CSE 406 Project Final Report

Optimistic TCP ACK Attack

By Using TCP socket connection for sending a Large video file (from server to client)

Lab Group: B2

Student Name: Pronob Das

Student ID: 1605098

Introduction:

An optimistic TCP ACK attack is a denial-of-service attack that makes the congestion-control mechanism of TCP work against itself.
The Transmission Control Protocol (TCP) means to provide reliable host-to-host transmission in a packet-switched computer network. In a normal case, a TCP client acknowledges (ACKs) the receipt of packets sent to it by the server. A TCP sender varies its transmission rate based on receiving ACKs of the packets it sends. The number of TCP packets allowed to be outstanding during a TCP communication session before requiring an ACK is known as a congestion window. As a server receives ACKs from a client, it dynamically adjusts the congestion window size, w, to reflect the estimated bandwidth available. By this way, the server sends packets and the client receives those and finally sends ACKs to the server.

An *optimistic* ACK is an ACK sent by a client for a data segment that it has not yet received by the client. A vulnerability exists in the potential for a client to craft optimistic ACKs timed in such a way that they correspond to legitimate packets that the sender has already injected into the network (often referred to as "in-flight" packets). As a result, the sender believes that the transfer is progressing better than it actually is and may increase the rate at which it sends packets.

So in a nutshell, a rogue client tries to make a server increase its sending rate until it runs out of bandwidth and cannot effectively serve anyone else. If performed simultaneously against many servers, this attack can also create Internet Wide congestion by overwhelming the bandwidth resources of routers between the victims and attacker.

Attack Implementation Idea:

Here I have simulated the whole idea of the attack. I have a server and a client and the server sends a large video file to the client after the client connects to it.

So a large video is going to be sent to the client but not the whole file will be sent in a single time. The large file will be divided into many smaller parts and they will be sent one by one to the client just like a normal tcp socket data sending.

In this simulation what I have done is, after receiving a packet by the client, the client will send an “ACK” message to the server.

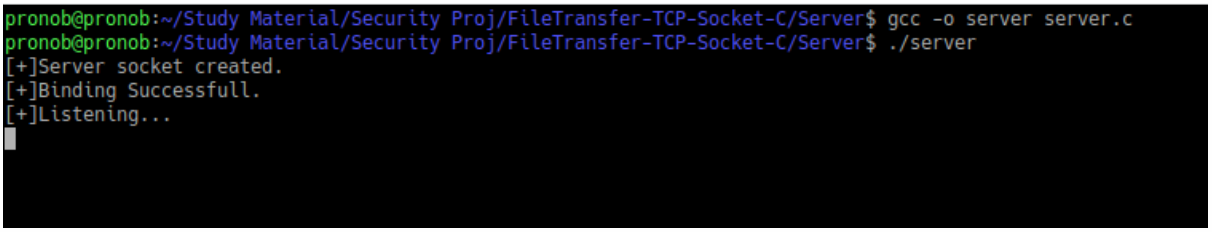
So the ideal case is, whenever the client receives a data packet, it will send an “ACK”. If somehow the client fails to send an “ACK”, the server will send at most 5 more data packets even if it doesn’t get any “ACK” message from the client. But after that, the server will stop sending data.

Now let's move to the attackers point of view. When the attacker receives a data packet, instead of sending a single “ACK” message, he will send more than one “ACK” message to the server.
From the server side, when the server sees that there are many “ACK”s stored, which is implemented by keeping a counter variable for “ACK” messages, it will speed up its transmission rate. This is implemented here by adding a sleep() function call with a certain waiting time. When there are more than 100 “ACK”s stored in the server, then the transmission speed will be upgraded.
We will observe the total amount of time to compare the transmission speed from the server to the client.

And when there are more “ACK”s than 999999, then the server will be collapsed.

Steps of Attack:

Step-1: Start the Server



Step-2: Configure the no of “ACK”s sent to the server in a single data packet receive:

```
// sending ack to the server.
char ack[4] = "ACK2";
send(sockfd, ack, sizeof(ack), 0);
```

Here “ACK2” means sending 2 “ACK”s after receiving a single data packet. Similarly “ACK1” means sending 1 “ACK”.

Step-3: Start the Client

```
pronob@pronob:~/Study Material/Security Proj/FileTransfer-TCP-Socket-C/Client$ gcc -o client client.c
pronob@pronob:~/Study Material/Security Proj/FileTransfer-TCP-Socket-C/Client$ ./client
[+]Server socket created.
[+]Connected to server.
[+]Data writing started in the file.
```

Snapshot of a Normal Case:

When the client sends 1 “ACK” message after one packet receive, the case will be as follows:

```
pronob@pronob:~/Study Material/Security Proj/FileTransfer-TCP-Socket-C/Server$ ./server
[+]Server socket created.
[+]Binding Successfull.
[+]Listening...
[+]Client connected.
[+]Data sending started.
[+]File data send successfully.
Time taken: 20.112944s
[+]Server closed.
pronob@pronob:~/Study Material/Security Proj/FileTransfer-TCP-Socket-C/Server$
```

Snapshot of an Attack:

When the client sends more than 1 “ACK” message after one packet receive, the case will be as follows:

```
pronob@pronob:~/Study Material/Security Proj/FileTransfer-TCP-Socket-C/Server$ ./server
[+]Server socket created.
[+]Binding Successfull.
[+]Listening...
[+]Client connected.
[+]Data sending started.
[+]File data send successfully.
Time taken: 19.468920s
[+]Server closed.
```

When the client sends more than 1 “ACK” message after one packet receive and when the count exceeds so much that the server gets shut down, the case will be as follows:

```
pronob@pronob:~/Study Material/Security Proj/FileTransfer-TCP-Socket-C/Server$ ./server
[+]Server socket created.
[+]Binding Successfull.
[+]Listening...
[+]Client connected.
[+]Data sending started.
Server Down.
pronob@pronob:~/Study Material/Security Proj/FileTransfer-TCP-Socket-C/Server$
```

Final Summary:

The way of implementing the attack simulation is successful according to me. When the attacker sends more than one “ACK” to the server at a time, then the process should take less time theoretically. And ultimately one time it will break down. But due to some other factors, sometimes the actual timing doesn’t reflect the ideal case.