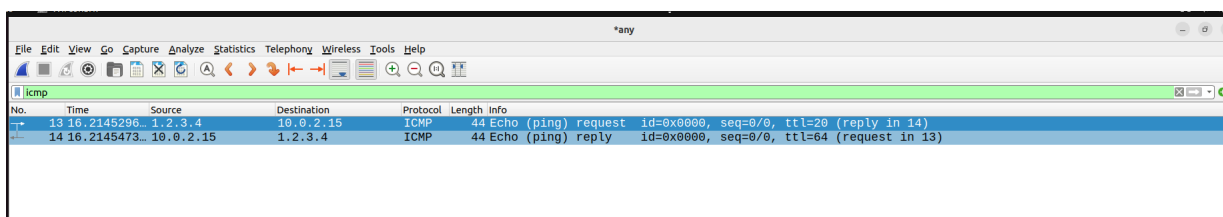
2.2.1 פלט בטרמינל

```
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$ sudo ./Spoofer
[sudo] password for yoad:
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$
```

לתוכנית זו אין פלט למשתמש, הטרמינל דורש רק סיסמה (כיוון שיש להריץ כמשתמש-על), אך חוץ מזה אין פלט כלל.

2.2.2 תעבורת Wireshark

כעת נציג תעבורה לאחר הפעלת התוכנית פעם אחת.



כאן אנו יכולים לראות ששלחו למחשבנו פינג ממחשב עם כתובת פיקטיבית, כמובן שיש תגובה מהמחשב שלנו (אך כנראה שבפועל היא לא תגיע לשום מקום).

2.3 זייפן תגובות

2.3.1 חלק א

פלט נציג את הפלט שהתקבל לאחר הרצת הפינג בין הדוקרים

```
yoad@yoad-VirtualBox: ~/Desktop/Labsetup-20230118/Labsetup
root@422d60f9302a: /home/hostA# ./Ping 10.9.0.6
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=0 ttl=64 time=0.22 ms
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=1 ttl=64 time=0.31 ms
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=2 ttl=64 time=0.39 ms
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=3 ttl=64 time=0.18 ms
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=4 ttl=64 time=0.43 ms
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=5 ttl=64 time=0.17 ms
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=6 ttl=64 time=0.18 ms
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=7 ttl=64 time=0.47 ms
29 bytes has been rcv from 10.9.0.6 to 10.9.0.5: icmp_seq=8 ttl=64 time=0.14 ms
^C
root@422d60f9302a: /home/hostA#
```

פשוט הרצנו את הפינג שאנו כתבנו במטלה 4 וקיבלנו פלט בהתאם.

תעבורה נציג את תוצאות התעבורה של הווירשארק.

No.	Time	Source	Destination	Protocol	Length	Info
125	41.9642221...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
126	41.9642766...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 127)
127	41.9643156...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 126)
128	41.9643547...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
132	42.9660391...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=256/1, ttl=64 (no response found!)
133	42.9661083...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=256/1, ttl=64 (reply in 134)
134	42.9661654...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) reply id=0x0000, seq=256/1, ttl=64 (request in 133)
135	42.9661865...	10.9.0.6	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=256/1, ttl=64
138	43.9672028...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=512/2, ttl=64 (no response found!)
139	43.9673964...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=512/2, ttl=64 (reply in 140)
140	43.9674600...	10.9.0.6	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=512/2, ttl=64 (request in 139)
141	43.9674834...	10.9.0.6	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=512/2, ttl=64
154	44.9702063...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=768/3, ttl=64 (no response found!)
155	44.9702487...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=768/3, ttl=64 (reply in 156)
156	44.9702788...	10.9.0.6	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=768/3, ttl=64 (request in 155)
157	44.9702883...	10.9.0.6	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=768/3, ttl=64
163	45.9712994...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=1024/4, ttl=64 (no response found!)
164	45.9713597...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=1024/4, ttl=64 (reply in 165)
165	45.9714138...	10.9.0.6	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=1024/4, ttl=64 (request in 164)
166	45.9714296...	10.9.0.6	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=1024/4, ttl=64
180	46.9727672...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=1280/5, ttl=64 (no response found!)
181	46.9728085...	10.9.0.5	10.9.0.6	ICMP	45	Echo (ping) request id=0x0000, seq=1280/5, ttl=64 (reply in 182)
182	46.9728374...	10.9.0.6	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=1280/5, ttl=64 (request in 181)

Frame 139: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface any, id 0
 Linux cooked capture v1
 Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
 Internet Control Message Protocol

כאן אנו רואים תעבורה סטנדרטית של פינג-פונג, בקשות ותגובות.

2.3.2 חלק ב

פלט נציג את הפלט שהתקבל לאחר הרצת פינג בין הדוקר הראשון לגוגל

```

yoad@yoad-VirtualBox: ~/Desktop/Labsetup-20230118/Labsetup
root@422d60f9302a: /home/hostA# ./Ping 8.8.8.8
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=0 ttl=114 time=15.21 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=1 ttl=114 time=6.76 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=2 ttl=114 time=12.95 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=3 ttl=114 time=10.97 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=4 ttl=114 time=8.14 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=5 ttl=114 time=9.62 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=6 ttl=114 time=9.75 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=7 ttl=114 time=10.69 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=8 ttl=114 time=7.84 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=9 ttl=114 time=8.29 ms
29 bytes has been rcv from 8.8.8.8 to 10.9.0.5: icmp_seq=10 ttl=114 time=7.51 ms
^C
root@422d60f9302a: /home/hostA#
  
```

תעבורה נציג את תוצאות התעבורה של הווירשארק.

No.	Time	Source	Destination	Protocol	Length	Info
122	16.4251467...	10.9.0.5	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=1792/7, ttl=64 (reply in 125)
123	16.4252342...	10.0.2.15	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=1792/7, ttl=63 (reply in 124)
124	16.4351266...	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=1792/7, ttl=115 (request in 123)
125	16.4352355...	8.8.8.8	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=1792/7, ttl=114 (request in 122)
126	16.4352580...	8.8.8.8	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=1792/7, ttl=114
127	17.4370661...	10.9.0.5	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2048/8, ttl=64 (no response found!)
128	17.4370661...	10.9.0.5	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2048/8, ttl=64 (reply in 131)
129	17.4371177...	10.0.2.15	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2048/8, ttl=63 (reply in 130)
130	17.4443873...	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=2048/8, ttl=115 (request in 129)
131	17.4445910...	8.8.8.8	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=2048/8, ttl=114 (request in 128)
132	17.4446228...	8.8.8.8	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=2048/8, ttl=114
133	18.4461083...	10.9.0.5	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2304/9, ttl=64 (no response found!)
134	18.4461083...	10.9.0.5	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2304/9, ttl=64 (reply in 137)
135	18.4461375...	10.0.2.15	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2304/9, ttl=63 (reply in 136)
136	18.4542097...	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=2304/9, ttl=115 (request in 135)
137	18.4542570...	8.8.8.8	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=2304/9, ttl=114 (request in 134)
138	18.4542667...	8.8.8.8	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=2304/9, ttl=114
139	19.4554227...	10.9.0.5	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2560/10, ttl=64 (no response found!)
140	19.4554227...	10.9.0.5	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2560/10, ttl=64 (reply in 143)
141	19.4554672...	10.0.2.15	8.8.8.8	ICMP	45	Echo (ping) request id=0x0000, seq=2560/10, ttl=63 (reply in 142)
142	19.4626193...	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=2560/10, ttl=115 (request in 141)
143	19.4627168...	8.8.8.8	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=2560/10, ttl=114 (request in 140)
144	19.4627416...	8.8.8.8	10.9.0.5	ICMP	45	Echo (ping) reply id=0x0000, seq=2560/10, ttl=114

Frame 71: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface any, id 0
 Linux cooked capture v1
 Internet Protocol Version 4, Src: 10.9.0.5, Dst: 8.8.8.8
 Internet Control Message Protocol

2.3.3 חלק ג

פלט כעת נציג את הפלט לאחר שליחת פניג לכתובת מאוייפת

```
yoad@yoad-VirtualBox: ~/Desktop/Labsetup-20230118/Labsetup
root@yoad-VirtualBox:/volumes# ./Sniff_and_spoof
Sniffing from: "enp0s3", with filters: "icmp"...
```

Replay has been send to: 10.0.2.15
 Reply has been send to: 10.0.2.15
 Reply has been send to: 10.0.2.15
 Reply has been send to: 10.0.2.15
 Reply has been send to: 10.0.2.15
 Reply has been send to: 10.0.2.15
 Reply has been send to: 10.0.2.15
 ^C
 root@yoad-VirtualBox:/volumes#

כאן אנו מפעילים הסנפה פעילה, בציפיה לקבל פניג לכתובת מאוייפת וכמובן כאשר אנו מקבלים הסנפה לכתובת מאוייפת אנו ישר שולחים תגובה כך שיראה שהכתובת אינה מאוייפת.

```
yoad@yoad-VirtualBox: ~/Desktop/Labsetup-20230118/Labsetup
root@422d6f9302a:/home/hostA# ./Ping 1.2.3.4
28 bytes has been rcv from 1.2.3.4 to 10.9.0.5: icmp_seq=1 ttl=20 time=258.50 ms
28 bytes has been rcv from 1.2.3.4 to 10.9.0.5: icmp_seq=2 ttl=20 time=22.67 ms
28 bytes has been rcv from 1.2.3.4 to 10.9.0.5: icmp_seq=3 ttl=20 time=20.34 ms
28 bytes has been rcv from 1.2.3.4 to 10.9.0.5: icmp_seq=4 ttl=20 time=19.45 ms
28 bytes has been rcv from 1.2.3.4 to 10.9.0.5: icmp_seq=5 ttl=20 time=13.78 ms
28 bytes has been rcv from 1.2.3.4 to 10.9.0.5: icmp_seq=6 ttl=20 time=24.04 ms
28 bytes has been rcv from 1.2.3.4 to 10.9.0.5: icmp_seq=7 ttl=20 time=22.25 ms
^C
root@422d6f9302a:/home/hostA#
```

כאן אנו מקבלים תגובות מהכתובת המאוייפת.

תעבורה נראה את תעבורת התוכנית.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
2	0.000000000	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
3	0.000123929	10.0.2.15	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=0/0, ttl=63 (no response found!)
4	0.258193834	1.2.3.4	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=20
5	0.258219370	1.2.3.4	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=20
6	1.260126948	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=256/1, ttl=64 (no response found!)
7	1.260126948	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=256/1, ttl=64 (no response found!)
8	1.260282211	10.0.2.15	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=256/1, ttl=63 (no response found!)
9	1.282561859	1.2.3.4	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=20
10	1.282573536	1.2.3.4	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=20
11	2.286064456	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=512/2, ttl=64 (no response found!)
12	2.286064456	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=512/2, ttl=64 (no response found!)
13	2.286113037	10.0.2.15	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=512/2, ttl=63 (no response found!)
14	2.306097891	1.2.3.4	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=20
15	2.306123913	1.2.3.4	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=20
16	3.312441927	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=768/3, ttl=64 (no response found!)
17	3.312441927	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=768/3, ttl=64 (no response found!)
18	3.312530288	10.0.2.15	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=768/3, ttl=63 (no response found!)
19	3.331525707	1.2.3.4	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=20
20	3.331548168	1.2.3.4	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=20
21	4.340344704	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=1024/4, ttl=64 (no response found!)
22	4.340344704	10.9.0.5	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=1024/4, ttl=64 (no response found!)
23	4.340440112	10.0.2.15	1.2.3.4	ICMP	45	Echo (ping) request id=0x0000, seq=1024/4, ttl=63 (no response found!)

נשים לב כי למרות שאנו שולחים לכתובת שאינה פעילה, אנו אכן מקבלים תגובה אקטיבית.

2.4 שער

כעת נציג פלט ותעבורה של השער.

2.4.1 פלט

```
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$ ./Gateway 1.2.3.4
wait for upd packet...
Received 7 bytes from 10.0.2.15:52502
Random number: 0.840188
send...
Received 7 bytes from 10.0.2.15:40980
Random number: 0.394383
Received 7 bytes from 10.0.2.15:43465
Random number: 0.783099
send...
Received 7 bytes from 10.0.2.15:51522
Random number: 0.798440
send...
Received 7 bytes from 10.0.2.15:44383
Random number: 0.911647
send...
^C
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$
```

```
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$ echo -n example | nc -4u -w 1 10.0.2.15 9090
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$ echo -n example | nc -4u -w 1 10.0.2.15 9090
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$ echo -n example | nc -4u -w 1 10.0.2.15 9090
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$ echo -n example | nc -4u -w 1 10.0.2.15 9090
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$ echo -n example | nc -4u -w 1 10.0.2.15 9090
yoad@yoad-VirtualBox:~/Desktop/Ex5_cnc$
```

כאן אנו מריצים מספר פעמים את השער שכתבנו, נשים לב שכאשר המספר המוגרל קטן מחצי לא נשלחת תגובה, התגובה נשלחת רק כאשר המספר המוגרל גדול מחצי כנדרש.

2.4.2 תעבורה

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	10.0.2.15	10.0.2.15	UDP	51	52502 → 9090 Len=7
2	0.000157898	10.0.2.15	1.2.3.4	UDP	51	9090 → 9091 Len=7
3	4.222606180	10.0.2.15	10.0.2.15	UDP	51	40980 → 9090 Len=7
4	7.999921103	10.0.2.15	10.0.2.15	UDP	51	43465 → 9090 Len=7
5	8.001141471	10.0.2.15	1.2.3.4	UDP	51	9090 → 9091 Len=7
6	11.7225866...	10.0.2.15	10.0.2.15	UDP	51	51522 → 9090 Len=7
7	11.7227946...	10.0.2.15	1.2.3.4	UDP	51	9090 → 9091 Len=7
8	14.8106044...	10.0.2.15	10.0.2.15	UDP	51	44383 → 9090 Len=7
9	14.8109587...	10.0.2.15	1.2.3.4	UDP	51	9090 → 9091 Len=7

כאן ניתן לראות תעבורה של שליחת הודעות (*Datagrams*) בפרוטוקול *UDP*, נשים לב שלהבדיל כאן אין חבילות של פתיחת או סגירת קשר וגם אין חבילות הנשלחות לאישור כך שקיבלנו חבילה. יתרה מזאת, נשים לב שגם לפי הדרישה, מספר הפורט משתנה וגדל לאחד בעת הצורך ולא משתנה כשאין צורך.

3 פרק ג' - מענה על השאלות

3.1 שאלה 1

כאשר הרצנו את הסניפר שכתבנו בעת ההרצה השתמשנו בהרצה על ידי `sudo` שמשמעותה היא הרצה כ-`SuperUser`, כלומר משתמש על - במילים פשוטות יותר (והקבלה למערכת ההפעלה `Windows`) הרצנו בתור מנהל מערכת (`Administrator`). היינו צריכים לתת הרשאה זו, כיוון שתוכניתנו - מטרה היא להסניף חבילות שעוברות דרך כרטיס הרשת ו-"לתפוס" את המידע שהן מחזיקות. לכידה זו של המנות, דורשת חיבור ישיר לכרטיס הרשת של המחשב ולממשק הרשת של המערכת, שכן (הרי לא יתנו גישה זו לכל משתמש שרירותי) הגישה לממשק הרשת של המערכת מוגבלת רק למשתמשים מוסמכים ואמינים (מורשים) וזאת על מנת למנוע פריצות, תחבולות ופגיעה על תהליכי תעבורת הרשת. בנוסף, אנו מפעילים הסנפה בסניפר שלנו, במצב של `Promiscuous-Mode`, גישה זו היא גישה יותר נרחבת ומלאה, שכן במצב זה אנו רואים לא רק חבילות ותעבורה המיועדת ישירות למחשב האישי ולמשתמש הנוכחי שלנו אלא רואים את כל התעבורה בטווח שכרטיס הרשת של מחשבנו מצליח לזהות ולקלוט, מתן אפשרות כזו רחבה לבצעה הסנפה לכל משתמש שלא בהכרח מורשה, יכולה להיות טעות חמורה ואף מחדל אבטחתי שמשתמשים שאינם מורשים יוכלו לנצל את מערכת התקשורת לרעה (לדוגמה - האקרים...).

כאשר מנסים להפעיל את הסניפר ולהתחיל לבצע הסנפה ללא הרשאות משתמש על התוכנית נופלת בשלב בו צריך להפעיל את מכשיר מכשיר הרשת ממנו אנחנו מעוניינים להתחיל לבצע הסנפה. (בפונקציה `(pcap_open_live)` כיוון שבשביל לפתוח מכשיר רשת להנספה פעילה, צריך לגשת לשכבות רגישות של השליטה בתעבורה ושוב, לא לכל משתמש (יש) אמורה להיות ההרשאה הנ"ל.

3.2 שאלה 2

כאשר אנו מגדירים חבילה שברצוננו לשלוח, אנו מבצעים השמה ידנית של השדות שחייבים להיות מוגדרים בחבילה וכמובן גם מגדירים את גודל שכבת האינטרנט פרוטוקול. אנו לא יכולים לשים כל ערך שרירותי בשדה של גודל החבילה כיוון שהפונקציה `(sendto)` תדרוש את השדה הזה כאשר אנו מעבירים את גודל החבילה שברצוננו לשלוח (בפרמטר `size`). אלא אם אנחנו ידנית ובאופן מודע נשלח את השדה שמתאר את הגודל ב-`*iphdr struct`, רק אז אנחנו נשפיע על השליחה. בכל מצב אחר בו נעביר גודל תקין, ישלח הגודל האמיתי בהתאם. בנוסף לכך, אם נשלח גודל שקטן מ-02 בתים, הפונקציה `(sendto)` תזרוק לנו שגיאה, כיוון שכל חבילה חייבת להיות לכל הפחות 02 בתים, שכן אחרת היא תיחשב חבילה פגומה ולא תקינה לשליחה (זאת מכיוון שצריך 02 בתים לשכבת האינטרנט פרוטוקול), וגרעין מערכת ההפעלה לא ידע כיצד לגשת ולפרש את החבילה שהתקבלה בפועל כתוצאה מהשליחה.

3.3 שאלה 3

כאשר משתמשים ב-`raw socket` אנו לא בהכרח חייבים לחשב באופן ידני את שדה ה-`checksum` בשכבת האינטרנט פרוטוקול. הדבר תלוי אם הפעלנו בשקע שיצרנו את האופציה של `IP_HDRINCL` - אם האופציה פועלת, גרעין מערכת ההפעלה (`OS Kernel`) יבנה את החבילה בצורה אוטומטית וכמובן כבר יכלול בשדה הרלוונטי חישוב תקין של ה-`checksum`. אם תכונה זו אינה פועלת על השקע שבו אנו משתמשים, עלינו יהיה לחשב בצורה את ערך ה-`checksum` כיוון שלא בהכרח מערכת ההפעלה תטפל בחישוב והשמה של ערך תקין בשדה הרלוונטי, כמובן, עלינו קודם כל לאפס ורק לאחר מכן לבצע השמה של החישוב הנכון של הערך.

4 פרק ד' - יכולות, הגבלות ומחקר נוסף

4.1 רחרחן

כאן נסקור מספר תכונות של הסניפר שכתבנו.

4.1.1 יכולות

ללא שינויים (הקוד המצורף) מסוגל לבצע הסנפה פעילה של חבילות המועברות בפרוטוקול *TCP* שנמצאות בממשק המקומי *lo : Loopback*. הרחרחן שלנו מדפיס למשתמש מספר דברים, בתחילת הריצה הוא מודיע למשתמש מאיזה מכשיר רשת מתבצעת הסנפה ואילו מסננים מופעלים, לאחר מכן כאשר הוא מזהה תעבורה הוא מדפיס למשתמש את מספר החבילה שנקלטה (אינדקס, כלומר מספר החבילה האחרונה היא כמה חבילות נקלטו בסך הכל) ולאחר מכן, מדפיס את שכבות התעבורה של החבילה - שכבת האינטרנט, שכבת האינטרנט פרוטוקול, שכבת ה-*TCP*, שכבת האפליקציה והמידע שהחבילה אכן מעבירה איתה. מודפסים כל השדות שיכולים להיות רלוונטיים מכל השכבות, לדוגמה בשכבת האינטרנט אנו מדפיסים כתובות פיזיות (*MAC*) של שני הצדדים, בשכבת האינטרנט פרוטוקול אנחנו מדפיסים כתובות *IP* של שני המקבל והשולח, ועוד כל מיני דגלים וגדלים נוספים, בשכבת ה-*TCP* אנו מדפיסים פורטים, בשכבת האפליקציה אנו מדפיסים את השדות הספייציפיים שנוצרים בחבילות ממטלה 2 שבניהם דגלים ובהדפסה של המידע, אנחנו מדפיסים את המידע שהחבילה מעבירה בפועל בתצורה הקסהדצימלית (ניסינו לדמות את תוכנת *Wireshark*), בנוסף התוכנית שלנו בסוף הריצה מייצאת לקובץ את כל החבילות שהוספו במהלך הריצה (מדמה את שמירת ההקלטות של וירשארק), עם כל השדות הרלוונטיים לפי הדרישה.

4.1.2 מגבלות

כרגע הסניפר שלנו דיי מוגבל, הוא מסניף רק ממקום ספציפי אחד (התעבורה המקומית) ומסניף רק פרוטוקול ספציפי מאוד (*TCP* של מטלה 2), בחבילות *TCP* "רגילות" אין בהכרח את אותם הדגלים כמו שיש בחבילות ממטלה 2, ולכן הסנפה של חבילות רגילות יבוצעו הדפסות מיותרות בשכבת האפליקציה. בנוסף, כרגע מסניפר זה אין אפשרות להסניף פרוטוקולים אחרים כגון *UDP*, *ICMP*, *QUIC* ועוד... כמו כן, הקוד קובע המון דברים שניתן לשפר בהמשך להחלטת המשתמש כדוגמה לתת למשתמש לבחור לבד מאיזה מכשיר רשת לבצע הסנפה, איזה פילטרים להפעיל וכדומה. בנוסף, בסוף הריצה, נשמר קובץ עם כל החבילות שהוספו, ללא מתן אפשרות בחירה למשתמש איזה חבילות הוא בפועל מעוניין לשמור.

4.2 זייפן חבילות

כאן נסקור את התכונות של זייפן החבילות

4.2.1 יכולות

כרגע הזייפן שכתבנו, מסוגל לשלוח פינג מזויף לכתובת המחשב הנוכחי - 10.0.2.15 מהכתובת (שידועה כמזויפת) - 1.2.3.4 את השליחה בפועל ניתן לראות דרך תוכנות שעוקבות אחר תעבורה כגון *Wireshark*.

4.2.2 מגבלות

אין שום פלט היוצא למשתמש בעת סיום הריצה, דבר שעלול להשאיר את המשתמש בחוסר ידיעה אם הפעולה בוצעה בהצלחה או לא (למרות שכן בכל שגיאה יש הודעת הדפסה - לכן ההנחה היא שאם אין שום פלט אזי התוכנית עברה בהצלחה), בנוסף ישנם דברים שקבועים כמו כתובת המחשב אליו שולחים והכתובת המזויפת ממנה שולחים דבר שניתן היה להגדיר כך שהמשתמש יזין את שניהם כקלט.

4.2.3 שינויים קטנים נוספים

הקוד שלנו בנוי כך שעם שינויים יחסית קטנים נוכל לזייף גם פרוטוקולים אחרים ולא רק חבילות בפרוטוקול *ICMP*. לדוגמה השינויים שניתן לעשות הם לשנות את ה-*Structionphdr* למבנה אחר שמתאר את הפרוטוקול שאותו אנו רוצים לשלוח, ולשנות את ההגדרה *IPPROTO_ICMP* בהגדרה של השקע להגדרה אחרת המתארת את הפרוטוקול שאנו מעוניינים לשלוח.

4.3 זייפן תגובות

4.3.1 הסבר מפורט

התוכנית פועלת במספר שלבים. ראשית לפני הכל, אנו מפעילים *Docker* שיהווה לנו שימוש כמספר מכונות שונות הפעולות אשר נוכל לבצע בניהם תקשורת. הדוקר יפעיל שלוש מכונות - הראשונה *Attacker* שהוא יהווה לנו תוקף תעבורת רשת, *HostA*, *HostB* שיהוו לנו הקבלה לשני מחשבים שנמצאים באותה רשת. לאחר שהפעלנו את כלל המערכת הנ"ל (לפי פעולות שנמצאות מלעיל) נוכל להתחיל לפעול לפי שלבי המטלה. בשלב הראשון - אנו שולחים פינג בין שני המחשבים *HostA*, *HostB* את השליחה הנ"ל נבצע בעזרת התוכנית שכתבנו במטלה 4, ניכנס לטרמינל של *HostA* ונכתוב `Ping 10.9.0.6`. הדבר יבצע פינג פשוט בין שתי המכונות. בשלב השני, עלינו לשלוח פינג מהמכונה הראשונה לכתובת הנמצאת ב-WAN (*Wide Area Network*) אנו בחרנו לשלוח פינג לדוגמה לגוגל (גם לפי ההמלצה במטלה) - ביצענו זאת על ידי אותה פקודה כמו קודם על ידי `Ping 8.8.8.8`. לבסוף, בשלב האחרון אנו משתמשים בתוכנית שכתבנו בשאלה זו. אנו שולחים ממכונה *HostA* בקשת פינג לכתובת מזויפת ומצפים לקבל תגובה. אנו עשינו זאת על ידי הרצת שתי פקודות, ב-*Attacker*, ראשית הרצנו את הפקודה `sudo./Sniff_and_poof` שהחלה האזנה לחבילות *ICMP*, לאחר מכן הרצנו על *HostA* את הפקודה `Ping 1.2.3.4`, הדבר שקרה הוא שזייפן התגובות זיהה תעבורה של פינג לכתובת וישר שלח תגובה ואז למרות שהכתובת מזויפת, לכאורה קיבלנו תגובה.

4.3.2 אופן המימוש

כמו בסעיף הראשון, יצרנו סניפר רק שהפעם האזנו למכשיר הרשת החיצונית שהוא *enp0s3* והסנפנו פקטות לפי סינון של *ICMP*. בפונקציה המטפלת בכל חבילה שהוסנפה, פתחנו שקע ולאחר מכן יצרנו חבילה כך שכתובת המקור היא כתובת היעד של החבילה שהסנפנו, לאחר מכן שלחנו אותה דרך השקע וסגרנו תקשורת. כך חזרנו על התהליך בכל חבילה שהתקבלה, ובכך הגבנו לכל פינג שהתקבל וזה נראה שאכן יש תעבורה תקינה לכתובת *IP* מזויפת.

4.3.3 יכולות

לזייפן התגובות שלנו יש יכולות דומות לסניפר בחלק של ההסנפה עצמה של החבילה רק עבור פרוטוקול אחר, (הוא לא מדפיס את החבילה או מייצא לקובץ אם כי אין בכך צורך), בנוסף יש לזייפן יכולות דומות לפינג ממטלה 4 שכן אנו שולחים פונג לאחר שקיבלנו פינג, בדומה לנעשה במטלה.

4.3.4 מגבלות

למרות שהזייפן שולח תגובות תקינות, *Wireshark* עדיין מצליח לזהות ולהראות למשתמש שהכתובת לא פעילה על ידי כיתוב ליד הפינג שלא התקבלה תגובה מתאימה בנושא. זאת כיוון ואנו לא באמת עוצרים פיזית את החבילה שנשלחה אלא רק בודקים האם נשלחה חבילה ואם כן אז שולחים תגובה - דבר הגורם לחוסר אמינות מסוים כאשר מקבלים תגובת פונג.

5 ביבליוגרפיה

רשימת מקורות בהם השתמשנו בעת מימוש וכתובת הפרוייקט:

1. <https://www.tcpdump.org/pcap.html>
2. <https://docs.docker.com/get-started/overview/>
3. <https://devopscube.com/what-is-docker/>
4. <https://www.geeksforgeeks.org/ip-spoofing/>
5. מצגות הקורס "תכנות מערכות 1" של ד"ר חוגי אסף.
6. מצגות וחוברת הקורס "רשתות תקשורת".