16. Web Security and Attacks (Part 1)

Blase Ur and David Cash

February 16th, 2022

CMSC 23200 / 33250

THE UNIVERSITY OF CHICAGO

JavaScript

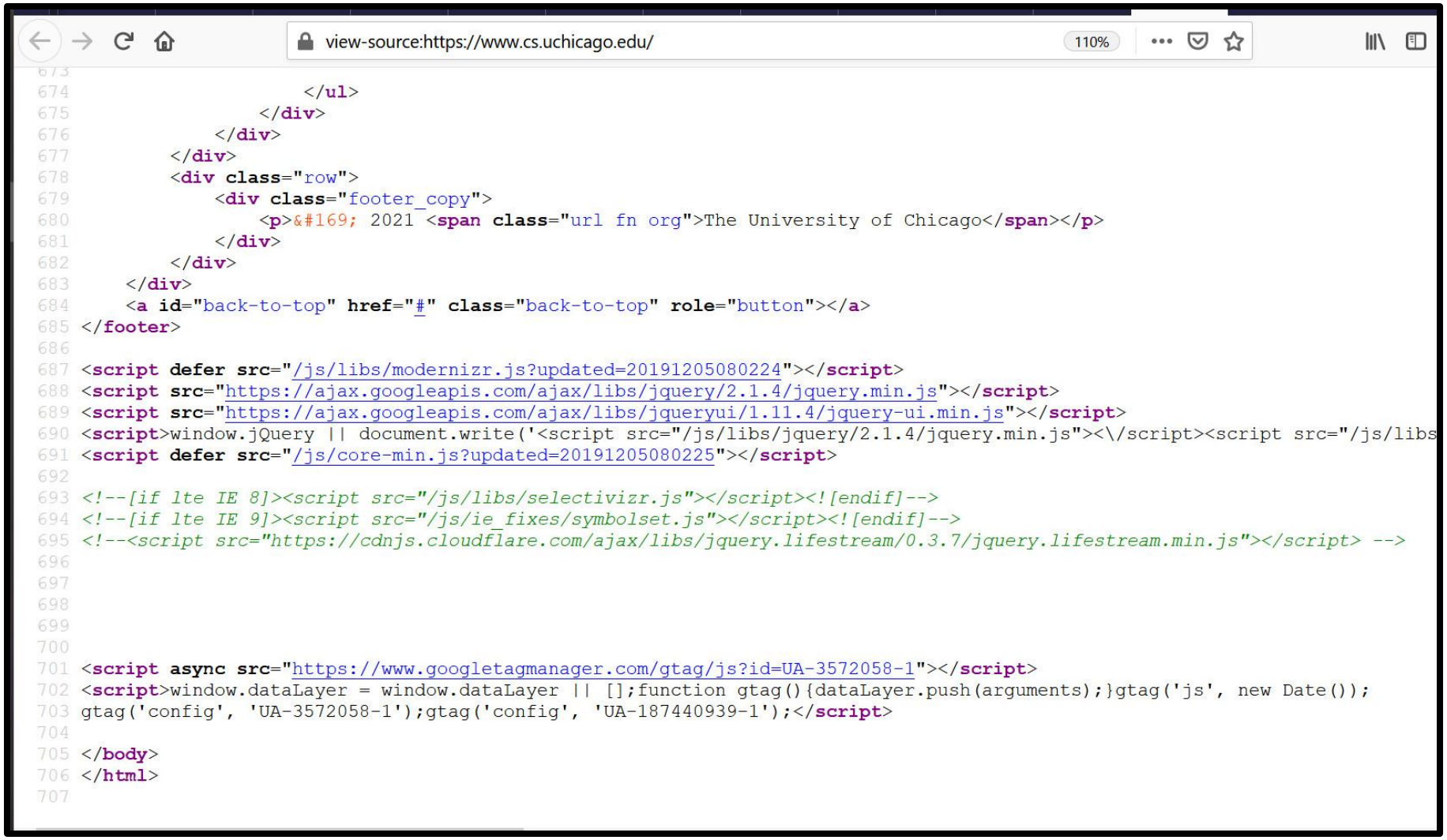
Interactive Pages?

- JavaScript!
  - The core idea: Let's run (somewhat) arbitrary 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 access to DOM)
  - $(".test").hide() hides all elements with class="test"
- JQueryUI
- Bootstrap
- Angular / React
- Google Analytics *(sigh)*

Importing Javascript Libraries




```
                    </ul>
                </div>
            </div>
        </div>
        <div class="row">
            <div class="footer_copy">
                <p>&#169; 2021 <span class="url fn org">The University of Chicago</span></p>
            </div>
        </div>
    </div>
    <a id="back-to-top" href="#" class="back-to-top" role="button"></a>
</footer>

<script defer src="/js/libs/modernizr.js?updated=20191205080224"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.4/jquery-ui.min.js"></script>
<script>window.jQuery || document.write('<script src="/js/libs/jquery/2.1.4/jquery.min.js"><\/script><script src="/js/libs
<script defer src="/js/core-min.js?updated=20191205080225"></script>

<!--[if lte IE 8]><script src="/js/libs/selectivizr.js"></script><![endif]-->
<!--[if lte IE 9]><script src="/js/ie_fixes/symbolset.js"></script><![endif]-->
<!--<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.lifestream/0.3.7/jquery.lifestream.min.js"></script> -->



<script async src="https://www.googletagmanager.com/gtag/js?id=UA-3572058-1"></script>
<script>window.dataLayer = window.dataLayer || [];function gtag(){dataLayer.push(arguments);}gtag('js', new Date());
gtag('config', 'UA-3572058-1');gtag('config', 'UA-187440939-1');</script>

</body>
</html>
```

Do You Have the Right .js File?

- Subresource integrity (SRI): https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity
- <script src="https://example.com/example-framework.js" integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC" crossorigin="anonymous"></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

Same-Origin Policy (SOP)

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

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 /quintuple/etc.)

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

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner /another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

Iframes (Inline Frames)

- Enable you to embed a webpage inside another webpage

Image from https://www.thoughtco.com/when-to-use-iframes-3468667

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.

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

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.

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

Revisiting SRI Relative to CORS

- <script src=https://example.com/example-framework.js integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC“ crossorigin="anonymous"></script>
  - anonymous = No credentials (e.g., cookies)
  - use-credentials

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

CSRF

Cross-Site Request Forgery (CSRF)

- Goal: Make a user perform some action on a website without their knowledge
  - Trick the browser into having them do this
- Main idea: Cause a user who's logged into that website to send a request that has lasting effects

Cross-Site Request Forgery (CSRF)

- Prerequisites:
  - *Victim* is logged into *important.com* in a particular browser
  - *important.com* accepts GET and/or POST requests for important actions
  - *Victim* encounters *attacker's* code in that same browser

CSRF Example

- *Victim* logs into *important.com* and they stay logged in (within some browser)
  - Likely an auth token is stored in a cookie
- *Attacker* causes *victim* to load `https://www.important.com/transfer.php?amount=1000000000&recipient=blase`
  - This is a GET request. For POST requests, auto-submit a form using JavaScript
- Transfer money, cast a vote, change a password, change some setting, etc.

CSRF: How?!

- On *blaseur.com* have Cat photos
- Send an HTML-formatted email with <img src="`URL`">
- Have a hidden form on *blaseur.com* with JavaScript that submits it when page loads
- Etc.

CSRF: Why Does This Work?

- Recall: Cookies for *important.com* are automatically sent as HTTP headers with every HTTP request to *important.com*
- *Victim* doesn't need to visit the site explicitly, but their browser just needs to send an HTTP request
- Basically, the browser is confused
  - "Confused deputy" attack

CSRF: Key Mitigations

- Check HTTP referrer *(less good)*
  - But this can sometimes be forged
- CSRF token *(standard practice)*
  - "Randomized" value known to *important.com* and inserted as a hidden field into forms
  - Key: not sent as a cookie, but sent as part of the request (HTTP header, form field, etc.)