The Complete Documentation of My Firewall – Ex4

Disclaimer: I omitted parts that I already explained in Ex 3 Documentation and only explained new mechanics I added in this assignment.

In this assignment, I had to implement a statefull filtering in addition to Ex 3 firewall implementation, and also add proxy to http and ftp connection.

**DO NOTE THAT I DIDN’T IMPLEMENTED FTP, AND ONLY HTTP IS SUPPORTED.**

Packet Extraction – To make things easier, I created a struct named packet with all the relevant information according to the packet. I used an extraction function to get all the data from the skb.

Statefull Filtering

The modulethat responsible for handling all the statefull filtering stuff is the FWConnectionDevice.

My main mechanism for Statefull filtering is using a Connection Table, that store all the connections that was created (after stateless filtering) and save a state that indicate the stage of the connection so we know what to expect from the next sender.

In the Connection Table, each entry is defined as 2 entities (one for internal, and one for external), a state that tells us what should be the next packet, and a direction to tell us who is gonna send next. The states are:

INIT - This state is the initial value for a connection created, at this point we will expect syn packet.

SYN – Syn packet already sent, so we expect syn-ack from the other side.

SYN\_ACK – both syn and syn-ack has been sent, so we expect ack to establish connection

ESTABLISHED – The connection is already established and we accept any packet from any side.

A\_SENT\_FIN – one side sent a FIN packet.

A\_FIN\_B\_ACK - one side sent a FIN packet but the other side sent an ack

A\_FIN\_B\_ACK\_FIN – both sided sent FIN and now waiting for an ack from A.

PROXY – special state that indicate that the connection is a proxy connection.

There are a lot of utility functions I used in the process of creating and maintaining the connection table, they are well documented in the code and the name is straightforward.

To filter based on the connection table I used a state machine that use the state and the direction of the packet to decide if we should accept the packet or drop it.

Proxy Connections

The modulethat responsible for handling all the proxy stuff is the FWProxyDevice.

To determine proxy connections I used a special state in the connection table called PROXY. This way if a connection is a proxy connection I can immediately find that it’s a proxy connection and route the packet to the FW, or change the sender info. The proxy is splited to 2 parts:

**Proxy packets at Pre Routing** - Those are packets that come from the client or the server. If it comes from the client, its easy to find If a connection is already exist as the client send a packet with the right information. However, when it comes from the server, the port doesn't match the one in the client connection, so we need to search the connection table by the sender (which is the server in this case) and check if the matched connection with the right dst IP (the client IP).

**Proxy packets at Local Out** - Those are packets that come from the FW itself to the Client or the Server, here we have a bigger challenge. If the Packet is sent to the client, we need to check the connection table for connection table. **We assume here that the client can only create ONE connection using this port unlike the server**.

If the packet is aimed for the server, we will use a proxy port table, that matches the proxy port to the connection.

This way we can identify connections that are a part of proxy connections, and if so we will change the routes of the packets:

Packet from the client and server will be changed to be destined to the FW.

Packet from the FW will be changed to so it looks like the sender is the original one (can be client or server).

Also, there are some utilities function that I used and they are self-explained and very documented in the code.

To use the proxy you need to run a python file named HTTPHandler in /proxy

One last thing to note is that there is a device that I used to get the proxy port from the user (when creating proxy connection from the FW to the server) to add it to the proxy port table.

**Important Functions That I Changed:**

Packet Handler – I decided to change the packet handler flow to be more accurate, clean and work better with the additions of this assignment. I also splited it, and implemented a specific function for each hook point

The Flow:

1. Extract the packet data
2. If the hook type is pre\_route call the pre\_route handler, else, call the local\_out Handler.

Pre\_Routing\_Handler – as mentioned above, I decided to make a specific function for each hook point, and as the name suggests, this function handle hooks on Pre\_Routing. The Flow of the function is:

1. Try to find if a connection in the connection table exist.
2. Check for special cases.
3. Check if the connection (if exist) is proxy connection. And if the connection is proxy connection, change the routes.
4. If the packet protocol is not TCP -> stateless filtering
5. Else, If no connection exist in the connection table AND the packet is a syn packet -> stateless filtering
6. If connection already exist -> statefull inspection using the state of the connection and the state machine.

Local\_Out\_handler – here we handler packets on local out, those are packets that the FW send (can be proxy or regular connections). Here we just check if the packet the FW sent is part of a proxy connection so we need to change the routes.