FTP

Or, how to get a file from point A to point B

FTP OVERVIEW

- FTP is the client/server protocol used to ship files across the network.
- FTP is an Application Layer protocol, implemented in plain text, having only one Transport Layer requirement: a reliable and ordered data stream. TCP is most commonly used.
- The standard FTP TCP port number is 21, though there are some complexities to FTP which call for additional ports.

PRIMARY FTP RFC'S

- The current specification for FTP is RFC 959.
  - `http://tools.ietf.org/html/rfc959`
- There are additional RFC's, including security extensions:
  - `http://tools.ietf.org/html/rfc2228`
- Internationalization:
  - `http://tools.ietf.org/html/rfc2640`
- Extensions:
  - `http://tools.ietf.org/html/rfc3659`

FTP OVERVIEW

- FTP is an interesting protocol, because the communications are actually split across two connections.
  - One connection is known as the command or control channel, and is used to run the entire session with a series of commands. This stream utilizes the telnet protocol, and can be attached to directly or via an FTP client program.
  - The other is a transient connection, known as the data connection. This connection is used strictly for sending the file data across, and by default, is dynamically created and destroyed for each file transferred.

FTP USAGE

- Generally, FTP is used via some form of client, be it a command line client or a graphical client.

- In this class, the focus will be on the commonly available command line client. This will permit a closer link to the protocol, as well as access to special commands and debugging options.

- The graphical clients are quite straightforward and easily learned, particularly after thorough instruction on the command line client.

STARTING THE CLIENT

- The first step is starting up the FTP client. The simplest way to do this is to run the ftp command with the ftp server address as the argument:
  - `ftp server1.example.com`
- This instructs the FTP client program to connect to server1.example.com and initiate an FTP session.
- Note: most clients will accept URL's as well:
  - `ftp ftp://server1.example.com/pub/`
- This example connects and automatically switches to the `pub` folder

AUTHENTICATING

- After connecting to an FTP server, the next step usually involves authenticating with a username and a password. There are generally two choices for authentication:
  - Anonymous - If enabled, logging in with a user name of “anonymous” permits guest access. Usually read only access.
  - Local account - Using credentials for an ftp account on the system. Usually allows full read/write access.
- Once authenticated, commands can be issued to move around the file server, view directory listings, transfer files and more.

LOCAL/REMOTE

- One important thing to keep straight in your head when connected to an FTP server is the notion of local and remote directories.

- You're basically interacting with two different file systems:
  - **Remote**: the filesystem on the FTP server
  - **Local**: your filesystem, on the machine running the FTP client software. Downloaded files are saved to this folder. Uploaded files must come from this folder.

LOCAL/REMOTE

- Most ftp client commands operate on the remote filesystem, but a few are important to the local filesystem:
  - `lcd`: Local Change Directory. This command changes the local current directory.
  - `lpwd`: Local Print Working Directory. Also, "`lcd .`" will usually show the local current directory.

NAVIGATION

- The basic idea with FTP is to use the `cd` command to move around the remote filesystem, and when you wish to transfer a file, you use:

- `get <remotename> [localname]`: Download a remote file to your local filesystem, optionally renaming in the process.

- `put <localname> [remotename]`: Upload a file from your local machine to the fileserver, optionally renaming.

DISCONNECTING

- When all transfers are complete, disconnecting from the server is simple:
  - `quit`: Terminate FTP session and exit client
  - `bye`: Same as `quit`
  - `close`: Terminate FTP session but remain in client
  - `disconnect`: Same as `close`

LAB 1

1) Connect to `ftp.gnu.org`. Locate and download `wget-latest.tar.gz`. Try uploading a file. Does it work? Why or why not, do you suppose?

2) Connect to `server1.example.com` as stationX, where X is your station number (Type `hostname` to see your station number). Try uploading a file.

3) Browse through RFC 959. Anything interesting?

FTP REPLIES

- FTP replies consist of a three digit number, possibly followed by a brief text message, normally all contained on one line, though multiple lines are possible depending on the server software and the command sent. Examples:

- `220 GNU FTP server ready.`
- `215 UNIX Type: L8`
- `257 "/"`
- `200 Switching to Binary mode.`

FTP REPLIES

- Each digit of the code has special meaning
- The first digit indicates the overall response: good, bad or incomplete.
- The second digit indicates the category of response, such as syntactical, connection, information, etc.
- The third digit provides additional detail about the category referred to in the second digit.

FIRST DIGIT

- 1__: Positive preliminary - action beginning
- 2__: Positive completion
- 3__: Positive intermediate, need more information
- 4__: Temporary negative completion
- 5__: Permanent negative completion

SECOND DIGIT

- `_0_`: Syntax errors
- `_1_`: Informational responses - status, help
- `_2_`: Connection responses - control and data
- `_3_`: Authentication and accounting
- `_4_`: Unspecified
- `_5_`: Filesystem responses

THIRD DIGIT

- The third digit provides additional meaning to the category given by the second digit. Accepted values given in RFC. Examples:
  - `500` - Syntax error, unrecognized command
  - `501` - Syntax error, invalid arguments
  - `502` - Command not implemented
  - `503` - Bad command sequence
  - `504` - Command parameter not implemented

TEXTUAL RESPONSE

- The textual component of the response is generally meant for human consumption, though there are a handful of responses which should be parsed by the client.

- For extra credit, note that if the reply text is longer than one line, every response except the last will start with the same code, followed immediately by a hyphen. The last line will not contain the hyphen:

```
250-First line

250-Second line

250 Last line
```

RESPONSES

- The official list of response codes and suggested texts is available in the RFC: ➥RFC 959 §4.2.2

- Additional response codes are possible for future use, and research into the server software will be necessary to understand their meaning.

CLIENT DEBUGGING

- Turning on debug mode in the client can be extremely useful to see the sequence of commands and replies.
  - `ftp -d ...`: Start up ftp client in debug mode
  - If the ftp client is already running, type "`debug`" to turn on debugging.
- All lines starting with "-->" are commands sent to the server.

LAB 2

1) For the following exercises, turn on debugging and analyze the control communications.

2) Connect to `ftp.gnu.org`. Locate and download `wget-latest.tar.gz` again.

3) Browse through RFC 959. Focus in on sections 2.3, 3.2, 3.3 and 4.2.

FTP CAVEAT #1

- FTP has a couple of caveats, and the first one is this:
  - There are two (technically four, but the other two aren't commonly used anymore) ways of encoding the data for transfer:
    - **Binary** - send data over precisely as stored on sending machine - no rewriting.
    - **ASCII** - convert data from local format to standardized NVT-ASCII format (see telnet protocol), transmit, and at the receiving end, convert back to a local storage form.

FTP CAVEAT #2

- The second FTP caveat:
  - How data connections are established. This single issue is the number one cause of FTP failures.
    - **Active** - Upon initiating a data transfer, the client provides a data port to the server, and the server connects to the client.
    - **Passive** - Upon data transfer, the server provides a port to the client, and the client connects to that port on the server.

ACTIVE/PASSIVE

- The active/passive modes in FTP are an artifact of an out of band protocol (meaning data is transferred separately from commands, unlike a protocol like SMTP, which is in-band).
- Active mode is commonly the default, and works fine in many situations. So long as the server can connect back to the client on the given port, all is well.
- What about firewalls, though?

FIREWALLS

- Firewalls are exactly the issue. Generally, firewalls are locked down to disallow inbound connections except to specific ports on a machine. And the randomly generated ports for active FTP present a serious headache for firewall administrators.

- Passive mode addresses this issue, because in passive mode, connections are always client to server, and outbound connections on a client usually aren't blocked by a firewall.

- Of course, this means the firewall on the ftp server needs to be opened up for any of those data ports. Most FTP servers provide configuration directives to control data port ranges.

FIXUP

- You should be aware that Cisco has a technology known as Fixup. Fixup is a feature that can be enabled on Cisco devices on a per protocol basis to analyze the protocol traffic and perform automatic self reconfiguration.

- Huh?

- Basically, if configured, a Cisco firewall can automagically open inbound ports based on passive mode FTP session commands! If it knows an internal machine is expecting a data connection from an external machine, the firewall will allow it!

PERFORMANCE

- Since we're on the topic, it should be noted that there is a certain cost associated with transferring a file with FTP.

- A control session must be established, then a data connection must be negotiated, connected, used and torn down again. For each file!

- If there are lots and lots of files to be transferred, it's often more efficient to zip them all up and transfer one large file, rather than dealing with lots and lots of separately negotiated, opened and closed data connections.

SWITCHING ACTIVE/ PASSIVE

- Switching between active and passive mode is simple:
  - `passive`: toggle between passive and active mode.
  - `status`: see all current session settings.

ACTIVE/PASSIVE

- When in active mode, the client will issue a PORT command to the server, supplying the necessary connection parameters for the server to connect back to the client.

- When in passive mode, the client will issue a PASV command, and the server will reply with the connection parameters for the client to use to connect back to the server. Then the client can issue a data transfer request and the server will prepare a connection on the PASV supplied parameters.

- Let's look at these various communications...

PASSIVE RESPONSES

- When a client is in passive mode, responses often look like:
  - `227 Entering Passive Mode (140,186,70,20,142,131)`
  - `229 Entering Extended Passive Mode (|||32388|)`
- The numbers in the parenthesis are what the client uses for the data connection parameters.

PASSIVE

- `227 Entering Passive Mode (140,186,70,20,142,131)`
- The first four numbers in the parenthesis tell the client each octet of the IPv4 address, in this case 140.186.70.20.
- The remaining two numbers are the high and low bytes of the port number. To get the port number:
  - `high * 256 + low`
  - `142 * 256 + 131 = port 36483`

EXTENDED PASSIVE

- `229 Entering Extended Passive Mode (|||32388|)`
- Extended passive mode is defined in RFC 2428. This extension was added mainly to support IPv6.
- The above response tells the client to connect to the server with the same network protocol and address on port 32388.

ACTIVE COMMANDS

- Active mode requires the client to tell the server what connection parameters to use for the data connection.

- You normally need to be in debug mode to see these commands:

  - `PORT 192,168,1,10,181,163`

  - `EPRT |1|192.168.1.133|63138|`

PORT

- Using the `PORT` command tells the server the parameters for connecting back to the client:
  - `PORT 192,168,1,10,181,163`
- Similar format to the passive response: each octet from the IPv4 address, followed by high and low order bytes of port number.

EPRT

- `EPRT` is defined in RFC 2428 as well, again for supporting IPv6:
  - `EPRT |1|192.168.1.133|63138|`
- This command tells the server to use IPv4 (the first 1. A 2 would indicate IPv6), connecting to 192.168.1.133 on port 63138.

LAB 3

1) Using server1.example.com and your stationX login, try uploading and downloading various files in both active and passive modes. Use debug mode to follow the FTP dialogue.

2) Try transferring a text file between systems and note the difference between binary and ascii mode. Generally, unix to windows and vice versa illustrates the behavior the best, due to line ending encoding.

ADDITIONAL COMMANDS

- Some additional and useful FTP commands:
  - `user` - Change or specify a username
  - `hash` - Toggle printing of hash progress meter
  - `mget` - Download multiple files at once. Supports wildcards.
  - `mput` - Upload multiple files at once. Supports wildcards.
  - `prompt` - Toggle prompting for each file in multiple operations.

SERVER LOGS

- There are many different FTP server programs out there, and there's no chance to cover all of them.

- Most provide basic logging options, and some include the ability to log full command conversations, which can occasionally be useful for debugging.

- Be sure to understand the capabilities of the FTP server being debugged, as the logs could be invaluable.

FTP SECURITY

- The notion of “security” when it comes to basic FTP is a joke.
- There is no inherent security in the FTP protocol whatsoever.
- Nothing is encrypted and there are no strong identification practices
- There are, however, a couple of related protocols that do provide various additional securities…

FTPS

- FTPS is a fairly rarely used extension to FTP that allows for negotiating an encrypted FTP session, via several newly defined commands.
- RFC 2228 officially defines the security extensions for FTP.
- Using FTPS encrypts the command and optionally the data channels with SSL or TLS.
- Not many servers support these features, as most public FTP servers are simply download repositories of public information.
- Consider: fixup + encrypted command channel? Will it work?

SFTP

- SFTP isn't actually FTP at all. :)
- SFTP is an FTP-like interface which is provided by the OpenSSH server.
- It can be accessed with the OpenSSH `sftp` command.
- Commands and behavior are very similar to original FTP, but that's where the similarities end.
- SFTP runs on top of the completely encrypted SSH protocol on port 22. Only one port to open on the firewall!

LAB 4

1) Use sftp to transfer a file to server1.example.com. Note any differences between the sftp and ftp clients.

2) Browse through RFC 2228.

3) Turn on hashmark printing. Upload all files from your home directory to server1.example.com.

4) Use a wildcard pattern to download 3-5 files at once from ftp.gnu.org.

```
slideshow.end();
```