School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg
Private Bag 3, 2050, Johannesburg, South Africa

ELEN4017: Project - 2017

Networks: Email Application System

May 2017

Benjamin Thomas (Ben)
545787
545787@students.wits.ac.za

Benjamin Shear (Benji)
749992
749992@students.wits.ac.za

Abstract

In this project the features of an email application were implemented. An email system comprised of an IMAP (Internet Message Access Protocol) client and server, POP3 (Post Office Protocol 3) client and server, and SMTP (Simple Mail Transfer Protocol) client and server was developed. The system also contains a Graphical User Interface (GUI) for some of the functionality as well as demonstration scripts to automate the system. The report will discuss the structure, features, response and replies, code and critical analysis of the system.

Introduction

Email plays a vital role in the day to day functioning of modern society. As one of the building blocks of the early internet, email has remained surprisingly relevant today as ever - despite the introduction of messaging technologies such as SMS, Whatsapp, Telegram and iMessage. Therefore, this project involves the investigation, design and implementation of an emailing system based on the robust protocols that define Email communication.

Implemented System and Overview of Features

The system implemented consists of an email client and email server. The system design was guided by the Request for Comments (RFC) publications as per the Internet Engineering Task Force and the Internet Society. The client side allows for communicating with an email IMAP server by the TCP protocol at the transport layer and using the IMAP protocol at the application level. The IMAP protocol was partially according to RFC 3501. The system also allows for interacting with an SMTP server to send email via the SMTP protocol. The SMTP server is a multithreaded server allowing for it to communicate with multiple client TCP connections simultaneously. The SMTP sever has the ability to relay any emails sent to it to another SMTP server. The SMTP design was partially according to RFC 5321. A multithreaded IMAP server was implemented which has the ability to interpret commands sent to it and respond according to the IMAP protocol. With POP3, a similair structure as IMAP with regards to the client and server was deployed. However, at the time of submission the code has not been fully tested and is therefore meant to mainly give an idea to the structure of the POP3 system as opposed to an actual implementation. The POP3 design was partially according to RFC 1393.

The system makes use of a simple GUI to demonstrate some of the functionality and help the user easily navigate through the application. To further demonstrate the workings of the servers and clients a number of demonstration scripts have been created. These scripts, with minimum modification can run client/servers to demonstrate certain key functionality.

Missed features

The application is able to effectively send and receive emails, however the application is not able to handle email attachments and only is able to handle text in the body of the email. Additionally many of the commands and specifications for formatting as specified in the RFC were not implemented. In most cases the bare functionality needed to perform basic email client and server tasks were implemented. In the case of the IMAP and POP3 servers the server only replies to a limited number of requests and does not have the ability for real email to be transferred onto these servers. Rather, mock accounts and emails have been set up in the code for demonstrative purposes.

Protocols

Below are the client commands and server response codes that have been implemented in the code.

SMTP

SMTP Client Commands

- **EHLO localhost** - Sends the EHLO command as per SMTP protocol.
- **AUTH LOGIN** - Sends the request to login. The server then responds, requesting client to send username and password.
- **MAIL FROM: [argument]** - Requests the senders mail ID.
- **RCPT TO: [argument]** - Requests the receivers mail ID.
- **DATA** - Requests the data to be sent. The server interprets a full-stop on an empty line as the token that informs the

server when the client has finished sending data.

- **NOOP** - Requests for the server to reply with a '250 OK'.
- **VRFY [argument]** - Verifies if a user exists on the server.
- **RSET** - Abort current mail transaction.
- **QUIT** - Terminate session with server.

SMTP Server Replies

- **250 Hello, I am delighted to meet you** - Sent upon receiving EHLO from client
- **250 Ok** - sent in reply to the following cases: MAIL FROM: [argument],RCPT TO: [argument], after an email has been relayed.
- **354 End data with <CR ><LF>.<CR><LF>** - Sent in response to DATA command.
- **221 Bye** - Sent in response to QUIT command from client [1].

POP3

POP3 Client Commands

- **USER [argument]** - This allows the user to pass their user-name to the POP3 server for authentication
- **PASS [argument]** - This allows the user to pass their password to the POP3 server for authentication against the user-name already entered
- **STAT** - This allows the user to view how many emails are in the inbox as well as the size (number of bytes) of the inbox
- **LIST [argument]** - This allows the user to view the size of a specific email. If no argument is passed, the server returns the size of each email in the inbox.
- **RETR [argument]** - This allows the user to retrieve a specific email corresponding to the argument passed in.
- **NOOP** - This does nothing.
- **DELE [argument]** - This marks a user specified email for deletion.
- **RSET** - This clears all emails that have been marked for deletion.
- **QUIT** - This closes the connection.

POP3 Server Replies

- **+OK Send PASS** - In reply to USER [argument]
- **+OK Welcome** - In response to PASS [argument] if the password matches the user-name, and **-ERR [AUTH] Username and password not accepted** if it does not.
- **+OK [number of messages in inbox] [size (in bytes) of inbox]** - in response to STAT.
- **+OK [message number] [size (in bytes) of message]** - in response to LIST [argument]. or **-ERR Message number out of range** if the argument passed is larger than the number of messages on the server.
- The server will either reply with the content of the message or **-ERR Message number out of range** if the argument passed is larger than the number of messages on the server in response to RETR [argument].
- **+OK** - In response to NOOP.
- - **+OK Message marked for deletion** - in response to DELE [argument] or **-ERR Message number out of range** if the argument passed is larger than the number of messages on the server.
- **+OK** - in response to RSET.
- **+OK Farewell** - in response to QUIT [2].

IMAP

IMAP Client Commands

- **LOGIN [username] [password]** - login to IMAP server using username, password.
- **SELECT INBOX** - Display details of messages.
- **CAPABILITY** - Request from server to send back its functionalities.
- **LIST INBOX** - List folders in INBOX.
- **DELETE inbox** - Delete folder 'inbox'.
- **FETCH [argument]** - get body/header of emails requested.
- **SUBSCRIBE [argument]** - Subscribe to a mailing list.
- **UNSUBSCRIBE [argument]** - Unsubscribe to mailing list.
- **CLOSE** - Terminate session with server.
- **COPY** - Copy specified emails to a folder
- **LOGOUT** - Logout from logged in session.

IMAP Server Replies

- **Ok LOGIN completed** - in reply to successful LOGIN [username] [password].
- *** [MAILBOX] exists** - in reply to SELECT INBOX.
- **Messages: [messages]** - in response to FETCH
- **No login failure: username or password rejected** - in reply to unsuccessful LOGIN [username] [password] [3].

Detailed Feature Implementation

Socket Handling

The IMAP, SMTP and POP3 clients have the ability to connect to their respective destination servers on the relevant port using Secure Sockets Layer. Once the client is able to establish a TCP socket connection with the destination server the socket handling functionality takes in a string passed to it from the application level. It then passes this to the TCP stream and sends to server. The response from the server will be received from the stream and passed to the application.

Likewise, the respective servers implemented have the ability to receive TCP connections from multiple hosts. The programme sets up a listener for incoming connections to a single port. In the case where a client connect to the server port, it establishes an SSL connection with them and; it has a public key and self signed certificate to authenticate to the client. One the connection is made the programme designates a socket and its own thread to that client and all future connections are via that designated port.

IMAP Client

The IMAP client has the ability to format strings for sending based on the IMAP protocol - such as appending each sent message to the server with an alphanumeric string and ending the message with Carriage Return and new line characters. The client functionality provides the user the ability to authenticate itself to the server and to view the server's capabilities. It can select mailboxes and list folders in the inbox. It can then select and view the header and body of the emails.

IMAP Server

The server does not provide the ability for the user to view a real email account. Rather a mock user has been set up in the code with mock emails. An email client may interact with the IMAP server and can only authenticate with that particular 'user account'. If the client requests to see the messages, the IMAP server will return them - however they are not in the format of a typical email. Rather, for demonstration purposes a string, with details of the sender and the actual message is sent back to the client upon request.

SMTP Client

The SMTP client code provides the user with the ability to authenticate him or herself to the server, sending the username and password base64 encoded. It provides the ability to send an email to the SMTP server, allowing for specifying of the recipient and sender as well as the body of the message.

SMTP Server

The SMTP server has the ability to receive input from the client's side and interpret the client's commands. The incoming command from the client is parsed and classified as a certain SMTP command accordingly. The SMTP can interpret and respond to commands such as EHLO, MAIL FROM, RCPT TO, and QUIT. Upon receiving a RCPT TO command the server will store the address of the recipient. The server then in turn becomes a client to another SMTP server (in the demonstration case - smtp.gmail.com) and will authenticate itself to that server and send an email to the server with the MAIL FROM being that of the relay server itself and the RCPT TO being that of the original senders email. The message pasted after the original DATA request is copied and then sent onto the next SMTP server.

POP3 Client and Server

Similar functionality as to that of the IMAP client and server was coded. However, at the time of submission the code had not been fully tested and is therefore not run in the demonstration code.

Graphical User Interface

Figure 1 of Appendix A shows the screen presented to the user upon running the application. The left half of the screen represents the receiving half of the screen and the right half of the screen represents the sending half of the screen. The user is first required to enter their username and password into the 'Username" and 'Password' fields, before clicking the 'Login' button. A pop-up message appears and indicates if the user-name or password are correct or incorrect. If correct, the program logs into the corresponding SMTP server.

Mail

As mentioned earlier, the right half of the screen represents the sending half of the screen and will be discussed in this section. Seeing that the user has already logged into the SMTP server, the user simply needs to type in the recipients email ID and the content of the email in order to send an email. Figure 3 in Appendix A shows the user networks4017tester@gmail.com logged in, sending an email to benjaminthomassa@gmail.com with the content of the message being some dummy text. If the message is able to send successfully, a pop-up message is displayed informing the user that the message has been successfully sent as shown by Figure 3.

Receiving mail

If the user wishes to receive mail, then the user needs to either select the IMAP or the POP3 check box. Upon doing so, the program will then either log into the IMAP or POP3 server, depending on which the user selected. This is shown by Figure 4 and 5, where the IMAP and POP3 servers were logged into respectively. If the user opted to log into the IMAP server, the drop-down box will contain items that correspond to the IMAP protocols and if the user opted to log into the POP3 server, the drop-down box will contain items that correspond to the POP3 protocols. Certain protocols (such as the POP3 RETR protocol) require the user to input additional data, in which case the user needs to enter the data into the 'Command' box. Once all the required fields have been selected or filled, the user can then hit the 'Send Command' button, which will send the command that the user has chosen, to the corresponding server (IMAP or POP3). The interaction between the client and server is displayed in the

Text Area at the bottom half of the receiving half of the screen. Figure 6 in Appendix A shows the Text Area being filled when a POP3 RETR protocol is run. The dummy text that was sent earlier can be seen, showing that the program is able to both send and receive emails.

Division of work

The work was divided as follows:

Percentage Division Of Work		
	Ben	Benji
SMTP	0	100
IMAP	0	100
POP	100	0
Multithreading	0	100
GUI	100	0
Report	80	20

Results

The Demonstration files were run and wireshark captures were taken of the programme running. All figures reffered to in this section can be found in Appendix B. All Python scripts referred to in this section are found in the Demonstration folder of the code. Fig. 9 shows a wireshark capture of the interaction between the running IMAP client (IP address 192.168.43.112) and that of imap.gmail.com (IP address 64.233.167.108). The file IMAPClientDemons.py was run to automate the client side interaction. Fig. 10 show the interaction between the IMAP client (run in IMAPserverDemo_2.py) and the IMAP server (run in IMAPserverDemo_1.py). The client programme which automates some client commands was run by the computer of IP adress 192.168.66.219 and the server programme was run on another computer (in the same network) at IP address 192.168.66.184. Fig. 7 shows the interaction between the SMTP client (run in SMTPclientDemo.py at IP address 192.168.1.14) and the server smtp.gmail.com (IP address 74.125.206.109). SMTPclientDemo.py automates client side SMTP commands. Fig. 8 shows the interaction between a SMTP client run on 192.168.66.219 and the SMTP server (SMTPserverDemo.py) run on 192.168.66.184.

Critical Analysis

The structure of the project may contain certain redundancy in code as much of the code across the servers was repeated in each case. Likewise for the clients. The implementation of the system is also minimal in that it was designed to provide minimal functionality and mainly for demonstrative purposes. On the other hand, by taking a modular approach to the design and separating the GUI, socket handling and client/server code it has been easy to interface and make changes between the system components and it allows for future scale and adding of more functionality.

Code structure

The project was coded in Python 2.7 and is designed to run on a Linux operating system. The main directory consists of four folders - Client, Server, GUI and Demonstration. A file called ServerClientDetails.py found in the main directory functions as a configuration file as to the ports and destination and/or source addresses of the servers and clients. All Client and Server functionality refer to this file to configure their addresses and ports. This section will discuss the class structure within each separation, as well as some of the important methods.

Server

ServerHandler Classes

The ServerHandler.py file contains classes for handling server interactions at the TCP level. The SMTPsocketThreadHandler, IMAPsocketThreadHandler and POPsocketThreadHandler classes deal with opening up the server socket port for connections and binding a listener for incoming connections. Once a connection has been received the socket thread handler will associate an SMTPserver_thread, IMAPserver_thread or POP3server_thread respectively with each incoming connection. The server_thread classes are responsible for designating each client making a connection its own port for subsequent interactions. Once the thread has started it will run an infinite loop that will continuously receive input from the TCP stream from the client and interpret and respond to this data until the client terminates the connection. Each server_thread object created (for the designated connection) will have its own instance of an IMAPserver, SMTPserver and POP3server object. This is discussed in the following section.

SMTP/IMAP/POP3server Classes

The SMTPserver, IMAPserver and POP3server classes contain code that interprets incoming messages passed to it by the server_thread classes and will parse the data, and according to the command received will respond with a reply code (SMTP) or the alphanumeric identifier (IMAP and POP3) with the response text appended to this. Each server_thread has its own instance of the server class which keeps track of the state of the current connection. Each server class has an 'interact' function which checks the received message against a number of conditions and if the condition is met the server class calculates a response and passes this response string back to the server_thread which then inserts it into the TCP stream to be received by the client.

How to set up a Server

To set up a server, a POP3/IMAP/SMTP-socketThreadHandler object must be instantiated. Once this has been done the server is ready and will wait for incoming connections to the designated port for processing.

Client

The client division was subdivided into separate sections according to the functionality of each section. This consisted of a client_sockets, a SMTPclient, a POP3client and a IMAPclient.

client_sockets class

The client_sockets.py file contains 3 classes which allocate TCP sockets on the client side for the SMTP, POP3 and IMAP services. These classes are clientIMAPSocket, clientSMTPSocket and clientPOP3Socket. These classes set up a socket that connects with the various remote server on the port of the service needed. For SMTP (SSL) the program connects to port 465 for the TCP connection. For IMAP, the program connects to port 993 and for POP3, the program connect to port 995. Each class has a sendMessageReceiveReply function that takes in a string as an argument, formats it correctly so that it can be sent via TCP stream to the server and it returns the reply of the server. Each SMTP/IMAP/POP3-client class contains a client_sockets object instance. The sendMessageReceiveReply method of the client_sockets object is then used by the client at the application level to pass data to the transport layer.

SMTPclient Class

The SMTP client class is responsible for interacting with the SMTP server. The class creates a client socket object to establish a connection with the SMTP server socket. The class has to follow all SMTP protocols, such as sending an EHLO command, before authentication can occur. The class has the primary ability to authenticate the user's user-name and password with the SMTP server as well as to send data from a sender to a receiver. The function authenticate() works by first encoding the user's username and password. A 'AUTH LOGIN' message is then sent to the SMTP server so that the server knows to expect the encoded user-name and password from the client.

IMAPclient Class

The IMAP client class is responsible for interacting with the IMAP server. The class creates a client socket object to establish a connection with the IMAP server socket. The class has the primary ability to authenticate the user's user-name and password with the IMAP server as well as to send data from a sender to a receiver. The function LOGIN() works by sending the user's user-name and password to the IMAP server, preceded by the word 'login'. If authenticated, the client then has the ability to carry out the LIST, DELETE, LOGOUT, SELECT and FETCH functions which are all self-explanatory.

POP3client Class

The POP3 client class is responsible for interacting with the POP server. The class creates a client socket object to establish a connection with the POP3 server socket. The class has to follow all POP3 protocols, such as first sending an USER command, with the client's user-name passed in as a parameter. The server then requests a password from the client before validating the user-name and password. The client sends the password through the PASS function, with the password being passed in as a parameter. Once authenticated, the client then once again has the ability to carry out the LIST, DELETE, RETRIEVE, RESET and QUIT functions. The only function which needs clarification is the LIST function, which has the option to either pass in an argument or not to. If an argument is passed in, then the server will list that specific email (the size of the email in bytes), if not then the server will list every single email belonging to the client with each corresponding email size (in bytes).

Graphical User Interface

The Graphical User Interface division was subdivided into separate sections according to the functionality of each section. This consisted of a GUI, a SMTPfunctions, a IMAPfunctions and a POP3functions file. The GUI file is responsible for presenting an easy to use graphical interface and each of the function files are responsible for linking the SMTP, IMAP and POP3 servers to the GUI. The GUI file makes use of the Tkinter package to form the graphical user interface. The GUI uses an array of Tkinter objects such as buttons, text boxes, check boxes, combo boxes and labels to form the GUI that is visible to the user.

Demonstrations

A number of demonstration scripts have been placed in the Demonstration file that are there to be run, and demonstrate interactions of the client/server. Each file contains specific instructions for how to run the demonstrations - such as requirement to run as super user, order to run them and in some cases the necessary changes that must be made to the serverClientDetails.py configuration file.

Conclusion

Clients and servers for SMTP, POP3 and IMAP have been developed. An overview of the system has been given, discussing what was and was not implemented. The server and client commands and responses implemented as per the protocols defined in the RFCs were listed. The code structure was discussed and wireshark captures of the actual packets sent in the demonstration code have been displayed.

References

[1] J. Klensin *Simple Mail Transfer Protocol* October 2008.

[2] J. Myers, M. Rose *Post Office Protocol - Version 3* May 1996.

[3] M. Crispin *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1* March 2003.

Appendix A: Graphical User Interface Images

Figure 1: Main Screen

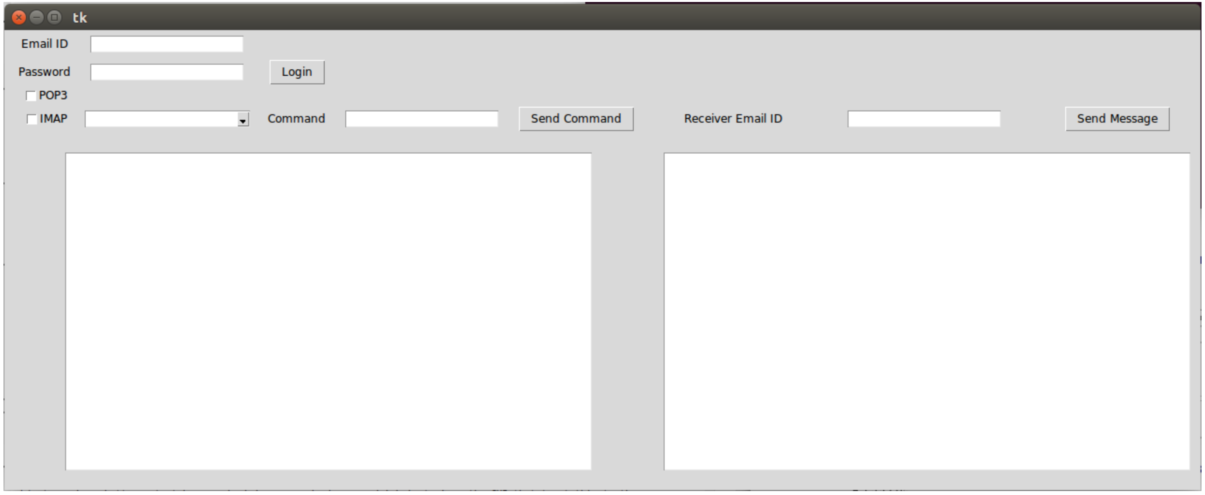
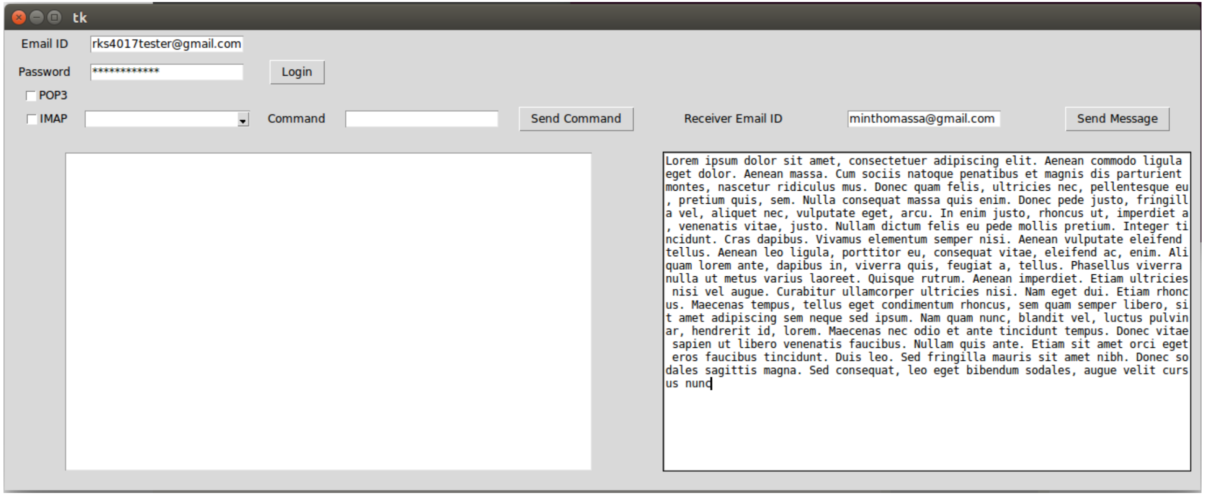


Figure 2: Sending Message Screen



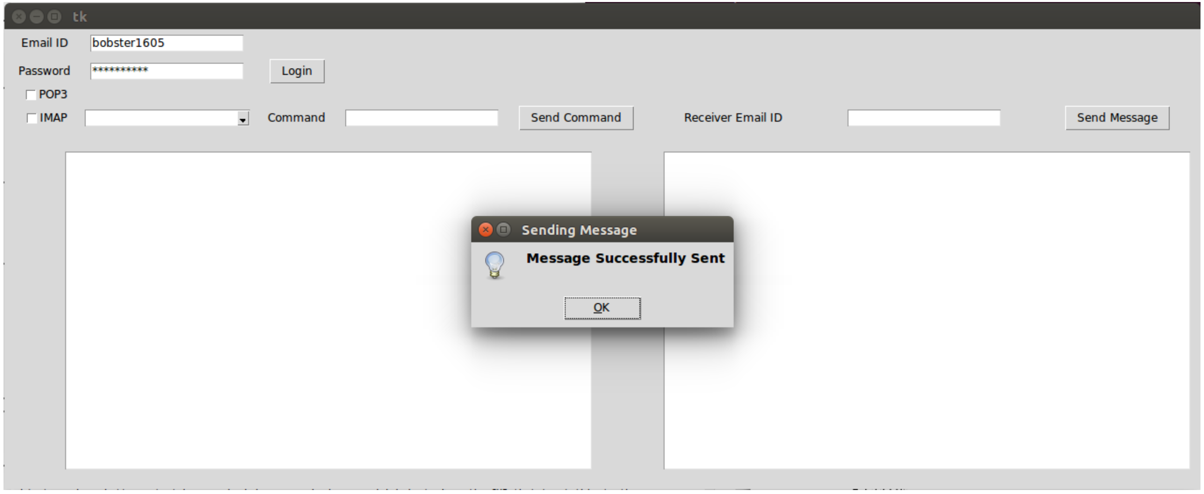
Email ID rks4017tester@gmail.com

Password *************

Receiver Email ID minthomassa@gmail.com

```
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula
eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient
montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu
, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringill
a vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a
, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer ti
ncidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend
tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Ali
quam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra
nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies
 nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhonc
us. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, si
t amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvin
ar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus. Donec vitae
 sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget
 eros faucibus tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec so
dales sagittis magna. Sed consequat, leo eget bibendum sodales, augue velit curs
us nunc
```

Figure 3: Message Sent Screen



Email ID: bobster1605

Password: **********

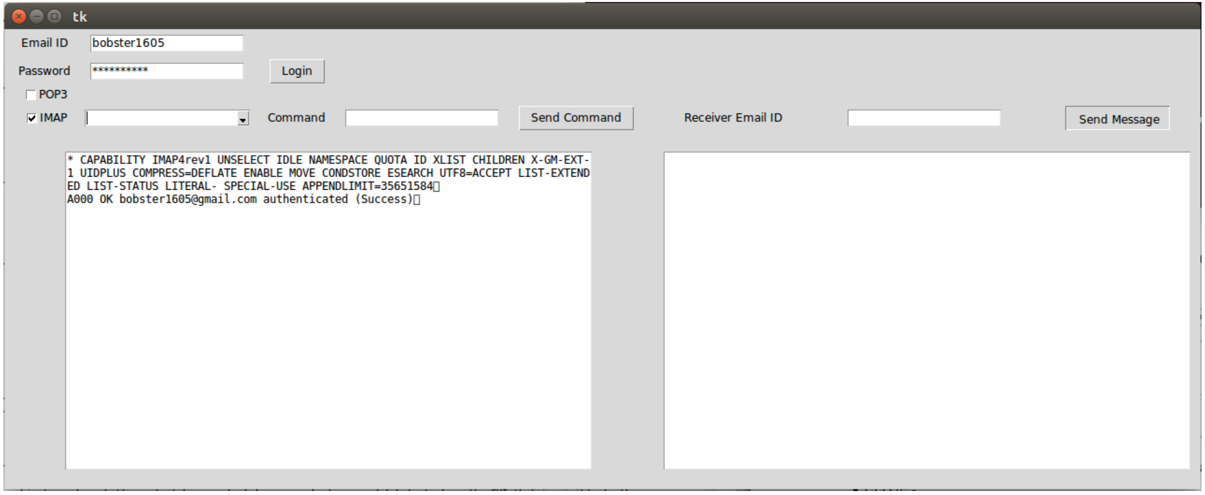
☐ POP3

☐ IMAP

Sending Message

Message Successfully Sent

Figure 4: IMAP Logged In Screen



Email ID: bobster1605

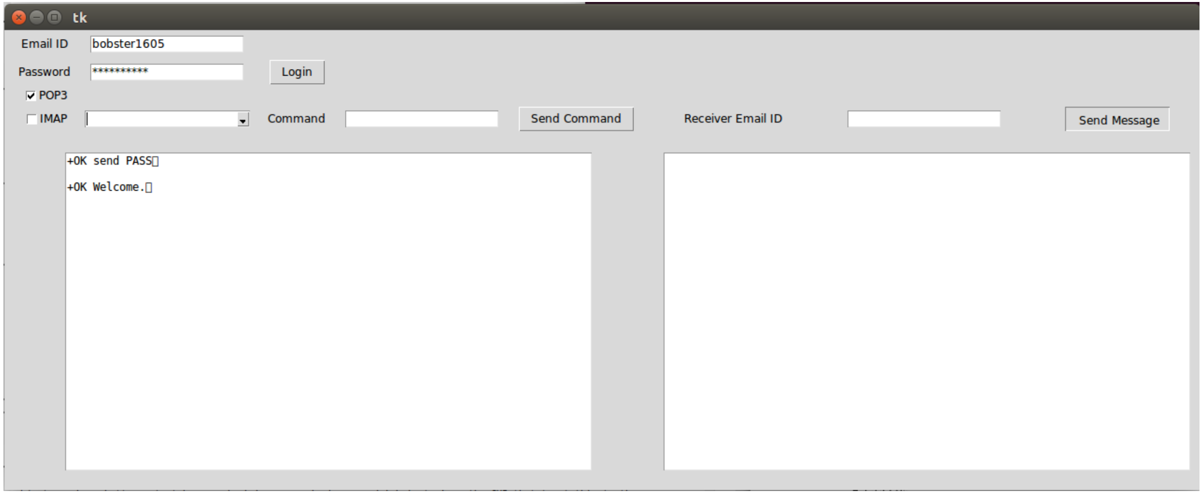
Password: **********

☐ POP3

☑ IMAP

```
* CAPABILITY IMAP4rev1 UNSELECT IDLE NAMESPACE QUOTA ID XLIST CHILDREN X-GM-EXT-
1 UIDPLUS COMPRESS=DEFLATE ENABLE MOVE CONDSTORE ESEARCH UTF8=ACCEPT LIST-EXTEND
ED LIST-STATUS LITERAL- SPECIAL-USE APPENDLIMIT=35651584
A000 OK bobster1605@gmail.com authenticated (Success)
```

Figure 5: POP3 Logged In Screen



Email ID: bobster1605

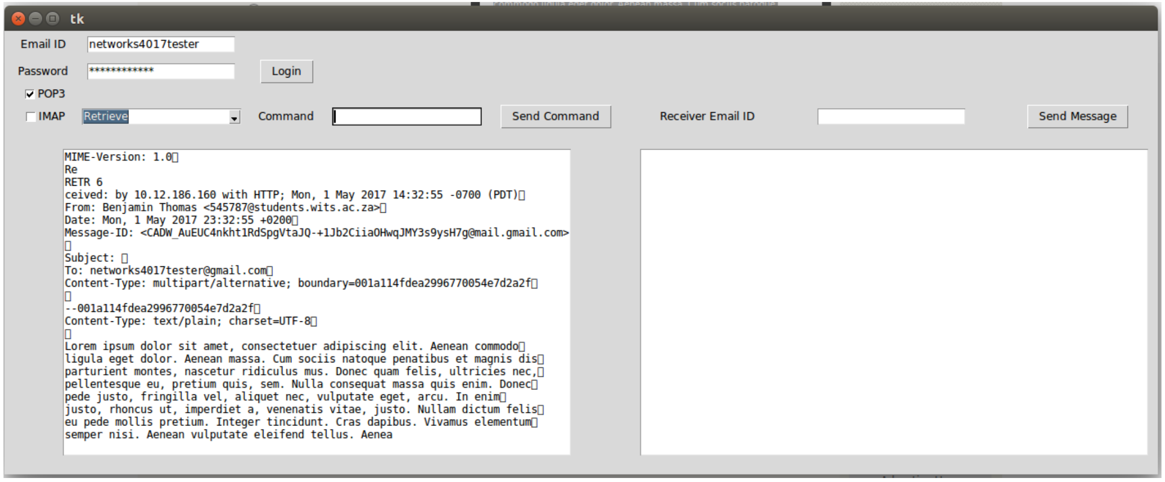
Password: **********

☑ POP3

☐ IMAP

```
+OK send PASS

+OK Welcome.
```

Figure 6: POP3 Retrieve Message Screen



Email ID networks4017tester

Password ************

☑ POP3

□ IMAP Retrieve

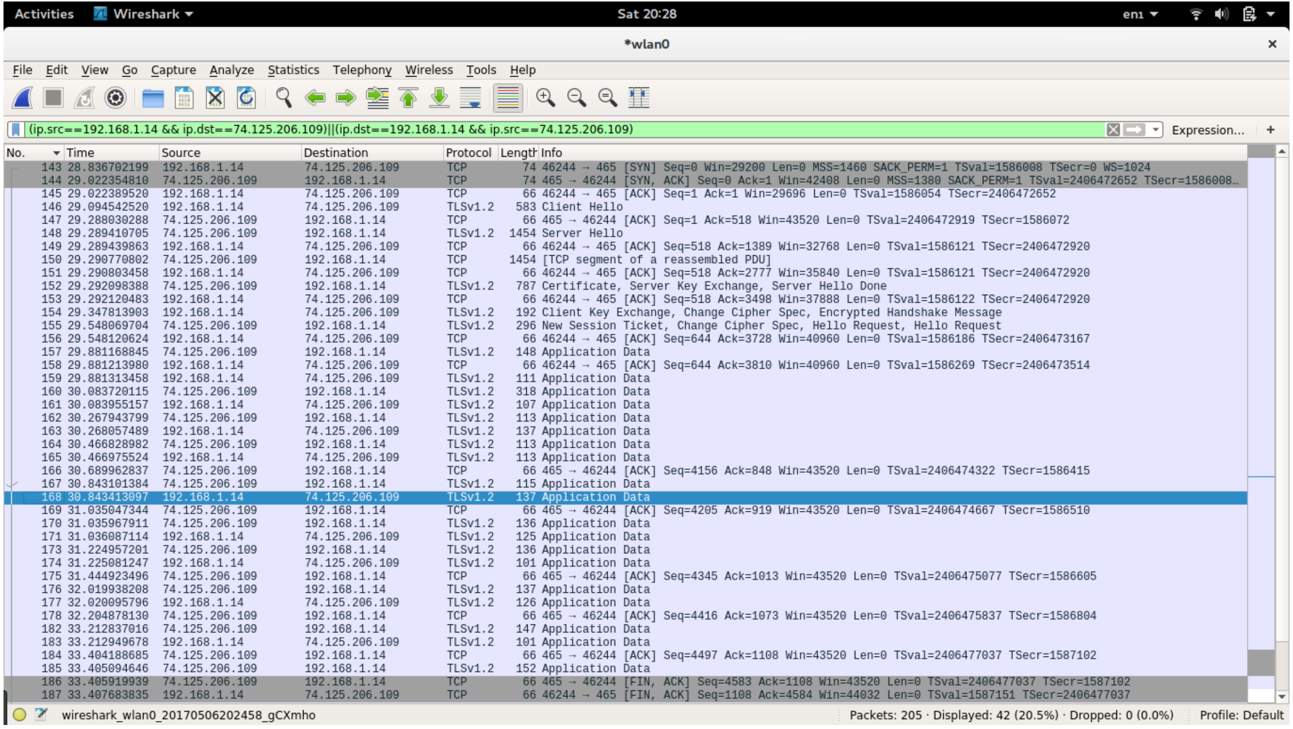
Command

Receiver Email ID

```
MIME-Version: 1.0
Re
RETR 6
ceived: by 10.12.186.160 with HTTP; Mon, 1 May 2017 14:32:55 -0700 (PDT)
From: Benjamin Thomas <545787@students.wits.ac.za>
Date: Mon, 1 May 2017 23:32:55 +0200
Message-ID: <CADW_AuEUC4nkht1RdSpgVtaJQ-+1Jb2CiiaOHwqJMY3s9ysH7g@mail.gmail.com>

Subject: 
To: networks4017tester@gmail.com
Content-Type: multipart/alternative; boundary=001a114fdea2996770054e7d2a2f

--001a114fdea2996770054e7d2a2f
Content-Type: text/plain; charset=UTF-8

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo
ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis
parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec,
pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec
pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim
justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis
eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum
semper nisi. Aenean vulputate eleifend tellus. Aenea
```

Appendix B: Wireshark Captures

Figure 7: Wireshark capture of SMTP client demonstration


Activities
Wireshark
Sat 20:28
en1
*wlan0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
(ip.src==192.168.1.14 && ip.dst==74.125.206.109)||(ip.dst==192.168.1.14 && ip.src==74.125.206.109)
```

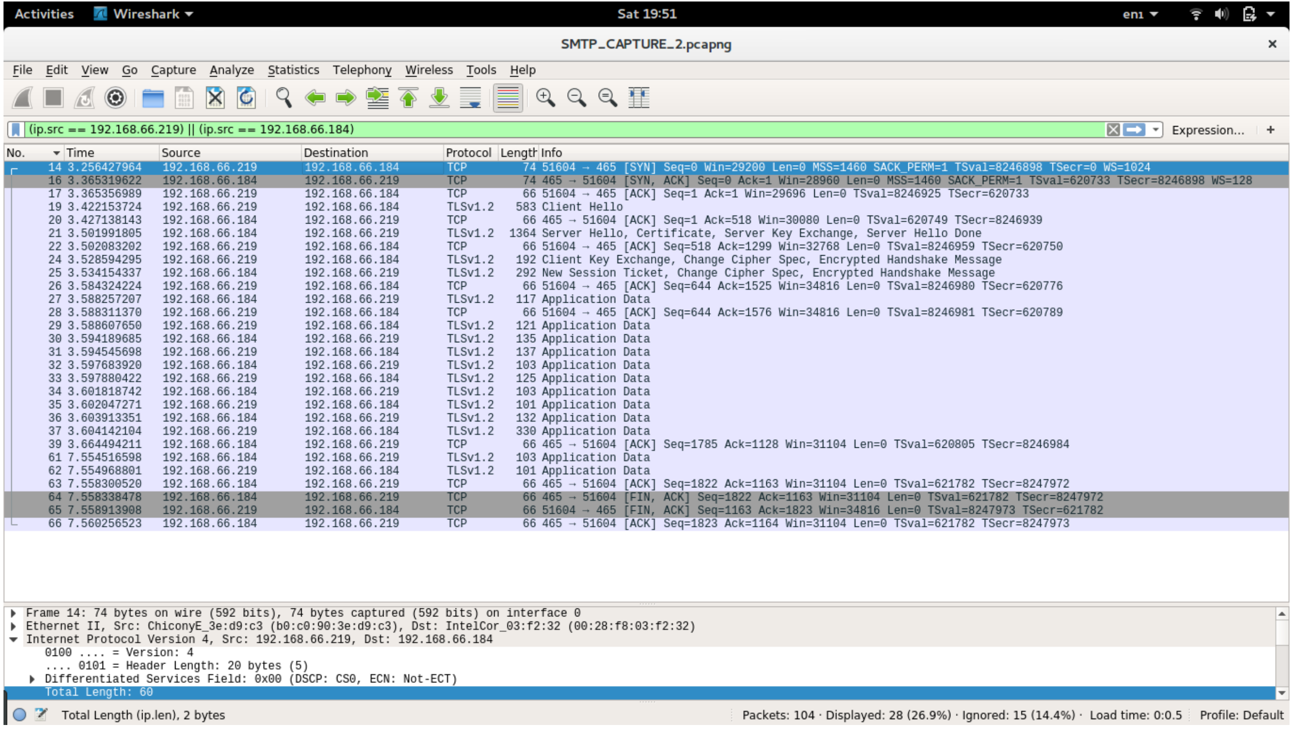
```
No.     Time          Source          Destination     Protocol Length Info
    143 28.836702199  192.168.1.14    74.125.206.109  TCP        74 46244 → 465 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1586008 TSecr=0 WS=1024
    144 29.022354810  74.125.206.109  192.168.1.14    TCP        74 465 → 46244 [SYN, ACK] Seq=0 Ack=1 Win=42408 Len=0 MSS=1380 SACK_PERM=1 TSval=2406472652 TSecr=1586008…
    145 29.022389520  192.168.1.14    74.125.206.109  TCP        66 46244 → 465 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=1586054 TSecr=2406472652
    146 29.094542520  192.168.1.14    74.125.206.109  TLSv1.2   583 Client Hello
    147 29.288030288  74.125.206.109  192.168.1.14    TCP        66 465 → 46244 [ACK] Seq=1 Ack=518 Win=43520 Len=0 TSval=2406472919 TSecr=1586072
    148 29.289410705  74.125.206.109  192.168.1.14    TLSv1.2  1454 Server Hello
    149 29.289439863  192.168.1.14    74.125.206.109  TCP        66 46244 → 465 [ACK] Seq=518 Ack=1389 Win=32768 Len=0 TSval=1586121 TSecr=2406472920
    150 29.290770802  74.125.206.109  192.168.1.14    TCP      1454 [TCP segment of a reassembled PDU]
    151 29.290803458  192.168.1.14    74.125.206.109  TCP        66 46244 → 465 [ACK] Seq=518 Ack=2777 Win=35840 Len=0 TSval=1586121 TSecr=2406472920
    152 29.292098388  74.125.206.109  192.168.1.14    TLSv1.2   787 Certificate, Server Key Exchange, Server Hello Done
    153 29.292120483  192.168.1.14    74.125.206.109  TCP        66 46244 → 465 [ACK] Seq=518 Ack=3498 Win=37888 Len=0 TSval=1586122 TSecr=2406472920
    154 29.347813903  192.168.1.14    74.125.206.109  TLSv1.2   192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
    155 29.548069704  74.125.206.109  192.168.1.14    TLSv1.2   296 New Session Ticket, Change Cipher Spec, Hello Request, Hello Request
    156 29.548120624  192.168.1.14    74.125.206.109  TCP        66 46244 → 465 [ACK] Seq=644 Ack=3728 Win=40960 Len=0 TSval=1586186 TSecr=2406473167
    157 29.881168845  74.125.206.109  192.168.1.14    TLSv1.2   148 Application Data
    158 29.881213980  192.168.1.14    74.125.206.109  TCP        66 46244 → 465 [ACK] Seq=644 Ack=3810 Win=40960 Len=0 TSval=1586269 TSecr=2406473514
    159 29.881313458  192.168.1.14    74.125.206.109  TLSv1.2   111 Application Data
    160 30.083720115  74.125.206.109  192.168.1.14    TLSv1.2   318 Application Data
    161 30.083955157  192.168.1.14    74.125.206.109  TLSv1.2   107 Application Data
    162 30.267943799  74.125.206.109  192.168.1.14    TLSv1.2   113 Application Data
    163 30.268057489  192.168.1.14    74.125.206.109  TLSv1.2   137 Application Data
    164 30.466828982  74.125.206.109  192.168.1.14    TLSv1.2   113 Application Data
    165 30.466975524  192.168.1.14    74.125.206.109  TLSv1.2   113 Application Data
    166 30.689962837  74.125.206.109  192.168.1.14    TCP        66 465 → 46244 [ACK] Seq=4156 Ack=848 Win=43520 Len=0 TSval=2406474322 TSecr=1586415
    167 30.843101384  74.125.206.109  192.168.1.14    TLSv1.2   115 Application Data
    168 30.843413097  192.168.1.14    74.125.206.109  TLSv1.2   137 Application Data
    169 31.035047344  74.125.206.109  192.168.1.14    TCP        66 465 → 46244 [ACK] Seq=4205 Ack=919 Win=43520 Len=0 TSval=2406474667 TSecr=1586510
    170 31.035967911  74.125.206.109  192.168.1.14    TLSv1.2   136 Application Data
    171 31.036087114  192.168.1.14    74.125.206.109  TLSv1.2   125 Application Data
    173 31.224957201  74.125.206.109  192.168.1.14    TLSv1.2   136 Application Data
    174 31.225081247  192.168.1.14    74.125.206.109  TLSv1.2   101 Application Data
    175 31.444923496  74.125.206.109  192.168.1.14    TCP        66 465 → 46244 [ACK] Seq=4345 Ack=1013 Win=43520 Len=0 TSval=2406475077 TSecr=1586605
    176 32.019938208  74.125.206.109  192.168.1.14    TLSv1.2   137 Application Data
    177 32.020095796  192.168.1.14    74.125.206.109  TLSv1.2   126 Application Data
    178 32.204878130  74.125.206.109  192.168.1.14    TCP        66 465 → 46244 [ACK] Seq=4416 Ack=1073 Win=43520 Len=0 TSval=2406475837 TSecr=1586804
    182 33.212837016  74.125.206.109  192.168.1.14    TLSv1.2   147 Application Data
    183 33.212949678  192.168.1.14    74.125.206.109  TLSv1.2   101 Application Data
    184 33.404188685  74.125.206.109  192.168.1.14    TCP        66 465 → 46244 [ACK] Seq=4497 Ack=1108 Win=43520 Len=0 TSval=2406477037 TSecr=1587102
    185 33.405094646  74.125.206.109  192.168.1.14    TLSv1.2   152 Application Data
    186 33.405919939  74.125.206.109  192.168.1.14    TCP        66 465 → 46244 [FIN, ACK] Seq=4583 Ack=1108 Win=43520 Len=0 TSval=2406477037 TSecr=1587102
    187 33.407683835  192.168.1.14    74.125.206.109  TCP        66 46244 → 465 [FIN, ACK] Seq=1108 Ack=4584 Win=44032 Len=0 TSval=1587151 TSecr=2406477037
```

wireshark_wlan0_20170506202458_gCXmho

Packets: 205 · Displayed: 42 (20.5%) · Dropped: 0 (0.0%)

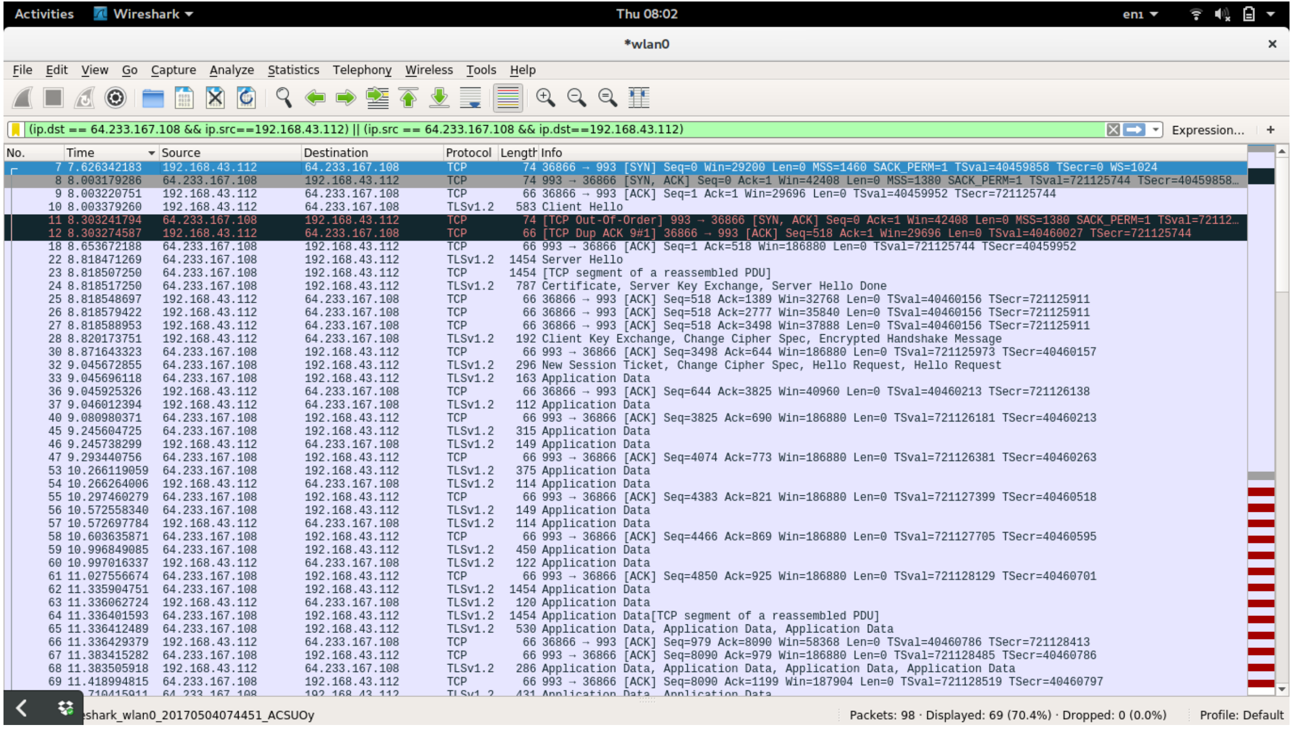
Profile: Default

Figure 8: Wireshark capture of SMTP server Demonstration



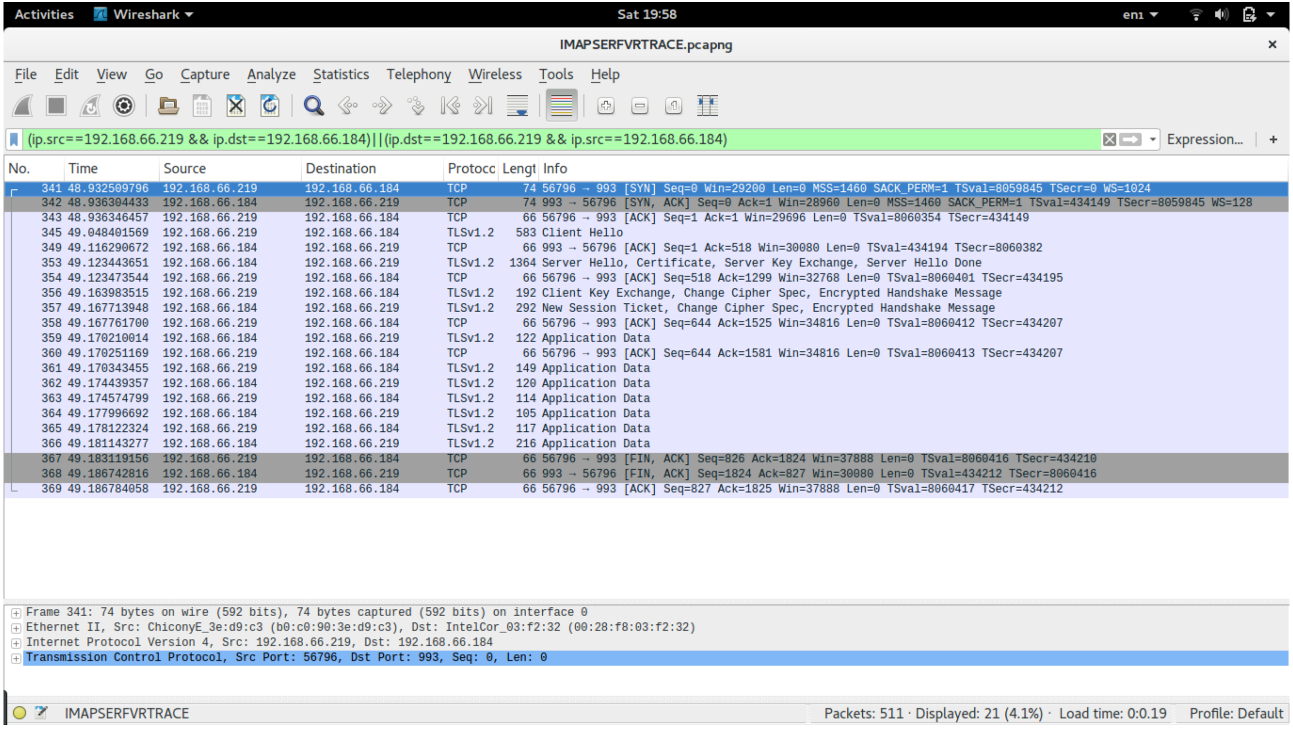
```
(ip.src == 192.168.66.219) || (ip.src == 192.168.66.184)

No.  Time         Source          Destination     Protocol Length Info
  14 3.256427964  192.168.66.219  192.168.66.184  TCP        74 51604 → 465 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=8246898 TSecr=0 WS=1024
  16 3.365319622  192.168.66.184  192.168.66.219  TCP        74 465 → 51604 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=620733 TSecr=8246898 WS=128
  17 3.365356989  192.168.66.219  192.168.66.184  TCP        66 51604 → 465 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=8246925 TSecr=620733
  19 3.422153724  192.168.66.219  192.168.66.184  TLSv1.2   583 Client Hello
  20 3.427138143  192.168.66.184  192.168.66.219  TCP        66 465 → 51604 [ACK] Seq=1 Ack=518 Win=30080 Len=0 TSval=620749 TSecr=8246939
  21 3.501991805  192.168.66.184  192.168.66.219  TLSv1.2  1364 Server Hello, Certificate, Server Key Exchange, Server Hello Done
  22 3.502083202  192.168.66.219  192.168.66.184  TCP        66 51604 → 465 [ACK] Seq=518 Ack=1299 Win=32768 Len=0 TSval=8246959 TSecr=620750
  24 3.528594295  192.168.66.219  192.168.66.184  TLSv1.2   192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
  25 3.534154337  192.168.66.184  192.168.66.219  TLSv1.2   292 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
  26 3.584324224  192.168.66.219  192.168.66.184  TCP        66 51604 → 465 [ACK] Seq=644 Ack=1525 Win=34816 Len=0 TSval=8246980 TSecr=620776
  27 3.588257207  192.168.66.184  192.168.66.219  TLSv1.2   117 Application Data
  28 3.588311370  192.168.66.219  192.168.66.184  TCP        66 51604 → 465 [ACK] Seq=644 Ack=1576 Win=34816 Len=0 TSval=8246981 TSecr=620789
  29 3.588607650  192.168.66.219  192.168.66.184  TLSv1.2   121 Application Data
  30 3.594189685  192.168.66.184  192.168.66.219  TLSv1.2   135 Application Data
  31 3.594545698  192.168.66.219  192.168.66.184  TLSv1.2   137 Application Data
  32 3.597683920  192.168.66.184  192.168.66.219  TLSv1.2   103 Application Data
  33 3.597880422  192.168.66.219  192.168.66.184  TLSv1.2   125 Application Data
  34 3.601818742  192.168.66.184  192.168.66.219  TLSv1.2   103 Application Data
  35 3.602047271  192.168.66.219  192.168.66.184  TLSv1.2   101 Application Data
  36 3.603913351  192.168.66.184  192.168.66.219  TLSv1.2   132 Application Data
  37 3.604142104  192.168.66.219  192.168.66.184  TLSv1.2   330 Application Data
  39 3.664494211  192.168.66.184  192.168.66.219  TCP        66 465 → 51604 [ACK] Seq=1785 Ack=1128 Win=31104 Len=0 TSval=620805 TSecr=8246984
  61 7.554516598  192.168.66.184  192.168.66.219  TLSv1.2   103 Application Data
  62 7.554968801  192.168.66.219  192.168.66.184  TLSv1.2   101 Application Data
  63 7.558300520  192.168.66.184  192.168.66.219  TCP        66 465 → 51604 [ACK] Seq=1822 Ack=1163 Win=31104 Len=0 TSval=621782 TSecr=8247972
  64 7.558338478  192.168.66.184  192.168.66.219  TCP        66 465 → 51604 [FIN, ACK] Seq=1822 Ack=1163 Win=31104 Len=0 TSval=621782 TSecr=8247972
  65 7.558913908  192.168.66.219  192.168.66.184  TCP        66 51604 → 465 [FIN, ACK] Seq=1163 Ack=1823 Win=34816 Len=0 TSval=8247973 TSecr=621782
  66 7.560256523  192.168.66.184  192.168.66.219  TCP        66 465 → 51604 [ACK] Seq=1823 Ack=1164 Win=31104 Len=0 TSval=621782 TSecr=8247973

▸ Frame 14: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▸ Ethernet II, Src: ChiconyE_3e:d9:c3 (b0:c0:90:3e:d9:c3), Dst: IntelCor_03:f2:32 (00:28:f8:03:f2:32)
▾ Internet Protocol Version 4, Src: 192.168.66.219, Dst: 192.168.66.184
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60

Total Length (ip.len), 2 bytes        Packets: 104 · Displayed: 28 (26.9%) · Ignored: 15 (14.4%) · Load time: 0:0.5    Profile: Default
```

Figure 9: Wireshark capture of IMAP client Demonstration



```
(ip.dst == 64.233.167.108 && ip.src==192.168.43.112) || (ip.src == 64.233.167.108 && ip.dst==192.168.43.112)

No.  Time          Source          Destination     Protocol Length Info
   7 7.626342183   192.168.43.112  64.233.167.108  TCP        74 36866 → 993 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=40459858 TSecr=0 WS=1024
   8 8.003179286   64.233.167.108  192.168.43.112  TCP        74 993 → 36866 [SYN, ACK] Seq=0 Ack=1 Win=42408 Len=0 MSS=1380 SACK_PERM=1 TSval=721125744 TSecr=40459858…
   9 8.003220751   192.168.43.112  64.233.167.108  TCP        66 36866 → 993 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=40459952 TSecr=721125744
  10 8.003379260   192.168.43.112  64.233.167.108  TLSv1.2   583 Client Hello
  11 8.303241794   64.233.167.108  192.168.43.112  TCP        74 [TCP Out-Of-Order] 993 → 36866 [SYN, ACK] Seq=0 Ack=1 Win=42408 Len=0 MSS=1380 SACK_PERM=1 TSval=72112…
  12 8.303274587   192.168.43.112  64.233.167.108  TCP        66 [TCP Dup ACK 9#1] 36866 → 993 [ACK] Seq=518 Ack=1 Win=29696 Len=0 TSval=40460027 TSecr=721125744
  18 8.653672188   64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=1 Ack=518 Win=186880 Len=0 TSval=721125744 TSecr=40459952
  22 8.818471269   64.233.167.108  192.168.43.112  TLSv1.2  1454 Server Hello
  23 8.818507250   64.233.167.108  192.168.43.112  TCP      1454 [TCP segment of a reassembled PDU]
  24 8.818517250   64.233.167.108  192.168.43.112  TLSv1.2   787 Certificate, Server Key Exchange, Server Hello Done
  25 8.818548697   192.168.43.112  64.233.167.108  TCP        66 36866 → 993 [ACK] Seq=518 Ack=1389 Win=32768 Len=0 TSval=40460156 TSecr=721125911
  26 8.818579422   192.168.43.112  64.233.167.108  TCP        66 36866 → 993 [ACK] Seq=518 Ack=2777 Win=35840 Len=0 TSval=40460156 TSecr=721125911
  27 8.818588953   192.168.43.112  64.233.167.108  TCP        66 36866 → 993 [ACK] Seq=518 Ack=3498 Win=37888 Len=0 TSval=40460156 TSecr=721125911
  28 8.820173751   192.168.43.112  64.233.167.108  TLSv1.2   192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
  30 8.871643323   64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=3498 Ack=644 Win=186880 Len=0 TSval=721125973 TSecr=40460157
  32 9.045672855   64.233.167.108  192.168.43.112  TLSv1.2   296 New Session Ticket, Change Cipher Spec, Hello Request, Hello Request
  33 9.045696118   64.233.167.108  192.168.43.112  TLSv1.2   163 Application Data
  36 9.045925326   192.168.43.112  64.233.167.108  TCP        66 36866 → 993 [ACK] Seq=644 Ack=3825 Win=40960 Len=0 TSval=40460213 TSecr=721126138
  37 9.046012394   192.168.43.112  64.233.167.108  TLSv1.2   112 Application Data
  40 9.080980371   64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=3825 Ack=690 Win=186880 Len=0 TSval=721126181 TSecr=40460213
  45 9.245604725   64.233.167.108  192.168.43.112  TLSv1.2   315 Application Data
  46 9.245738299   192.168.43.112  64.233.167.108  TLSv1.2   149 Application Data
  47 9.293440756   64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=4074 Ack=773 Win=186880 Len=0 TSval=721126381 TSecr=40460263
  53 10.266119059  64.233.167.108  192.168.43.112  TLSv1.2   375 Application Data
  54 10.266264006  192.168.43.112  64.233.167.108  TLSv1.2   114 Application Data
  55 10.297460279  64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=4383 Ack=821 Win=186880 Len=0 TSval=721127399 TSecr=40460518
  56 10.572558340  64.233.167.108  192.168.43.112  TLSv1.2   149 Application Data
  57 10.572697784  192.168.43.112  64.233.167.108  TLSv1.2   114 Application Data
  58 10.603635871  64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=4466 Ack=869 Win=186880 Len=0 TSval=721127705 TSecr=40460595
  59 10.996849085  64.233.167.108  192.168.43.112  TLSv1.2   450 Application Data
  60 10.997016337  192.168.43.112  64.233.167.108  TLSv1.2   122 Application Data
  61 11.027556674  64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=4850 Ack=925 Win=186880 Len=0 TSval=721128129 TSecr=40460701
  62 11.335904751  64.233.167.108  192.168.43.112  TLSv1.2  1454 Application Data
  63 11.336062724  192.168.43.112  64.233.167.108  TLSv1.2   120 Application Data
  64 11.336401593  64.233.167.108  192.168.43.112  TLSv1.2  1454 Application Data[TCP segment of a reassembled PDU]
  65 11.336412489  64.233.167.108  192.168.43.112  TLSv1.2   530 Application Data, Application Data, Application Data
  66 11.336429379  192.168.43.112  64.233.167.108  TCP        66 36866 → 993 [ACK] Seq=979 Ack=8090 Win=58368 Len=0 TSval=40460786 TSecr=721128413
  67 11.383415282  64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=8090 Ack=979 Win=186880 Len=0 TSval=721128485 TSecr=40460786
  68 11.383505918  192.168.43.112  64.233.167.108  TLSv1.2   286 Application Data, Application Data, Application Data, Application Data
  69 11.418994815  64.233.167.108  192.168.43.112  TCP        66 993 → 36866 [ACK] Seq=8090 Ack=1199 Win=187904 Len=0 TSval=721128519 TSecr=40460797
     .710415911    64.233.167.108  192.168.43.112  TLSv1.2   431 Application Data, Application Data

eshark_wlan0_20170504074451_ACSUOy        Packets: 98 · Displayed: 69 (70.4%) · Dropped: 0 (0.0%)    Profile: Default
```

Figure 10: Wireshark capture of IMAP server Demonstration



(ip.src==192.168.66.219 && ip.dst==192.168.66.184)||(ip.dst==192.168.66.219 && ip.src==192.168.66.184)

```
No.    Time          Source          Destination     Protocc Lengt Info
   341 48.932509796  192.168.66.219  192.168.66.184  TCP       74 56796 → 993 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=8059845 TSecr=0 WS=1024
   342 48.936304433  192.168.66.184  192.168.66.219  TCP       74 993 → 56796 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=434149 TSecr=8059845 WS=128
   343 48.936346457  192.168.66.219  192.168.66.184  TCP       66 56796 → 993 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=8060354 TSecr=434149
   345 49.048401569  192.168.66.219  192.168.66.184  TLSv1.2  583 Client Hello
   349 49.116290672  192.168.66.184  192.168.66.219  TCP       66 993 → 56796 [ACK] Seq=1 Ack=518 Win=30080 Len=0 TSval=434194 TSecr=8060382
   353 49.123443651  192.168.66.184  192.168.66.219  TLSv1.2 1364 Server Hello, Certificate, Server Key Exchange, Server Hello Done
   354 49.123473544  192.168.66.219  192.168.66.184  TCP       66 56796 → 993 [ACK] Seq=518 Ack=1299 Win=32768 Len=0 TSval=8060401 TSecr=434195
   356 49.163983515  192.168.66.219  192.168.66.184  TLSv1.2  192 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
   357 49.167713948  192.168.66.184  192.168.66.219  TLSv1.2  292 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
   358 49.167761700  192.168.66.219  192.168.66.184  TCP       66 56796 → 993 [ACK] Seq=644 Ack=1525 Win=34816 Len=0 TSval=8060412 TSecr=434207
   359 49.170210014  192.168.66.184  192.168.66.219  TLSv1.2  122 Application Data
   360 49.170251169  192.168.66.219  192.168.66.184  TCP       66 56796 → 993 [ACK] Seq=644 Ack=1581 Win=34816 Len=0 TSval=8060413 TSecr=434207
   361 49.170343455  192.168.66.219  192.168.66.184  TLSv1.2  149 Application Data
   362 49.174439357  192.168.66.184  192.168.66.219  TLSv1.2  120 Application Data
   363 49.174574799  192.168.66.219  192.168.66.184  TLSv1.2  114 Application Data
   364 49.177996692  192.168.66.184  192.168.66.219  TLSv1.2  105 Application Data
   365 49.178122324  192.168.66.219  192.168.66.184  TLSv1.2  117 Application Data
   366 49.181143277  192.168.66.184  192.168.66.219  TLSv1.2  216 Application Data
   367 49.183119156  192.168.66.219  192.168.66.184  TCP       66 56796 → 993 [FIN, ACK] Seq=826 Ack=1824 Win=37888 Len=0 TSval=8060416 TSecr=434210
   368 49.186742816  192.168.66.184  192.168.66.219  TCP       66 993 → 56796 [FIN, ACK] Seq=1824 Ack=827 Win=30080 Len=0 TSval=434212 TSecr=8060416
   369 49.186784058  192.168.66.219  192.168.66.184  TCP       66 56796 → 993 [ACK] Seq=827 Ack=1825 Win=37888 Len=0 TSval=8060417 TSecr=434212
```

```
⊞ Frame 341: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
⊞ Ethernet II, Src: ChiconyE_3e:d9:c3 (b0:c0:90:3e:d9:c3), Dst: IntelCor_03:f2:32 (00:28:f8:03:f2:32)
⊞ Internet Protocol Version 4, Src: 192.168.66.219, Dst: 192.168.66.184
⊞ Transmission Control Protocol, Src Port: 56796, Dst Port: 993, Seq: 0, Len: 0
```

IMAPSERFVRTRACE | Packets: 511 · Displayed: 21 (4.1%) · Load time: 0:0.19 | Profile: Default