

EECS 388: Lab 5

- Project 1 Part 2 Review
- HTML and JavaScript
- Mechanics of XSS and CSRF



Review: Project 1, part 2

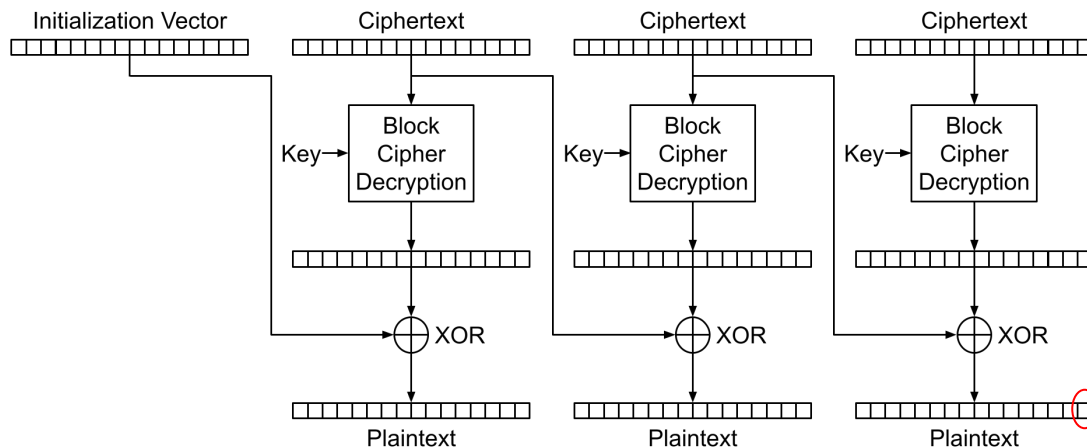
Padding Oracle Attack

- General Idea
 - There exists some “oracle” that allows the attacker to differentiate between failure and success
- Key idea
 - If an attacker can tell whether an integrity check fails due to incorrect padding or an incorrect MAC, they can figure out the padding scheme.
 - Then an attacker can spoof message on the system
- Protection
 - **ALWAYS** check the MAC **before** decrypting and checking the padding scheme
 - Cryptographic Doom Principle!



Padding Oracle Attack

- Why is it dangerous for a server to send an error for incorrect padding?
- Explicitly explain how you could exploit this to solve for this byte



Hint: you can manipulate whatever bytes of ciphertext you'd like, send the ciphertext to Bob, and see whether he gives you a padding error or not

Cryptographic doom principle: If you need to do any work before checking the MAC, then you are doomed!

Padding Oracle Attack

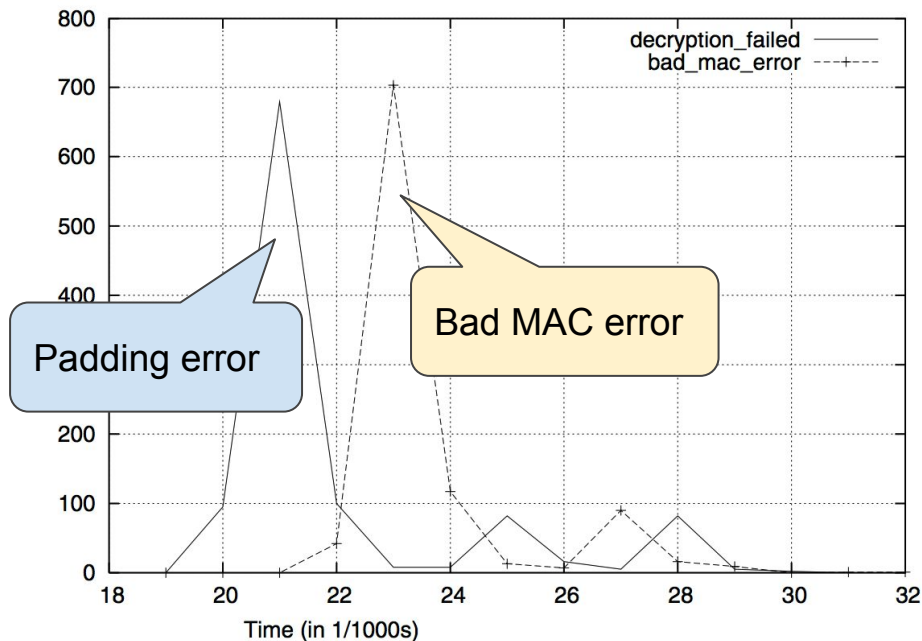


Image credit: <https://link.springer.com/content/pdf/10.1007/b11817.pdf#page=596>

- What if we don't get a "padding error" or "MAC error"?
 - What if the server just returns "error"?
 - Is there still the potential for a padding oracle?

Approaches to AEAD

1. MAC-then-Encrypt

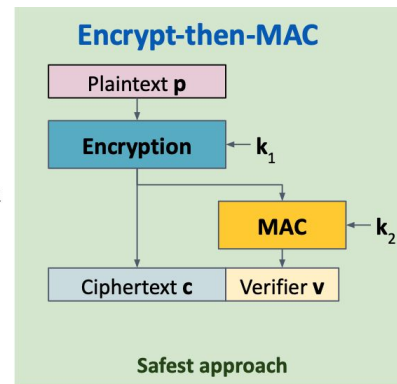
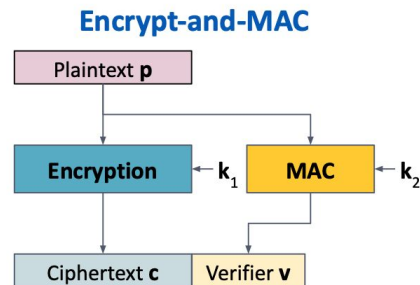
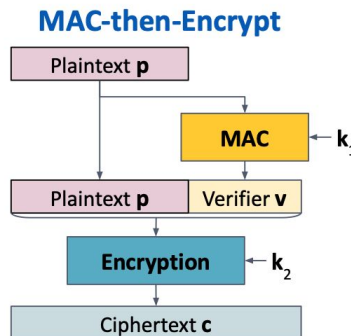
- Never!

2. Encrypt-and-MAC

- Still nope

3. Encrypt-then-MAC

- The only safe way



Cryptographic doom principle: If you need to do any work before checking the MAC, then you are doomed!

Bleichenbacher Attack

- *Vulnerability*: Small public key and improper verification means we can forge a signature
- *Attack*: Construct proper padding, take cube root, do “magic” math. Why?
 - When it was cubed, the signature was floored
 - Adding 1 only messes with arbitrary bytes and the signature will be valid (even though it shouldn't be)

00 01 FF 00 30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 XX XX XX ... XX YY YY YY ... YY
ASN.1 “magic” bytes denoting type of hash algorithm SHA-256 digest (32 bytes) $k/8 - 55$ arbitrary bytes



Current Assignments

Reminder: Canvas quizzes due the day before the next lecture

- Project 2: Web Security
 - **Due Thursday, October 3rd 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

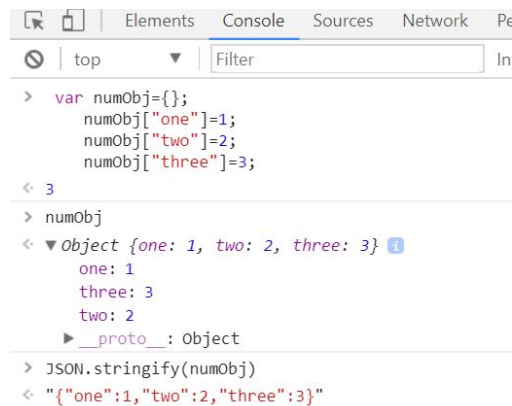
Hypertext Transfer Protocol (HTTP)

- 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!



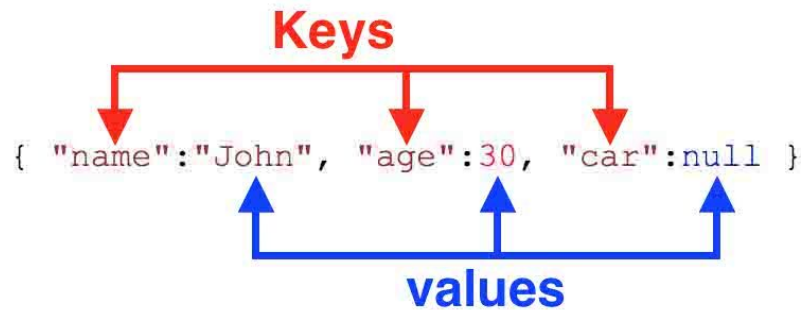
JavaScript

- 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



```
var numObj={};
numObj["one"]=1;
numObj["two"]=2;
numObj["three"]=3;

numObj
// Object {one: 1, two: 2, three: 3}
JSON.stringify(numObj)
// '{"one":1,"two":2,"three":3}'
```



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



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**

JQUERY :: AJAX

```
$.ajax({  
  url: '/api/posts'  
  type: 'POST',  
  data: {},  
  success: function () {},  
  error: function () {}  
});
```

`$(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.getElementsByClassName('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!

Rules for Setting and Reading Cookies

Setting a cookie:

A site can set a cookie for its own domain or any *parent domain*, as long as the parent domain is not a **public suffix**.

Example: **login.site.com** attempts to set cookie

login.site.com	allowed
site.com	allowed
other.site.com	prohibited
different.com	prohibited
com	prohibited (public suffix)

Caution: You don't know which domain set a cookie when you receive it.

Example: **club.eecs.umich.edu** can set cookies for **umich.edu** (since latter isn't a public suffix).

Reading a cookie:

A site can read cookies set for its own domain **or any parent domains**.*

Example: **login.site.com** can read cookies for

login.site.com	yes
site.com	yes
other.site.com	no
different.com	no

Caution: Cookies also specify a *path* on the site, but (without `HttpOnly`) it's for efficiency only. DOM origins aren't isolated by path, so scripts can read cookies set for any path in the origin.

Suppose cookie set for **x.com/b**. Then **x.com/a** can do: **alert(frames[0].document.cookie);**

* If **domain=** attribute is *unset*, some browsers disallow reading by subdomains, but can't rely on this behavior.

When are these cookies sent?

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

name = cookie3
value = mycookievalue
domain = site.com
path = /my/home

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>	Yes	Yes	Yes
site.com/my	No	Yes	No

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

ALL COMMENTS (322,186)



`<script>alert(1)</script>`

Your comment will be visible to people outside of your domain.

Cancel

Post

- 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>
```

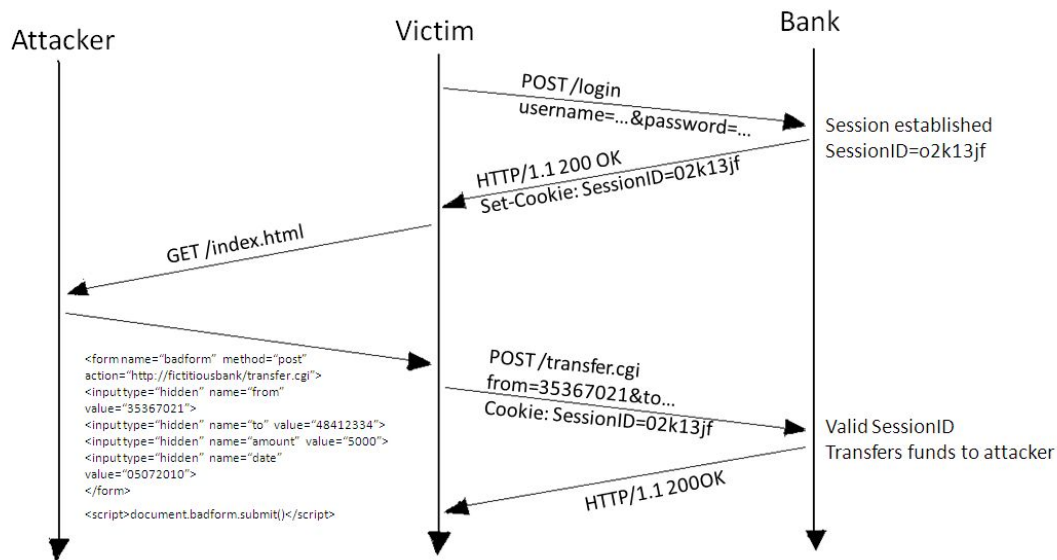
Since **BUNGLE!** already has jQuery loaded, you **don't** need to import it yourself when performing XSS



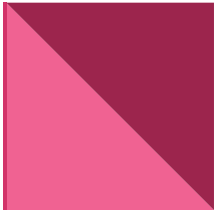
Cross-Site Request Forgery (CSRF)

What is CSRF?

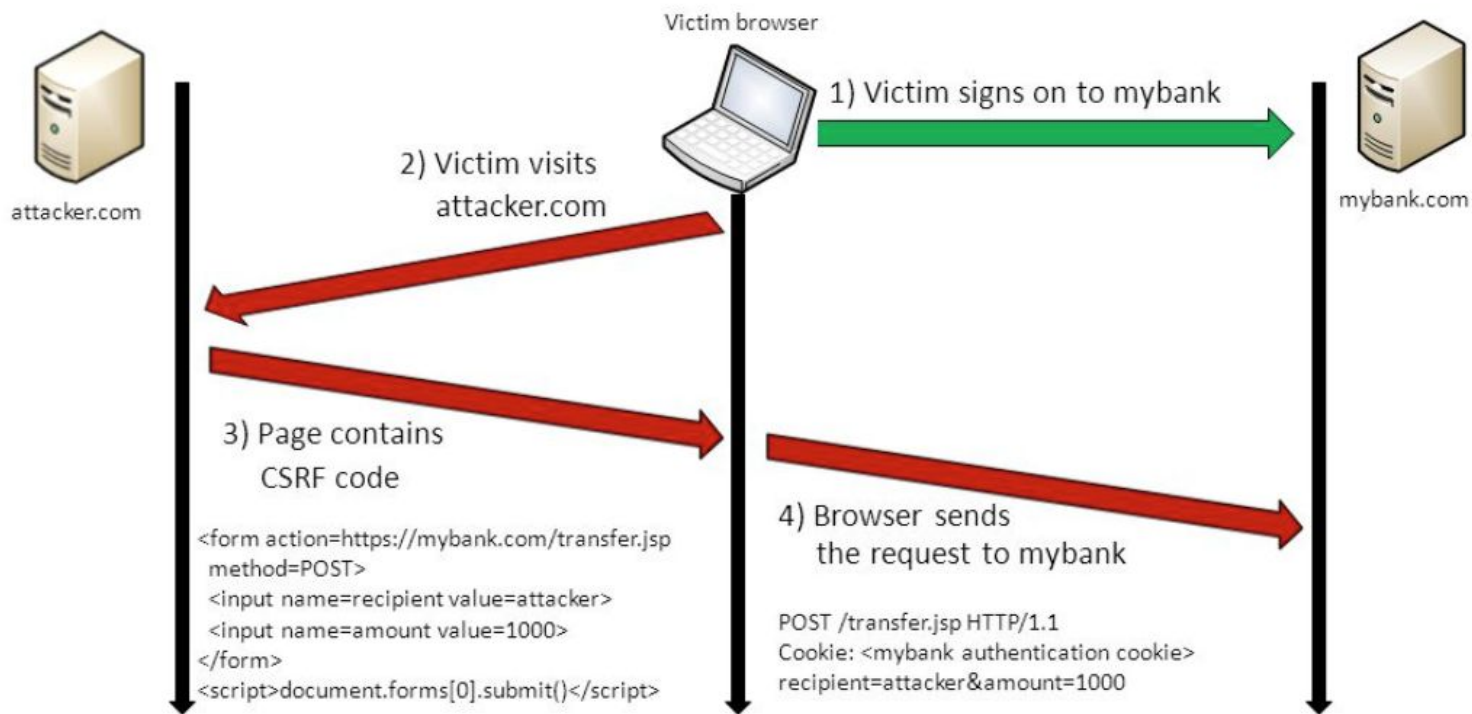
- An attack that makes a victim *execute commands* against their will on another website to which they are *currently authenticated*
- In the project, you are trying to force a user to log into your own attacker account without their knowledge
(Why would this benefit the attacker?)



Step By Step



Step By Step



HTML Forms

- Forms are a common way to send POST requests

```
<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 action is a URL where the POST is sent

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!

Analyzing Requests

- Right click on a webpage and click 'Inspect'
- Go to the network tab



filter by type of request

a request that was made

A screenshot of the Chrome DevTools Network tab. The 'All' filter is selected, and a red oval highlights the filter buttons. Below the filters, there are checkboxes for 'Blocked response cookies', 'Blocked requests', and '3rd-party requests'. A timeline at the top shows request durations. Below the timeline is a table of network requests. The first request, 'example.com', is highlighted with a red box. The second request, 'pageScript.bun...', is partially visible.

Name	Status	Type	Initiator	Size	Time
example.com	304	document	Other	84 B	32 ms
pageScript.bun...	200	script	contentScript.bundle	12.4 kB	164 ms

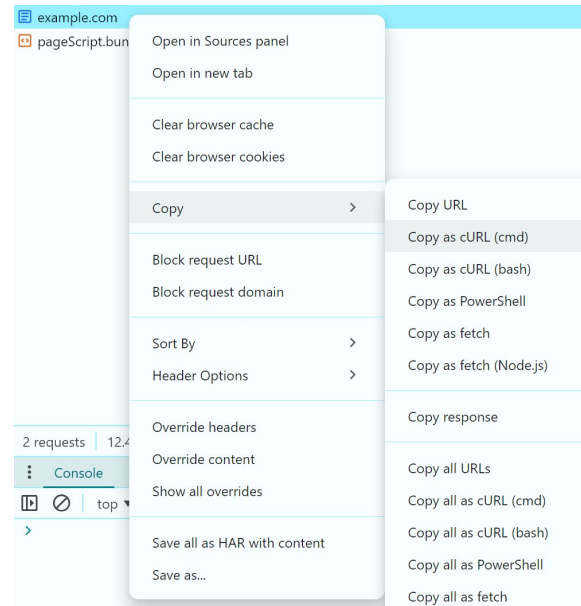
Replicating Requests

Right click on request -> Copy -> Copy as cURL (cmd)

Example response:

```
curl "https://example.com/" ^  
-H "accept:  
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,app  
lication/signed-exchange;v=b3;q=0.7" ^  
-H "accept-language: en-US,en;q=0.9" ^  
-H "cache-control: max-age=0" ^  
-H "if-modified-since: Thu, 17 Oct 2019 07:18:26 GMT" ^  
-H "if-none-match: ^\^"3147526947+gzip^\^" ^  
-H "priority: u=0, i" ^  
-H ^"sec-ch-ua: ^\^"Chromium^\^";v=^\^"128^\^", ^\^"Not;A=Brand^\^";v=^\^"24^\^", ^\^"Google  
Chrome^\^";v=^\^"128^\^" ^  
-H "sec-ch-ua-mobile: ?0" ^  
-H ^"sec-ch-ua-platform: ^\^"Windows^\^" ^  
-H "sec-fetch-dest: document" ^  
-H "sec-fetch-mode: navigate" ^  
-H "sec-fetch-site: none" ^  
-H "sec-fetch-user: ?1" ^  
-H "upgrade-insecure-requests: 1" ^  
-H "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/128.0.0.0 Safari/537.36"
```

request headers



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

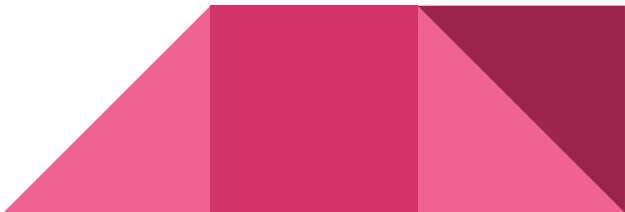
```
<html><body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<form id="form1" method="post" action="https://example.com/submitcomment" style="display: none;">
  <input name="comment" value="hello world">
</form>
<script>
  $.ajax({
    type: 'POST',
    data: $('#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
- 



See you next week!

