Spase Maximillien Sandoval

DETER Project

How The TCP SYN Flood Attack Works

The TCP SYN flood attack is a form of DoS or denial-of-service attack that exploits the normal TCP three-way handshake so it can consume excessive resources on the target server and making it unable to respond to legitimate traffic efficiently. As the name describes, the attacker sends a flood of TCP/SYN packets, every SYN packet that the server gets tells the server to reply with a SYN-ACK or synchronize-acknowledgement response, and to wait for a final ACK that completes the handshake, leaving a temporary half open connection. Since the ACK never arrives back to the server the server gets overwhelmed with these half open connections ramping up the hardware usage significantly and taking away resources like memory or CPU power.

What Can SYN Cookies Do

Syn cookies can help prevent SYN flood attacks by allowing the server to not have to maintain a state, referring back to that half open connection, without the half open connection the SYN flood isn't as effective. Syn cookies work so that instead of allocating recourses towards a connection when the server gets a SYN request, the server sends back a SYN-ACK response where a sequence number is calculated as a hash of client info (ip, port) and that sequence number is used as a cookie. If the client is legitimate, it responds with an ACK that includes the hash value, once the server gets the ACK it can validate the cookie, then if the cookie is valid the server can allocate resources and establish a connection. So in short, syn cookies allow the server to avoid keeping state on all the SYN requests, making the impact of SYN flood attacks pretty small in comparison to a server not using syn cookies.

Client Script

fetch_index.sh

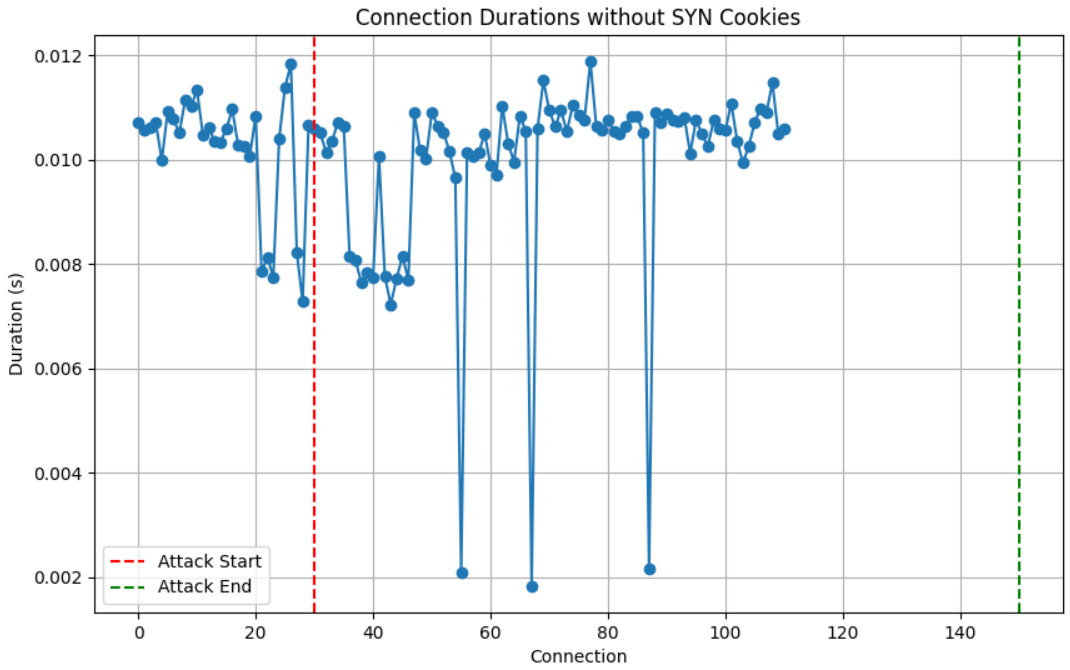
```
#!/bin/bash

while true; do

  curl http://5.6.7.8/index.html

  sleep 1

done
```

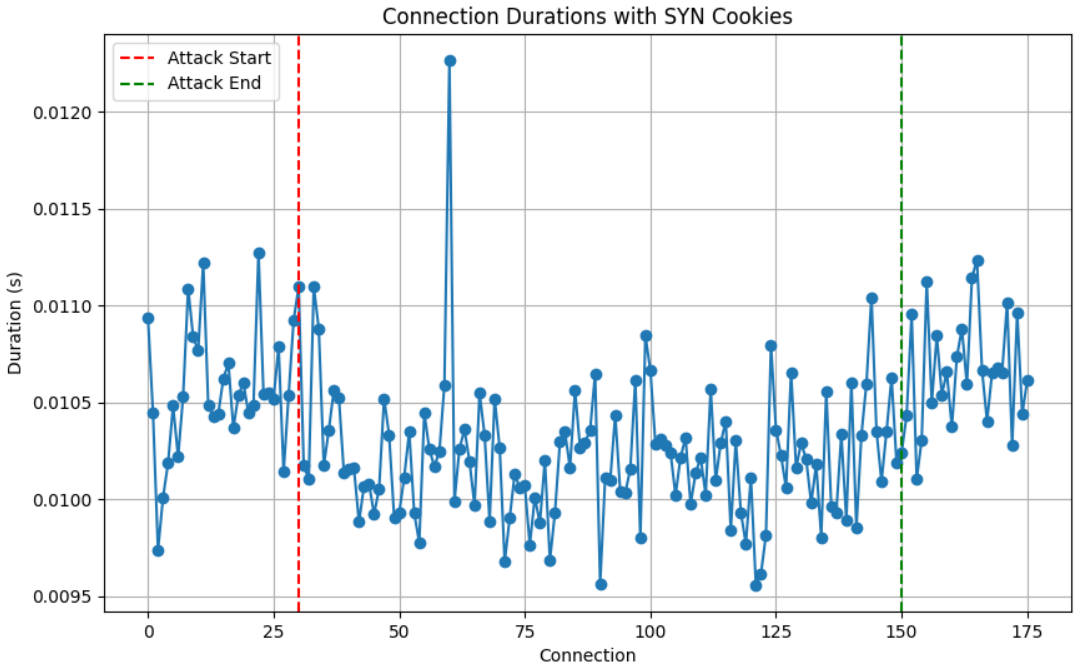
Attack Command

```
sudo flooder --dst 5.6.7.8 --src 1.1.2.0 --srcmask 255.255.255.0 --highrate 100 --proto 6 --

dportmin 80 --dportmax 80
```

Refer to next page for graphs:

Graphs





Graphs Explained

The graphs show accurate effectiveness of the attack, and how SYN cookies can help prevent SYN flood attacks. As seen in the graph with SYN cookies on, the connection is mostly stable and there aren't any significant drops. When we preform the attack without SYN cookies, the connection gets much more unstable, and we can visibly see connections being dropped or reset as they get so close to zero, likely because of the resource exhaustion being caused by the SYN flooder. Also, without syn cookies, the connection duration is much shorter implying the server was unable to maintain the connection/handle the SYN requests properly.

Graphing Script

graph_durations.py

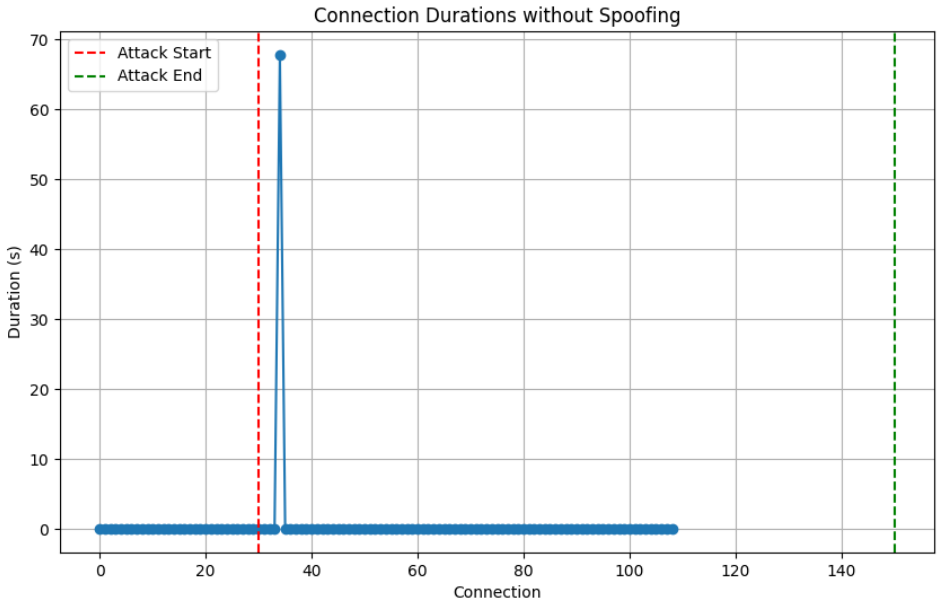
```
import pandas as pd
import matplotlib.pyplot as plt
from scapy.all import rdpcap, TCP

def parse_pcap(file_path):
    packets = rdpcap(file_path)
    connections = {}
    for packet in packets:
        if packet.haslayer(TCP):
            if packet[TCP].flags == 'S':  # SYN
                connection_id = (packet['IP'].src, packet['TCP'].sport,
packet['IP'].dst, packet['TCP'].dport)
                connections[connection_id] = {'syn_time': packet.time,
'fin_time': None}
            elif packet[TCP].flags & 0x01 or packet[TCP].flags & 0x04:  # FIN or
RST
                connection_id = (packet['IP'].src, packet['TCP'].sport,
packet['IP'].dst, packet['TCP'].dport)
                if connection_id in connections:
                    connections[connection_id]['fin_time'] = packet.time

    data = []
    for conn, times in connections.items():
        syn_time = times['syn_time']
        fin_time = times['fin_time'] if times['fin_time'] else syn_time + 200  #
200 if no FIN/RST
        duration = fin_time - syn_time
```

```
        data.append((*conn, duration))

    return pd.DataFrame(data, columns=["Src_IP", "Src_Port", "Dst_IP",
"Dst_Port", "Duration"])

def plot_durations(df, title, output_file, attack_start, attack_end):
    plt.figure(figsize=(10, 6))
    plt.plot(df.index, df["Duration"], marker='o', linestyle='-')
    plt.axvline(x=attack_start, color='r', linestyle='--', label='Attack Start')
    plt.axvline(x=attack_end, color='g', linestyle='--', label='Attack End')
    plt.xlabel("Connection")
    plt.ylabel("Duration (s)")
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.savefig(output_file)
    plt.show()

# parse pcap
df_without_syncookies = parse_pcap("synfloodexp_nocookies.pcap")
df_with_syncookies = parse_pcap("synfloodexp_syncookies.pcap")

# csv
df_without_syncookies.to_csv("durations_without_syncookies.csv", index=False)
df_with_syncookies.to_csv("durations_with_syncookies.csv", index=False)

attack_start = 30  # atk start
attack_end = 150  # atk end

# plot
plot_durations(df_without_syncookies, "Connection Durations without SYN Cookies",
"durations_without_syncookies.png", attack_start, attack_end)
plot_durations(df_with_syncookies, "Connection Durations with SYN Cookies",
"durations_with_syncookies.png", attack_start, attack_end)
```

(modified this script slightly to preform the extra credit below)

Extra Credit

When we perform this attack without SYN cookies and without IP spoofing, the graph shows that the connections become very short, implying the server could handle the connections efficiently. Since all the attack packets came from one IP address, the server was easily able to implement rate limiting reducing the overall impact, not entirely but mostly, as we can see there is one connection that appears with a strangely long duration but other than that the connections seem to be handled well. Since there is no spoofing, the traffic patterns become predictable making this an easy attack to prevent. There are some ways you could make an attack without spoofing more efficiently, first that comes to mind would be increasing the rate at which SYN packets are sent. One could also use multiple legitimate IP addresses, but this seems tedious. Another option would be using a variety of different source ports, if each connection is coming from a different source port it could help bypass rate limiting procedures, could also exhaust resources more efficiently.



Graph: