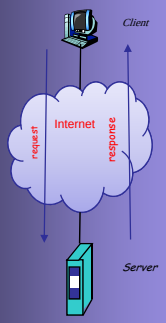


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 10.4: Mail Protocols (Part 4)

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives

- Learn about the Post Office Protocol version 3, POP3,[RFC 1939]: http://www.ietf.org/rfc/rfc1939.txt
- Introduction to Secure Socket Layer (SSL)
- Lean how to write a POP client to retrieve mails

Introduction

- The purpose of the **Post Office Protocol** is to allow users to retrieve their mails from a mail server to their local machines.
  - It does not support transmission of messages to the server.
- Some of the advantages of **POP** are:
  - Saving storage space on the mail server
  - Relieving the network admin from managing mail boxes.
- POP may not be convenient for users accessing their mails through multiple machines.
  - Once downloaded on one machine and deleted from the server, the mails cannot be accessed from another machine.
- The alternative is **Internet Mail Access Protocol** (**IMAP) –** RFC 1730, which allows users to manipulate their mails directly on the mail server.
- The POP server runs on the well-known port **110**.

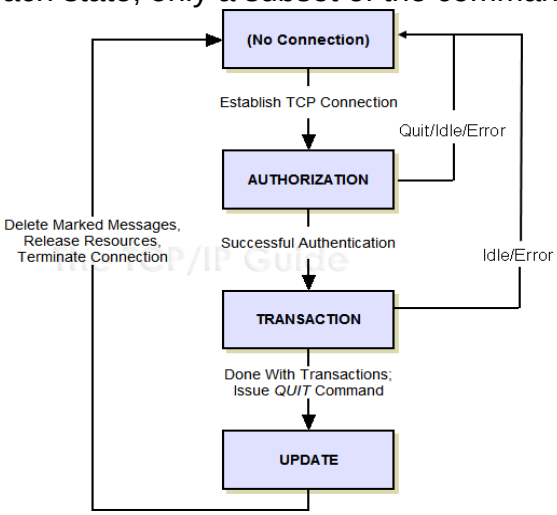
POP Commands

- POP protocol provides commands that the client can use to communicate with the server.
- Each POP command consists of a command string followed by zero or more parameters.

Command	Description
USER name	Specifies username for USER/PASS authentication
PASS password	Specifies password for USER/PASS authentication
STAT	Requests a “drop listing” indicating number of messages and size of the mail store
LIST [msg]	Requests a “scan listing” for specific or all message(s) indicating message number and size of the message
TOP msg n	Retrieves specified message’s headers and top n lines of body
RETR msg	Retrieves specified message
DELE msg	Marks specified message as deleted
RSET	Resets initial state by unmarking deleted messages
NOOP	No operation
QUIT	Initiates session termination

POP Commands …

- POP is a **stateful** protocol with four states as summarized below. At each state, only a subset of the commands can be issued.



POP Commands …

- The following describes the various states:

No Connection:

- The client must first establishes a TCP connection with the POP server on port **110**.
- Once a connection is established, the server sends a welcome message and enters into the Authorization state.

Authorization State:

- The client either sends a "**QUIT**" command to terminate the connection or issues a "**USER username**" command, followed by "**PASS password**" command to login.
- If login is successful, the server acquires an exclusive access lock on the *maildrop*, to prevent messages from being modified or removed before the session enters the UPDATE state.

POP Commands …

Authorization State (continued):

- If the lock is successfully acquired, the server responds with a positive status indicator.
- If the *maildrop* cannot be opened for some reason (example, a lock can not be acquired), the client is denied access.
- After the POP3 server has opened the *maildrop*, it assigns a message number to each message, and notes the size of each message in bytes.
- The first message in the *maildrop* is assigned a message-number of "1", the second is assigned "2", etc.
- The server then enters into the Transaction State.

POP Commands …

Transaction State:

- In the transaction state, the user can issue any of the following transaction commands repeatedly until he enters the **QUIT** command: STAT, TOP, RETR, LIST, DELE, RSET, NOOP, QUIT
- If a message is marked for deletion with the DELE command, then its message number is not a valid argument for any command. Also the message will not be shown as part of a response.
- While still in the Transaction State, messages marked for deletion can be unmarked by issuing a RSET command.
- Issuing the QUIT command puts the server into the **UPDATE** state.

POP Commands …

Update State:

- When the client issues the QUIT command from the TRANSACTION state, the server enters the UPDATE state.
- In this state, the server removes any message marked for deletion from the maildrop, then releases the exclusive-access lock on the maildrop and closes the TCP connection.

Note:

- If a session terminates for some reason other than a client-issuing the QUIT command, the server does NOT enter the UPDATE state and MUST not remove any messages from the maildrop.

POP Responses …

- For each command, the server responds with one of two responses called status indicators;
  - **"+OK"** followed by some textual comments if the commands succeeds
  - **"-ERR"** followed by textual comments if the command fails.
- For multi-line responses such as those from LIST, RETR & TOP commands, the response lines ends with "." on a line by itself.

Command	Response	Example
STAT	+OK nn mm	STAT +OK 2 320
LIST [msg]	+OK scan listing follows -ERR no such message	LIST +OK 2 messages (320 octets) 1 120 2 200 . LIST 2 +OK 2 200

POP Responses …

Command	Response	Example
RETR msg	+OK message follows -ERR no such message	RETR 1 +OK 120 octets < the server sends the entire message here ending with '.'>
TOP msg n	+OK top of msg follows -ERR no such message	TOP 1 10 +OK top of message follows < first 10 lines of the message # 1 >
DELE msg	+OK message deleted -ERR no such message	DELE 2 +OK message deleted
RSET	+OK reset state	RSET +OK reset state
NOOP	+OK no transaction	NOOP +OK No-op to you too!
QUIT	+OK bye	QUIT +OK goodbye

Sample POP3 Session with Telnet

```
telnet bareed.ccse.kfupm.edu.sa 110
+OK POP3 khuzama v2000.69 server ready
user bmghandi (in plain text)
+OK User name accepted, password please
pass actual-password (in plain text)
+OK Mailbox open, 24 messages
stat
+OK 24 3535201
list
+OK Mailbox scan listing follows
1 355411
2 3733
...//deleted
24 4590
.
top 10 5
+OK Top of message follows
Received: from khuzama.ccse.kfupm.edu.sa by
    khuzama.ccse.kfupm.edu.sa (8.11.0/8.9.3) with ESMTP id
...//deleted
```

Sample SMTP Session using TELNET …

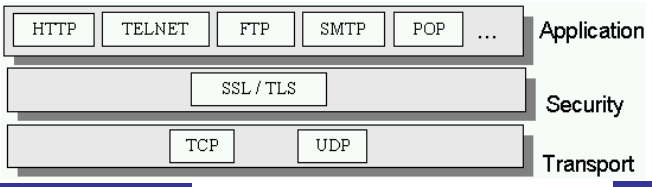
```
To: <bmghandi@ccse.kfupm.edu.sa>
Subject: SWE344 - Project proposal...
Message-ID:
    <1070190821.3fc9d0e5c948f@webmail.ccse.kfupm.edu.sa>
Date: Sun, 30 Nov 2003 14:13:41 +0300
From: "Al-Tawfiq, Hani" <st204920@ccse.kfupm.edu.sa>
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
User-Agent: IMP/PHP IMAP webmail program 2.2.0-pre13
X-Originating-IP: 212.93.193.82
Content-Type: text/plain
Content-Length: 838
Status: RO

Assalam Alaikum
Dear Mr. Ghandi

Eid Mubarak to you. I hope you enjoyed the vacation and I hope
    you are in good health
.
```


Introduction to SSL

- Communication using normal Sockets is not secure.
  - Messages (including passwords) can be easily captured on transit by malicious users using network traffic capturing tools such as Analyzer.
- SSL, also called Transport Layer Security (TLS : RFC 2246) is designed to provide security for Internet applications.
  - Its sits between application layer (Socket layer) and Transport Layer.
- SSL ensures that messages exchanged between server and clients are encrypted so that even if they are captured on transit, they cannot be interpreted.
- SSL also provides a mechanism for servers and clients to authenticate themselves using digital certificates and signatures



Introduction to SSL …

- SSL handshake takes place immediately after establishing a connection but before any application data is exchanged. Such hand shake involves:
  - Server sends its certificate for authentication by the client.
  - The client optionally authenticate itself to the server.
  - Client and server negotiate a cryptographic algorithms to use.
  - Server and Clients use public-key encryption techniques to generate shared secrets.
- For SSL communications, different port numbers are provided for the well-known internet protocols:

Protocol	Normal Port	SSL Port
FTP	20/21	989/990
HTTP	80	443
IMAP	143	993

Protocol	Normal Port	SSL Port
POP3	110	995
SMTP	25	465
TELNET	23	992

Introduction to SSL …

- .NET 2.0 provides many classes to support SSL communication in the System.Net.Security namespace.
- The SslStream class is used to wrap a normal network stream to support SSL.
- The wrapping is done immediately after establishing connection.
- The SslStream has the following constructors:

SslStream (Stream)	• Boolean: if sets to true, the underline stream is closed after closing the SslStream. • RemoteCertificateValidationCallback : is a delegate that allows client to control server authentication. See .NET documentation for signature of the delegate. • LocalCertificateSelectionCallback: is a delegate that allows server to control client authentication.
SslStream (Stream, Boolean)	
SslStream (Stream, Boolean, RemoteCertificateValidationCallback)	
SslStream (Stream, Boolean, RemoteCertificateValidationCallback, LocalCertificateSelectionCallback)	

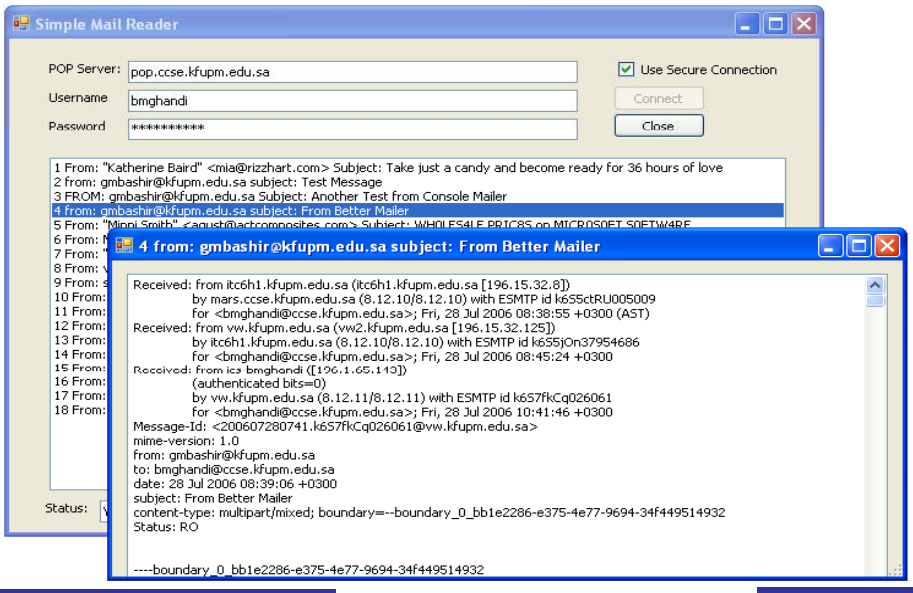
Introduction to SSL …

- The SslStream also has the following important methods:

void AuthenticateAsClient (string server) void AuthenticateAsClient (string server, X509CertificateCollection serverCertficates, SslProtocols enabledProtocols, bool checkCertificateRevocation)	Called by clients to authenticate the server. The authentication process uses the specified certificate collection and SSL protocol
void AuthenticateAsServer (X509Certificate serverCertificate) void AuthenticateAsServer (X509Certificate serverCertificate, bool clientCertificateRequired, SslProtocols enabledSslProtocols, bool checkCertificateRevocation)	Called by server to authenticate the client. The authentication process uses the specified certificate collection and SSL protocol

- The SslStream also has many useful properties including:
  CipherAlgorithm, CipherStrength, IsAuthenticated, IsEncrypted, RemoteCertificate, SslProtocol, etc.

Example : A Simple Mail Client



Example : A Simple Mail Client …

```
1.  using System;
2.  using System.Windows.Forms;
3.  using System.ComponentModel;
4.  using System.Net;
5.  using System.Net.Sockets;
6.  using System.IO;
7.  using System.Net.Security;
8.  using System.Security.Cryptography.X509Certificates;

9.  namespace SimpleMailReader {
10.     public partial class ListMessages : Form     {
11.         private TcpClient client;
12.         private Stream stream;
13.         private StreamReader reader;
14.         private StreamWriter writer;
15.
16.         public ListMessages()         {
17.             InitializeComponent();
18.         }
```


Example : A Simple Mail Client …

```
19.    private void btnConnect_Click(object sender, EventArgs e) {
20.      txtStatus.Text = "Checking for messages....";
21.      btnConnect.Enabled = false;    btnClose.Enabled = true;
22.      string response;
23.      string from = null;
24.      string subject = null;
25.      int messageCount = 0;
26.      try   {
27.          if (chkSsl.Checked)  {
28.              client = new TcpClient(txtHostname.Text, 995);
29.              stream = client.GetStream();
30.              stream.ReadTimeout = 120000;
31.              //For SSL Connection, the following lines are needed
32.              SslStream sslStream = new SslStream(stream, false,
33.               new RemoteCertificateValidationCallback(
34.                       CertificateValidationCallback));
35.              sslStream.AuthenticateAsClient(txtHostname.Text);
36.              stream = sslStream;
37.          } else  {
38.              client = new TcpClient(txtHostname.Text, 110);
39.              stream = client.GetStream();
40.              stream.ReadTimeout = 120000; //
41.          }
```

Example : A Simple Mail Client

```
42.          reader = new StreamReader(stream);
43.          writer = new StreamWriter(stream);
44.          response = reader.ReadLine(); //read welcome message
45.          if (response.StartsWith("-ER"))
46.          {
47.              txtStatus.Text = "Unable to connect to server";
48.              return;
49.          }
50.          IssueCommand("User " + txtUsername.Text);
51.          IssueCommand("Pass " + txtPassword.Text);
52.          response = IssueCommand("Stat");

53.          string[] tokens = response.Split(' ');
54.          messageCount = int.Parse(tokens[1]);
55.          if (messageCount > 0)
56.              txtStatus.Text = "You have " + messageCount + "
57.                 messages";
58.          else
59.              txtStatus.Text = " You have no messages";
```


Example : A Simple Mail Client

```
60.          for (int i = 1; i <= messageCount; i++)      {
61.              IssueCommand("top " + i + " 0"); //get only the headers

62.              while (true)
63.              {
64.                  response = reader.ReadLine();
65.                  if (response == ".")
66.                      break;
67.                  if (response.ToLower().StartsWith("from"))
68.                      from = response;
69.                  else if (response.ToLower().StartsWith("subject"))
70.                      subject = response;
71.              }
72.              lstMessages.Items.Add(i + " " + from + " " + subject);
73.          }
74.      }
75.      catch (Exception ex)
76.      {
77.          txtStatus.Text = "Unable to connect to server: "+ex.Message;
78.      }
79.  }
```

Example : A Simple Mail Client

```
80.     bool CertificateValidationCallback(object sender, X509Certificate
     certificate, X509Chain chain, SslPolicyErrors sslPolicyErrors)
81.     {
82.         if (sslPolicyErrors != SslPolicyErrors.None &&
83.             sslPolicyErrors != SslPolicyErrors.RemoteCertificateChainErrors)
84.         {
85.             txtStatus.Text = "SSL Certificate Validation Error!";
86.             return false;
87.         }
88.         else
89.             return true;
90.     }
91.     private void btnClose_Click(object sender, EventArgs e)
92.     {
93.         if (stream != null)
94.         {
95.             stream.Close();
96.             client.Close();
97.         }
98.         Close();
99.     }
```


Example : A Simple Mail Client

```
100.     string IssueCommand(string command)     {
101.         string response = null;
102.         try    {
103.             writer.WriteLine(command);
104.             writer.Flush();

105.             response = reader.ReadLine();
106.             if (response.StartsWith("-ER"))
107.                 txtStatus.Text = "Unable to connect to server";
108.         }
109.         catch (SocketException)  {
110.             txtStatus.Text = "Unable to connect to server";
111.         }

112.         return response;
113.     }

114.     private void lstMessages_DoubleClick(object sender, EventArgs e)
115.     {
116.         string title = (string)lstMessages.SelectedItem;
117.         ShowMessage sm = new ShowMessage(stream, title);
118.         sm.ShowDialog();
119.     }
```

This method is used to register with the DoubleClick event of the messages listbox.

Example : A Simple Mail Client

```
1.  using System;
2.  using System.Windows.Forms;
3.  using System.Net;
4.  using System.Net.Sockets;
5.  using System.IO;
6.  using System.Threading;
7.  using System.ComponentModel;

8.  namespace SimpleMailReader
9.  {
10.     public partial class ShowMessage : Form
11.     {
12.         private Stream stream;
13.         StreamReader reader;
14.         StreamWriter writer;
15.         Thread handler = null;
16.         int messageNumber = 0;
```


Example : A Simple Mail Client …

```
17.     public ShowMessage(Stream stream, string title) {
18.         InitializeComponent();
19.         this.stream = stream;
20.         reader = new StreamReader(stream);
21.         writer = new StreamWriter(stream);
22.         this.Text = title;
23.         string[] tokens = title.Split(' ');
24.         messageNumber = int.Parse(tokens[0]);

25.         handler = new Thread(new ThreadStart(ReadMessage));
26.         Control.CheckForIllegalCrossThreadCalls = false;
27.         handler.Start();
28.     }
29.     void ReadMessage()  {
30.         writer.WriteLine("retr " + messageNumber);
31.         writer.Flush();
32.         string response = reader.ReadLine(); //read status
```

Example : A Simple Mail Client …

```
33.     string message = "";
34.             response = reader.ReadLine();
35.             while (response != ".")
36.             {
37.                 message += response + "";
38.                 response = reader.ReadLine();
39.             }
40.             txtDisplay.Text = message;
41.         }
42.     }
43. }
```


Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- [RFC 1939]: http://www.ietf.org/rfc/rfc1939.txt
- Books
  - Richard Blum, C# Network Programming. Sybex 2002.
- Lecture notes of previous offerings of SWE344 and ICS343
- Some other web sites and books; check the course website at
  - http://faculty.kfupm.edu.sa/ics/alfy/files/teaching/swe344/index.htm