CSE 406
Computer Security Sessional

Optimistic TCP ACK attack

Name: Gourab Saha
Roll no: 1605053
Lab Group: A2

Other Members:
1. 1605034
2. 1605054

Title

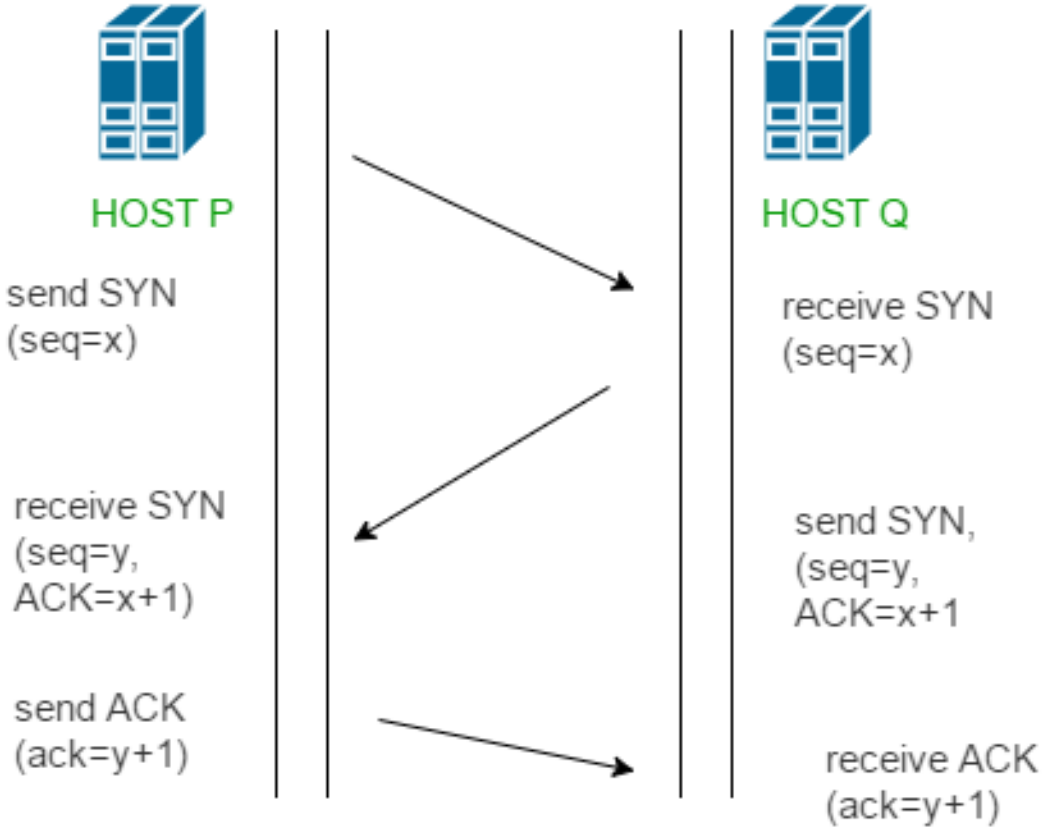
Optimistic TCP ACK attack on a streaming server

Why does this attack work?

This attack uses the idea of TCP congestion control mechanism against itself. It is a DOS attack where an attacker forcefully increases the sending rate of the server by sending ACK packets for data it has not received yet. Because of the nature of the protocol itself, the server has to reply to these ACKs. This allows the attacker to force the victim server to run out of bandwidth. This in turn makes the server unable to reply to other clients at a regular rate.

Idea of the attack

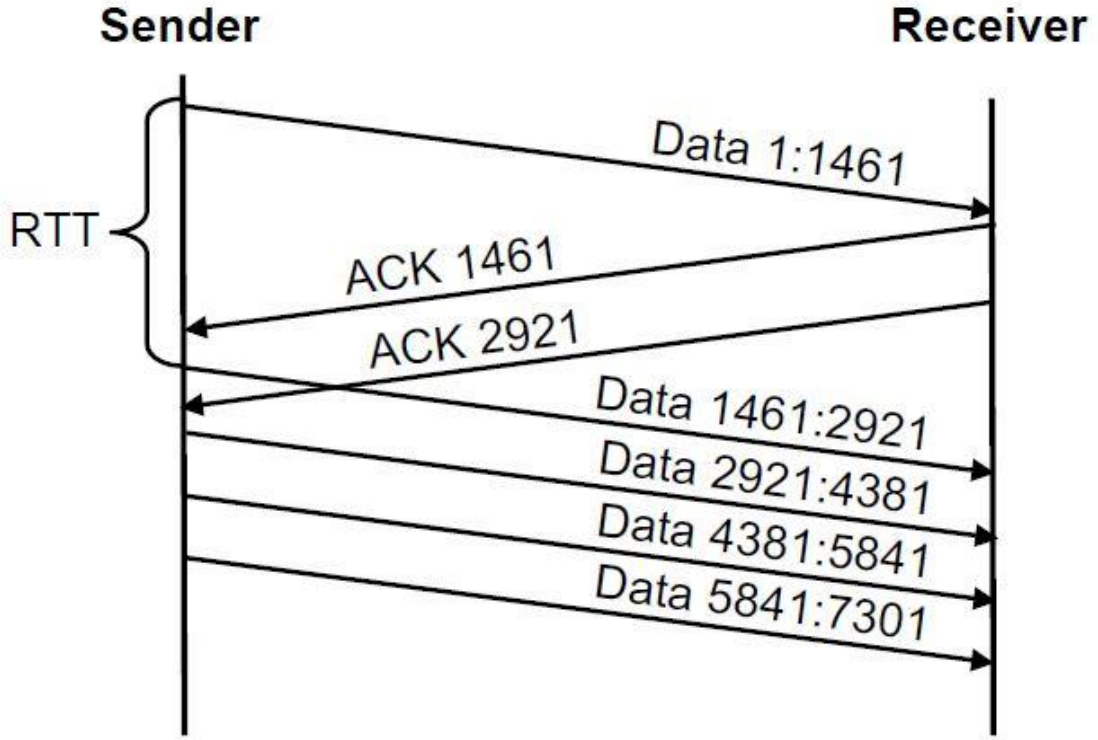
To launch a successful attack, an attacker has to connect to a server normally like any other client. So we need to implement a TCP 3 way handshake algorithm.



Here, in the image HOST P is our attacker while HOST Q is the server that is being attacked.

Once a connection has been established, it is time for attack. We need to first extract the information from the packets received from the server.

In the next pass, we send out sequential acks. As long as we set the correct ack number using the window size of the packets, the server will regard them as unique packets and thus will reply to them.



Participants:

1. Server
2. Normal client(s)
3. Attacking client(s)

Tools:

1. Operating System: Ubuntu 20.04
2. Wireshark for observing the packets

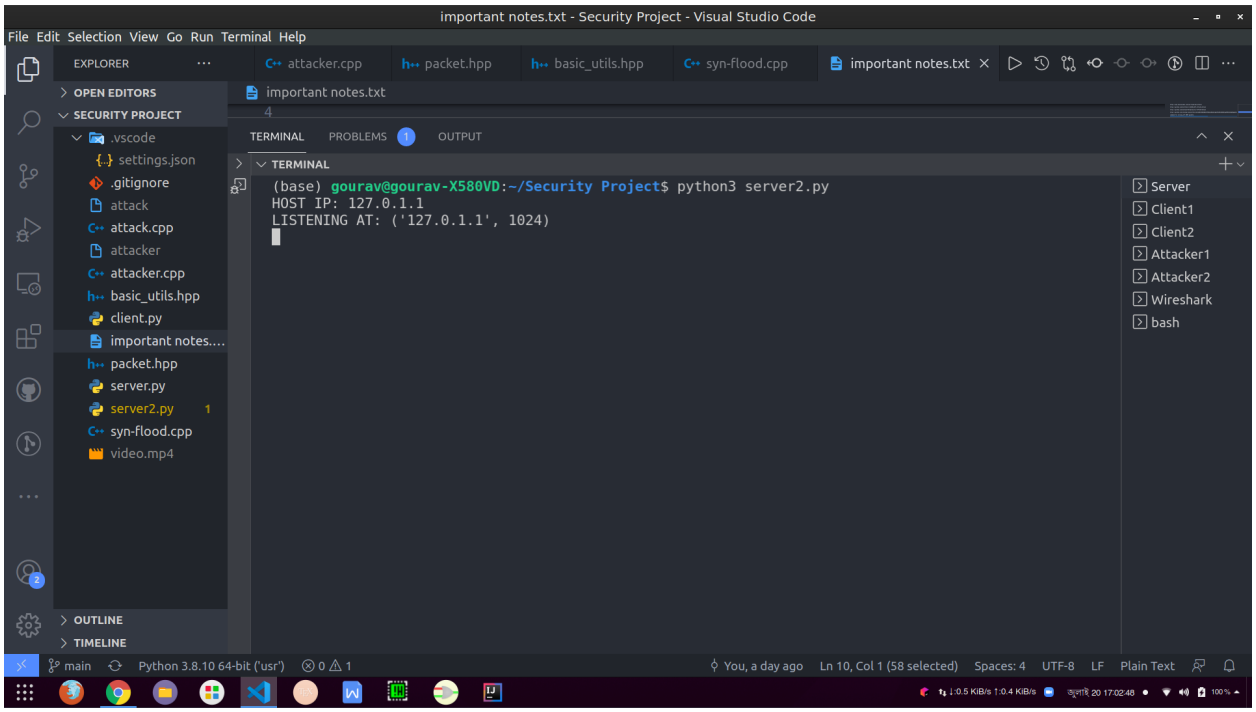
Language:

1. Server and regular client- Python
2. Attacker- C++

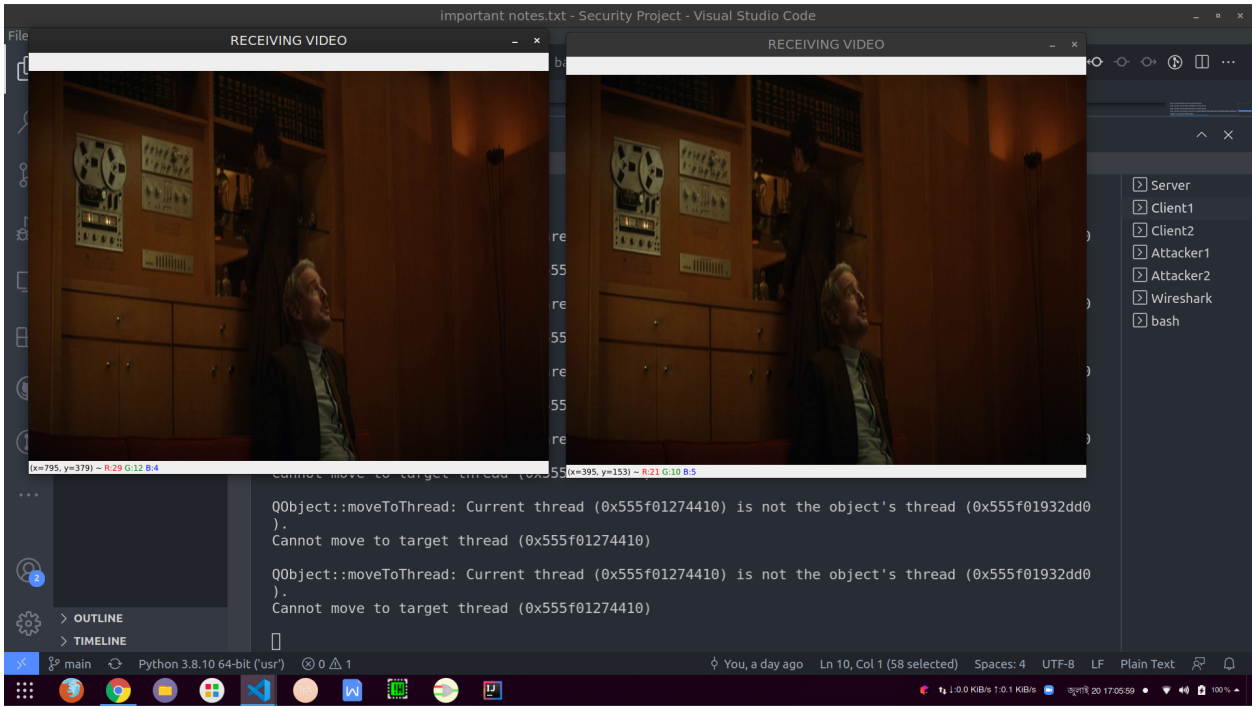
Steps of the Attack:

1. First we need to run the following command to turn off RST packets in the attacker end.
   sudo iptables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP

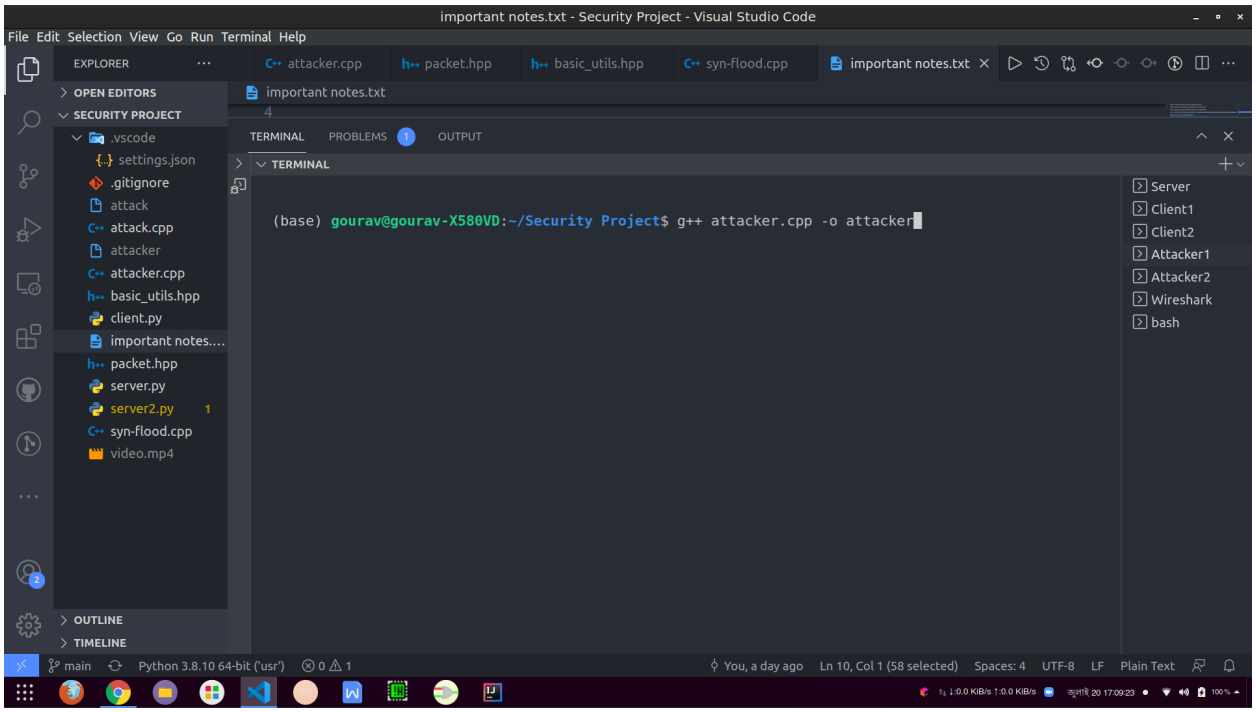
2. We use the following command to run the python server.
   python3 server2.py



3. We use the following command to wire up multiple clients. This will open multiple OpenCV windows each representing a client.
   python3 client.py

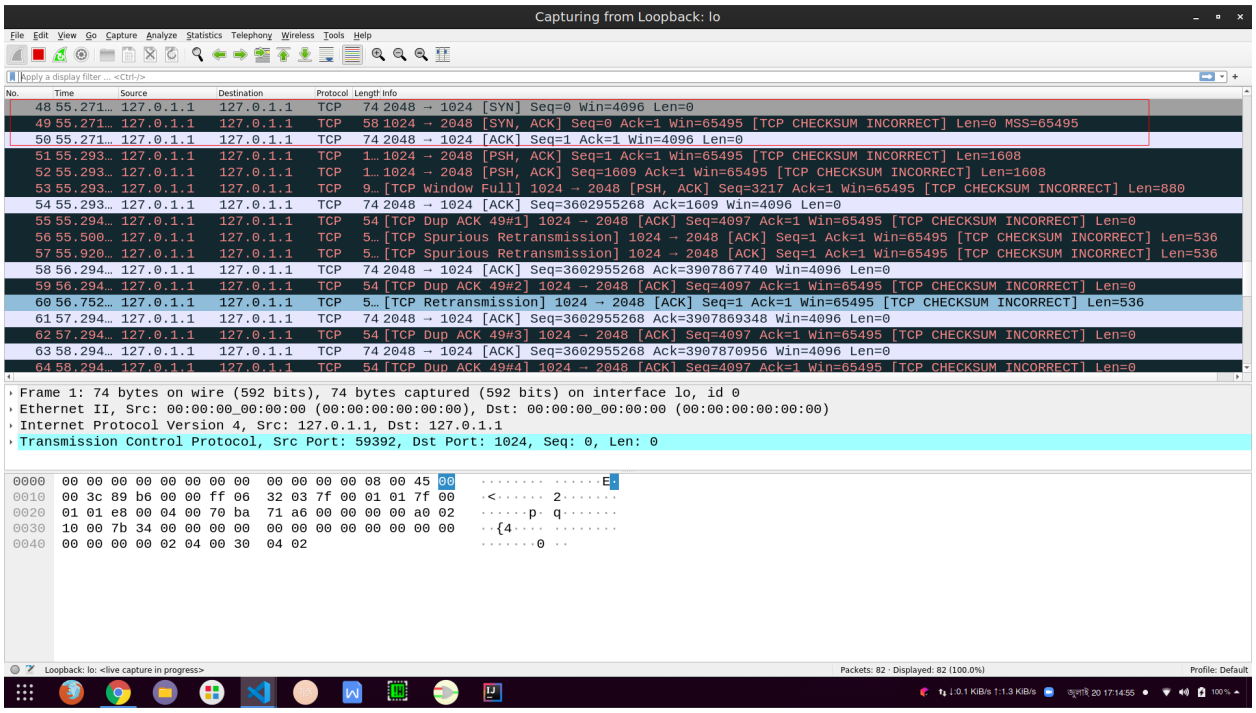


4. We use following command to compile attacker program
   g++ attacker.cpp -o attacker

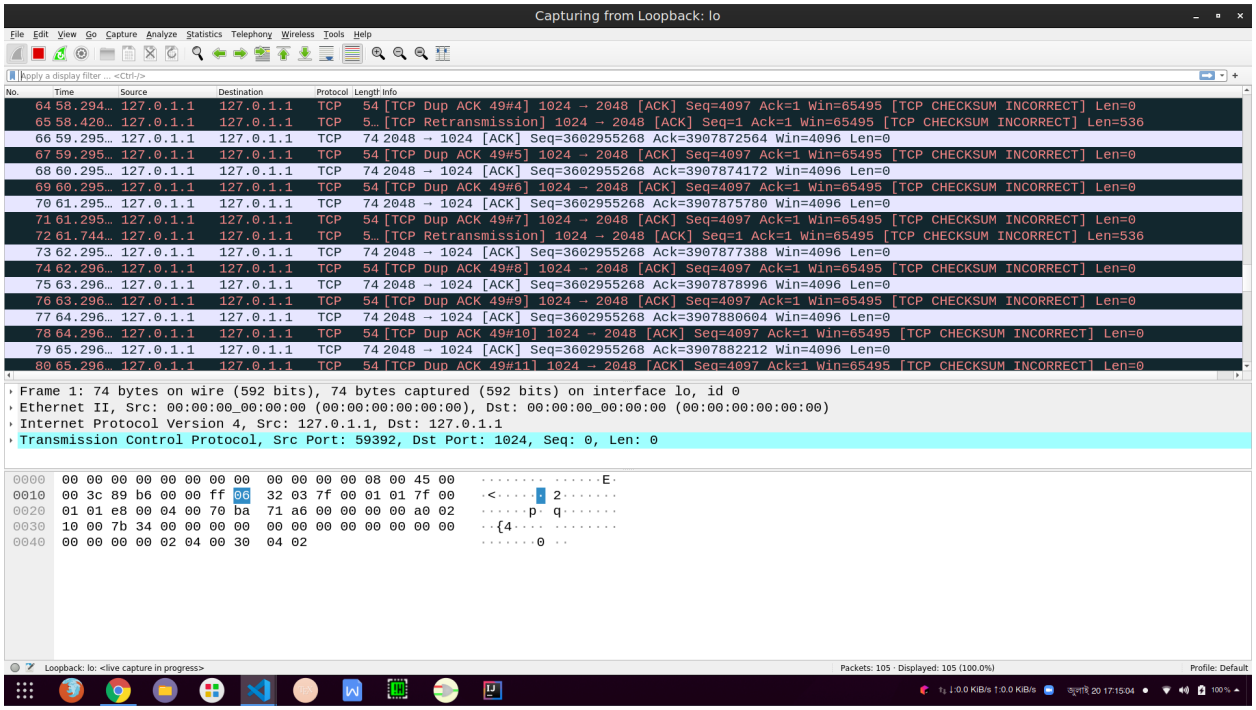


5. We run the attacker using the following command
   sudo ./attacker
   We need sudo privilege as we used TCP RAW_SOCK. RAW_SOCK requires sudo access to run.

6. We'll see the initial 3 way handshake.



7. After that the client will send a lot of errant packets and we'll see a reply from the server for all those packets.



Proof of Success:

My attack was successful. From the screenshot in step-7 we can see the server replying to the packets I sent. If we run the attack while regular clients are running, we will see a slowdown in the frame rate of the videos at the client end. This indicates that clients are receiving less data or data at a slower rate.

Problems Faced:

1. Very first problem I faced was establishing a connection between the server and my attacker. That is where I found out about RST packets.
2. The other problem I faced during establishing a 3-way handshake connection was setting the proper window size. I had to experiment with multiple window sizes before inspecting the packets to find out how a legitimate client sets window size.
3. The problem of window size came again while establishing the ACK number for the packets I sent.

4. If the server has to be shut down manually or by crash then it keeps the port engaged. Then the PORT has to be manually killed in order for that specific port to be available again. We use the following command to do so.
   sudo fuser -k PORT_NUMBER/tcp
   Or,
   sudo fuser -n tcp -k PORT_NUMBER
   However, it takes a while for some reason before fuser can kill a PORT or it shows permission denied. Sometimes, it requires a full system restart to disengage the port.
5. If I send too many packets, the system detects it as an attack and uses built in counter measures to prevent said attack.

Countermeasures:

Like many other TCP connection related attacks, Linux has built in countermeasures to prevent said attack. As such if we send too many packets, the system will just block the packets. On the other hand, if we send too little, the server will not have any noticeable difficulty in sending replies to these packets and still processing other clients as is. Real servers have way more bandwidth than the dummy server that I have used to demonstrate this attack. The built in countermeasures are not exclusive to Linux actually. Windows has it as well. Most importantly, streaming servers usually use UDP for their connections. So it would be impossible to launch a TCP ACK attack on a UDP server.

If I were to design a countermeasure for such a hypothetical situation, then my solution would be to use a heuristic to determine average connection time and use that as a standard to reject any connection that sends at rate faster than the heuristic time. This however is not ideal, as there may be users with different latencies which may

increase the connection time and then the server will reject harmless connections who just happen to be very near to the server itself.