09. How the Web Works

Blase Ur and Grant Ho
February 5th, 2024
CMSC 23200
Your interface to the web

• Your web browser contacts a web server
A 10,000 Foot View of Technologies

- Where things run:
  - HTML / CSS
  - JavaScript (Angular/React)
  - Browser Extensions
  - Django (Python) / CGI (Perl) / PHP / Node.js / Ruby on Rails
  - Databases (MySQL)
The Anatomy of a URL

- https://www.uchicago.edu/fun/funthings.htm?query=music &year=2024#topsection
The Anatomy of a URL

• **https://www.uchicago.edu/fun/funthings.htm?query=music &year=2024#topsection**
  – Protocol: https
  – Hostname: www.uchicago.edu
    • .edu is the top level domain (TLD)
  – Path: /fun/funthings.html
  – Parameters: (key=value pairs, & delimited)
  – Named anchor: #topsection

• Some technologies (e.g., Django) parse the path differently (e.g., parameters in path)
The Anatomy of a Webpage

• view-source:https://www.cs.uchicago.edu/

• HTML (hypertext markup language)
  – Formatting of a page
  – All sorts of formatting: `<div><p>Hi</p></div> <br />
  – Links: `<a href="blaseur.com">Click here</a>`
  – Pictures: `<img src="unicorn.jpg" />
  – Forms
  – Audio/video
The Anatomy of a Webpage
The Anatomy of a Webpage

• CSS (cascading style sheets)

  
  <link href="/css/main.css?updated=20181020002547" rel="stylesheet" media="all">


• #id (*intended* to be unique)

• .class (not intended to be unique)
The Anatomy of a Webpage

- DOM (document object model)
Typing Something into a Browser:

• DNS (domain name service)
  – `www.cs.uchicago.edu` resolves to IP address 128.135.164.125
• https://www.cs.uchicago.edu/
  – Protocol: https
  – Hostname: www.cs.uchicago.edu
  – Default file name (since none is listed): index.html (and similar)
HTTP Request

• HTTP = Hypertext Transfer Protocol
• Start line: method, target, protocol version
  – GET /index.html HTTP/1.1
  – Method: GET, PUT, POST, HEAD, OPTIONS
• HTTP Headers
  – Host, User-agent, Referer, many others
• Body (not needed for GET, etc.)
• In Firefox: F12, “Network” to see HTTP requests
HTTP Request

• GET /index.html HTTP/1.1
Sending Data to a Server

• GET request
  – Data at end of URL (following “?”)

• POST request
  – Typically used with forms
  – Data *not* in URL, but rather (in slightly encoded form) in the HTTP request body

• PUT request
  – Store an entity at a location
URL Parameters / Query String

- End of URL (GET request)
  - https://www.cs.uchicago.edu/?test=foo&test2=bar
HTTP Response

  - 200 (OK)
  - 404 (not found)
  - 301 (moved permanently)
  - 302 (moved temporarily)

- HTTP Headers

- Body
HTTP

Requests

POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh; ..) AppleWebKit/51.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9, image/webp,image/apng,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=--12656974
Content-Length: 345

--12656974
(more data)

Responses

HTTP/1.1 403 Forbidden
Server: Apache
Content-Type: text/html; charset=iso-8859-1
Date: Wed, 10 Aug 2016 09:23:25 GMT
Keep-Alive: timeout=5, max=1000
Connection: Keep-Alive
Age: 3464
Date: Wed, 10 Aug 2016 09:46:25 GMT
X-Cache-Info: caching
Content-Length: 220

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 2.0//EN">
(more data)
HTTPS

• Simply an HTTP request sent over TLS!
  – That is, the request and response are encrypted
• An extension of HTTP over TLS (i.e., the request/response itself is encrypted)
• Which CAs (certificate authorities) does your browser trust?
  – Firefox: Options → Privacy & Security → (all the way at the bottom) View Certificates
Keeping State Using Cookies

- Cookies enable persistent state
- Set-Cookie HTTP header
- Cookie HTTP header
  - Cookie: name=value; name2=value2; name3=value3
- Cookies are **automatically sent** with all requests your browser makes
- Cookies are *bound to an origin* (only sent to the origin that set them)
Keeping State Using Cookies

- Session cookies (until you close your browser) vs. persistent cookies (until the expiration date)
- Secure cookies = only sent over HTTPS, not HTTP
- HTTPOnly cookies are not accessible to JavaScript, etc.
- View cookies: “Application” tab in Chrome developer tools, “Storage” in Firefox
Authorization Tokens = Cookies

• You log into a website, and it presents you an authorization token (typically a hash of some secret)
• Subsequent HTTP requests automatically embed this authorization token
Other Ways to Keep State

- Local storage
- Flash cookies
- (Many more)
JavaScript
Interactive Pages?

• JavaScript!
  – The core idea: Let’s run code on the client’s computer
• Math, variables, control structures
• Imperative, object-oriented, or functional
• Modify the DOM
• Request data (e.g., through AJAX)
• Can be multi-threaded (web workers)
Common Javascript Libraries

• JQuery (easier to specify access to DOM)
  – \$\(".test"\).hide() hides all elements with class="test"

• JQueryUI

• Bootstrap

• Angular / React

• Google Analytics \((sigh)\)
Importing Javascript Libraries
Do You Have the Right .js File?


- `<script src="https://example.com/example-framework.js" integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"></script>`

- `cat FILENAME.js | openssl dgst -sha384 -binary | openssl base64 -A`
Patching JavaScript Libraries

• Many outdated (and sometimes vulnerable) JavaScript libraries continue to be used

• Very complex chain of dependencies!
  – How do you determine if a given change is for good or evil?
Core Web Defense: Same-Origin Policy
Same-Origin Policy

- Prevent malicious DOM access
- Origin = URI scheme, host name, port
- Only if origin that loaded script matches can a script access the DOM
  - Not where the script ultimately comes from, but what origin loads the script
# Definition of an origin

Two URLs have the same origin if the protocol, port (if specified), and host are the same for both. You may see this referenced as the "scheme/host/port tuple", or just "tuple". (A "tuple" is a set of items that together comprise a whole — a generic form for double/triple/quadruple/tr quintuple/etc.)

The following table gives examples of origin comparisons with the URL http://store.company.com/dir/page.html:

<table>
<thead>
<tr>
<th>URL</th>
<th>Outcome</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://store.company.com/dir2/other.html">http://store.company.com/dir2/other.html</a></td>
<td>Same origin</td>
<td>Only the path differs</td>
</tr>
<tr>
<td><a href="http://store.company.com/dir/inner/another.html">http://store.company.com/dir/inner/another.html</a></td>
<td>Same origin</td>
<td>Only the path differs</td>
</tr>
<tr>
<td><a href="https://store.company.com/page.html">https://store.company.com/page.html</a></td>
<td>Failure</td>
<td>Different protocol</td>
</tr>
<tr>
<td><a href="http://store.company.com:81/dir/page.html">http://store.company.com:81/dir/page.html</a></td>
<td>Failure</td>
<td>Different port (http:// is port 80 by default)</td>
</tr>
<tr>
<td><a href="http://news.company.com/dir/page.html">http://news.company.com/dir/page.html</a></td>
<td>Failure</td>
<td>Different host</td>
</tr>
</tbody>
</table>
Iframes (Inline Frames)

- Enable you to embed a webpage inside another webpage
CORS (Relaxes SOP)

• Cross-Origin Resource Sharing
  – Specifies when specific other origins can make a request for data on a different origin

• https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

  • Access-Control-Allow-Origin: https://foo.example
  • Access-Control-Allow-Methods: POST, GET, OPTIONS
  • Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
  • Access-Control-Max-Age: 86400
When CORS is Not Needed

Some requests don't trigger a CORS preflight. Those are called simple requests, though the Fetch spec (which defines CORS) doesn't use that term. A simple request is one that meets all the following conditions:

- One of the allowed methods:
  - GET
  - HEAD
  - POST

- Apart from the headers automatically set by the user agent (for example, Connection, User-Agent, or the other headers defined in the Fetch spec as a forbidden header name), the only headers which are allowed to be manually set are those which the Fetch spec defines as a CORS-safelisted request-header, which are:
  - Accept
  - Accept-Language
  - Content-Language
  - Content-Type (please note the additional requirements below)

- The only type/subtype combinations allowed for the media type specified in the Content-Type header are:
  - application/x-www-form-urlencoded
  - multipart/form-data
  - text/plain

- If the request is made using an XMLHttpRequest object, no event listeners are registered on the object returned by the XMLHttpRequest.upload property used in the request; that is, given an XMLHttpRequest instance xhr, no code has called xhr.upload.addEventListener() to add an event listener to monitor the upload.

- No ReadableStream object is used in the request.
When CORS is Needed

What requests use CORS?

This cross-origin sharing standard can enable cross-origin HTTP requests for:

- Invocations of the XMLHttpRequest or Fetch APIs, as discussed above.
- Web Fonts (for cross-domain font usage in @font-face within CSS), so that servers can deploy TrueType fonts that can only be loaded cross-origin and used by web sites that are permitted to do so.
- WebGL textures.
- Images/video frames drawn to a canvas using drawImage().
- CSS Shapes from images.

This is a general article about Cross-Origin Resource Sharing and includes a discussion of the necessary HTTP headers.

Revisiting SRI Relative to Crossing Origins

- `<script src=https://example.com/example-framework.js integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlIGYl1kPzQho1wx4JwY8wC" crossorigin="anonymous"></script>`
  - anonymous = No credentials (e.g., cookies)
  - use-credentials

From https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/crossorigin