EECS 388: Lab 5

- HTML and JavaScript
- Mechanics of XSS and CSRF

Current Assignments

Reminder: Canvas quizzes due the day before the next lecture

- Project 2: Web Security
  - **Due Thursday, October 5th at 6 PM**
  - Coverage:
    - SQL Injection (Lecture 8 and previous lab)
    - XSS Attack (Lecture 8 and today's lab)
    - CSRF Attack (Lecture 8 and today's lab)
  - Please see supplemental lecture videos for material that didn't fit in Lecture 8

If you haven't started, start now!
Partners are optional.

HTML, HTTP & JavaScript

Hypertext Markup Language (HTML)

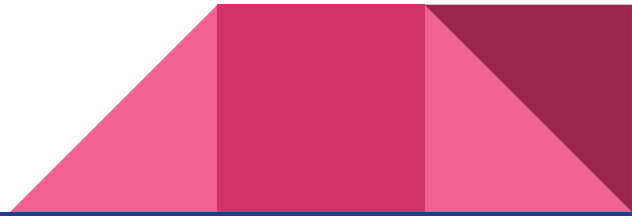
- Opening & closing tags build a tree structure (Document Object Model (DOM))
- Describes objects to display on the web page
- Tags `<a>` may contain attributes (`href=`)

```
<html>
    <head>
        <title>Hello</title>
    </head>
    <body>
        <p>Some paragraph.
       <a href="https://example.com">This is a link</a>.</p>
    </body>
</html>
```

- Try it: `curl -v https://example.com` ; Developer Tools

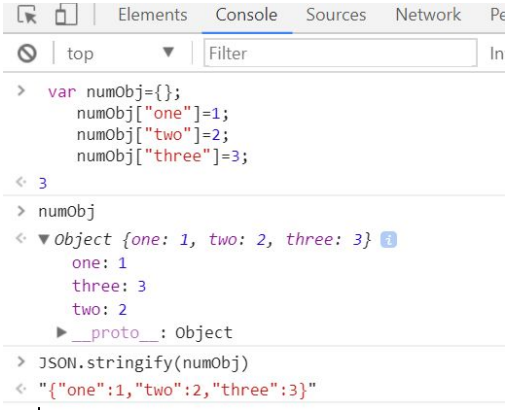
Hypertext Transfer Protocol (HTTP)

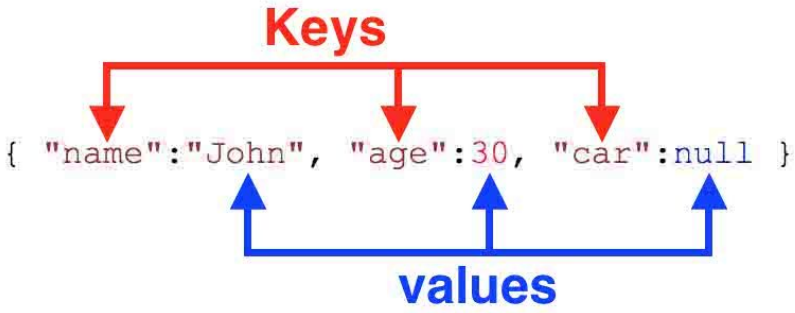
- Requests
  - `GET`
    - Request to “get” the page at a given URI
    - Can send data in the URI, but not in the request body
    - *Asking someone for information*
    - e.g., clicking a link to load a web page
    - Not supposed to have side-effects
  - `POST`
    - Submitting some data, usually in the request body
    - *Sending someone a package*
    - e.g., signing in with username and password
    - Side-effects allowed
  - Many others too—take EECS 485!



JavaScript

- Scripting language that runs in the context of a page and can directly interact with HTML (i.e., modify elements in the DOM)
- See example:
  - https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onclick_html
- JavaScript values are either objects (similar to dictionaries), arrays, or primitives





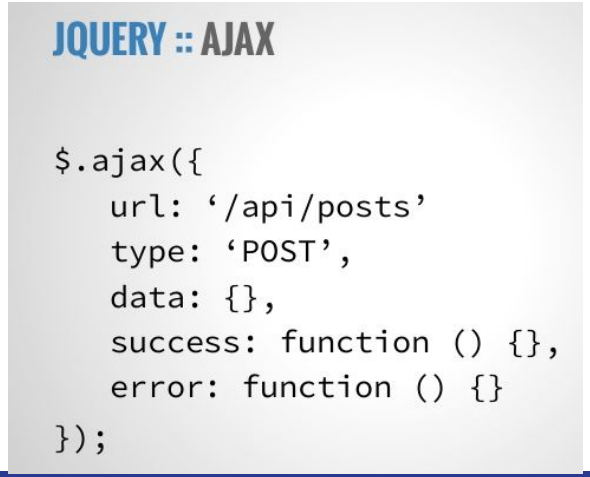
JSON: JavaScript Object Notation

- Standard text-based format for representing structured data in JavaScript object format
  - Consists of attribute-value pairs, arrays, nested objects
- Commonly used to transmit data in web applications
  - Sending data from server to client, vice-versa

https://www.json.org/json-en.html

jQuery

- jQuery is a **JS library** that simplifies DOM interaction and event handling
  - Provides simple **selector** functions for *matching HTML elements by ID, class*, etc.
  - Allows for easy HTTP request creation/sending
    - **`$.ajax`** is the most generic; **`$.get`** and **`$.post`** are more ‘simple’ versions of **`$.ajax`**



$(document).ready()

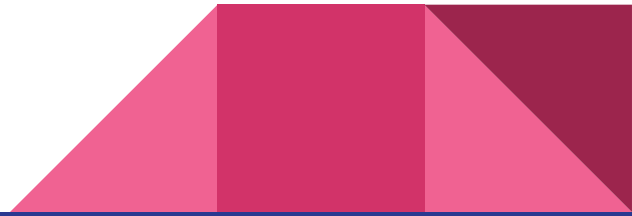
- A page can’t be manipulated safely until the DOM is “ready”
- This will run the code only after the DOM is fully loaded

```
$(document).ready(function() {
    console.log(“DOM ready!”);
});
```

Shorthand:

```
$(function() {
    console.log(“DOM ready!”);
});
```

- Takeaway: If you’re modifying/accessing data within the DOM, wait until it’s fully loaded!



JavaScript vs. jQuery

JavaScript

A scripting language to work with HTML

ID selection:

```
var el = document.getElementById('hello');
```

Class selection:

```
var el = document.getElementByClass('bye');
```

Get text (and print to console):

```
console.log(el.innerHTML);
```

jQuery

JS library that simplifies DOM interaction

ID selection:

```
var el = $("#hello");
```

Class selection:

```
var el = $('.bye');
```

Get text (and print to console):

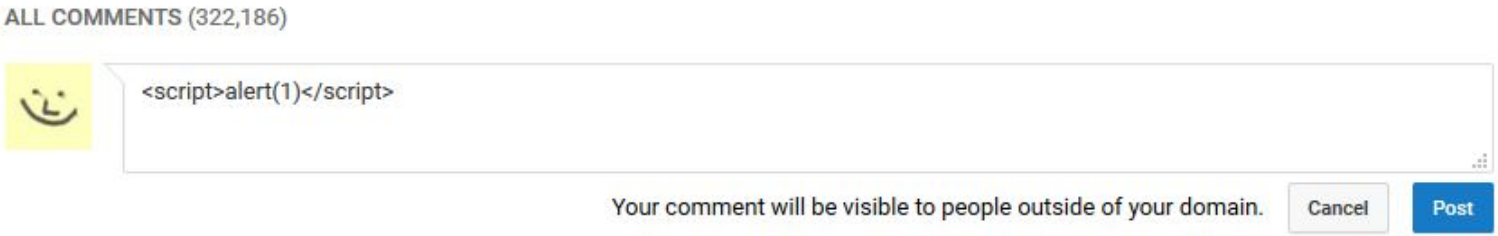
```
console.log(el.text());
```

There is nothing that jQuery can do that JavaScript can't, but jQuery makes it way easier!

Cross-Site Scripting (XSS)

XSS

- Cross Site Scripting (XSS)
  - Injecting code into the DOM that is executed when the page loads
  - Can be either **reflected** (mirrored from URL into the page) or **stored** (e.g. in a comment)
- An innocent example



- On load, the script is executed

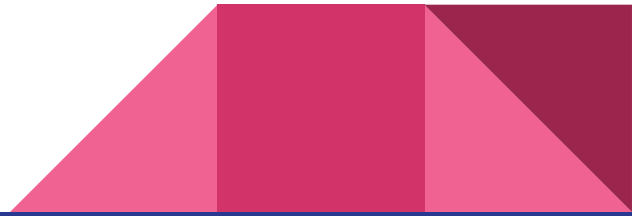
Exercise: XSS Attack

- Navigate to https://goo.gl/CivpX2
- Working alone or with people around you, see how many levels you can get through in the next 5–10 minutes
- Let me know if you have any questions!

(Protip: 4 hands on 1 keyboard = double the hacking)

Project 2: XSS

- Your goal
  - Steal a user’s search history and send it to yourself
  - You will submit a URL
  - Must work in specific Firefox version within Docker



Exercise: Try JS in the Browser Console

1. Open **BUNGLE!** inside your Docker Firefox
2. Create a new user, login, and follow along
    - ***DO NOT use an important password to test an insecure site!!!***
3. Open Dev Tools, go to Console
4. Try some sample code w/ JS and JQuery:

Remember!

NEVER use an important password for an account on an insecure site!

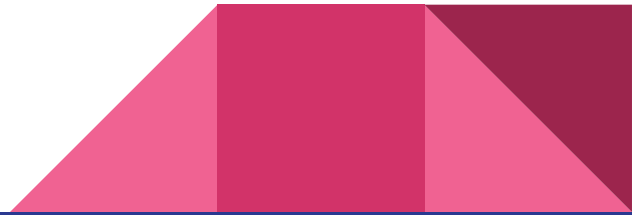
(Also, never reuse a password at all)

Project 2: JavaScript

You can use jQuery within Project 2 (but not other external scripts)

```
<html>
    <body>
        <script>
            $.get("https://example.com"); // Send an http get request using jQuery
            // Cannot read the response if the origin is different, due to same origin policy.
        </script>
    </body>
</html>
```

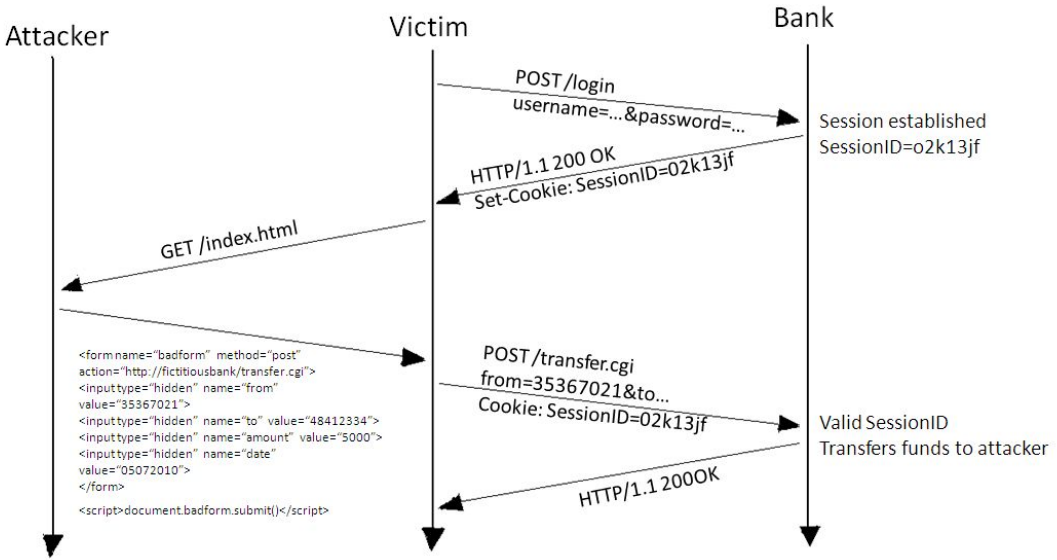
Since **BUNGLE!** already has jQuery loaded, you **don't** need to import it yourself when performing XSS



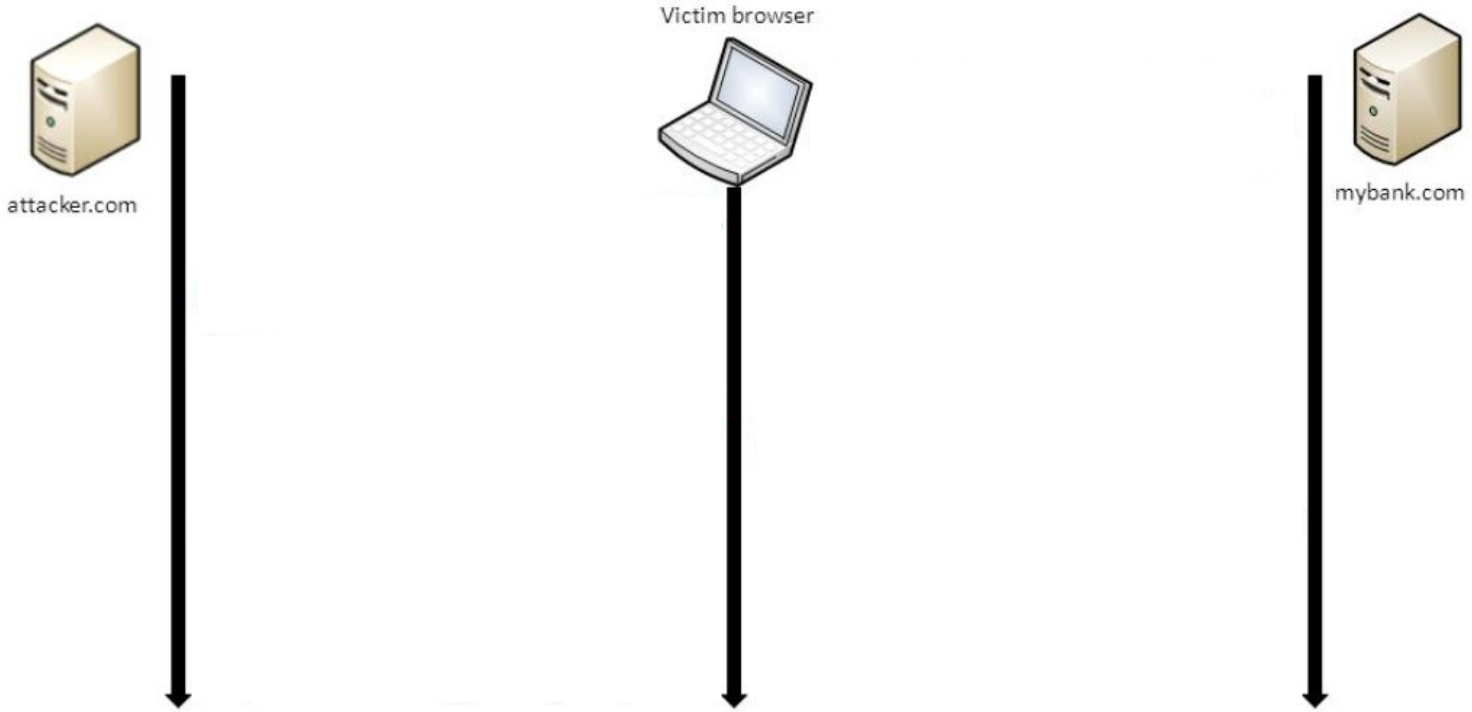
Cross-Site Request Forgery (CSRF)

What is CSRF?

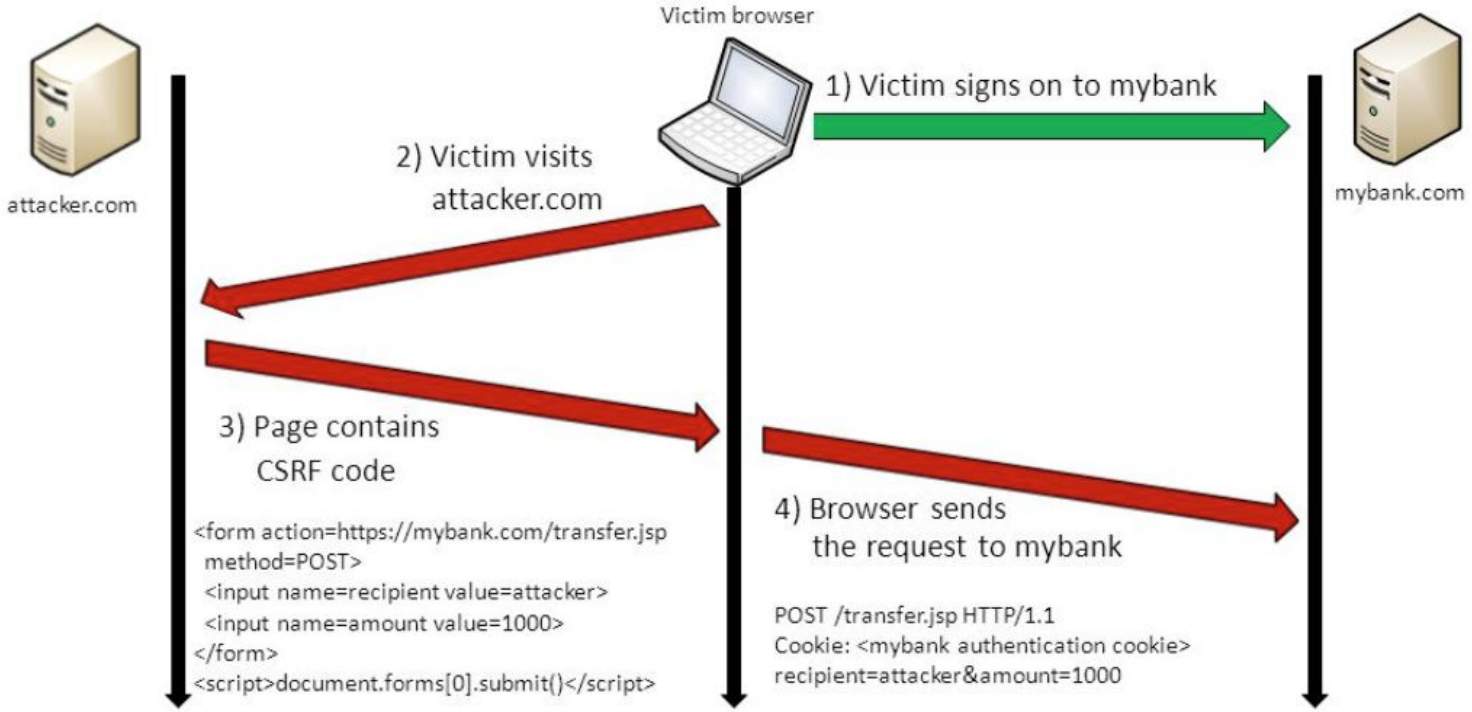
- An attack that makes a victim *execute commands* against their will on another website to which they are *currently authenticated*
- In the project, you are trying to force a user to log into your own attacker account without their knowledge
  (Why would this benefit the attacker?)



Step By Step



Step By Step


Victim browser
attacker.com
mybank.com
1) Victim signs on to mybank
2) Victim visits attacker.com
3) Page contains CSRF code
<form action=https://mybank.com/transfer.jsp method=POST>
<input name=recipient value=attacker>
<input name=amount value=1000>
</form>
<script>document.forms[0].submit()</script>
4) Browser sends the request to mybank
POST /transfer.jsp HTTP/1.1
Cookie: <mybank authentication cookie>
recipient=attacker&amount=1000

HTML Forms

- Forms are a common way to send POST requests

The action is a URL where the POST is sent

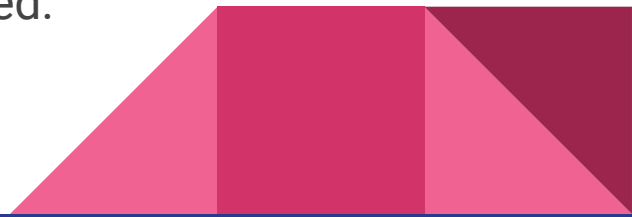
```
<html>
    ...
    <body>
        <form action="https://example.com/submitcomment" method="post" id="commentForm">
            <input type="text" name="comment">
            <input type="hidden" name="id" value="someval">
            <input type="submit" value="Submit">
        </form>
        <script>$("#commentForm").submit()</script>
    </body>
</html>
```

The inputs define the data sent in the request

Loading this page will send the POST request, writing a comment on https://example.com/submitcomment
But also redirects the user to another page!

Project 2: CSRF

- Your goal
  - Get someone to login to the attacker account without their knowledge or consent
- What this should look like
  - Victim is logged into **BUNGLE!**
  - Victim clicks on your link while browsing another page
    - *“Click this link for $1,000,000!!!”*
  - When Victim goes back to **BUNGLE!** and does another search (or refreshes the page) they are now logged in as `attacker`
- Your attack page may need to import jQuery.
  See spec: only one particular jQuery version is allowed.



Project Resources: HTML Forms with Ajax

```
<html><body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<form id="form1" method="post" action="https://example.com/submitcomment" style="display: none;">
      <input name="comment" value="hello world">
</form>
<script>
      $.ajax({
            type: 'POST',
            data: $('#form1').serialize(),
            url: 'https://example.com/submitcomment',
            xhrFields: {withCredentials: true} // Necessary if you want to send and set cookies
      });
</script>
</body></html>
```

Ajax → Asynchronous. Doesn't redirect the user!

- What does this code do differently?

Project 2: Big Picture

Exploit three classic attacks and know how to prevent them

- **SQL Injection** provides malicious input that gets interpreted as SQL code by the server
  - Defense: Always use SQL prepared statements
- **XSS** provides malicious input that gets interpreted as JavaScript in the victim's browser
  - Defense: Validate inputs and escape outputs; use Content Security Policy (CSP)
- **CSRF** performs actions as the user on another site
  - Defense: SameSite cookie attribute, dynamic token validation in forms