COMP 416 – Computer Networks
Project #2

Oğuzhan Kadaifçiler

Naim Berk Tümer

Caner Korkmaz

Term : Fall 2018

Date : 21.11.2018

Table of Contents

EXTRACTION OF THE IP ADDRESS AND PORT NUMBERS

Step 1 : Getting the IP Address and Port Numbers as Hexadecimals

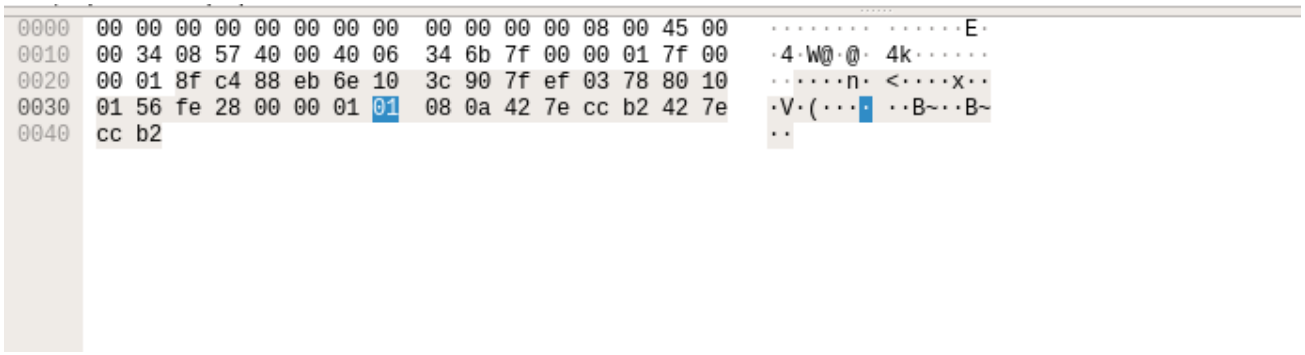

Figure 1: A sample TCP package from our server/client transfer

First of all, we needed to identify the parts of the hexadecimal that are used to identify the IP addresses and the port numbers. For this, we begin by selecting a TCP packet sent from our client to server by filtering the packets in the packet list pane to the ones including our predetermined server port using the command: “tcp.port == *server_tcp_port”*. After selecting the packet in figure 1, we selected the source and destination IP numbers and port numbers in the packet details pane of wireshark to determine the locations of the ip addresses and port numbers in the hexadecimal format. Moreover, we compare the locations shown in packets by bytes pane shown in figure 1, with our knowledge: According to our knowledge and research, the first 14 bytes of hexadecimal format is the Ethernet Header including MAC addresses of the

sender and the receiver, the next 20 bytes are the IP Header with bytes between 13 and 16 symbolizing the source ip address and 17-20 symbolizing the destination ip address, and lastly the next 20 bits are the TCP Header of which the first two bytes symbolized the Source Port and the next two bytes symbolized Destination Ports.

Step 2 : Parsing Hexadecimals in the Sniffer Program and Transforming them to Decimals

```
public static int hexadecimalToDecimal(String
input) {
    String digits = "0123456789ABCDEF";
    input = input.toUpperCase();
    int result = 0;
    for (int i = 0; i < input.length(); i++) {
        char c = input.charAt(i);
        int d = digits.indexOf(c);
        result = 16*result + d;
    }
    return result;
}
```

Figure 2: Hexadecimal to Decimal Method

After identifying the locations of the IPs and Ports in the hexadecimal stream, next step was the parse them. To parse the IPs, first the headers were separated into Ethernet, IP and TCP headers as described above. Later the sub strings were parsed from the positions described above. Then code in figure 2 was used to transform hexadecimals into decimals. Basically, the code in figure 2 multiplies every digit with 16. After this step, IP numbers and Port numbers were obtained.

EXTRACTION THE PAYLOAD FROM THE CAPTURED PACKET

We determined that payload was at the end of the hexadecimal packet format, after the 20 byte of TCP header. After parsing the TCP, extraction of payload was as simple as taking a sub-string of the hexadecimal for after the 20 byte of TCP header.

RETRIEVING THE EXCHANGED APPLICATION LAYER'S MESSAGE

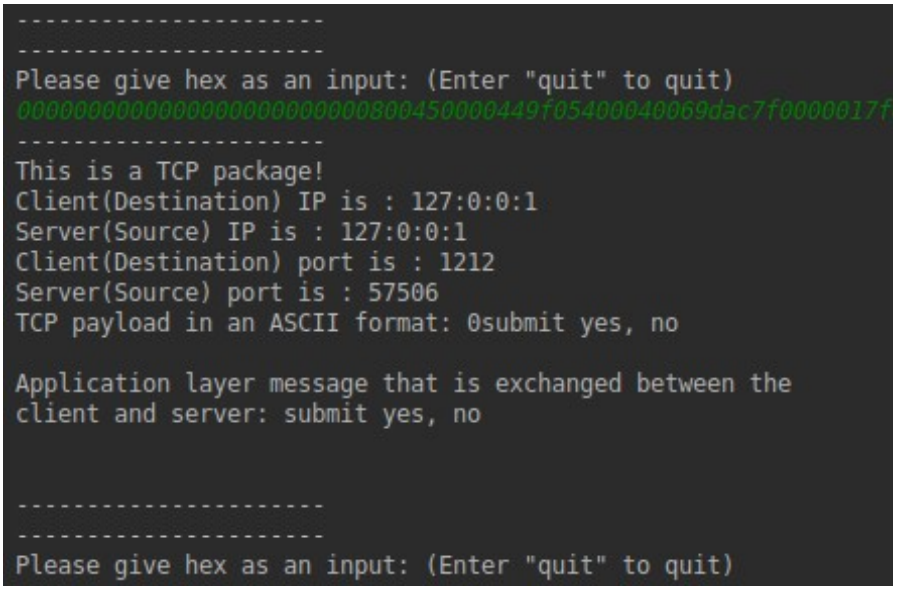

Figure 3: The output of the sniffer program for a sample package hexadecimal.

To retrieve the exchange message, our first task was to transform the hexadecimal payload into ASCII format. By first parsing the integers from the hexadecimal format and then appending the characters for those integers. After removing the “OK” message added to payload, we were left with the original message as can be observed in figure 3.

DIFFERENCES BETWEEN THE SSL AND TCP PARTS

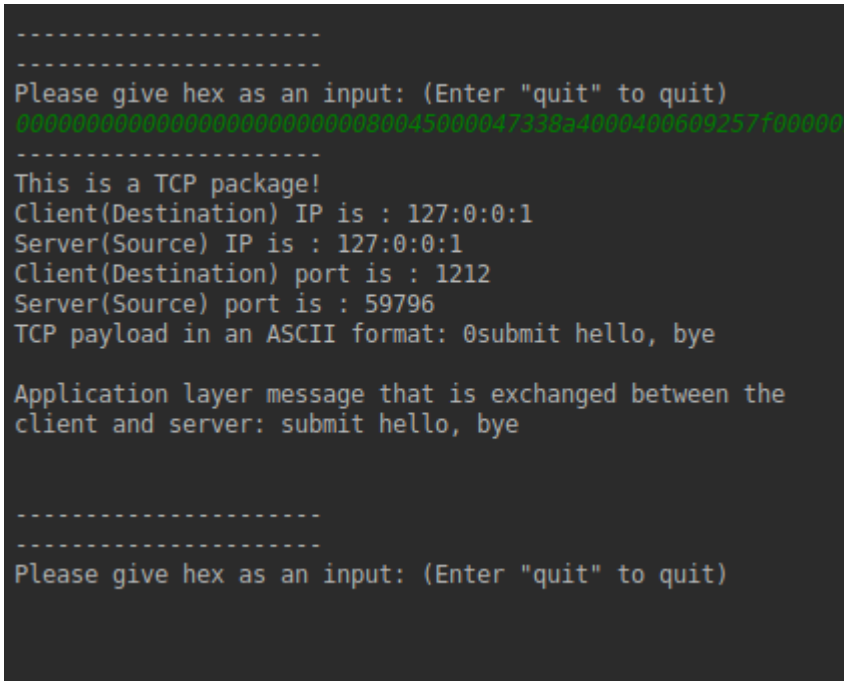

Figure 4: An example of the parts of TCP message without SSL.

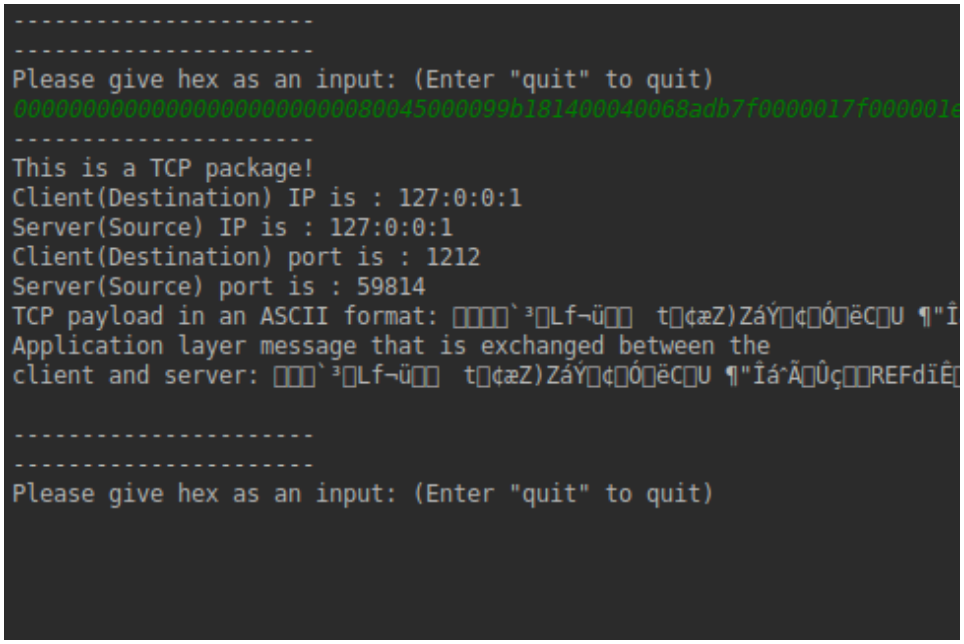

Figure 5: An example of the parts of TCP message with SSL.

As can be observed in the outputs of sniffer program in figures 4 and 5, both when the SSL is used and when it is not used the IP addressees and port numbers of destination and source and be found. However, when SSL is used, any data inside the payload is not visible. As it is encrypted and the decryption without knowing the appropriate keys is in the set of NP complete problems, retrieving the contents of the payload is only possible using the keys. What is more, the encryption of the payload and thus the message is established with the help of an another difference between TCP without SSL and with SSL: The SSL handshakes, although certainly cause a latency, enables secure transfer of the data through establishing a secure connection.

RE-TRANSMISSIONS BY TCP PROTOCOL

Figure 6: TCP Retransmission warning in info section

We observed TCP retransmissions in slow network connections when the connection speed changed a lot. The Retransmission was displayed in the info section and also the duplicate ACKs were shown in the info section. Hence we checked info section in packets list pane for the retransmissions.

INITIAL SEQUENCE NUMBER FOR TCP

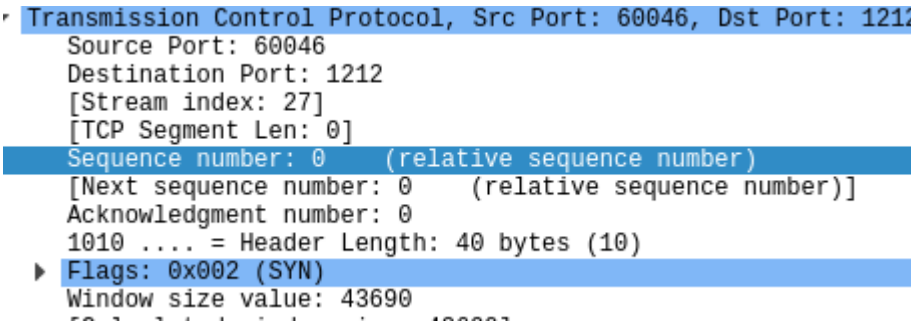

Figure 7: Sequence Number Sample

As indicated in the Wireshark interface, the sequence numbers are relatively shown. Hence the initial sequence number shown (at least in our version of wireshark) is a relative sequence number that always started with “0”.

NUMBER OF CHECKSUMS OF SSL/TCP PACKETS

We noted that there were two checksums: One checksum for the IP Header and the other Checksum for the TCP header. There are several reasons for this:

First of all, IP checksum is just a header checksum meaning that it does not include the checksum for the data. Secondly, it is possible that after parsing the TCP header from IP a corruption in the data may occur hence the TCP checksum is important for such a case. Thirdly, this is the convention that was observed to be efficient in both speed and the error detection.

BONUS PART

For the bonus part, we implemented the database using SQLite. Our implementation

```
private synchronized void initialize() {    try {        connection =
DriverManager.getConnection(JDBC_CONNECTION_STRING);    } catch (SQLException e) {
throw new RuntimeException(e);    }    try (Statement statement =
connection.createStatement()) {        statement.executeUpdate("CREATE TABLE IF NOT
EXISTS SERVER (id TEXT PRIMARY KEY, value TEXT NOT NULL)");    } catch (SQLException e)
{        e.printStackTrace();    }}
```

Figure 8: Initializing SQLite Table

of SQLite runs directly on a local storage. In addition to SQLite, we also made it possible to use concurrent hash map. As we created an abstract class named

IkeyStorage and implemented both hashmap and SQLite using this abstract class, we made it possible for server thread to use either of them.

For the SQLite implementation, the database is first initialized by creating a table called “SERVER” which can hold key, value pairs in text format as can be seen in the initialize function in figure 8. Then three functions from the abstract class were implemented: containsKey, setKey and getKey methods.

In set key method a query statement was executed with the new key, value pair and thus the new pair would be added or the old pair would be updated. In contains key, a query was executed to select the values with given key and return true if such a value exists. In the getKey method, a query was sent to retrieve the value corresponding to the given key from the database. This completes the implementation of the bonus part.

TASK DIVISIONS EXPLAINED

As suggested in the project guideline, the project was initially divided into three parts:

- Server (TCP and SSL) was implemented by Naim Berk Tümer.

- Client (TCP and SSL) was implemented by Caner Korkmaz.

- Sniffer Program and Wireshark packet captures was implemented and done by Oğuzhan Kadaifçiler.

- Each member of the team tested Client/Server and Sniffer.

- This report was written by Oğuzhan Kadaifçiler.