MobileProxy Milestone Three:

Author: Amanda Triplett

The packet format is set by the function setPacket(). In this function, there are four arguments which consist of an integer type, a character payload, an integer length and an integer sequence. In the type category, it can be set as type one, which is the heartbeat and type two, which is any other message. Each of the integers is allotted four characters of space, and the pointer in the character buffer is moved to account for this when assigning the values into the header. Last, the character buffer, which is the payload, is appended. For the heartbeat, this is simply the characters "hb". For the other message type, it is the body of the message. This means the header of the messages is always 12 characters long. For the heartbeat message, the sequence integer is the random number session ID generated by the function getSessionID(). This session ID should be generated uniquely between sessions. Even though the length may vary, as it is a random number, it is still an integer and thus, when the pointer points where the start of the integer is stored, it should be able to extract the entire number. For the second message type, this sequence is used to track the data packets and perform reliable transfer. Different parts of the header can be extracted by moving the pointer as described previously and this is accomplished with functions getPacketType, getsesh and getPacketMsg.

On the client and server side, when the user initiates the program. It will wait in listening mode. Once the telnet local host command is given, it sets up the socket connections between telnet and cproxy, cproxy and sproxy, and sproxy and the daemon in that order. At that point, the two programs enter the while loop controlled by select and begin sending heartbeats. The unique session ID is communicated by the client to the server in the first heartbeat sent, and the server changes the ID in its own header. The two will continue sending heartbeats until it detects the connection is lost after three unacknowledged heartbeats. This is caused by an IP address change within the same session. The client will close the connection, followed by the server. Then, because a new IP has been given to the client, it is able to reestablish the connection and the server is able to accept it. It will then exit the connection lost condition, and reenter heartbeat mode. This goal is fully functional.

For the functionality of the second section, again heartbeats will be sent until the connection is lost. At this point, the client will not be able to reestablish a connection with the server because it has no IP address. In the code, this flags a statement to execute that will return as false so that the inner while loop controlled by select on the client side can be broken. When the client's loop is broken, it returns to the outer while loop where it sets a new session ID. The server detects the change in session ID and breaks its own inner while loop. This returns both the client and the server to listening mode with the appropriate sockets closed, so that upon a new telnet local host, the user will be prompted to log-in again and start a new session. While I got this to work, unfortunately, the statement that I had return as false (a telnet bind command) causes an error if the client has its IP address restored on the same line it is removed. So unfortunately, I reverted the code to a state where it only executes the first goal. While this method worked, because it cannot work simultaneously with the first goal, it is clearly not an optimal way to accomplish this behavior. Even though I had a unique ID with the heartbeat message, it was difficult to find an appropriate condition where the client would be able to communicate the change in ID to the server. That is, a loss in connection is what triggers the behavior, so how can the client communicate the new session ID without an established connection. Additionally, without causing aberrant behavior for the other goal. I do have a

version of the code that can accomplish this goal but again, it cannot execute and also maintain the functionality of the first goal.

Last, I set up some functionality for reliable transfer by including a sequence in an argument of the setPacket() function. However, it was not able to be implemented. The plan was to store messages in an overflow buffer if a timeout occurred, and then, to check the sequence number of the first message received after reconnection against the sequence numbers of the messages in the buffer to determine which needed to be sent and which can be discarded.