**SMTP Protocol Packet Crafting for Testing Network Forensics Solution**

**Introduction:**

Packet crafting is a process of manually creating of editing an existing packet to test network behaviors and to test network forensic products. In packet crafting, one creates a completely new packet or edits the existing packet to change the information packet contains. Sometimes we do not get our requirement packets on the internet or capture those packets from simulated systems can be difficult.

**High level Problem definition:**

Our main aim was to test encrypted-email (PGP and S/MIME) feature tests for our product. But the problem was that we were not able to get any such packets available on the internet. After reading rfc4880 about those encrypted emails we can get the basic packet structure and working of encryption and decryption for emails. Then we tried to set up an email server with PGP and S/MIME encryption, which required a valid certificate for public key and private key. Getting a valid certificate from a Certificate Authority (CA) is a long formal process. Then we got a great documentary on “Protected Headers for Cryptographic E-mail", where we got a clear idea of encrypted email header with body with example. Then we thought of editing unencrypted email packets with those encrypted email headers and body.

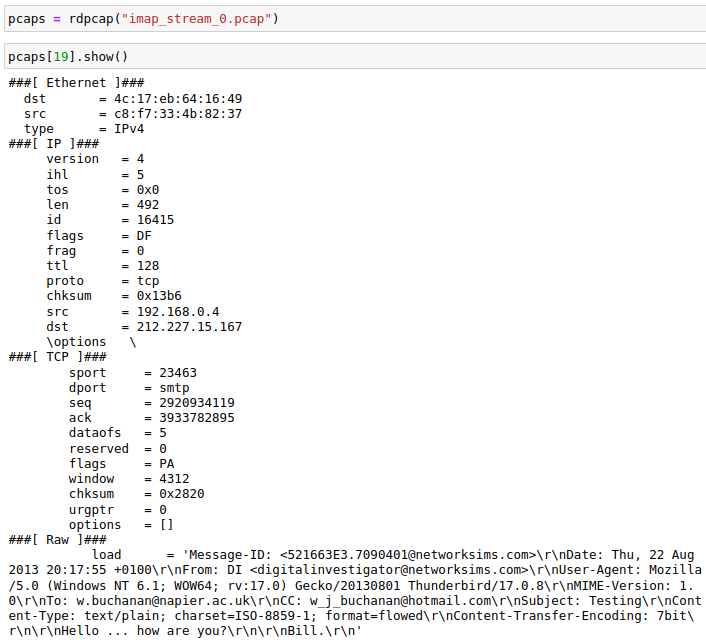
Our main aim was to capture/create packets for encrypted email (PGP and S/MIME). To send encrypted emails, we need to have an email server that has a valid certificate (CA certified) and support for sending PGP and S/MIME encrypted email. Setting up this type of server is non-viable as we need to buy a valid CA certificate. After reading some documentation about the difference between simple SMTP and encrypted email packet payload structures, we decided to craft simple SMTP packets to get encrypted packets.

**High Level Sketch of the Solution:**

For editing we choose to work with SMTP non-fragmented email packet. Which contain multiple packets of client and server communication. In those packets to replace unencrypted email with encrypted email, we needed to change 3-4 packets with payloads which are “MAIL FROM:”, “RCPT TO:” and actual mail with email-header.

**Detailed Description and Analysis of the Solution:**

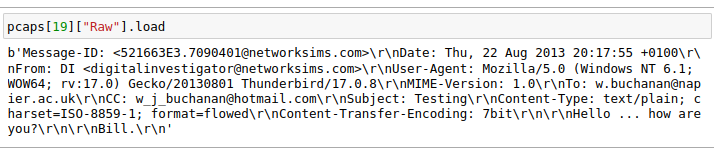
For editing payload of existing packet, we have used Scapy library of python, as python is a very high-level language and extremely easy to write code. In scapy, packets in one pcap file are represented as a list of packets which can be accessed by index. We can read pcap file using **rdpcap** function and write pcap file using **wrpcap** function. Scapy has **show()** function to view formatted packet.



In this output we can view and edit all layer's information except data-link layer.

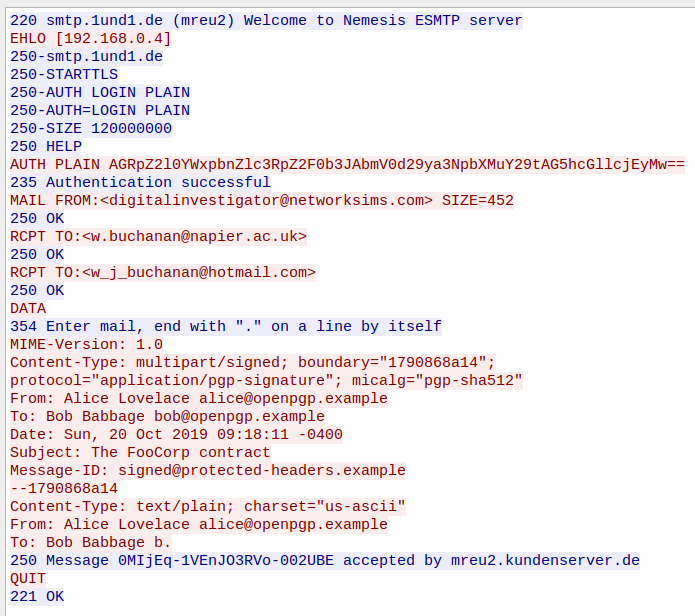
1. **Payloads replace:**

In Scapy we can access payload using **pcaps[index][“Raw”].load,** which takes string and bytes type of values. Before replacing payload, we need to replace every newline with “\r\n”.



1. **IP Header Length:**

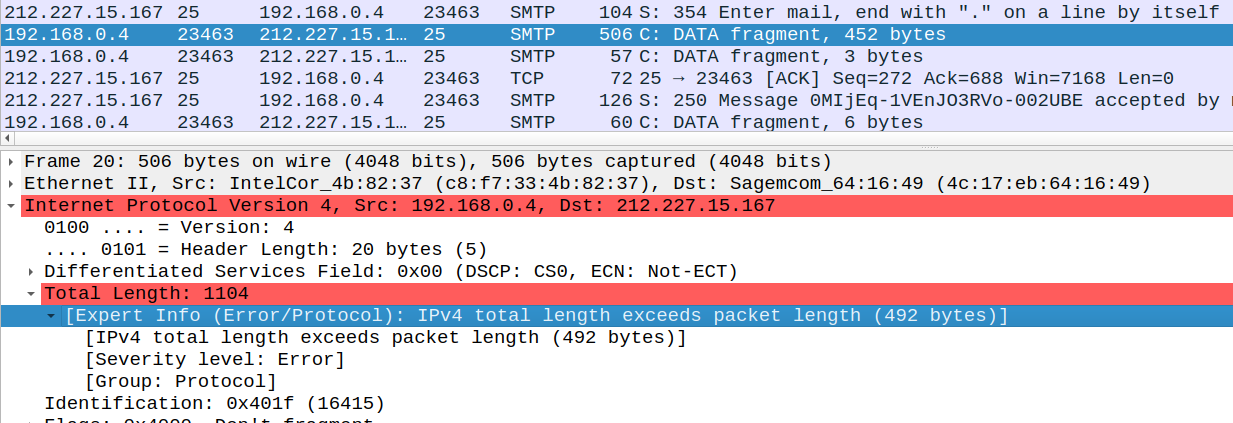
After editing/replacing payload. Now if we follow TCP Stream, we will see cropped out message that is because we need to change IP Header length.



We need to rectify IP Header length, which we will get from **len(pcaps[index][“IP”]).** We can directly replace IP header length value with new length of all the changed packets using **pcaps[index][“IP”].len = len(pcaps[index][“IP”]).**

1. **Frame length:**

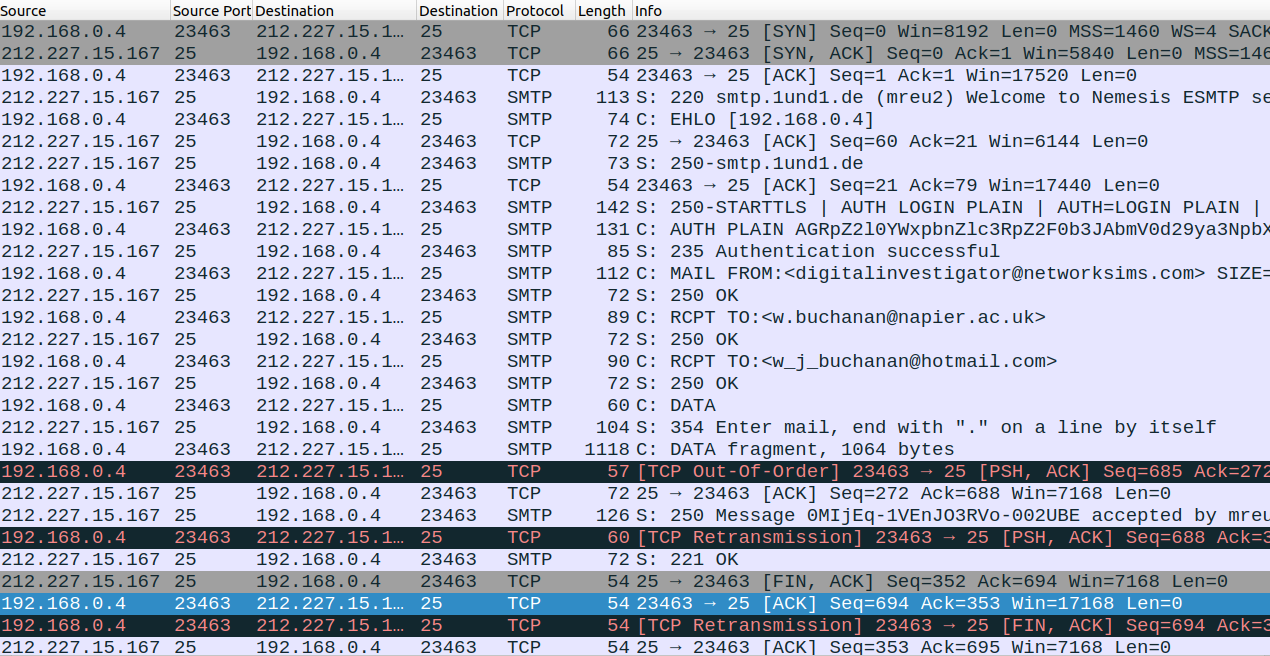
Now if we view the pcap file using Wireshark we will encounter a “total length exceeds packet length”.

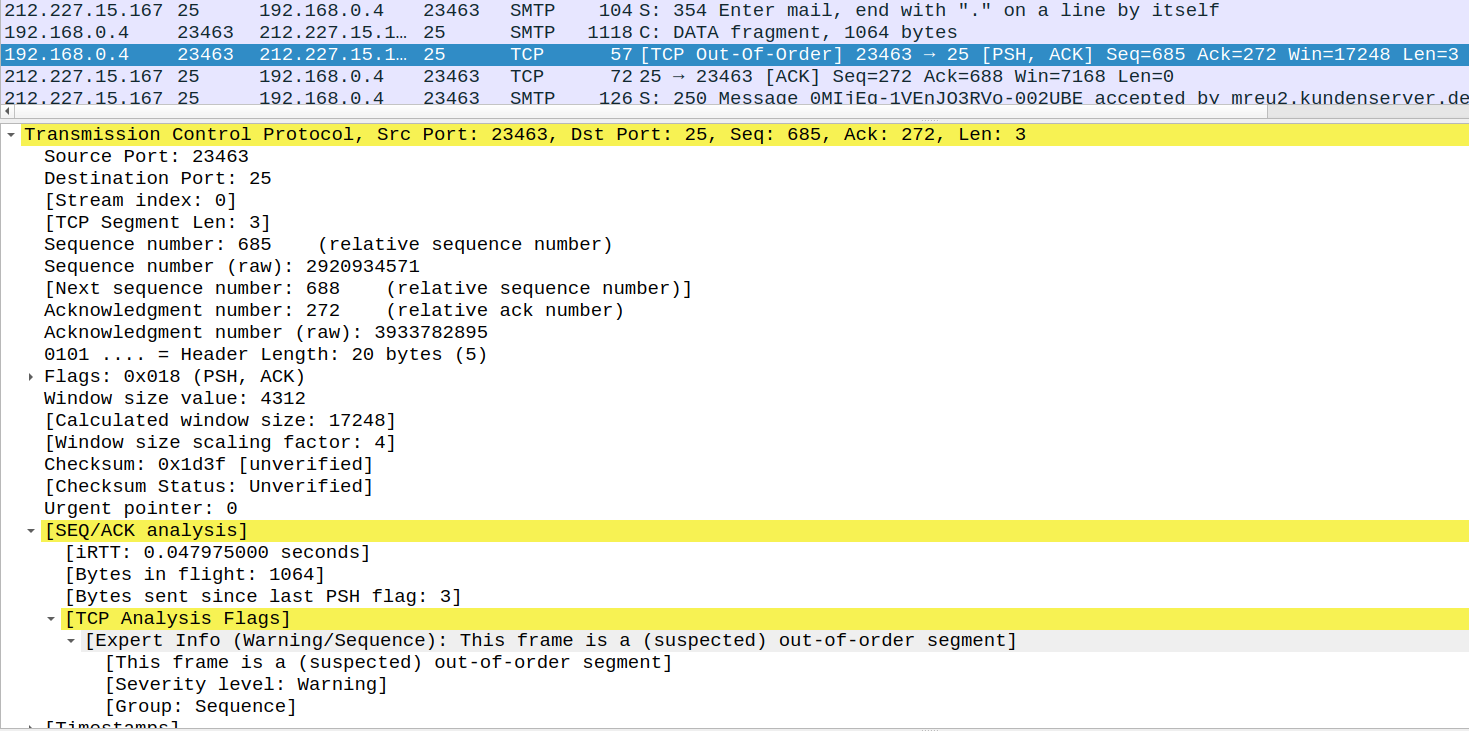


This means that the specified packet length in **Frame length** is less than total packet length. So, now we need to change **Frame length.** Scapy does not provide the option to change **Frame Length.** From Scapy we can get total length of the packet using **len(pcaps[index])** and convert that in hexadecimal. The last four bytes of data-link layer are for Frame length after that Ethernet layer starts and can view those hexadecimal values start bytes in Wireshark. Using Ghex application we can edit Frame length value to hexadecimal value of packet length.

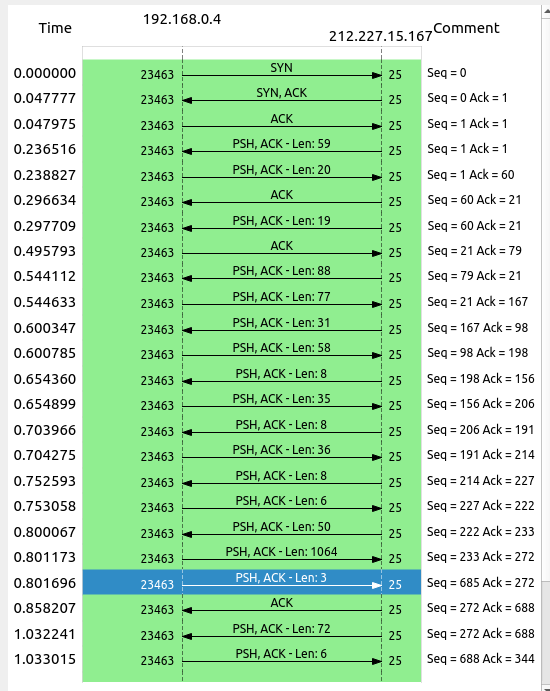
1. **Sequence and Acknowledgement numbers:**

Now if we view the packet with Wireshark, we will encounter with “TCP Out-Of-Order" error and all the packets with same sequence series will show error of “TCP Retransmission”.



This is coming as TCP sequence number not matching the expected data length.

The client of TCP keeps track of the amount of data sent (payload size) on each packet. This sequence number is included on each transmitted packet and acknowledged by the opposite host as an acknowledgement number to inform the sending host that the transmitted data was received successfully. Verified sequence and acknowledgement number from Wireshark by navigating **Statistics -> Flow Graph** and then select **TCP flow** and click **Ok**. Wireshark shows graphical view of the TCP sequence and acknowledgement number with zero initialization.

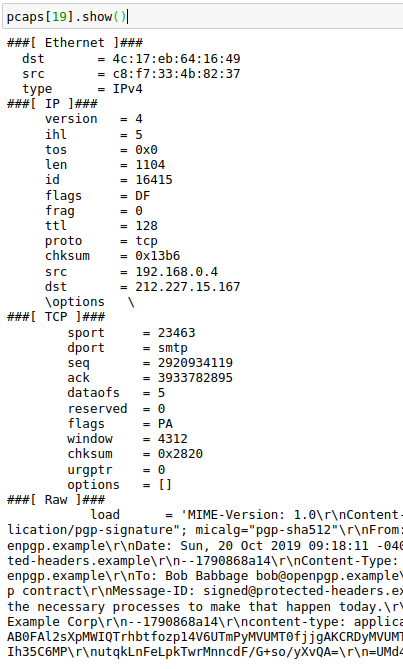
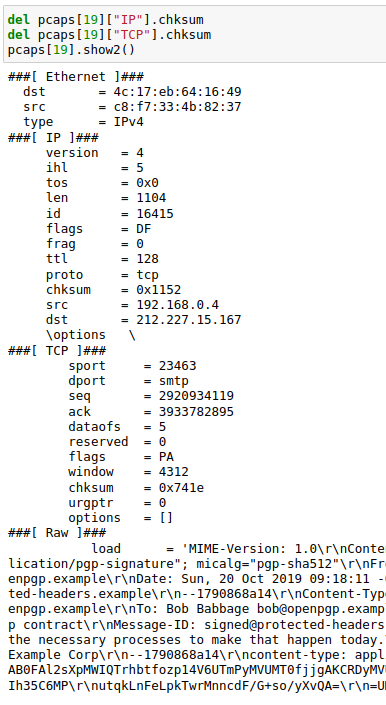


The increase of the sequence series in the next packet will be same as current packet’s payload length. Here, in this example showing in blue the sequence number from the previous packet from 233 and payload length of 1064 should be (233+1064) = 1297, but here it is 685. So, we need to increase that sequence for all the next packets.

Using Scapy we have done that increment programmatically, and then viewed that capture by Wireshark.

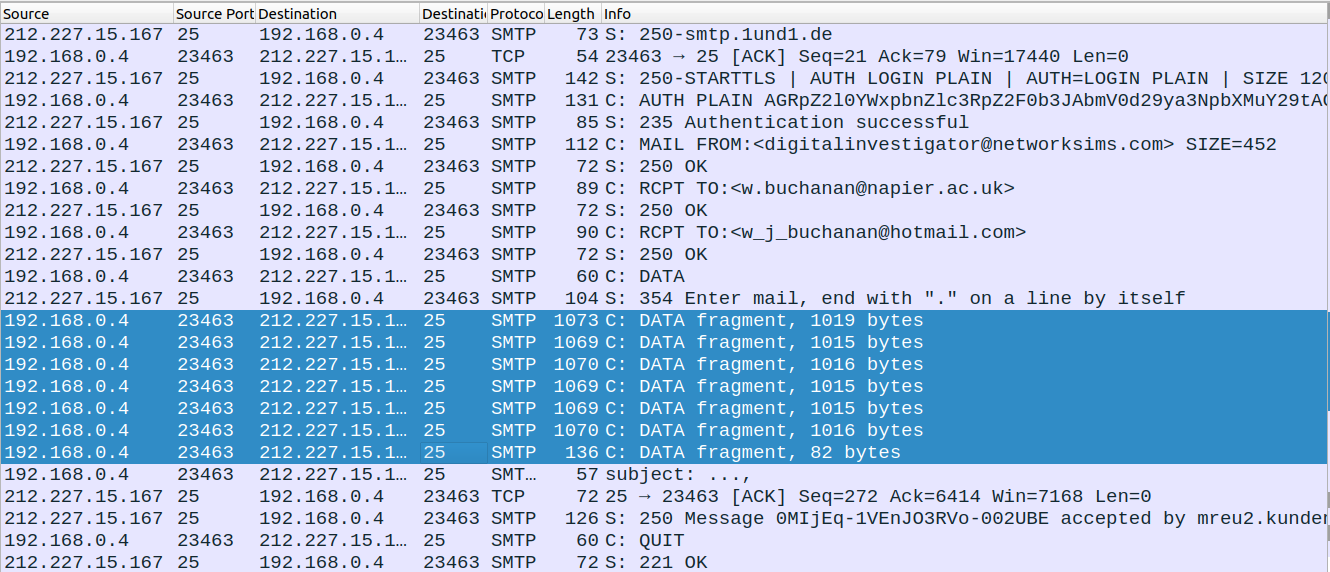
1. **Rectifying IP and TCP checksum:**

As we are changing headers data the IP and TCP checksum will become wrong. Scapy provides the option to recalculate those checksums. If we delete IP and TCP checksum of every packet, then scapy will automatically recalculate checksum while saving and we can view calculated checksum using **show2()** function. Below figures are shown before correcting checksum and after correcting checksum.

1. **Fragmented payload:**

Every SMTP server may have a fixed upper limit on message size. Any attempt by a client to transfer a message which is larger than that fixed upper limit will fail. For the message size limit constrain we have made fragmented packets if the message size is more than approximately 1000 bytes. For fragmented payload we need to add multiple packets as per message size and we need to rectify the above stated parameters and Identification number of IP layer.

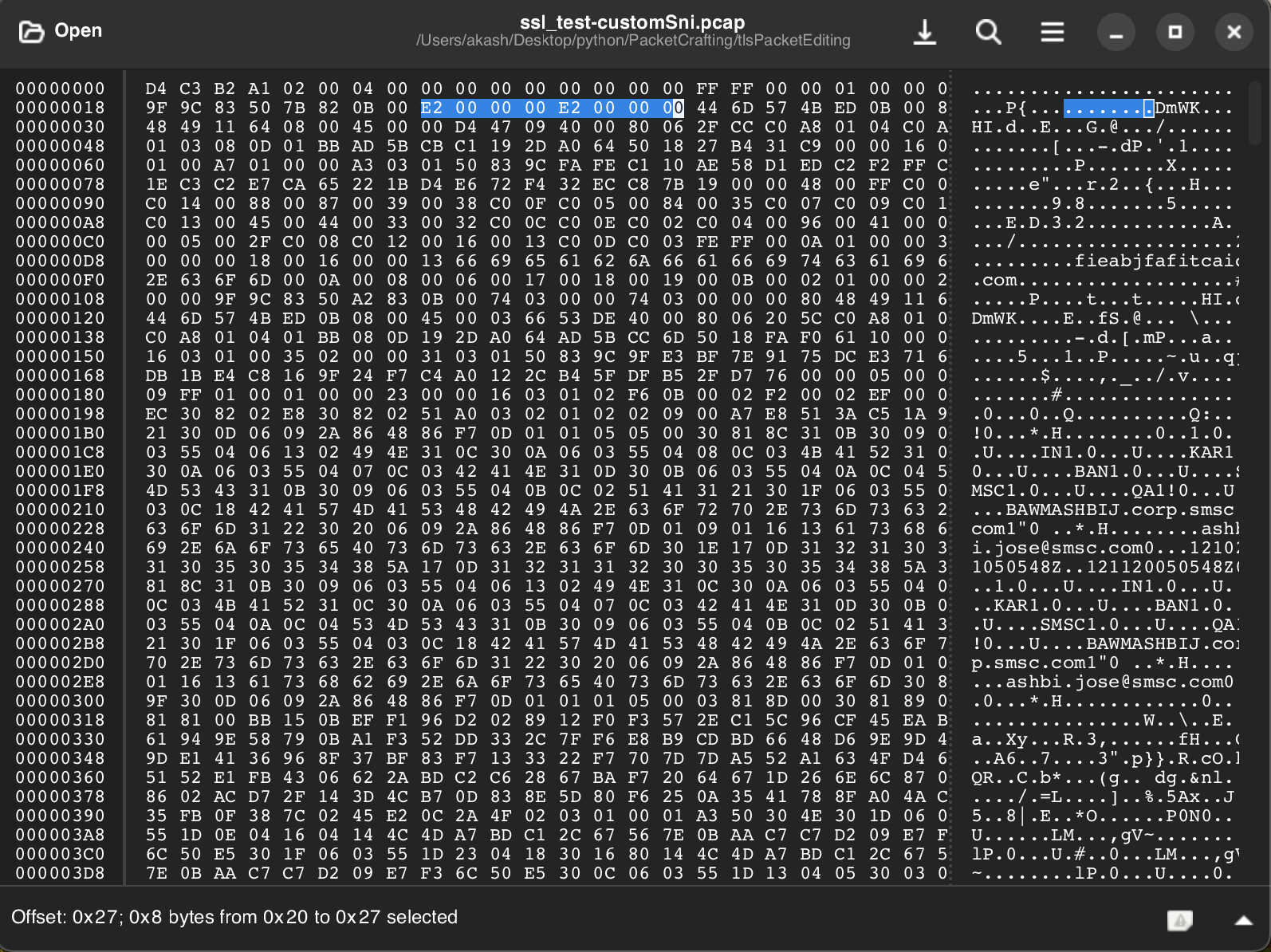
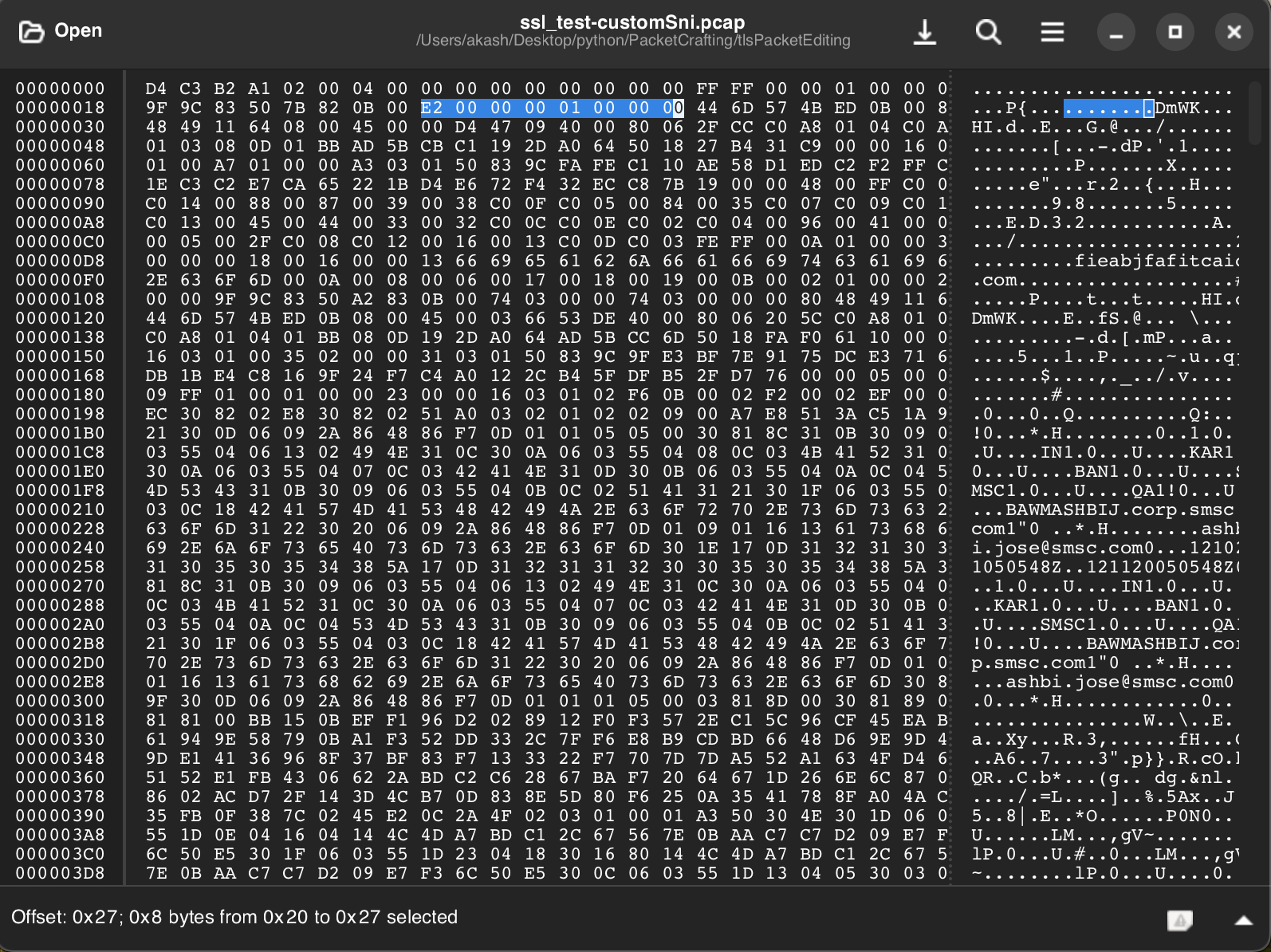


1. **IP Identification number:**

IP identification numbers are used for packet reassembly and should be unique within the pcap. In this case, we have two series of IP Identification numbers. In the fragmented packets, we are incrementing the series by one.

1. **Frame layer correction:**

Scapy does not have the option to correct the frame layer for the edited packets, for this we will get error for the Frame layer while viewing .pcap file on the Wireshark. We can edit the frame layer using “GHex” editor.



**Conclusion:**

Packet crafting is a good way to audit network security and exploit vulnerably. This document gives a detailed description of Packet editing from an existing packet by using one of powerful Packet crafting tool Scapy. Here we have discussed editing SMTP protocol, but we can do this for email protocols (POP3, IMAP), HTTP and other protocols.

**References:**

1. [Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification (rfc8551).](https://datatracker.ietf.org/doc/html/rfc8551)
2. [OpenPGP Message Format (rfc4880).](https://datatracker.ietf.org/doc/html/rfc4880)
3. [Protected Headers for Cryptographic E-mail](https://tools.ietf.org/id/draft-autocrypt-lamps-protected-headers-01.html).[Bjarni Rúnar Einarsson, Daniel Kahn Gillmor.](https://tools.ietf.org/id/draft-autocrypt-lamps-protected-headers-01.html)
4. [**https://github.com/autocrypt/protected-headers**](https://github.com/autocrypt/protected-headers)**.**
5. [Test Vectors for E-mail Header Protection](https://header-protection.cmrg.net/).
6. [TCP Sequence and Acknowledgement Numbers Explained](https://madpackets.com/2018/04/25/tcp-sequence-and-acknowledgement-numbers-explained/).
7. [Understanding TCP Sequence and Acknowledgment Numbers.](https://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/)
8. [How to Inject Code into HTTP Responses in the Network in Python (Scapy).](https://www.thepythoncode.com/article/injecting-code-to-html-in-a-network-scapy-python)