

Denial-of-Service Attacks using Socket Programming

by

Aditya Mandeep Vakani(01FB16EEC020), Akshar Deepankar
Athreya(01FB16EEC028), Anurag Vinayak Muttur (01FB16EEC042), Ashwin Kumar
Singh(01FB16EEC052)

Department of Electronics and Communication Engineering, PES University,
Bangalore 560085

Supervised by

Professor M. Rajasekar
Department of Electronics and Communication Engineering, PES University,
Bangalore 560085

GitHub Repository:

<https://www.github.com/akshar-Athreya97/CN-Project>

Introduction

This project demonstrates a Denial-of-Service (DoS) Attack from one end system to another with the help of Python and Socket Programming. A Denial-of-Service attack is a cyber-attack in which the attacker seeks to make a machine or network resource unavailable to its intended users by temporarily disrupting services of a victim connected to the internet. In this project we have demonstrated different types of DoS attacks - SYN Flood attack and UDP Flood attack using Socket Programming, RST Flood attacks and ICMP Flood attack using hping3 command line tool. The incoming packets have been captured and analyzed using Wireshark which is a free and open-source network packet analyzer. The effects of the attacks on the computer have been shown using Windows Task Manager.

The two main types of DoS attacks are classified as:

- 1. Bandwidth Depletion:**

This type of attack involves an attacker sending large amounts of traffic to a victim's system, usually causing the system to slow down, crash or preventing access to it by legitimate users, also known as packet flooding. The following two types have been demonstrated:

- a. UDP Flood**
- b. ICMP Flood**

- 2. Resource Depletion:**

This type of attack involves the attacker sending packets that misuse network protocol communications or are malformed, leading to network resources being tied up and none left for legitimate users. Two common resource depletion attacks that have been shown are:

- a. TCP SYN flood**
- b. TCP RST flood**

Software Used

Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Python's simple, easy to learn syntax emphasizes readability and therefore reduces cost of program maintenance. Python supports modules and packages,

which encourages program modularity and code reuse. In this project we've mainly made use of the Socket module to write our code.

Wireshark

Wireshark is a free and open-source packet analyzer which is used for network troubleshooting, analysis, protocol development and for educational purposes. It is a cross platform application that using pcap to capture packets. Wireshark “understands” the structure of different networking protocols and can parse and display the fields with their meanings as specified by different networking protocols.

Photoshop

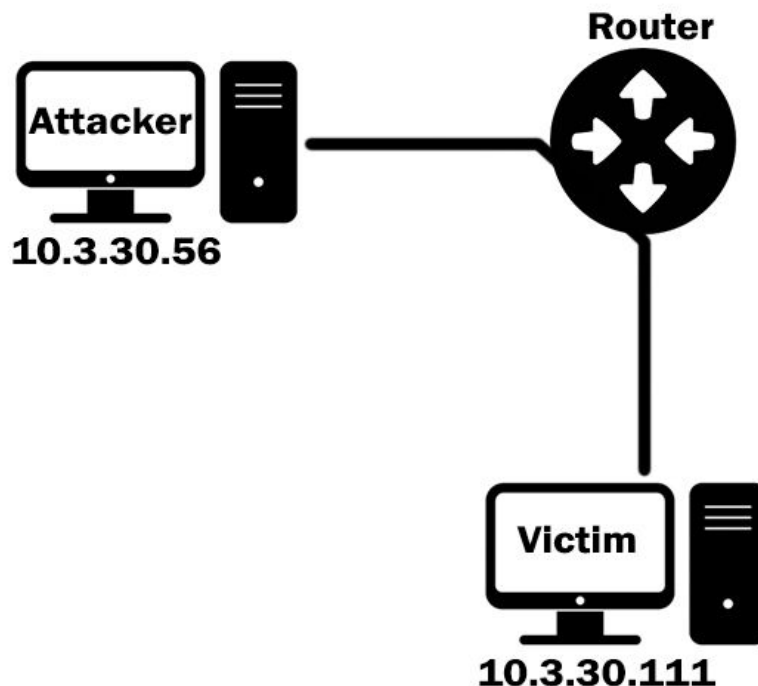
Graphic designing software which has been used here to make illustrations and label diagrams.

The Setup:

We will be using 2 end systems on a private network.

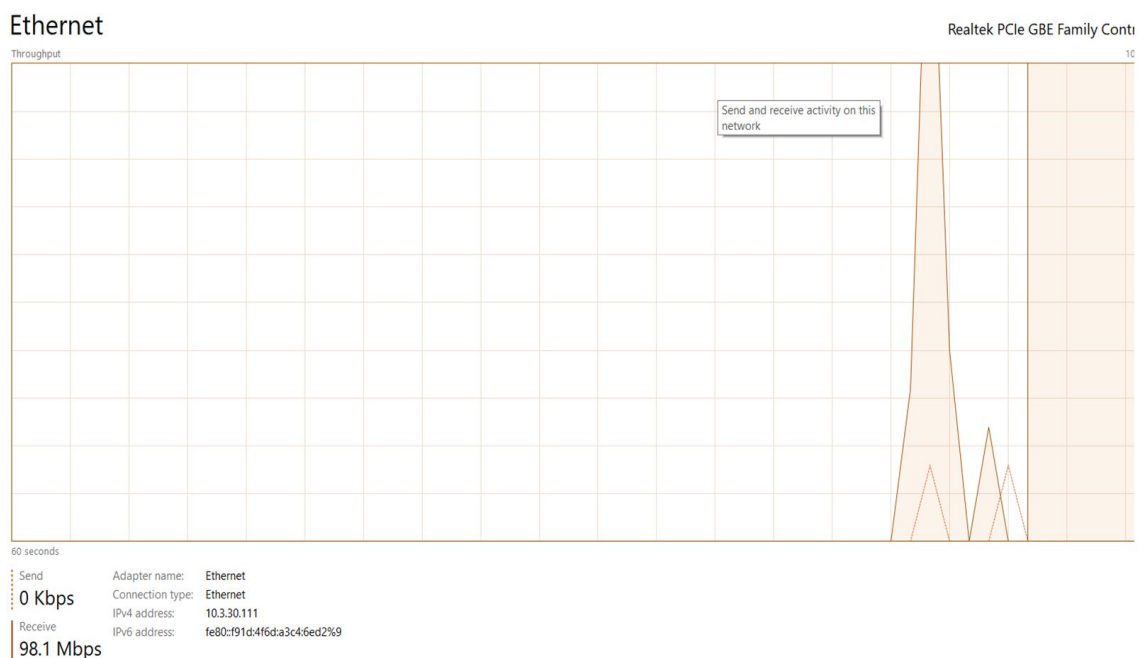
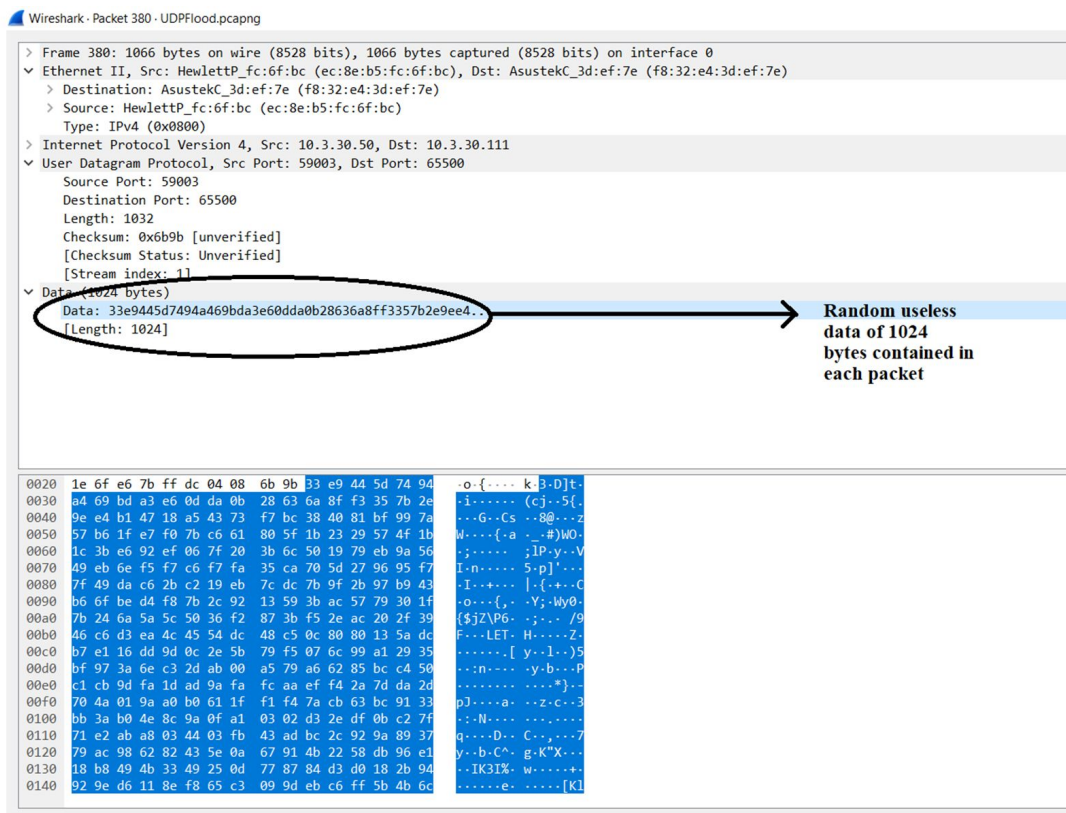
Victim's system IP during test -> *10.3.30.111* and *10.3.30.56* (**Windows 10 64bit**)

Attacking system IP during test -> *10.3.30.50* (**Ubuntu 16.04LTS**)



- Attacker's IP address
- Victim's IP address
- Incoming, continuous flood of UDP packets
- Source and Destination port numbers
- Total Packet length

1. Source and Destination IP address (source - attacker's ip, Destination - victim's ip)
2. The protocol using which the packet has been sent: UDP
3. Port Numbers: Source port: 59003; Destination port: 65500
4. Packet length: 1066 Bytes



If,

We can see that the victims system is being spammed with random junk packets which do not contain any useful data from the same ip address (attackers ip).

The incoming data rates shoots up from few Kbps to 98Mbps which indicates that the victims bandwidth resources are being attacked

First we create a UDP socket within a try-except block, assign a random packet size.

We specify the end time for the attack using the 'endtime' variable.

Until this end time is encountered, the packets are sent using a while loop , with the 'sendto' function containing packet size, target IP and port number as parameters.

The program is called from command line using ->

\$: sudo python2 dos.py udp [target IP] [port] [duration]

ICMP Flood (Ping Flood attack using hping3):

Illustrations:

Wireshark monitor:

Time	Source	Destination	Protocol	Length	Info
75793 14.620449	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=39849/43419, ttl=64 (no response found!)
75794 14.620450	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=40105/43420, ttl=64 (no response found!)
75795 14.620450	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=40361/43421, ttl=64 (no response found!)
75796 14.620450	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=40617/43422, ttl=64 (no response found!)
75797 14.620450	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=40873/43423, ttl=64 (no response found!)
75798 14.620450	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=41129/43424, ttl=64 (no response found!)
75799 14.620450	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=41385/43425, ttl=64 (no response found!)
75800 14.620450	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=41641/43426, ttl=64 (no response found!)
75801 14.620450	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=41897/43427, ttl=64 (no response found!)
75802 14.620451	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=42153/43428, ttl=64 (no response found!)
75803 14.620451	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=42409/43429, ttl=64 (no response found!)
75804 14.620451	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=42665/43430, ttl=64 (no response found!)
75805 14.620451	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=42921/43431, ttl=64 (no response found!)
75806 14.620452	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=43177/43432, ttl=64 (no response found!)
75807 14.620452	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=43433/43433, ttl=64 (no response found!)
75808 14.620452	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=43689/43434, ttl=64 (no response found!)
75809 14.620452	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=43945/43435, ttl=64 (no response found!)
75810 14.620452	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=44201/43436, ttl=64 (no response found!)
75811 14.620452	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=44457/43437, ttl=64 (no response found!)
75812 14.620452	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=44713/43438, ttl=64 (no response found!)
75813 14.620453	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=44969/43439, ttl=64 (no response found!)
75814 14.620453	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=45225/43440, ttl=64 (no response found!)
75815 14.620453	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=45481/43441, ttl=64 (no response found!)
75816 14.620453	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=45737/43442, ttl=64 (no response found!)
75817 14.620453	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=45993/43443, ttl=64 (no response found!)
75818 14.620453	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=46249/43444, ttl=64 (no response found!)
75819 14.620453	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=46505/43445, ttl=64 (no response found!)
75820 14.620454	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=46761/43446, ttl=64 (no response found!)
75821 14.620725	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=47017/43447, ttl=64 (no response found!)
75822 14.620725	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=47273/43448, ttl=64 (no response found!)
75823 14.620725	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=47529/43449, ttl=64 (no response found!)
75824 14.620726	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=47785/43450, ttl=64 (no response found!)
75825 14.620726	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=48041/43451, ttl=64 (no response found!)
75826 14.620726	2.2.2.2	10.3.30.111	ICMP	60	Echo (ping) request id=0x1311, seq=48297/43452, ttl=64 (no response found!)

Frame 75798: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
ethernet II, Src: HewlettP_fc:6f:bc (ec:8e:b5:fc:6f:bc), Dst: AsustekC_3d:ef:7e (f8:32:e4:3d:ef:7e)
Internet Protocol Version 4, Src: 2.2.2.2, Dst: 10.3.30.111
Internet Control Message Protocol
0 f8 32 e4 3d ef 7e ec 8e b5 fc 6f bd 00 00 45 00 -2.-.-.-.-.-E.

In the above illustration we can observe the following:

1. Source IP: 2.2.2.2(attackers spoofed IP address)
2. Destination IP :10.3.30.111 (victims IP address)
3. Protocol using which packets have been sent:ICMP(Internet control Message Protocol)
4. Packet Length:60
5. Each packet is an Echo Request type packet.

```

> Frame 34854: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
> Ethernet II, Src: HewlettP_fc:6f:bc (ec:8e:b5:fc:6f:bc), Dst: AsustekC_3d:ef:7e (f8:32:e4:3d:ef:7e)
▼ Internet Protocol Version 4, Src: 2.2.2.2, Dst: 10.3.30.111
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 28
    Identification: 0xd389 (54153)
    > Flags: 0x0000
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x7ae2 [validation disabled]
    [Header checksum status: Unverified]
    Source: 2.2.2.2
    Destination: 10.3.30.111
▼ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x6782 [correct]
    [Checksum Status: Good]
    Identifier (BE): 4881 (0x1311)
    Identifier (LE): 4371 (0x1113)
    Sequence number (BE): 32108 (0x7d6c)
    Sequence number (LE): 27773 (0x6c7d)
▼ [No response seen]
    > [Expert Info (Warning/Sequence): No response seen to ICMP request]

```

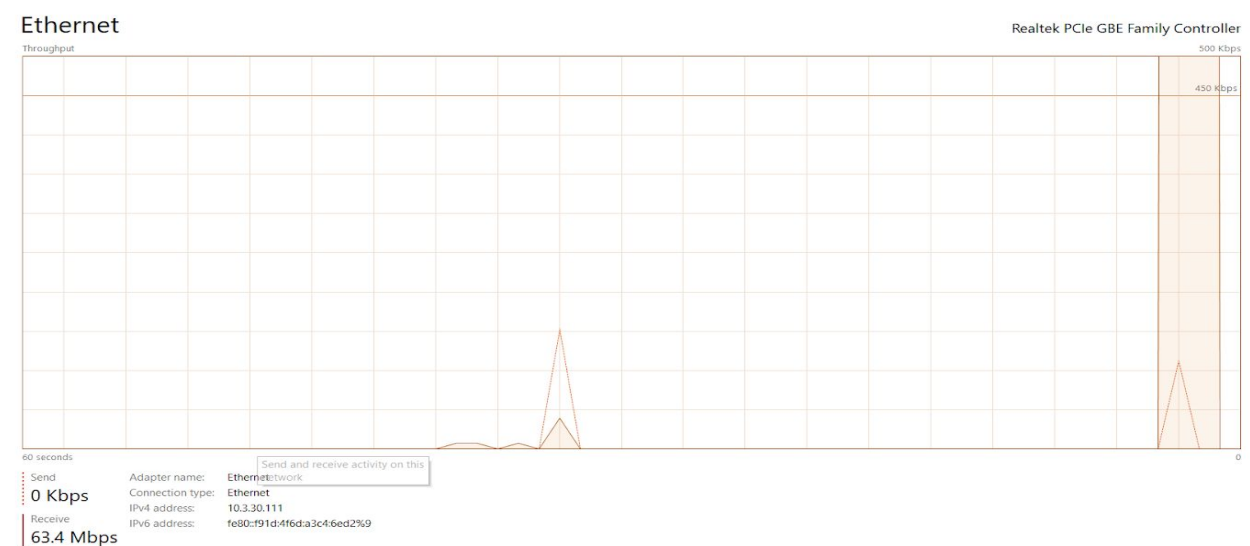
Each packet sent is an Echo Request packet. Hence the type is 8

The packets do not wait for an Echo Reply from the victim's system

```

0000 f8 32 e4 3d ef 7e ec 8e b5 fc 6f bc 08 00 45 00 -2-=-... ..o...E-
0010 00 1c d3 89 00 00 40 01 7a e2 02 02 02 02 0a 03 -...@. z.....
0020 1e 6f 08 00 67 82 13 11 7d 6c 00 00 00 00 00 00 -c.g... }l.....
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```



From the above snapshots we can see that the attacker floods the victim's network with request packets(Echo Request).This strains both the incoming and outgoing channels of the network, consuming significant bandwidth and resulting in a denial of service. The outgoing data rate goes from a few Kbps to 63Mbps.

TCP SYN FLOOD

Illustrations:

No.	Time	Source	Destination	Protocol	Length	Info
4744	25.561310	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4745	25.561310	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4746	25.561310	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4747	25.561311	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4748	25.561312	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4749	25.561312	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4750	25.561313	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4751	25.561313	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4752	25.561313	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4753	25.561313	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4754	25.561314	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4755	25.561314	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4756	25.561314	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4757	25.561314	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4758	25.561315	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4759	25.690684	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4760	25.690684	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4761	25.690685	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4762	25.690685	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4763	25.690685	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4764	25.690685	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4765	25.690686	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4766	25.690686	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4767	25.690686	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4768	25.690686	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4769	25.690686	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4770	25.690686	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4771	25.690687	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4772	25.690687	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4773	25.690687	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4774	25.690687	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4775	25.690688	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4776	25.690688	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4777	25.690688	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4778	25.690688	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4779	25.690688	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21
4780	25.690689	55.55.55.55	10.3.30.111	TCP	75	[TCP Retransmission] 65500 → 80 [SYN] Seq=0 Win=53270 Len=21

In the above illustrations we can observe the following:

1. Source IP: 55.55.55.55(attackers spoofed IP)
2. Destination IP :10.3.30.111(Victims IP address)
3. Protocol used to send the packets:TCP
4. There is a continuous flood of SYN packets without waiting for Acknowledgement (ACK)
5. Source port:65500
6. Destination port:80
7. Packet Length:75

Wireshark - Packet 4765 - Ethernet

```

> Frame 4765: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0
> Ethernet II, Src: HewlettP_fc:6f:bc (ec:8e:b5:fc:6f:bc), Dst: AsustekC_3d:ef:7e (f8:32:e4:3d:ef:7e)
> Internet Protocol Version 4, Src: 55.55.55.55, Dst: 10.3.30.111
v Transmission Control Protocol, Src Port: 65500, Dst Port: 80, Seq: 0, Len: 21
  Source Port: 65500
  Destination Port: 80
  [Stream index: 4]
  [TCP Segment Len: 21]
  Sequence number: 0 (relative sequence number)
  [Next sequence number: 22 (relative sequence number)]
  Acknowledgment number: 0
  0101 .... = Header Length: 20 bytes (5)
  v Flags: 0x002 (SYN)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ...0 .... = Congestion Window Reduced (CWR): Not set
    ....0. .... = ECN-Echo: Not set
    ....0. .... = Urgent: Not set
    ....0. .... = Acknowledgment: Not set
    ....0. .... = Push: Not set
    ....0. .... = Reset: Not set
    > ....1. .... = Syn: Set
    ....0. .... = Fin: Not set
    [TCP Flags: .....S.]
  Window size value: 53270
  [Calculated window size: 53270]
  Checksum: 0x0000 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  > [Seq/ACK analysis]
  > [Timestamps]
  TCP payload (21 bytes)
  Retransmitted TCP segment data (21 bytes)
  0000 f8 32 e4 3d ef 7e ec 8e b5 fc 6f bc 08 00 45 00 -2=-...-...0...E-
  0010 00 3d d4 31 00 00 ff 06 50 a9 37 37 37 0a 03 -m-1...P:7777..
  0020 1e 6f ff dc 00 50 00 00 01 c6 00 00 00 00 50 02 -...0...0...0...
  0030 d0 16 00 00 00 00 54 75 72 6e 20 6f 6e 20 79 f2 -...Tu rn on yo
  0040 75 72 20 66 69 72 65 77 61 6c 6c                ur firew all
  
```

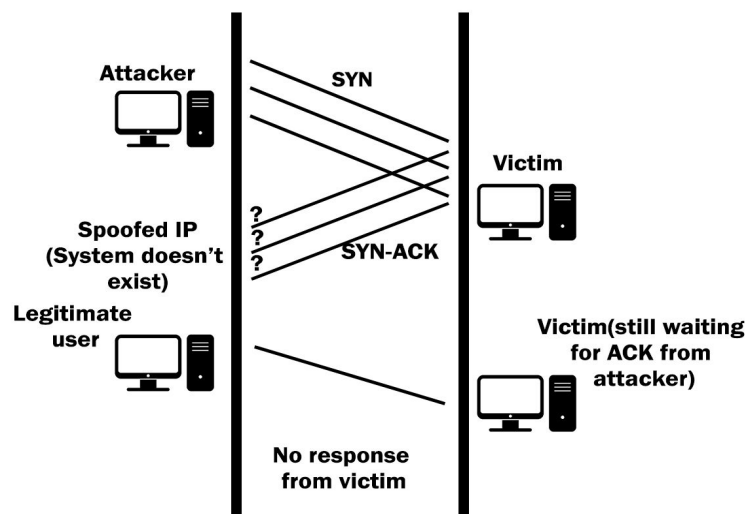
SYN flag set in each incoming packet

Since there is no ACK packet being sent, TCP Retransmission occurs which further depletes resources on victims system

Unnecessary data sent by Attacker

The attacker rapidly sends continuous SYN packets to the victim from a spoofed IP address, The victim attempts to send a SYN ACK packet back to the attacker but since its a spoofed IP address it is not received by the attacker so no ACK packet is generated and the Victim keeps waiting for an ACK reply.

Since there is no reply from the attacker the victims system requests retransmission. This process takes place for every SYN packet sent, and so it depletes the victims resources greatly



TCP RST FLOOD

Illustrations:

No.	Time	Source	Destination	Protocol	Length	Info
40	5.884881	172.231.79.124	10.3.30.56	TCP	60	1449 → 5555 [RST] Seq=1 Win=512 Len=0
41	5.885080	18.27.219.152	10.3.30.56	TCP	60	1450 → 5555 [RST] Seq=1 Win=512 Len=0
42	5.885081	252.53.79.53	10.3.30.56	TCP	60	1455 → 5555 [RST] Seq=1 Win=512 Len=0
43	5.885081	171.6.99.58	10.3.30.56	TCP	60	1456 → 5555 [RST] Seq=1 Win=512 Len=0
44	5.885103	128.15.240.65	10.3.30.56	TCP	60	1452 → 5555 [RST] Seq=1 Win=512 Len=0
45	5.885103	28.222.180.176	10.3.30.56	TCP	60	1451 → 5555 [RST] Seq=1 Win=512 Len=0
46	5.885103	68.160.139.139	10.3.30.56	TCP	60	1457 → 5555 [RST] Seq=1 Win=512 Len=0
47	5.885103	159.55.79.195	10.3.30.56	TCP	60	1453 → 5555 [RST] Seq=1 Win=512 Len=0
48	5.885103	208.126.28.141	10.3.30.56	TCP	60	1454 → 5555 [RST] Seq=1 Win=512 Len=0
49	5.886064	222.166.98.102	10.3.30.56	TCP	60	1460 → 5555 [RST] Seq=1 Win=512 Len=0
50	5.886065	134.218.64.85	10.3.30.56	TCP	60	1467 → 5555 [RST] Seq=1 Win=512 Len=0
51	5.886065	194.16.255.139	10.3.30.56	TCP	60	1476 → 5555 [RST] Seq=1 Win=512 Len=0
52	5.886065	128.144.210.139	10.3.30.56	TCP	60	1478 → 5555 [RST] Seq=1 Win=512 Len=0
53	5.886065	139.79.48.253	10.3.30.56	TCP	60	1496 → 5555 [RST] Seq=1 Win=512 Len=0
54	5.886065	48.112.30.112	10.3.30.56	TCP	60	1499 → 5555 [RST] Seq=1 Win=512 Len=0
55	5.886065	79.42.233.73	10.3.30.56	TCP	60	1500 → 5555 [RST] Seq=1 Win=512 Len=0
56	5.886066	128.23.239.113	10.3.30.56	TCP	60	1503 → 5555 [RST] Seq=1 Win=512 Len=0
57	5.886067	107.253.18.64	10.3.30.56	TCP	60	1458 → 5555 [RST] Seq=1 Win=512 Len=0
58	5.886067	249.230.27.141	10.3.30.56	TCP	60	1472 → 5555 [RST] Seq=1 Win=512 Len=0
59	5.886067	38.61.191.0	10.3.30.56	TCP	60	1462 → 5555 [RST] Seq=1 Win=512 Len=0
60	5.886067	227.201.248.78	10.3.30.56	TCP	60	1463 → 5555 [RST] Seq=1 Win=512 Len=0
61	5.886067	99.78.139.180	10.3.30.56	TCP	60	1464 → 5555 [RST] Seq=1 Win=512 Len=0
62	5.886067	64.133.181.115	10.3.30.56	TCP	60	1468 → 5555 [RST] Seq=1 Win=512 Len=0
63	5.886068	54.248.81.21	10.3.30.56	TCP	60	1474 → 5555 [RST] Seq=1 Win=512 Len=0
64	5.886068	10.49.11.26	10.3.30.56	TCP	60	1480 → 5555 [RST] Seq=1 Win=512 Len=0
65	5.886068	23.68.114.187	10.3.30.56	TCP	60	1465 → 5555 [RST] Seq=1 Win=512 Len=0
66	5.886069	39.30.128.255	10.3.30.56	TCP	60	1471 → 5555 [RST] Seq=1 Win=512 Len=0
67	5.886069	68.249.241.114	10.3.30.56	TCP	60	1473 → 5555 [RST] Seq=1 Win=512 Len=0
68	5.886069	151.159.113.165	10.3.30.56	TCP	60	1477 → 5555 [RST] Seq=1 Win=512 Len=0
69	5.886069	154.30.125.46	10.3.30.56	TCP	60	1479 → 5555 [RST] Seq=1 Win=512 Len=0
70	5.886069	11.112.190.130	10.3.30.56	TCP	60	1483 → 5555 [RST] Seq=1 Win=512 Len=0
71	5.886070	83.47.96.93	10.3.30.56	TCP	60	1482 → 5555 [RST] Seq=1 Win=512 Len=0
72	5.886070	78.113.73.230	10.3.30.56	TCP	60	1484 → 5555 [RST] Seq=1 Win=512 Len=0
73	5.886070	76.42.144.165	10.3.30.56	TCP	60	1487 → 5555 [RST] Seq=1 Win=512 Len=0
74	5.886070	82.160.190.252	10.3.30.56	TCP	60	1485 → 5555 [RST] Seq=1 Win=512 Len=0
75	5.886070	158.181.10.181	10.3.30.56	TCP	60	1488 → 5555 [RST] Seq=1 Win=512 Len=0
76	5.886071	21.251.151.217	10.3.30.56	TCP	60	1494 → 5555 [RST] Seq=1 Win=512 Len=0
77	5.886071	162.120.23.235	10.3.30.56	TCP	60	1495 → 5555 [RST] Seq=1 Win=512 Len=0

Multiple spoofed IP addresses sending packets at once

Random source port numbers sending packets to the same destination port specified by user

Flood of RST packets

In the above illustrations we can observe the following:

1. Source IP: Randomly generated IP's (Attacker uses a new spoofed IP with every new packet sent)
2. Destination IP: 10.3.30.56 (Victims IP address)
3. Protocol used to send the packets : TCP
4. There is a continuous flood of TCP packets with RST flag set
5. Source port: randomly generated port numbers for each packet sent
6. Destination port: 5555
7. Packet length: 60

```

Wireshark · Packet 47 · RST.pcapng
> Frame 47: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
> Ethernet II, Src: HewlettP_fc:6f:bc (ec:8e:b5:fc:6f:bc), Dst: CompalIn_16:8a:1d (98:28:a6:16:8a:1d)
> Internet Protocol Version 4, Src: 159.55.79.195, Dst: 10.3.30.56
▼ Transmission Control Protocol, Src Port: 1453, Dst Port: 5555, Seq: 1, Len: 0
  Source Port: 1453
  Destination Port: 5555
  [Stream index: 42]
  [TCP Segment Len: 0]
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 1 (relative sequence number)]
  > Acknowledgment number: 26645043
  0101 ... = Header Length: 20 bytes (5)
  ▼ Flags: 0x004 (RST)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    ....0... = Congestion Window Reduced (CWR): Not set
    ....0... = ECN-Echo: Not set
    ....0... = Urgent: Not set
    ....0... = Acknowledgment: Not set
    ....0... = Push: Not set
    > ....1... = Reset: Set
    ....0... = Syn: Not set
    ....0... = Fin: Not set
    [TCP Flags: .....R..]
  Window size value: 512
  [Calculated window size: 512]
  [Window size scaling factor: -1 (unknown)]
  Checksum: 0x149d [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  > [Timestamps]

0000 98 28 a6 16 8a 1d ec 8e b5 fc 6f bc 08 00 45 00  .(-.....-o...E-
0010 00 28 f9 36 00 00 40 06 6a 64 9f 37 4f c3 0a 03  -(-6-@.jd.70...
0020 1e 38 05 ad 15 b3 5c 5b 76 89 01 96 92 33 50 04  -8-...\[ v...3P-
0030 02 00 14 9d 00 00 00 00 00 00 00 00 00 00 00 00  .....

```

In this attack the attacker sends TCP packets with the RST flag set.

RESET is a flag in TCP packets to indicate that the connection is not longer working. So, if any of the two participants in a TCP connection send a packet contains such a RESET flag, the connection will be closed immediately.

Thus it can be use to attack TCP connections once the attacker can forge TCP packets from any of the two parties if he or she know their IPs, ports and the sequence number of current TCP connection. The attack can be used to make certain users to fail to use certain network services based on TCP if we know the information above.

Contributions by each student:

The main tasks needed to complete the project were researching different types of DoS attacks, understanding socket programming and the different functions of the socket module in python, writing the python script, capturing and analyzing packets in Wireshark and finally writing the project report. The four of us worked together in making this project possible taking equal responsibility in each of the tasks.

Conclusion and Future Work:

This project shows that, with a little understanding of Python and networks, a user can successfully carry out a Denial-Of-Service attack against an unprotected victim. Hence, now more than ever, there is a need for Network Security to prevent these attacks and to stay safe. We further would like to extend this project by working towards providing security by designing a Firewall that is capable of preventing such attacks from a wide range of attacking sources.

References:

Cryptography and Network Security:Principles and Practise by William Stallings

<https://www.geeksforgeeks.org/socket-programming-python/>

<https://docs.python.org/2/howto/sockets.html>

<https://www.incapsula.com/ddos/ddos-attacks.html>

<https://www.github.com/akshar-Athreya97/CN-Project>

<https://www.wikipedia.org>

<https://www.google.com>