CS-410—Networks and Networking
Assignment 2, Due 5:00pm, 21 Sept

Fall '12

Objectives

In this lab you will examine several Application Layer protocols. In Part I you will do this by interacting with the relevant servers using `telnet`. In Part II you will observe HTTP in nature using the ethereal packet analyzer.

Due: Friday, 5:00pm, 21 Sept

Part I
Exploring Well-Known Applications using `telnet`

This assignment is structured as a lab. The Procedure section takes you thorough a sequence of explorations of the protocols. The Assignment section asks for a specific set of transcripts of sessions with the servers. You should read through the lab completely before starting so you will know what sessions you are required to gather before you start. That way you can probably gather them as you go. But you should *not* skip the Procedure section. The *point* of the lab is to carry out the whole set of explorations. The assignment section is asking you for a record of some subset of those explorations as evidence that you did them.

A note about font conventions. In the example interactions below, user input is typeset in *`a slanted typewriter font`* and program output is typeset in `an upright typewriter font`. The distinction is sometimes subtle.

> *Warning:* There are parts of this lab in which you must enter your password in circumstances in which it will be echoed to the screen. It is very important that you be aware of who is around you when you do these parts and that you make sure the terminal window in which you do this is destroyed before you leave the machine. (Depending on the terminal tool you are using, you may need to make sure that scroll-back lines are cleared as well.) Moreover, I recommend changing your password (using the `yppasswd` command) once you have completed the lab. (You almost certainly are overdue to change it anyway, right?)

Background

The `telnet` service provides an (unsecured) remote shell, functionally similar to `ssh`. As a result of it's lack of security (and the fact that it has been superseded by the secure `ssh`

service) it is usually configured to refuse connections. So the server, itself, is pretty much useless. But the `telnet` client, in essence, provides an interface between a TCP socket and the user—it simply copies user input to the socket and copies data from the socket to the display. Thus, it is a convenient mechanism for exploring those Application Layer protocols which use plain text messages over TCP. One simply `telnet`s to the appropriate server and interacts with it in the role of the client.

In this lab you will do this for HTTP, FTP, SMTP and POP.

Procedure

HTTP

You should have Section 2.2 of the text handy while doing this part.

1. Open a `telnet` connection to the HTTP server on quark:

   ```
   bash$ telnet quark.cs.earlham.edu 80
   ```

   The first argument is the hostname (more properly this would be `quark.cs.earlham.edu` or, perhaps, `www.cs.earlham.edu`, or, most properly, just `cs.earlham.edu`). The second argument is the port number, the standard HTTP service port, 80. If this is omitted, `telnet` will attempt to connect to port 23, the standard `telnet` port, and the connection will be refused.

   You should get the response

   ```
   Trying 159.28.230.3...
   Connected to cs.earlham.edu.
   Escape character is '^]'.
   ```

   The `telnet` client is waiting for your input. The significance of the escape character (which is ⟨`ctrl`⟩-`]`, *not* ⟨`Esc`⟩) is that it will allow you to break out of the connection and talk to the client itself. If, for some reason, your connection gets hung, you can escape from the session and tell the client to close it. (At the `telnet>` prompt enter '?' for help.)

2. Using the `GET` request, retrieve an HTML page. (Choose something short; the page in the example works well.) Initially, you should make this a version 1.0 request.

   GET /∼bchaudhry/cs410/bang.html HTTP/1.0
   ⟨`empty line`⟩

   You signal the end of the request header by sending an empty line. The page should be dumped to your terminal and the HTML server should close the `telnet` connection. Examine the response message and compare it to the response discussed in the text.

3. Try this again with a `HEAD` request. This will return only the header of the response.

```
bash$ telnet quark.cs.earlham.edu 80
Trying 159.28.230.3...
Connected to cs.earlham.edu.
Escape character is '^]'.
HEAD /~bchaudhry/cs410/bang.html HTTP/1.0
⟨empty line⟩
HTTP/1.1 200 OK
...

Connection closed by foreign host.
```

4. Repeat the `GET` request, but request HTTP version 1.1 this time. (Don't change anything but the HTTP version.)

```
bash$ telnet quark.cs.earlham.edu 80
Trying 159.28.230.3...
Connected to cs.earlham.edu.
Escape character is '^]'.
HEAD /~bchaudhry/cs410/bang.html HTTP/1.1
⟨empty line⟩
HTTP/1.1 400 Bad Request
...

Connection closed by foreign host.
```

5. Review the text's coverage of the HTTP protocol to determine the reason that this is a Bad Request. Repeat the request, this time with the required header line(s).

6. If you did not include a "`Connection: close`" header line in the last exchange (it is not a required line) you are likely to have noticed the delay between the end of the response message and the closing of the connection. Since this is HTTP 1.1, the default (in absence of an explicit `close`) is to use persistent mode. The delay is the interval during which the HTTP server will wait for the HTTP client to make additional requests. (Note that it is actually an error for the client to not explicitly request that the connection be closed unless it intends to make additional requests.) If you did not observe a persistent connection on your last request, try it again without the "`Connection: close`" header line. In any case, try the request with "`Connection: close`" as well.

7. The goal now is to make multiple requests over the same data connection using persistent mode. The tricky part of this is that the timeout is often quite short and the port is likely to be closed before you can type your follow-up request. One way to get around this is to type your sequence of commands into an editor window and then cut-and-paste them into the `telnet` window.

In the editor or your choice type a `GET` request for the ∼`bchaudhry/cs410/bang.png` image, specifying `Connection: close` and terminating it with an empty line. Now highlight the lines of the request (including the terminating empty line) in the editor window (to get it into the X cut-buffer), open the `telnet` session with the HTTP server and type in the `GET` request for the ∼`bchaudhry/cs410/bang.html` page (without a "`Connection: close`" line, followed by an empty line. As soon as the response completes, while the server is still waiting for additional requests insert the `GET` command for the image into the `telnet` window (most probably by center-clicking the mouse). You should get the image file in response following which the connection should be closed. (A caveat, the image file contains binary data which may drive your terminal program into fits. If it does, try again but just get the html page twice.)

8. In waiting for the first response before making the second request, you are using the persistent connection in non-pipelined mode. To use pipelined mode all you have to do is to send the second command without waiting for the response to the first. To do this, insert the `GET` request for the ∼`bchaudhry/cs410/bang.html` page, followed by an empty line, in front of the `GET` request for the ∼`bchaudhry/cs410/bang.png` image in the editor window. Highlight both of these requests, open the `telnet` session with the HTTP server, and insert the requests into the `telnet` window. You should get both files returned in sequence, following which the connection should be closed.

FTP

You should refer to Section 2.3 of the text while doing this section.

File Transfer Protocol uses two ports, one for control communications (port 21) and one for data communications (port 20). While the FTP daemon and the FTP agent are in their normal server and client roles for the control port, their roles are reversed in using the data port—the FTP agent is responsible for listening on the data port and the FTP daemon makes the request to set up the connection. This complicates exercising the FTP daemon with `telnet` because you have no way of listening on a port with (most) `telnet` clients.

9. Open a `telnet` connection to quark's FTP port:

   ```
   telnet quark.cs.earlham.edu 21
   ```

   You will get a `220 FTP Server Ready.` response.

10. Register yourself as the user:

    `USER` ⟨`your-userid-here`⟩

    You should get a "`331 Password required for` ⟨`your-userid-here`⟩`.`" response. (If not, you've got real problems. Don't tell anyone but the sysadmins.)

11. Send your password. *Warning:* there is no way to do this except in visible text. Make sure you are in a reasonably discrete location when you do this and make sure you close this window when you are done with this part of the lab.

`PASS ⟨your-passwd-here⟩`

You should get a couple of "`230`" responses, letting you know the terms of use and that you are logged in.

12. Here is where you first run into the need for the data connection. Try to get a directory listing for this directory:

    `LIST`

    You should get "`425 Can't build data connection: Connection refused`" as a response. Your reason message may vary, but the underlying cause is still the inability to build the data connection.

13. The way around this is to use *passive* mode. This tells the FTP daemon that it is going to have to be in the server role for *both* connections. (This is included in the protocol to accommodate firewalls that will not allow incoming TCP connections.) Put the server in passive mode:

    `PASV`

    The response you get will tell you the address of the port the server will listen to for the next data connection:

    `227 Entering Passive Mode (159,28,230,3,156,149).`

    Here the IP address and port number are both given in "dotted octet" form. The '`159,28,230,3`" is almost the familiar form for the IP address of quark. To decode the port number "`156,149`", though, you have to understand that it is a sixteen bit number given as the decimal representation of the most significant byte followed by the decimal representation of the least significant byte. So the actual port number (in this case) is

    $$156 * 256 + 149 = 40085$$

    In order to use this port, you will need to use a second instance of `telnet` to connect to it. The easiest way to do this is to open a second shell window and run the second `telnet` in it.

14. Compute the port number for the Passive Mode response you actually got. Open a second shell window but don't initiate the `telnet` yet.

15. In the first `telnet` session (the control session) enter the `LIST` command again. You will initially get no response—the FTP daemon is waiting for you to open the data connection.

16. In the new shell (the data session) `telnet quark.cs.earlham.edu ⟨portno⟩`, where ⟨`portno`⟩ is the port number you computed from the Passive Mode response. You should get a directory listing in the data session and

```
150 Opening ASCII mode data connection for file list
226 Transfer complete.
```

in the control session.

17. You must set Passive mode for each data transfer, as each one uses a new TCP connection. Enter passive mode again and retrieve a short file.

```
PASV
227 Entering Passive Mode (159,28,230,3,176,126).
RETR bang.html
150 Opening ASCII mode data connection for bang.html (187 bytes)
226 Transfer complete.
```

18. Close the session with `QUIT`.

You can find the complete set of commands in the FTP protocol in Section 4 of RFC-959 (`ftp://ftp.rfc-editor.org/in-notes/rfc959.txt`). (This is an `ftp` host which does not support `http` retrieval. Take a moment to contemplate the pleasure of using `ftp` to retrieve this RFC.)

SMTP

During this phase of the lab you should consult Section 2.4 of the text and RFC-2821 (`ftp://ftp.rfc-editor.org/in-notes/rfc2821.txt`) and, if you wish, RFC-822 (`ftp://ftp.rfc-editor.org/in-notes/rfc822.txt`).

19. Simple Mail Transport Protocol (SMTP) runs on all of our machines. While most properly you should communicate directly with the target mail server (`cs.earlham.edu` for mail to the `cs` domain) but the SMTP server will relay mail that it receives appropriately. Thus you can, and should, run on the local machine, which you can address as `localhost`.

```
bash$ telnet localhost 25
```

You should get a "`220...`" message confirming that you are connected.

20. You can open the session with the `HELO` command. You should include the hostname of the machine you are connecting from.

```
HELO kleene.cs.earlham.edu
250 kleene.cs.earlham.edu Hello localhost.localdomain [127.0.0.1], ...
```

21. Each mail message is initiated by identifying the sender's address...

```
MAIL FROM: <your-userid@cs.earlham.edu>
250 2.1.0 your-userid@cs.earlham.edu...  Sender ok
```

22. . . . and the recipient's address (Send the message to yourself.)

```
RCPT TO: <your-userid@cs.earlham.edu>
250 2.1.5 your-userid@cs.earlham.edu...  Recipient ok
```

    The SMTP server is usually quite lenient about the recipient's address. You can typically send just the userid and it will expand this to the full default address for its domain.

23. The body of the message is introduced with a `DATA` command followed by lines of arbitrary ASCII text. It is terminated with a line containing just a '.' on a line by itself

```
DATA
354 Enter mail, end with "." on a line by itself
This is the body of the message
It contains four lines, one of which is blank

(The terminating dot line is not part of the body, but this line is.)
.
250 2.0.0 h0M2Ybd10296 Message accepted for delivery
```

24. Following this, additional mail messages can be sent (starting again with the `MAIL TO:` line) or the session can be ended with `QUIT`

```
QUIT
221 2.0.0 kleene.cs.earlham.edu closing connection
Connection closed by foreign host.
```

25. If you haven't already, send yourself a message (at your usual address). Give the mail system a minute or so to pass it along and pick it up. Although this is dependent on the mail server, if it is following the standard the message will not show up as being addressed to you but rather to something like "undisclosed-recipients:". This is because the addresses you gave to the SMTP server are used for mail transport only. The addressing information that is communicated between the mail agents at both ends is contained in the message header, which, from SMTP's perspective, is just more of the `DATA` section. The content of the message header is standardized in RFC-822. We only care about three header lines here: `To:`, `From:` and `Subject:`. These are separated from the actual body of the message by an empty line. Send yourself another message, this time including at least the `To:` field.

```
...
DATA
354 Enter mail, end with "." on a line by itself
To:  your-userid@cs.earlham.edu
From:  your-userid@cs.earlham.edu
```

```
Subject:  RFC 822 Mail headers
<blank line>
This is the body of the message
.
250 2.0.0 h0M2Ybd10296 Message accepted for delivery
```

When you pick this up, you should see the "To:", "From:" and "Subject:" data you expect.

26. Look at the `Received:` header lines of the message. (If you use a mail agent with fascist tendencies you may have to coerce it to show you the header.) What hosts has your message passed through on the way to your mail drop? What is the order in which the `Received:` headers appear?

27. Browse RFC-2821. Look, in particular, at Section 2.2, (*The Extension Model*) and Section 4.1 (*SMTP Commands*). What happens if you open a session with `EHLO` rather than `HELO`?

28. The mechanism SMTP uses to delineate the end of the `DATA` section, a '.' on a line by itself, belies the antiquity of its origins. (This dates back at least to the unlamented editor `ed`—give it a try, it's still lurking about on almost all unix systems.) One drawback to this approach shows up if you decide to send a message with lines containing just a '.'. Find out how to do this by browsing in the RFC.

29. By the time you have gotten to this point in the lab you have probably figured out that there is nothing that requires the client of the SMTP service to be honest about reporting the sender's address, etc. Lying about this is known as *spoofing* and is so obviously simple to do that it really isn't particularly interesting. Read Section 7 (*Security Considerations*) of RFC-2821, particularly Section 7.1 (*Mail Security and Spoofing*). What are the reasons for not even trying to make spoofing more difficult?

POP3

This phase of the lab references Section 2.4.4 of the text.

Post Office Protocol provides a means for transitory hosts to pull mail from a mail server on demand. If you read your mail on a system that is permanently attached to the network, then you likely don't use POP. Even if you do read your mail on a transitory host, you may well use IMAP (see the text). As POP is more or less the lowest common denominator, we will look at it.

30. You will need to contact the POP server of the system which handles your mail. (For the `cs` domain, this is, again, `quark`.)

```
bash$ telnet quark.cs.earlham.edu 110
...
+OK POP3 cs.earlham.edu v2001.78 server ready
```

31. As with FTP, you identify your userid and password with `user` and `pass` commands. (*Once again your password will appear in plain text in your terminal window. You must exercise care here.*)

```
user your-userid
+OK User name accepted, password please
pass your-soon-to-be-outdated-password
+OK Mailbox open, 27 messages
```

The `+OK` indicates that the command was executed successfully. Errors are flagged with `-ERR`. The POP server tells you how many messages are in the maildrop.

32. You can get a listing of the lengths (in octets) of the messages in the maildrop with the `list` command

```
list
+OK Mailbox scan listing follows
1 713831
2 4541
3 1761
...
27 691
.
```

All multiline responses, such as this one, follow the `ed` delimiter convention—the end of the response is indicated with a line containing only '.'.

33. To retrieve a message, use `retr` and the message number

```
retr 15
+OK 719 octets
Return-Path:  <bchaudhry@cs.earlham.edu>
Received:  from kleene.cs.earlham.edu (kleene.cs.earlham.edu [159.28.230.27])
by quark.cs.earlham.edu (8.12.3/8.12.3) with ESMTP id h0M1YAIZ041032
for <bchaudhry@cs.earlham.edu>; Tue, 21 Jan 2003 20:34:10 -0500 (EST)
(envelope-from bchaudhry@cs.earlham.edu)
Received:  from kleene.cs.earlham.edu (localhost.localdomain [127.0.0.1])
by kleene.cs.earlham.edu (8.11.6/8.11.6) with ESMTP id h0M1VTd10216
for <bchaudhry@cs.earlham.edu>; Tue, 21 Jan 2003 20:31:53 -0500
Date:  Tue, 21 Jan 2003 20:31:53 -0500
From:  bchaudhry@cs.earlham.edu
Message-Id:  <200301220131.h0M1VTd10216@kleene.cs.earlham.edu>
To:  undisclosed-recipients:;
Status:  RO

first line
second line
```

```
last line
.
```

Again, the final '.' is the not part of the message but, rather, marks the end of the response.

34. Retrieving a message does not remove it from the maildrop. That must be done explicitly with the `dele` command

```
dele 15
+OK Message deleted
```

35. Finally, you exit with `quit`

```
quit
+OK Sayonara
Connection closed by foreign host.
```

Part II
Wireshark Lab 2

This part is the text's Wireshark Intro and HTTP labs. You can get the lab either from the text's website or from the class web:

`http://cs.earlham.edu/~bchaudhry/teaching/CS410`

To be able to capture packets on the interface you need to run `wireshark` as root. The members of this class are all members of the `cs410` group, which has `sudo` privileges for `wireshark`. (You can check which groups you are a member of with the `groups` command.) `sudo` is a utility that provides a way for designated users to run specific commands as other users. In this case, those of us in the `cs410` group may run `/usr/bin/wireshark` as `root`, allowing access directly to the ethernet interface.

A caveat. While, given our network topology, the risk is relatively minor, all mechanisms that permit one to examine packets captured directly from the network compromise system security significantly. (That's why `root` privileges are required.) You are responsible for using this tool only for the legitimate purposes of this class. It is a breach of academic honesty to use it in any other way or for any other purpose. Irresponsible use will be treated as a serious breach of Earlham's Academic Integrity policy.

Run `wireshark` under sudo: `sudo /usr/bin/wireshark`
You will be required to enter your password in order to proceed. Note that you do not *have* to run wireshark as root, but since ordinary users don't have access to the raw ethernet interface you will not be able to capture any traffic. You can use it this way to analyze data that has been acquired and dumped previously. One reason for doing this would be to use the dumps provided for these labs in the text's web. *If, for some reason, you feel the need*

to work from these dumps rather than from a live capture you should talk with me before proceeding.

If you did not do `WireShark` Lab 1, you should work through the *Getting Started* (`intro`) lab to get familiar with it. You should complete the questions in the What to hand in section, but you are not required to hand these in at this time.

You should next move on to the HTTP lab, completing the each of 19 questions that occur in the text of the lab. These you will hand in.

Assignment

Part I

Turn in transcripts of each the following sessions and answers to the interspersed questions. To make transcripts you may cut-and-paste from the `telnet` window to a file or you can run the `script` command prior to running `telnet`. (See `man script`.) You can print the transcripts two-up using `a2ps`. (See `man a2ps`.)

1. A transcript of an HTTP session in which you retrieve both `/~bchaudhry/cs410/bang.html` and `/~bchaudhry/cs410/bang.png` using pipelined persistent mode.

2. A transcript of an FTP session (only the control session) in which you retrieve the file `~bchaudhry/cs410/bang.html`. *IMPORTANT:* You *must* edit this transcript to remove your password. Failure to do so will result in no credit for this part of the lab (and a uniform substitution of 'w' for 'e' in every ASCII file in your filespace).

3. Look up the `PORT` command in RFC-959. What does it do? Is it useful for what we are doing in this lab? Why or why not?

4. A transcript of an SMTP session which you initiate with `EHLO`. What extended SMTP commands does the server support?

5. A transcript of an SMTP session (possibly the same one) in which you include at least `To:`, `From:` and `Subject:` RFC-822 header lines and in which the body of the message, as delivered, contains at least one line containing just a '.'.

6. A transcript of a POP session in which you retrieve the message you sent in the previous step. (Again, you *must* delete your password from the transcript.)

Part II

Submit your answers to the 19 questions in the Wireshark HTTP lab.