secure|set
A FLATIRON SCHOOL

SecureSet and Flatiron School have joined forces with a mission of enabling people to pursue careers they love.

We are tackling the cyber skills gap together, using our collective strengths to provide transformative cybersecurity education.

Networking 100

OSI Layers 5 through 7

LAYER 3 PROTOCOLS (REVIEW)

- Global addressability and reachability
  - Remote Attacks / spoofable
- Routing Protocols and ways to exploit them
  - Route Hijacking, Data blackhole-ing, More Specific Wins!
  - Subnet Numbers (No host Information)
- Routers process packets based on destination address.
  - DDOS Direct/Indirect

flatironschool.com

LAYER 3 SERVICES (REVIEW)

ARP

- Have your IP, give me your MAC

ICMP

- ping (echoing)
- traceroute (ttl expire, footprinting)

DNS

- Have your name, Whats your IP?

DHCP

- Give me an IP Address / Mask / Default Gateway / DNS / Domain Name / etc

INTERNET RECAP

Most connections are unauthenticated by default

- TLS only provides encryption and one-way authentication of the server (see notes)

Most of them spoof-able

All of them are essential for basic communication.

While TLS/SSL does provide a level of authentication, most implementations are one-way authentication, in that the client authenticates the certificate provided by the TLS-enabled server, but no other credentials or authentications exist. This can be viewed as appropriate authentication (of the server), but it is potentially susceptible to Man-in-the-Middle and other attacks. More detail will be covered in CRY200 and later in the Network course.

TRANSPORT RECAP

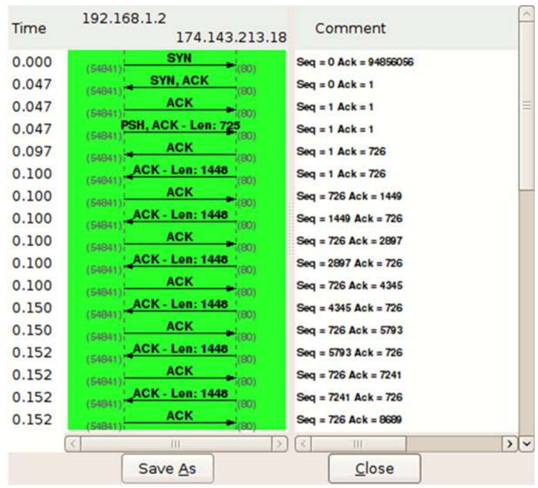
UDP	**TCP**
Connectionless	Connection Oriented (handshake)
Unreliable	Reliable (retransmissions / acks)
No flow control (out of band)	Flow Control (Sliding Window)
Streamlined	Suitable for file/content transfers
Suitable for time critical applications	

TCP Sequence Numbers

TCP SEQUENCE NUMBERS

The SEQ and ACK numbers here are *relative* numbers, not actual.

Sequence number is incremented for each byte of application data sent (payload)



http://media.packetlife.net/media/blog/attachments/429/tcp_flow.png

When using Wireshark to trace packets, right click on a TCP packet, choose Protocol Preferences and uncheck Relative Sequence Numbers.

TCP SEQUENCE NUMBERS

#1 Client SYN relative zeros (it's actually a random number on each side)
#2 Server (sseq 0, sack 1)
#3 C (cseq 1, cack 1)
#4 C http request - 725 bytes (cseq1,cack 1)
#5 S ack request (sseq 1, sack 726)
#6 S http reply – 1448 bytes (sseq 1, sack 726)
#7 C ack (cseq 726, cack 1449)
#8 S http reply – 1448 bytes more (sseq 1449, sack 726)
#9 ?

RULES:

- Increase own sequence number after the other side acknowledges reception of data you transmitted. (how much data have I transmitted)
- Increase own ack value, after data is received from a peer (how much data have I received?)

https://taosecurity.blogspot.com/2004/01/tcp-sequence-numbers-explained-today-i.html

Above: Richard goes in-depth and uses real numbers….easier (as in slides) to use small numbers because it can make it confusing. The simple numbers in the slides are not real… just an example

ABSOLUTE VS RELATIVE SEQUENCE NUMBERS

```
Info
50379 → 80 [SYN] Seq=3014546115 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
80 → 50379 [SYN, ACK] Seq=2486097343 Ack=3014546116 Win=8192 Len=0 MSS=1440
50379 → 80 [ACK] Seq=3014546116 Ack=2486097344 Win=66048 Len=0
GET /?ld=d3V5-lM_QFyt7PV9Ddn6ipKzVUCUzdKAcGjelTJMAJAWeEE5QGTXe5U9-jM-34Wo-g
80 → 50379 [ACK] Seq=2486097344 Ack=3014547212 Win=132096 Len=0
80 → 50379 [ACK] Seq=2486097344 Ack=3014547212 Win=132096 Len=1452 [TCP seg
80 → 50379 [PSH, ACK] Seq=2486098796 Ack=3014547212 Win=132096 Len=566 [TCP
HTTP/1.1 302 Object Moved
50379 → 80 [ACK] Seq=3014547212 Ack=2486099367 Win=66048 Len=0
```

```
Info
50379 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
80 → 50379 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1440 WS=256
50379 → 80 [ACK] Seq=1 Ack=1 Win=66048 Len=0
GET /?ld=d3V5-lM_QFyt7PV9Ddn6ipKzVUCUzdKAcGjelTJMAJAWeEE5QGTXe5U9
80 → 50379 [ACK] Seq=1 Ack=1097 Win=132096 Len=0
80 → 50379 [ACK] Seq=1 Ack=1097 Win=132096 Len=1452 [TCP segment
80 → 50379 [PSH, ACK] Seq=1453 Ack=1097 Win=132096 Len=566 [TCP s
HTTP/1.1 302 Object Moved
50379 → 80 [ACK] Seq=1097 Ack=2024 Win=66048 Len=0
```

Application Protocols

OSI MODEL - LAYERS 5-7

Note that the TCP/IP model doesn't make a distinction between the OSI's layers 5-7.

> (From the point of view of a networking stack, everything about the transport layer is handled by the application.)

There is more gray area in the upper layers.

It's good to keep in mind that the OSI model is just a model, to help think abstractly about network communication.

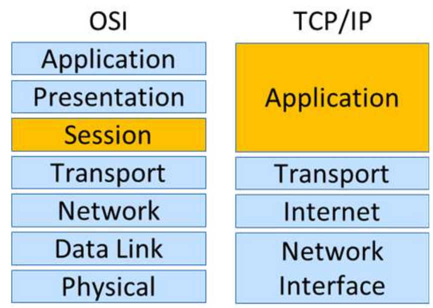
OSI LAYER 5 - SESSION LAYER

Handles the creation, use, and tear-down of a “session.”

Allows for stateful connections

Token management

Think cookies



flatironschool.com

OSI LAYER 6 – PRESENTATION LAYER

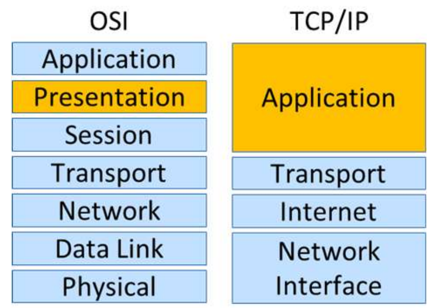
Preserve the syntax of the transmitted data

Compression/decompression

Encryption/decryption

Blur between presentation and application layer

- Either could handle the identification of character encoding, for example.

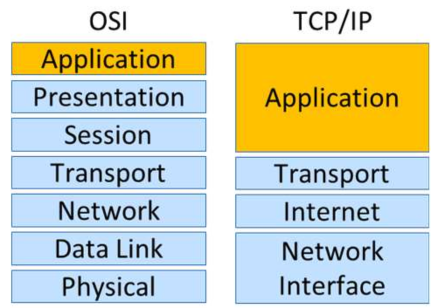


OSI LAYER 7 – APPLICATION LAYER

Level at which applications have specific communication protocols

Examples: HTTP, FTP, SSH, SMTP…

Think “User interface”



Encapsulation

ENCAPSULATION CASE STUDY: HTTP

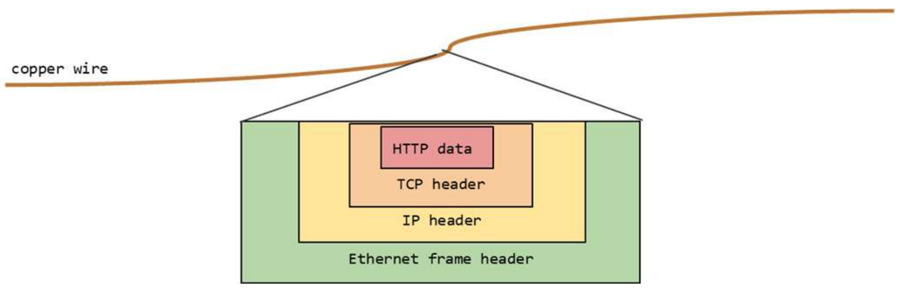
Not every protocol fits perfectly into the OSI model, but here is an example interpretation of HTTP, using the model.

OSI	
Application	GET / HTTP/1.1
Presentation	Gzip compression
Session	Cookies
Transport	TCP port 80
Network	IP
Data Link	Ethernet Frame
Physical	Copper / Radio

ENCAPSULATION

Each protocol adds headers to information it receives from the layer above it.

This is easier to describe by combining layers 5-7…



ENCAPSULATION (ETHERNET AND IP HEADERS)

Ethernet Header

```
0000  58 8b f3 ba 69 b9 30 65  ec a6 ce 4a 08 00 45 00   X···i·0e ···J··E·
0010  04 70 2b 7d 40 00 80 06  00 00 0a 00 00 d3 cc 4f   ·p+}@··· ·······O
0020  c5 c8 c4 cb 00 50 b3 ae  52 c4 94 2e d5 c0 50 18   ·····P·· R··.··P·
0030  01 02 a1 4d 00 00 47 45  54 20 2f 3f 6c 64 3d 64   ···M··GE T /?ld=d
0040  33 56 35 2d 6c 4d 5f 51  46 79 74 37 50 56 39 44   3V5-lM_Q Fyt7PV9D
0050  64 6e 36 69 70 4b 7a 56  55 43 55 7a 64 4b 41 63   dn6ipKzV UCUzdKAc
```

IP Header

```
0000  58 8b f3 ba 69 b9 30 65  ec a6 ce 4a 08 00 45 00   X···i·0e ···J··E·
0010  04 70 2b 7d 40 00 80 06  00 00 0a 00 00 d3 cc 4f   ·p+}@··· ·······O
0020  c5 c8 c4 cb 00 50 b3 ae  52 c4 94 2e d5 c0 50 18   ·····P·· R··.··P·
0030  01 02 a1 4d 00 00 47 45  54 20 2f 3f 6c 64 3d 64   ···M··GE T /?ld=d
0040  33 56 35 2d 6c 4d 5f 51  46 79 74 37 50 56 39 44   3V5-lM_Q Fyt7PV9D
0050  64 6e 36 69 70 4b 7a 56  55 43 55 7a 64 4b 41 63   dn6ipKzV UCUzdKAc
```

ENCAPSULATION (TCP AND HTTP LAYERS)

TCP Header

```
0000  58 8b f3 ba 69 b9 30 65  ec a6 ce 4a 08 00 45 00   X···i·0e ···J··E·
0010  04 70 2b 7d 40 00 80 06  00 00 0a 00 00 d3 cc 4f   ·p+}@··· ·······O
0020  c5 c8 c4 cb 00 50 b3 ae  52 c4 94 2e d5 c0 50 18   ·····P·· R··.··P·
0030  01 02 a1 4d 00 00 47 45  54 20 2f 3f 6c 64 3d 64   ···M··GE T /?ld=d
0040  33 56 35 2d 6c 4d 5f 51  46 79 74 37 50 56 39 44   3V5-lM_Q Fyt7PV9D
0050  64 6e 36 69 70 4b 7a 56  55 43 55 7a 64 4b 41 63   dn6ipKzV UCUzdKAc
```

HTTP (Application Layers)

```
0000  58 8b f3 ba 69 b9 30 65  ec a6 ce 4a 08 00 45 00   X···i·0e ···J··E·
0010  04 70 2b 7d 40 00 80 06  00 00 0a 00 00 d3 cc 4f   ·p+}@··· ·······O
0020  c5 c8 c4 cb 00 50 b3 ae  52 c4 94 2e d5 c0 50 18   ·····P·· R··.··P·
0030  01 02 a1 4d 00 00 47 45  54 20 2f 3f 6c 64 3d 64   ···M··GE T /?ld=d
0040  33 56 35 2d 6c 4d 5f 51  46 79 74 37 50 56 39 44   3V5-lM_Q Fyt7PV9D
0050  64 6e 36 69 70 4b 7a 56  55 43 55 7a 64 4b 41 63   dn6ipKzV UCUzdKAc
```

HTTP

HYPERTEXT TRANSFER PROTOCOL (HTTP)

HTTP is the protocol to exchange or transfer hypertext

Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. (URL/URI: http\ftp\mailto)

Request – Response

Client – Server

Data Can be HTML a File an image a sound

Can also open multiple additional connections to same server to download additional resources

TCP mostly but can also be UDP (SCCP)

HTTP uses TCP port 80

HTTP METHODS

GET

HEAD

POST (chat)

PUT (file to URI)

TRACE (ack to client)

OPTIONS (HTTP methods supported by server for URL)

CONNECT (SSL) / PATCH (Update)

flatironschool.com

HTTP: https://www.computerhope.com/jargon/h/http.htm

Short for HyperText Transfer Protocol, HTTP is a set of standards that allow users of the World Wide Web to exchange information found on web pages. When accessing any web page entering http:// in front of the address tells the browser to communicate over HTTP. For example, the URL for Computer Hope is https://www.computerhope.com. Today's browsers no longer require HTTP in front of the URL since it is the default method of communication. However, it is kept in browsers because of the need to separate protocols such as FTP. Below are a few of the major facts on HTTP.

The term HTTP was coined by Ted Nelson.
The standard port for HTTP connections is port 80.
HTTP/0.9 was the first version of the HTTP, and was introduced in 1991.
HTTP/1.0 is specified in RFC 1945, and was introduced in 1996.
HTTP/1.1 is specified in RFC 2616, and was officially released in January 1997.

HTTP (80)

Return Status Codes

- 1xx Informational
- 2xx Success (200 OK)
- 3xx Redirection (301 Moved Permanently)
- 4xx Client Error (403 forbidden, 404 not found)
- 5xx Server Error (500 Internal Server Error, 501 not implemented)

HTTP status codes: https://www.computerhope.com/jargon/h/http.htm
Below is a listing of HTTP status codes currently defined by Computer Hope. These codes enable a client accessing another computer or device over HTTP to know how to proceed or not proceed. For example, 404 tells the browser the request does not exist on the server.

1xx - 2xx - 3xx - 4xx -5xx
100 (Continue)
101 (Switch protocols)
102 (Processing)

200 (Success)
201 (Fulfilled)
202 (Accepted)
204 (No content)
205 (Reset content)
206 (Partial content)
207 (Multi-Status)

301 (Moved permanently)
302 (Moved temporarily)

304 (Loaded Cached copy)
307 (Internal redirect)

400 (Bad request)
401 (Authorization required)
402 (Payment required)
403 (Forbidden)
404 (Not found)
405 (Method not allowed)
406 (Not acceptable)
407 (Proxy authentication required)
408 (Request timeout)
409 (Conflict)
410 (Gone)
411 (Length required)
412 (Precondition failed)
413 (Request entity too large)
414 (Request URI too large)
415 (Unsupported media type)
416 (Request range not satisfiable)
417 (Expectation failed)
422 (Unprocessable entity)
423 (Locked)
424 (Failed dependency)

500 (Internal server error)
501 (Not Implemented)
502 (Bad gateway)
503 (Service unavailable)
504 (Gateway timeout)
505 (HTTP version not supported)
506 (Variant also negotiates)
507 (Insufficient storage)
510 (Not extended)

HTTP HEADERS

Both client and server can send additional information about the HTTP data that is being transmitted.

We will see several examples of these in the lab.

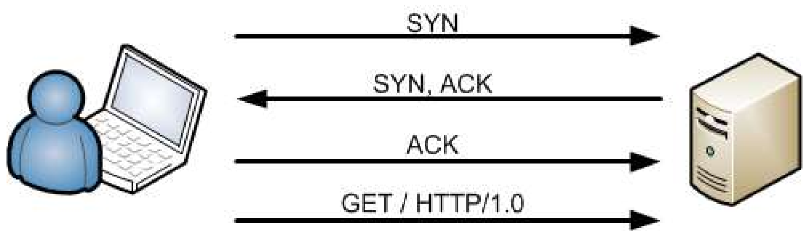
To tie it back to the OSI model, there is a header called Content-Encoding that describes the presentation of the data.

For example:

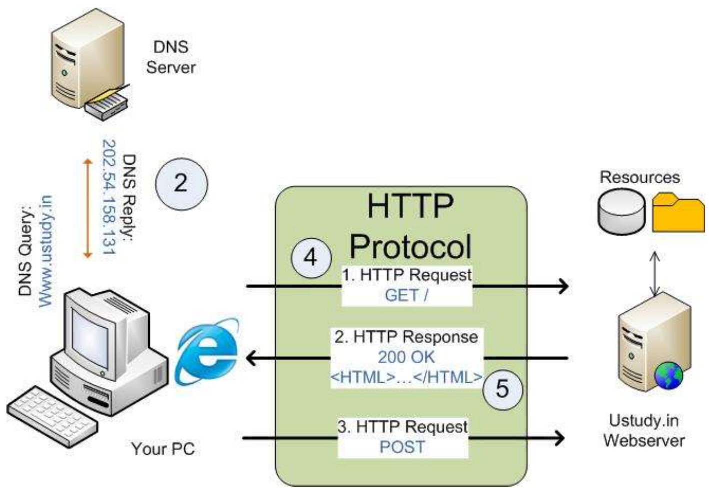
- Content-Encoding: gzip

HTTP and TCP

HTTP AND TCP



HTTP BASIC SESSION



www.ustudy.in

Alternative link: https://github.com/foundersandcoders/old-coursebook/blob/master/patterns/week2/httprequest.md

HTTP and TLS

SECURE HTTP – HTTPS (443)

TLS / SSL (Encryption)

- Authentication
- Privacy
- Integrity
- Man-in-the-Middle (MITM) Protection

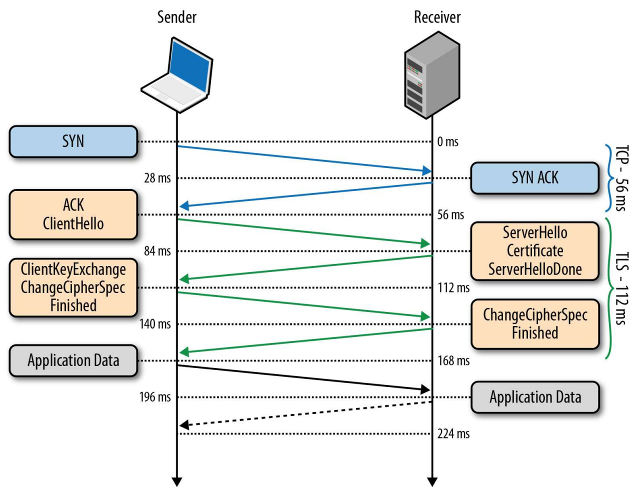
Web Browser

- Certificate Authority (Discuss)
- Preloaded CA Certificates
- 85-90% of global browsing is secured today

HTTPS: https://www.computerhope.com/jargon/h/http.htm

Short for Hypertext Transfer Protocol Secure, HTTPS is a protocol which uses HTTP on a connection encrypted by transport-layer security. HTTPS is used to protect transmitted data from eavesdropping. It is the default protocol for conducting financial transactions on the web, and can protect a website's users from censorship by a government or an ISP.
HTTPS uses port 443 to transfer its information.
HTTPS is first used in HTTP/1.1 and is defined in RFC 2616.

HTTPS SESSION



serverfault.com

flatironschool.com

https://www.youtube.com/watch?v=4nGrOpo0Cuc

How HTTPS Works: https://www.addictivetips.com/vpn/https-explained/

The first thing that happens is establishing a secure SSL connection. This begins with a quick handshake between the client (your computer, smartphone, etc.) and the server. The goal of this is to verify each other's identity and agree upon encryption protocols, setting things up for an impending data transmission.
If an SSL handshake were a conversation, it might play out something like this:
CLIENT: I'm looking for Server #SS1978-IJ56. Is that you?
SERVER: Yes. Are you the client I'm supposed to be working with?
CLIENT: Yes. Let's use Encryption Method 742 to chat.
SERVER: 742, no problem.
The handshake serves as a brief introduction. No data is transmitted during this process, it's just a quick superficial nod to make sure both parties are who they should be. The next part of the process is where the server and the host verify their identities and actually start exchanging information. This is still just the SSL part of the interaction, by the way. HTTP is waiting to do its job once SSL gives it the go-ahead.
After the handshake, the following steps take place, in order:
1. Greeting – This phase is somewhat similar to the handshake, only now that the

client/server identities are established, they can actually send data to each other. Verification begins with the client sending the equivalent of a hello message. This encrypted message contains all the information the server will need to communicate with the client via SSL, including encryption keys. The server then sends its own hello message back, containing similar information the client needs in order to hold up its end of the communication.

2. Certificate swap – Now that the server and client are ready to communicate securely, they need to verify their identity. This is a crucial step that ensures third parties can't pretend to be the intended server, which is what keeps encryption keys out of their hands. This is accomplished through an SSL certificate swap between the client and the server, roughly the equivalent of showing someone your ID in real life. SSL certificates contain data like the party's domain name, its public key, and who owns the device. These are checked against a centralized Certificate Authority (CA) source to make sure it's valid. CAs issue these certificates, which helps keep them out of malicious third party hands.

3. Key swap – Everyone knows who everyone else is, encryption protocols have been agreed upon, so it's finally time to get started. The key swap begins with the client (your device) generating a cipher key to use in a symmetrical algorithm. This means the encrypted data can be unlocked and fully accessed by anyone with the key, hence the symmetry. Since the key styles were agreed upon during the verification phase, all the client has to do is share the key and the two parties can communicate efficiently and securely.

All of these phases with SSL verification and data swapping seem like a lot of extra steps, but they're crucial to establishing a secure connection between the right computers. Without verifying identities, other computers can steal data and decrypt it. Without verifying encryption methods, other computers can share fake keys and gain access to data. Only with all of these pre-sharing steps can the HTTP transfer take place securely.

Once the SSL portion of the transfer takes place, HTTP steps in and does its thing. Here data is broken into packets, labeled with your IP address, stuffed inside the SSL envelope and sent along their way. SSL ensures only the client and the intended server can read the information being sent. The process is completed thousands of times for each request, and it happens in a fraction of a second.

HTTPS in Your Browser

You've probably seen your browser display a little padlock icon in the URL bar from time to time. This simply means the site is secured with HTTPS. It normally happens with sites that legitimately collect private data, such as credit card information for online shopping, passwords for checking your e-mail, or anything involving banking or financial transactions. More and more websites are using HTTPS these days, however, which is great for online privacy in general.

HTTPS is done on the server's side. In other words, you can't force a site to use HTTPS if its servers aren't set up to handle it. Many websites will only switch to HTTPS if your browser specifically demands it, and others will load unsecured content within HTTPS pages, which

defeats the purpose entirely.
There's a fantastic browser extension called HTTPS Everywhere that alleviates a lot of the above issues. The plug-in rewrites your browser requests to use HTTPS whenever it's available. It can't create a secure connection where none exists, and it doesn't encrypt anything itself, but HTTPS Everywhere ensures you always take advantage of the extra security whenever possible.

FTP

FTP (21/20)

File Transfer Protocol (CLI /GUI)

- Session (21)
- Data (20)

Commands (80+)

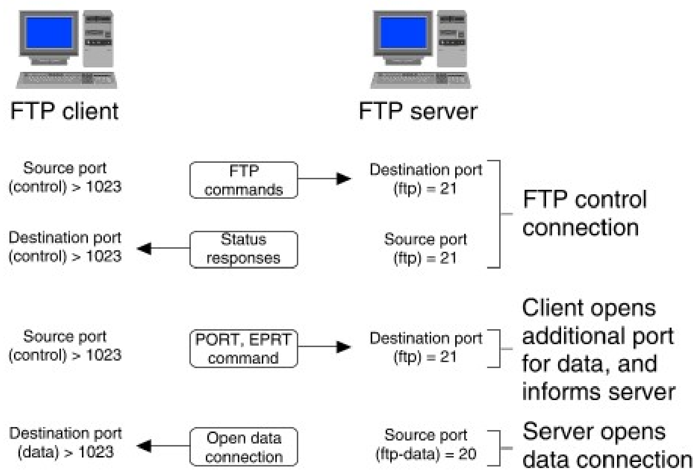
- LIST
- MKD
- RETV
- QUIT

Secure Versions (FTPS –SSL / SFTP –SSH)

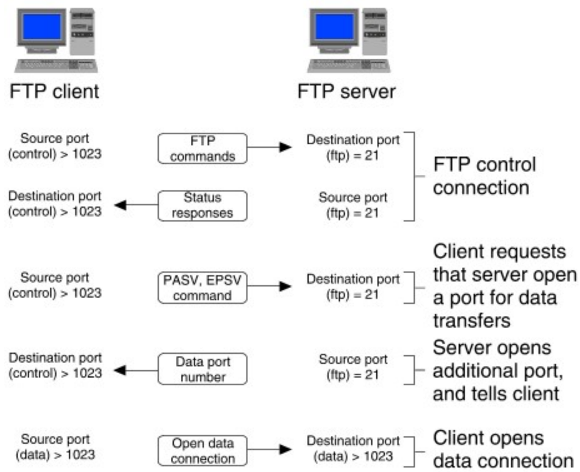
Return Codes

- 1xx Positive Preliminary
- 2xx Positive Completion
- 3xx Positive Intermediate (Pending User Action)
- 4xx Transient / try again later
- 5xx Command not accepted

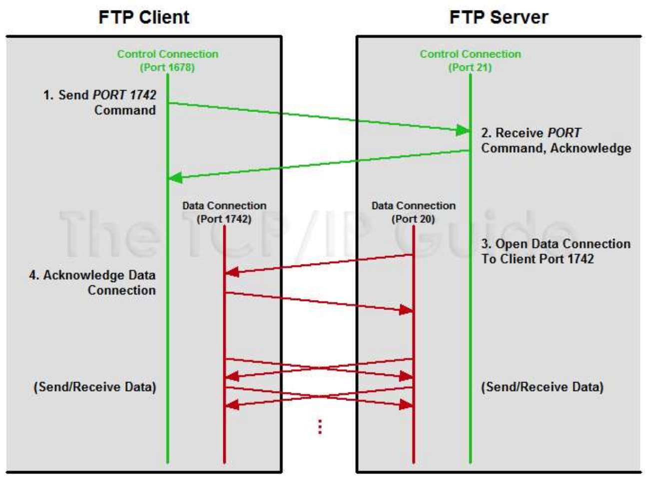
FTP ACTIVE


FTP client
FTP server
Source port (control) > 1023
FTP commands
Destination port (ftp) = 21
FTP control connection
Destination port (control) > 1023
Status responses
Source port (ftp) = 21
Source port (control) > 1023
PORT, EPRT command
Destination port (ftp) = 21
Client opens additional port for data, and informs server
Destination port (data) > 1023
Open data connection
Source port (ftp-data) = 20
Server opens data connection

FTP PASSIVE


FTP client
FTP server
Source port
(control) > 1023
FTP
commands
Destination port
(ftp) = 21
FTP control
connection
Destination port
(control) > 1023
Status
responses
Source port
(ftp) = 21
Source port
(control) > 1023
PASV, EPSV
command
Destination port
(ftp) = 21
Client requests
that server open
a port for data
transfers
Destination port
(control) > 1023
Data port
number
Source port
(ftp) = 21
Server opens
additional port,
and tells client
Source port
(data) > 1023
Open data
connection
Destination port
(data) > 1023
Client opens
data connection

2 PORTS



Great Resource: http://www.slideshare.net/PeterREgli/ftp-6027338

SMTP

SIMPLE MAIL TRANSFER PROTOCOL

SMTP (25) (587 & 455 Secure)

Email transmission

Server to Server Usually

Client server use POP3 /IMAP for access SMTP for Relay Services

POP AND IMAP PROTOCOLS

POP3 (110) POP3S-TLS (995)

Used by email clients to access and retrieve email

- Authentication
- Simpler than IMAP
- Gets email to local computer
- 1 client / 1 account

IMAP (143) IMAPS-TLS (993)

Internet Message Access Protocol

- Storage/Caching
- Download copy of email but leave original on server
- Multiple clients / 1 account
- MIME/ Partial Fetch / Search / Offline Mode

FINAL APPLICATION LAYER THOUGHTS

There are several hundreds of Application layer protocols used today

While most of them are standardized, any application developer can create their own (Tor)

Most of these well-known protocols originated without security in mind, and while new versions are being secured, not all Internet usage is secure.

Don’t forget that bad guys can use same security protocols to their advantage to masquerade their activities‼ Or hide behind anonymity (What’s in an ICMP packet payload?)

[redacted]