HTTP:
Hypertext Transfer Protocol

Lecture 12

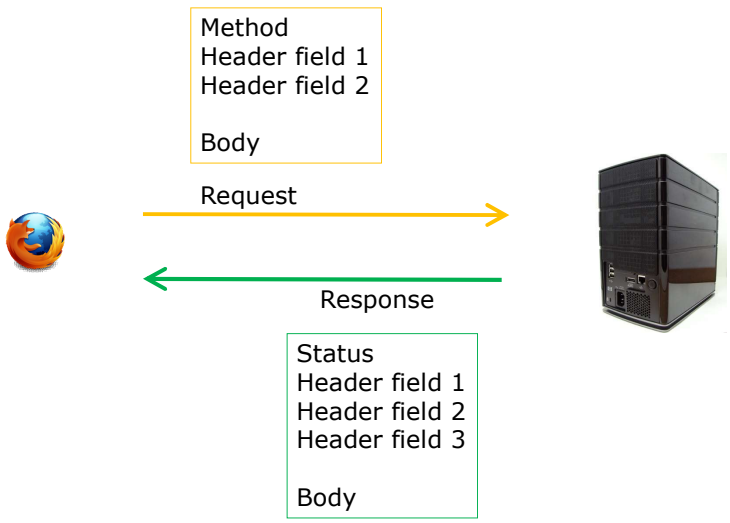
HTTP

- Hypertext Transfer Protocol
- History
  - Early 90's: developed at CERN, Tim Berners-Lee
  - 1996: version 1.0
  - 1999: version 1.1 (ubiquitous today!)
  - May 2015: version 2
    - Performance improvements: binary, server push…
    - Backwards compatible
    - Adoption: https://w3techs.com/technologies/details/ce-http2/all/all
- Simple request/response (client/server)
  - Client sends request to (web) server
  - (Web) server responds
  - "stateless" protocol

Request/Response Anatomy

- An HTTP request/response consists of
  1. Header: meta information
  2. Body (sometimes): payload
- The header consists of
  1. Method/Status (for request/response)
  2. Header fields, separated by newlines
  3. Blank line

Protocol: Request, Response



Request Header: First Line

- Syntax of first line:

  verb path version

  - Verb: **GET, HEAD, POST, PUT, DELETE,…**
  - Path: part of URL (path and query)

    scheme://FQDN:port/path?query#fragment
  - Version: **HTTP/1.1, HTTP/2**
- Example:
  - For URL

    http://news.osu.edu/news/
  - First line of request is

    GET /news/ HTTP/1.1

Request Header: Header Fields

- Each field on its own line, syntax:

  name: value
- Examples (only "Host" is required)

```
Host: cse.ohio-state.edu
Accept: text/*
Accept: image/gif
If-Modified-Since: Sat, 12 May 2016
19:43:31 GMT
Content-Length: 349
User-Agent: Mozilla/5.0 (X11; Ubuntu;
Linux x86_64; rv:51.0) Gecko/20100101
Firefox/51.0
```

- Blank line indicates end of headers

Header Fields cont'd

- Host
  - Only required field
  - Q: Why is host field even needed?
- Accept
  - Browser preference for MIME type(s) to receive
- If-Modified-Since
  - Send payload only if changed since date
  - Date must be GMT
- Content-Length
  - Required if request has a body
  - Number of bytes in body
- User-Agent
  - Identifies application making request

Steiner, The New Yorker (1993)

"On the Internet, nobody knows you're a dog."

"Nobody knows you're a dog"


```
GET / HTTP/1.1
Host: news.osu.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu;…etc
```

Request

"Nobody knows you're a dog"

```
$ telnet
```

```
GET / HTTP/1.1
Host: news.osu.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu;…etc
```

Request

```
$ curl -A "Mozilla/5.0" news.osu.edu
```

Demo: HTTP Request with telnet

- Example URL
  - web.cse.ohio-state.edu/~sivilotti.1/
- At console

  `$ telnet web.cse.ohio-state.edu 80`
  - Opens connection to port 80, where a web server is listening
- Send the following HTTP request:

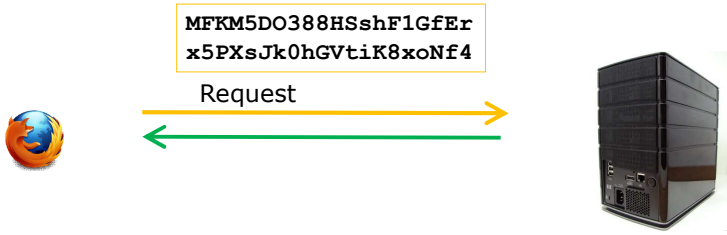
  `GET /~sivilotti.1/ HTTP/1.1`

  `Host: web.cse.ohio-state.edu`

  `<blank line>`

HTTP Traffic Transparency

- Everything is visible to an eavesdropper
  - HTTP headers are plain text
  - HTTP payload may be binary
- To protect communication, use encryption
  - SSL, TLS: protocols to create secure channel
  - Initial handshake between client and server
  - Subsequent communication is encrypted
- HTTP over secure channel = HTTPS
  - Default port: 443



Demo: HTTPS with openssl

- Use openssl instead of telnet
  - Negotiates initial handshake with server
  - Handles encryption/decryption of traffic
- Example URL
  - https://www.osu.edu/
- At console

```
$ openssl s_client -connect www.osu.edu:443
```

  - Note connection to port 443 (ie https)
- Syntax of subsequent request is the same
- Send the following HTTP request:

```
GET / HTTP/1.1
Host: www.osu.edu
<blank line>
```

HTTP Response Anatomy

- Recall, four parts
  1. Status (one line)
  2. Header fields (separated by newlines)
  3. Blank line
  4. Body (ie payload)
- Parts 1-3 collectively called "the header"
- Part 1 (status line) syntax:

```
http-version status-code text
```

  - Examples

```
HTTP/1.1 200 OK
HTTP/1.1 301 Moved Permanently
HTTP/1.1 404 Not Found
```

Taxonomy of Status Codes

Code	Meaning
1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error

Some Common Status Codes

- 200 OK
  - All is good!
  - Response body is the requested document
- 301 Moved Permanently
  - Requested resource is found somewhere else (please go there in the future)
- 304 Not Modified
  - Document hasn't changed since date/time in If-Modified-Since field of request
  - No response body
- 404 Not Found
  - Server could not satisfy the request
  - It is the client's fault (design-by-contract?)
- 500 Internal Server Error
  - Server could not satisfy the request
  - It is the server's fault (design-by-contract?)

Response Header: Header Fields

- Each field on its own line, syntax:

  name*: *value

- Examples

```
Date: Mon, 16 Sep 2019 14:51:38 GMT
Server: Apache/2.4.6 (Red Hat)
Content-Type: text/html; charset=UTF-8
Content-Length: 333
```

- Blank line indicates end of headers

Demo: Using Terminal

- Use telnet to retrieve

  http://web.cse.ohio-state.edu/~paolo

  - Fails (see status code)

  http://web.cse.ohio-state.edu/~paolo/

  - Body is incomplete (no images)
  - Body is chunked

- Use curl to retrieve
  - Handles https, headers, redirection, chunking,…

```
$ curl -L http://web.cse.ohio-sta…
```

Demo: Using Firefox

- Developer > Network
- One GET results in many requests

```
http://www.cse.osu.edu/~paolo
```

- For each request, see:
  - Request headers
  - Response status code
  - Response headers
  - Response (and preview)

Demo: Using Ruby

- Mechanize: A Ruby gem for HTTP

```
require 'mechanize'
```

- Create an agent to send requests

```
agent = Mechanize.new do |a|
  a.user_agent_alias = "Mac Safari"
end
```

- Use agent to issue a request

```
page = agent.get "http://www.osu.edu"
```

- Follow links, submit forms, etc

```
page.link_with(text: "Carmen").click
s = page.form_with action: /search/
```

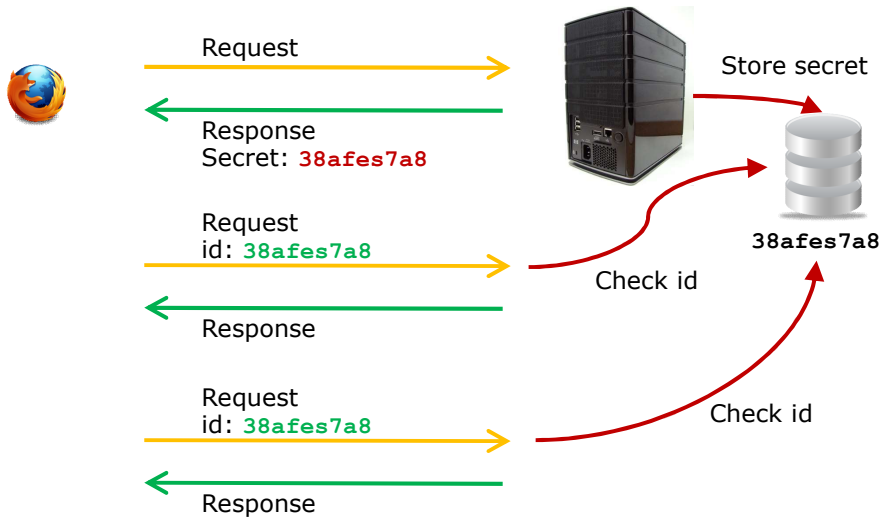
Request Methods

- GET, HEAD
  - Request: should be *safe* (no side effects)
- PUT
  - Update (or create): should be *idempotent*
- DELETE
  - Delete: should be *idempotent*
- POST
  - Create (or update): changes server state
  - Beware re-sending!
- HTTP does not enforce these semantics

HTTP is Stateless

- Every request looks the same
- But maintaining state between requests is really useful:
  - User logs in, then can GET account info
  - Shopping cart "remembers" contents
- One solution: Keep a shared secret
  - Server's response contains a unique session identifier (a long random value)
  - Subsequent requests from this client include this secret value
  - Server recognizes the secret value, request must have come from original client

HTTP Session



HTTP Cookies

- Popular mechanism for session manag'nt
- Set in *response* header field

  `Set-Cookie: session=38afes7a8`
  - Any name/value is ok
  - Options: expiry, require https
- Client includes cookie(s) in subsequent requests to that domain
- Sent in *request* header field:

  `Cookie: session=38afes7a8`
- Cookies also used for
  - Tracking/analytics: What path did they take?
  - Personalization

Passing arguments: GET

- Arguments are key-value pairs

```
Mascot: Brutus Buckeye
Dept: CS&E
```

- Can be encoded as part of URL

```
scheme://FQDN:port/path?query#fragment
```

- application/x-www-form-urlencoded
  - Each key-value pair separated by & (or ;)
  - Each key separated from value by =
  - Replace spaces with + (arcane!)
  - Then normal URL encoding

```
Mascot=Brutus+Buckeye&Dept=CS%26E
```

Examples

- Wikipedia search

```
http://en.wikipedia.org/
   w/index.php?
   search=ada+lovelace
```

- OSU news articles

```
https://news.osu.edu/
   ?
   q=Rhodes+Scholarship&search.x=1
```

- Random numbers (link)

```
https://random.org/
   passwords/?
   num=5&len=8&format=plain
```

  - Demo: use FF Dev to edit/resubmit request
  - See guidelines and API for http clients

Passing Arguments: POST

- Encoded as part of the request *body*
- Advantages:
  - Arbitrary length (URLs are limited)
  - Arguments not saved in browser history
  - Result not cached by browser
  - Slightly more secure (not really)
    - Args not in location bar, so less likely to be accidentally shared
- Content-Type indicates encoding used
  - application/x-www-form-urlencoded
    - Same encoding as used in GET
  - multipart/form-data
    - Better for binary data (else 1 byte→3 bytes)
  - More options too:
    - application/xml, application/json, …

Passing Args: GET vs POST

- GET

```
GET /passwords/?num=5&len=8&format=plain
HTTP/1.1
Host: www.random.org
```

- POST

```
POST /passwords/ HTTP/1.1
Host: www.random.org
Content-Type: application/x-www-form-
urlencoded
Content-Length: 24

num=5&len=8&format=plain
```

Summary

- HTTP: request/response
- Anatomy of request
  - Methods: GET, PUT, DELETE, POST
  - Headers
  - Body: arguments of POST
- Anatomy of response
  - Status Codes: 200, 301, 404, etc
  - Headers
  - Body: payload
- Tools
  - Curl, FF Developer, Mechanize