## horizontal line



CSE-406

Optimistic TCP Ack Attack

08.09.2019

**─**

Md. Asif Tanvir

1505088

Section - B

Group - 1

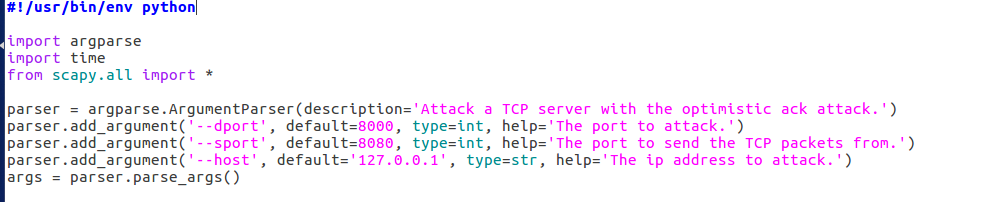
# Introduction

Optimistic TCP-Ack acknowledges a packet before it is even received. In effect, the connection’s round trip time is reduced and the total throughput increased. An attack can be initiated on this technique where attacker keeps acknowledging the sender without getting the information to make the system flooded with packets.

For **TCP ack attack**, upon receiving the first data packet, the receiver sends a stream of ACKs to the sender for data which it has not yet received. The sender, confused by these ACKs, will put more data into the network before the previous data has left the network.This will force the network to increase its bandwidth. Eventually after a certain time the network will run out of bandwidth and it will create a **Denial of Service.** It means that other clients will not be able to get access to that server.

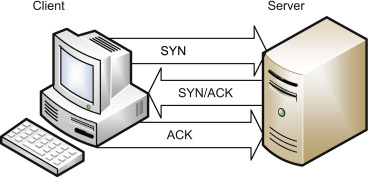
# Steps of Attack

1. For implementing this attack we need to only implement the attacker file. Here **SCAPY** and **PYTHON** is being used to create the TCP attacker client.
2. Firstly, the attacker port and host port is fixed using **argparse.** The destination port(**dport)** is set at default 8000 and the source port(**sport**) is set at default 8080. The host is also set using this **argparse** and it is set to **127.0.0.1 .** The snapshot is given below of this part.



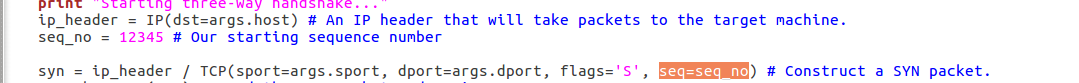
**Fig 1 - Setting the port and host**

1. Next, we need to start the TCP connection using a **three way handshaking**. The handshaking works as following-



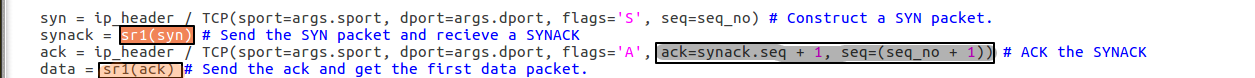
**Fig 2 - Three way Handshake**

So the attacker needs to send the first **SYN** packet. Then the server will reply with **SYN/ACK**. Each side acknowledges each other’s **sequence** number by **incrementing** it: this is the **acknowledgement** number. The use of sequence and acknowledgement numbers allows both sides to detect missing or out-of-order segments. The snapshot of this part is given below



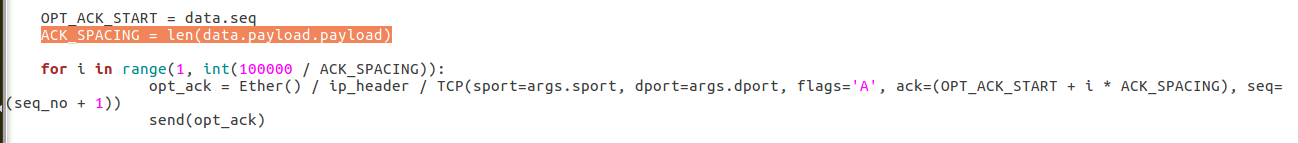
**Fig 3- Sending SYN packet**

Using the **sr1** command we can send the **SYN** packet and receive the **SYN/ACK** message from the server. For building the **ACK** message the **sequence number** is incremented so the server can know that the packet has been received. Then it is also sent using the **sr1** command of **SCAPY.** The **sr1** command does the work of both **sending** and **receiving**.

****

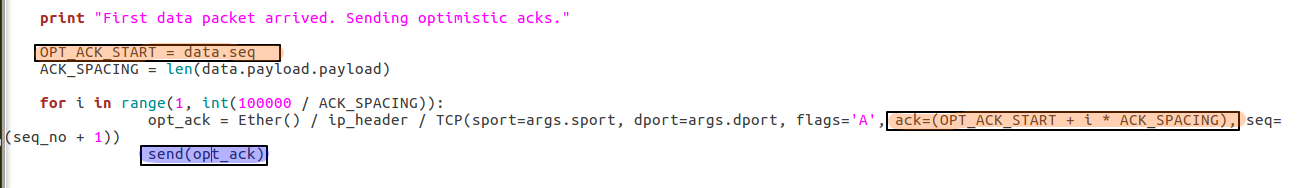
**Fig 4- Receiving the SYN/ACK and sending ACK**

1. After the **three way handshake** is done then the server starts to send data. Then the main attack starts. The attacker then starts to send continuous **ACK** whether or not it has received any data. So, a for loop is written here to send ACKs. The for loop runs for **100000 / ACK\_SPACING** times where **ACK\_SPACING** is the length of the **payload.**



**Fig 5- Starting the loop**

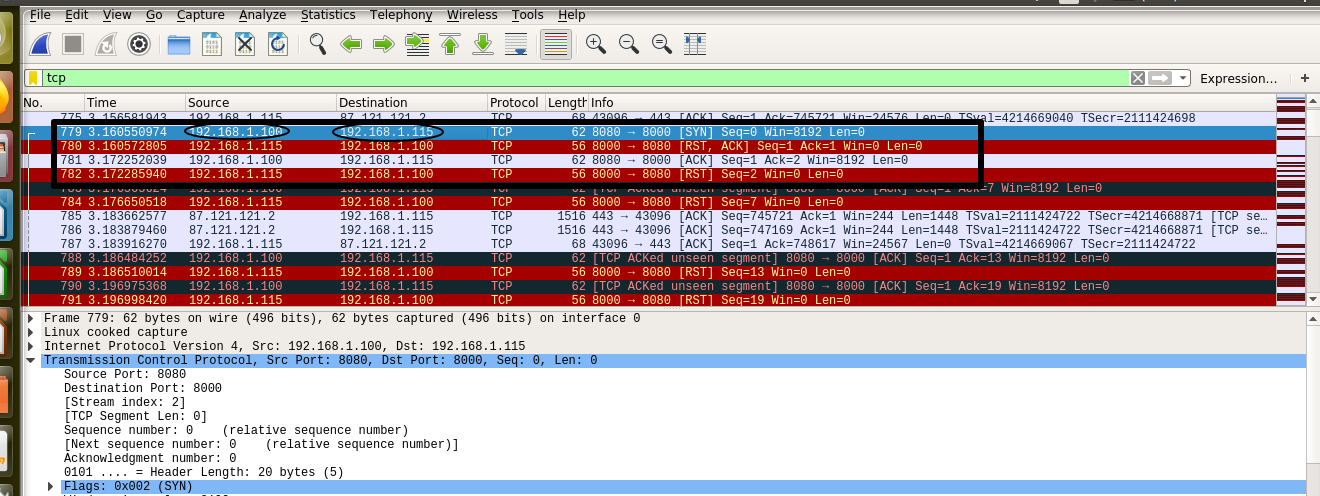
1. The main challenge is to how to send the **acknowledgement** number. We start the **acknowledgement** from the **sequence** number of the data that the server has sent. Then we increment the **acknowledgement** number by a multiple of the **ACK\_SPACING** ( length of the **payload** )**.** This is because the **TCP** protocol increases its window size if it gets **ACK**s from the client. So, if we increase the **acknowledgement** by a multiple of the **ACK\_SPACING** then the server will get **ACKs** of the same number as the **window size**. So it will think that all data upto that window is received by the receiver and it will **increase**  the window size by two times. In this way the server will soon run out of bandwidth.



**Fig 6- Sending Continuous ACKs**

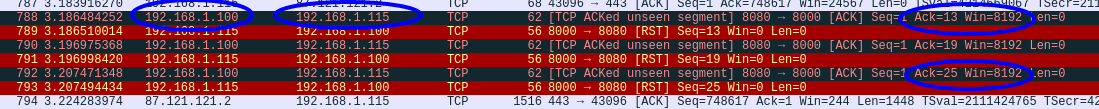
# Snapshot of the Victim

Here are the outputs from the console of the victim servers **wireshark** :



**Fig 7- Three way handshaking done**

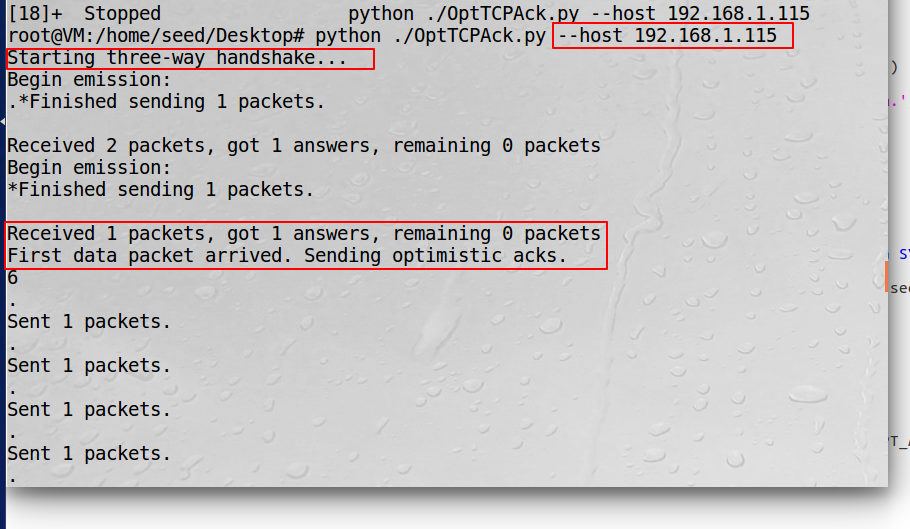
Here we can see that the **three way handshaking** is done in the marked columns.Here attacker has the IP of **192.168.1.100** and destination IP is **192.168.1.115**. After that the server starts sending data packets and then the attack starts and the server receives continuous **ACKs**.



**Fig 8- Continuous coming ACKs**

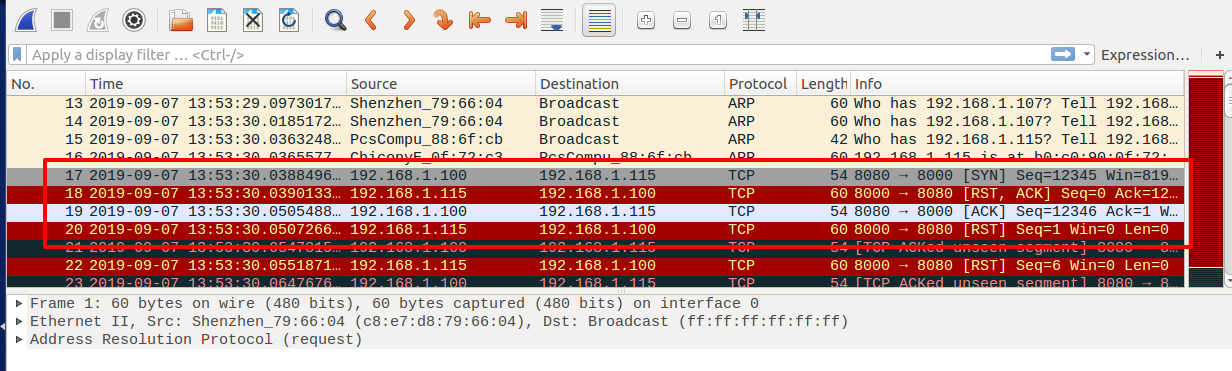
# Snapshot of the Attacker

The outputs of the **python** file that the attacker runs is given below:



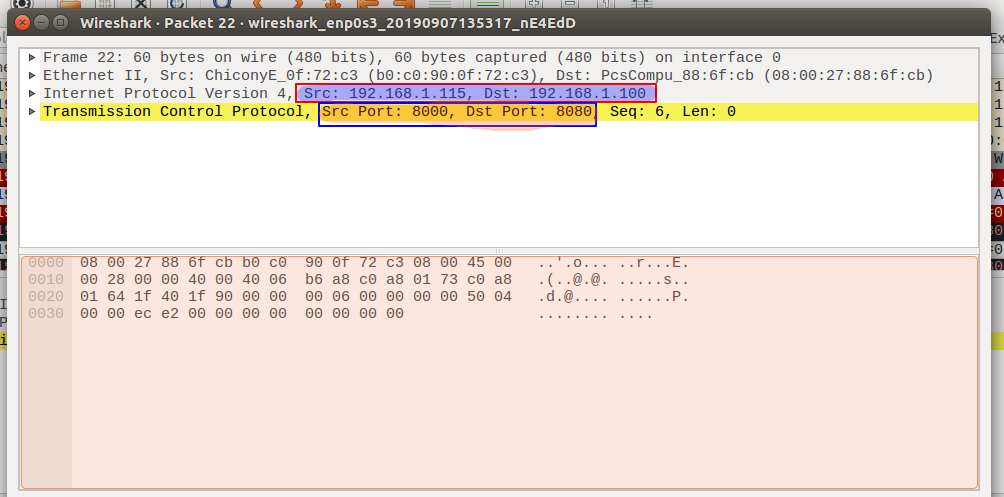
**Fig 9 - Output of the python file**

Here are the outputs from the console of the attackers **wireshark :**



**Fig 10- Output from the attackers wireshark**

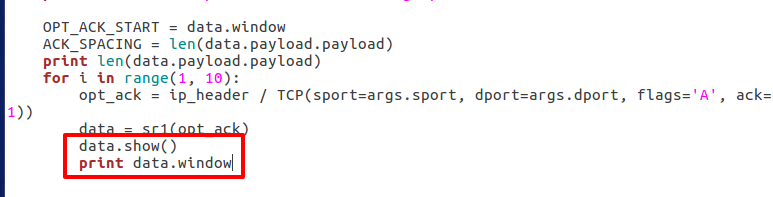
Here we can see the three way handshaking happening and the data coming from the server. The detail information of the packets looks like this:

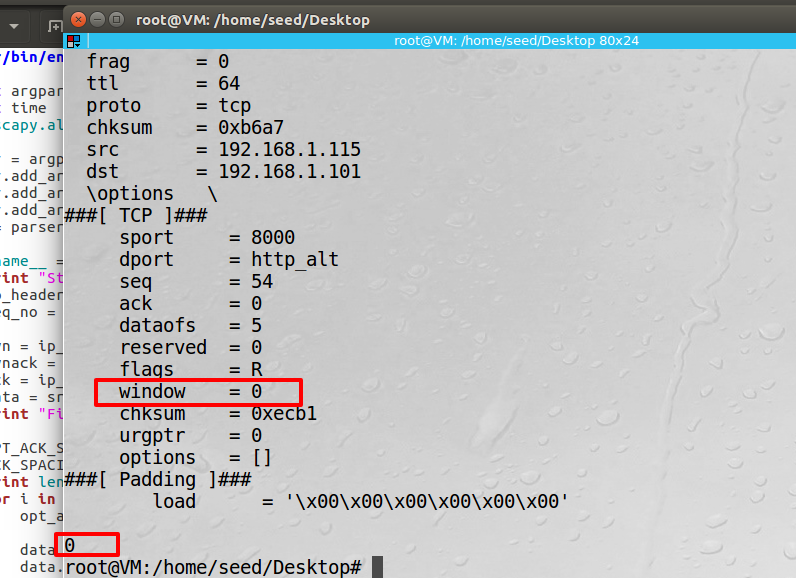


**Fig 11- Details of the packet Coming from server**

# Does the attack work?

The attack is taking place as the continuous **ACKs** are going from the attacker to the server as given in the previous pictures. But the main purpose of the attack is not fulfilled. Because the server does not extend its window and it does not run out of bandwidth. So, the attack is not complete. There can be many reasons for this attack to not work. Firstly, the major Linux distros have already released the fix. So, it is a possibility that the linux os is preventing this attack from happening. Another problem is that we are incrementing the **acknowledgement** number by a multiple of the **ACK\_SPACING** ( length of the **payload** ), while we should actually increment it by the **window size**. We are not using the **window size** here because the **window size** returning from the server has zero value in it.



Fig 12- Printing the Window size

Here we see that the window size received from the server is 0. So, we can not use the **window size**. Without knowing the **window size** it is quite hard to overflow the current window. This might be another possible reason for failure.

# Conclusion

Optimistic TCP ACK attack is mainly a **denial of service(DOS)**  attack. It can be implemented easily if the details information of the packets are available. Here the attack is not creating any denial of service because of the prevention mechanism of linux and the **zero window** problem. But building a separate python socket server and then attacking that server with this code might be successful.