

HTTP and Friends

I.G.Batten@bham.ac.uk

FTP(s)

- Wide range of protocols available to “send a filename, receive a file”.
 - **FTP**, which we will discuss later, probably oldest and certainly ugliest
 - **kermit** and **uucp** different ways to use serial lines (complete with own “transport” layers)
 - uucp mostly Unix-only, still shipped on OSX
 - kermit multi-platform
 - both can be coerced into running over a TCP connection

pre-HTTP

- Range of protocols for finding resources to then fetch (probably with FTP)
 - gopher
 - archie
 - There were others
- All completely killed by rise of HTTP and (later) Search Engines

Why HTTP?

- Hypertext Transport Protocol
- Originally designed to deal with downloading HTML with support for hyperlinks (“HTTP GET”)
- Extensible to a wide range of other tasks with other commands (“HTTP POST”, “HTTP PUT”)
- Flexible concept of URL means that it can do many other tasks apart from shipping files

Why HTTP?

- Protocol is relatively easy to implement, but very flexible
- Protocol is not a screaming nightmare for networking engineers, unlike FTP
- Protocol decouples names from things (URLs look like Unix paths, but don't have to be)

Basic structure

- Client sends some (optional) options
- Client sends command
- Client sends a blank line
- Server sends a status
- Server sends some commentary and information
- Server sends a blank line
- Server sends some data

Lines

- This being the Internet, lines are terminated with `\r\n` (carriage-return line feed, aka control-M control-J, aka 0x0d, 0x0a)
- “man ascii”
 - Just to be annoying, Unix convention is `\n`, DOS convention is `\r`.

Example

- We are examining the behaviour of the command
`curl -v -6 \`
`https://www.batten.eu.org/index.html`

HTTPS vs HTTP

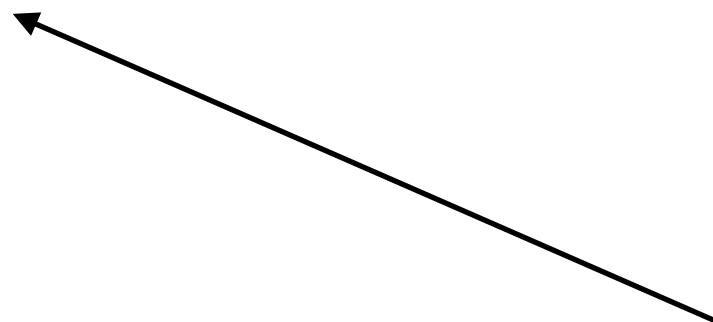
- HTTP (port 80) talks HTTP (amazingly)
- HTTPS(port 443) talks HTTP over an immediately-negotiated TLS session

First, TLS is set up

```
* Hostname was NOT found in DNS cache
*   Trying 2a00:1630:66:25:87:65:43:21...
* Connected to www.batten.eu.org (2a00:1630:66:25:87:65:43:21) port 443 (#0)
* successfully set certificate verify locations:
*   CAfile: none
   CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS handshake, Server hello (2):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Server key exchange (12):
* SSLv3, TLS handshake, Server finished (14):
* SSLv3, TLS handshake, Client key exchange (16):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / DHE-RSA-AES256-GCM-SHA384
* Server certificate:
*   subject: C=GB; CN=batten.eu.org
*   start date: 2016-09-17 12:07:45 GMT
*   expire date: 2019-09-17 12:07:45 GMT
*   subjectAltName: www.batten.eu.org matched
*   issuer: C=IL; O=StartCom Ltd.; OU=StartCom Certification
Authority; CN=StartCom Class 1 DV Server CA
*   SSL certificate verify ok.
```

What is sent

```
GET /index.html HTTP/1.1  
User-Agent: curl/7.38.0  
Host: www.batten.eu.org  
Accept: */*
```



Note there is a blank line here

Line formats

- The **is no colon** in the GET/POST/etc command line (and in the status response to it)
- There **is a colon** in the remaining option lines.
 - Format taken from email headers
 - Key: Value
 - Lines starting with whitespace are continuations of the previous line. Yes, this is a pain to code.

GET /index.html HTTP/1.1

- Operation (others are POST, the less used PUT, the far less used DELETE, the sometimes useful HEAD)
- Name of resource to be fetch. Note, does **not** include hostname: dates back to era when there was a 1:1 relationship between IP numbers and name-spaces
- Note that this isn't the (obsolete) HTTP 1 protocol.

User-Agent: curl/7.38.0

- Optional, but useful: sometimes a server will send different content based on its knowledge of the capabilities of the client
- Remember, people can lie (most browsers have a debug option to allow you to set pretty well any User-Agent field)

Host: www.batten.eu.org

- Indicates the namespace, and allows multiple virtual servers to live on one IP number
- Historically, this required one IP number per namespace, which clearly isn't going to end well for (eg) Wordpress — even if they had enough IP numbers, there are limits to how many you can stick on one interface

Accept: */*

- I will accept any content type (cf. `text/html`).

What comes back?

```
HTTP/1.1 200 OK
Server: nginx/1.11.4
Date: Mon, 24 Oct 2016 21:17:31 GMT
Content-Type: text/html
Content-Length: 656
Last-Modified: Sun, 18 Sep 2016 20:47:08 GMT
Connection: keep-alive
ETag: "57defd4c-290"
Strict-Transport-Security: max-age=31536000
Public-Key-Pins:
    pin-sha256="z0rKil8EjF1RmB6shUYfITbsq2+r8HWhTgj0wIFtBcI=";
    pin-sha256="YwBM31CHmLowHjIwSGTZX5PBj4w9i0Gt9dMDXpU87M=";
    pin-sha256="+Z7a6RhW5wAU+PTNmoAy2p/fVgQUdxXQ6U57dI7CK5E=";
    max-age=15552000
Accept-Ranges: bytes
```

Or...

HTTP/1.1 200 OK

Date: Mon, 24 Oct 2016 23:06:08 GMT

Server: Apache

Vary: Accept-Encoding

Content-Type: text/html; charset=utf-8

HTTP/1.1 200 OK

- Only compulsory element
- List of error codes taken from (essentially) SMTP and previously FTP
- See also 404 Not found, 403 Forbidden, etc

Server: nginx/1.11.4

Date: Mon, 24 Oct 2016 21:17:31 GMT

- Useful commentary
- Case can be made that both are revealing information of possible use to attacker, although (a) it's trivial to fingerprint servers and (b) everyone synch's their clock these days.

Content-Type: text/html
Content-Length: 656

- Type allows client to deal with content appropriately (in this case, hint to render it as HTML)
- Content-Length can optimise subsequent read ()
 - Might be missing if length unknown prior to sending

End of File: Nightmare

- FTP opens a separate TCP connection for each file (using a “control connection” to orchestrate them)
- File can then be sent unmodified (“BINARY MODE”) followed by closing the connection to mark its end.
 - Inefficient, complex, hard on the network
- SMTP, POP3, others use “.” on a line on its own to mean “end of file”, with an extra dot stuffed into lines that start with a dot.
 - Means sender and receiver have to scan whole file, and inserting a single character breaks the flow of “read block, send block”.
 - Prevents protocol from being “8 bit clean” as it has to avoid sequence “\r\n.\r\n” at all costs.
 - One of the most broken things about existing email standards.
- Byte count (IMAP, HTTP) allows sender to stat() a file and then read it as one operation and send it as one operation, and allows receiver to just route next n bytes to the client software. HTTP doesn’t mandate \r\n for data, and very high performance IMAP servers store messages with \r\n so they can do the same “stat(), open(), read()” trick.

Last-Modified: Sun, 18 Sep 2016 20:47:08 GMT

ETag: "57defd4c-290"

- Last-Modified and ETag helpful for caching (storing copies of someone else's content)
- Etag is opaque fingerprint of exact contents, which is a stronger check than comparing the last-modified string.

Connection: keep-alive

- Indicates to client that the connection can be re-used for more requests once this one has completed
- Saves the building of a fresh connection and has performance benefits we will talk about later
- Early browsers built a fresh TCP connection for every item on a webpage: catastrophic performance

Strict-Transport-Security: max-age=31536000

- HSTS extension: says that this namespace is only available from HTTPS encrypted HTTP, and all requests for the unencrypted form should be rewritten to the encrypted version for the next 365 days
- Ignored unless send over HTTPS, cached by client
- Can be “pre-loaded”
- Use this whenever possible: all content should be encrypted to ensure authenticity

Public-Key-Pins: pin-sha256="z0rKil8EjF1RmB6shUYfITbsq2+..."
max-age=15552000

- HPKP Experimental Extension
- Lists the set of public keys that should be accepted for this domain over the coming period of time (here 180 days)
- HSTS+HPKP+HSTS pre-load substantially reduce risk of “rogue AP”
 - Network Security next semester

Accept-Ranges: bytes

- Indicates server is willing to accept requests for partial transfers, specified by byte-offsets

Then the content

Note blank line

Accept-Ranges: bytes

<html>

<head>

<!-- <base href="https://www.batten.eu.org/"> -->

<link rel="stylesheet" type="text/css" href="https://www.batten.eu.org/batten.css">

<title>batten.eu.org</title>

</head>

Flexibility

- Trivially, the requested name can be a file in a directory rooted somewhere in the filestore
- But can just as easily be a key into a database, a parameter to a program, whatever.
- The server converts URNs into content however it wants (and, presumably, the user expects)

By convention

- You can pass multiple parameters with
 - `/some/path?var1=value1&var=value2`
- But note this is just a convention (“CGI”, Common Gateway Interface) and nothing stops you turning URL `/foo/bar/baz` into a call to `foo` with arguments `bar` and `baz`.
- Passing arguments via GET makes them trivially available in logs

POST

```
curl -d foo=bar -v -v -6 https://www.batten.eu.org/index.html
```



```
> POST /index.html HTTP/1.1  
> User-Agent: curl/7.38.0  
> Host: www.batten.eu.org  
> Accept: */*  
> Content-Length: 7  
> Content-Type: application/x-www-form-urlencoded
```

HEAD

```
HEAD /index.html HTTP/1.1
User-Agent: curl/7.38.0
Host: www.batten.eu.org
Accept: */*
```

Request sent



```
HTTP/1.1 200 OK
Server: nginx/1.11.4
Date: Mon, 24 Oct 2016 21:58:16 GMT
Content-Type: text/html
Content-Length: 656
Last-Modified: Sun, 18 Sep 2016 20:47:08 GMT
Connection: keep-alive
ETag: "57defd4c-290"
Strict-Transport-Security: max-age=31536000
Public-Key-Pins:
  pin-sha256="z0rKil8EjF1RmB6shUYfITbsq2+r8HWhTgj0wIFtBcI=";
  pin-sha256="YwBM31CHmLowHjIwSGTZX5PBj4w9i0Gt9dMDXpU87M=";
  pin-sha256="+Z7a6RhW5wAU+PTNmoAy2p/fVgQUdxXQ6U57dI7CK5E=";
  max-age=15552000
Accept-Ranges: bytes
```

Response Received



PUT

- Permits file upload
- Subtle differences to POST, which we can talk about if we have time to discuss REST

DELETE

- Rarely used for files today, but used in RESTful web APIs.

Accept-Encoding

- Client can indicate that it can handle encodings (distinguish from encryption) of the content
- `Accept-Encoding: deflate, gzip`
- Server can (at its discretion) respond with
- `Content-Encoding: gzip`

Languages

- Similarly for Accept-Language and Content-Language.
- And Accept-Charset and Content-Charset.
- More generally, client sends requests and capabilities, server indicates formats and content.

Username and Passwords

- HTTP Basic Auth is a cesspit of security problems
- But, if we must...
- If there is no authentication in the request, it is rejected as follows:

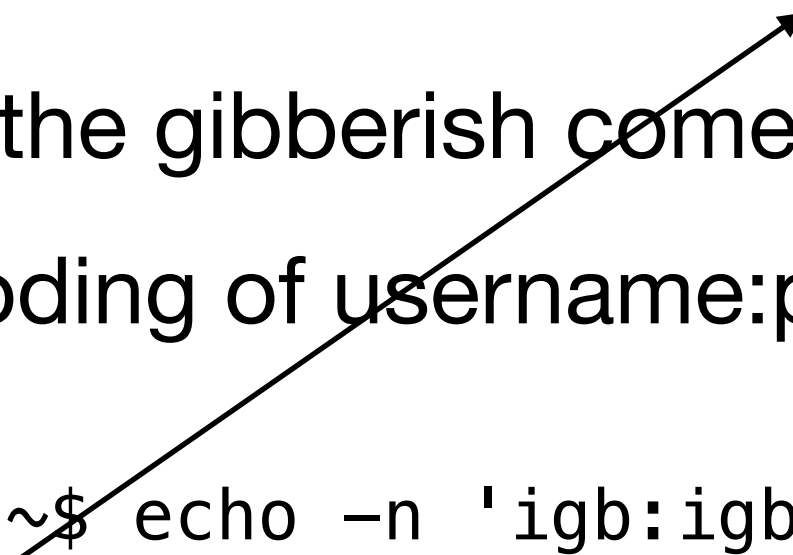
```
HTTP 401 Unauthorized
```

```
WWW-Authenticate: Basic realm="Some Tag"
```

Username and Passwords

- If you have a login and password for the domain, you send
 - Authorization: Basic aWdiOm1nYjEyMw==
 - Where does the gibberish come from?
 - Base64 encoding of username:password

```
igb@pi-one:~$ echo -n 'igb:igb123' | base64
aWdiOm1nYjEyMw==
igb@pi-one:~$
```



HTTPS Only

- Obviously, don't do this over unencrypted channels!
- There is a challenge/response mechanism ("Digest", RFC 2069) using hashes, but
 - it involves the server storing plaintext passwords
 - almost no-one implements it anyway.

Cookies

- Another security cesspit
- And wildly abused for a wide range of things they aren't suitable for

Cookies

- Any operation can return a “Set-Cookie” operation, which logs a key=value pair against (usually) the current domain and host
- Whenever a request is made for that domain and host, all relevant cookies are sent with “Cookie” header
- Cookies can have lifetimes, various security properties

Cookies

- One common use is/was to use HTTPS to protect username and password, returning a cookie for successful authentication, and then rely on that being safe to pass unencrypted to prove authentication has happened
 - HTTPS no longer expensive, and should be used for all operations, particularly...
 - ...those involving cookies
- More generally, anything which involves storing client-side state: navigation, shopping carts, etc
- Best limited to a random session-id and nothing else, with everything else stored server-side.

RESTful Interfaces

- Later lecture (was requested last year, so I have it good to go)
- Way to interact with APIs over standard HTTP

Caching

- Rises and falls in popularity
- Idea is that you can keep a copy of “all the internet that matters” at the edge of a **consuming** network
 - Ian F brought the cs.bham.ac.uk caching strategy to ftel.co.uk; when I came back, I brought its end
- Alternatively, a more usefully, you keep a copy of “all our content from lots of slow machines” at the edge of a **producing** network
- Caches and proxies tend to be the same boxes: we will talk about proxying when we talk about NAT.

Caching

- Usually done with special software (“squid” is the most common) or with special appliances
- Trick is to cache stuff which is static and used repeatedly, while not interfering with content that changes rapidly or is rarely accessed
- My gut feel is that 2016-stylee, all caching is out of favour