Thea Traw

Cookies & cross-site scripting

COOKIES

1. Go to FDF and use your browser's Inspector to take a look at your cookies for cs338.jeffondich.com. Are there cookies for that domain? What are their names and values?

   Before logging in, there's one cookie → theme: default

   After logging in as Alice, there's two cookies → theme: default; session: .eJwlzjEOwzAIQNG7eO5gMDaQy0QEg9o1aaaqd6-lrl9v-J-y5xnXs2zv845H2V-zbEUJTeY0Te_B6mzqg7kRA9GEPJRbxcBxJDq7KHFrddFRvYK0HpqBMhMDvFtQlSkWSEDm3dM4qnVXMCNZ1akHcFuySvfpZY3cV5z_GyjfH73ML3Q.ZUlMYg.0gJvFyiryYV3VdiBg1bVDiMU8Gg

2. Using the "Theme" menu on the FDF page, change your theme to red or blue. Look at your cookies for cs338.jeffondich.com again. Did they change?

   Yes. They change each time you switch the theme–whether to red, blue, or default. The session cookie remains the same.

3. Do the previous two steps (examining cookies and changing the theme) using Burpsuite (either on your base OS or on Kali). What "Cookie:" and "Set-Cookie:" HTTP headers do you see? Do you see the same cookie values as you did with the Inspector?

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Wed, 08 Nov 2023 05:48:34 GMT
4 Content-Type: text/html; charset=utf-8
5 Connection: close
6 Set-Cookie: theme=default; Expires=Tue, 06 Feb 2024 05:48:34 GMT; Path=/
7 Vary: Cookie
8 Content-Length: 32165
```

   There is a Set-Cookie header in the server response. This tells the browser to store a specific cookie (name: theme; value: default). When the theme is updated upon browser request, then a new Set-Cookie is sent from the server, with the updated value.

```
POST /fdf/login HTTP/1.1
Host: cs338.jeffondich.com
Content-Length: 40
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://cs338.jeffondich.com
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/119.0.6045.105 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q
=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,a
pplication/signed-exchange;v=b3;q=0.7
Referer: http://cs338.jeffondich.com/fdf/login
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: theme=default
Connection: close

email=alice%40example.com&password=alice
```

Then for all subsequent browser requests, there is a Cookie header that stores the theme cookie. When the theme is updated (red, default, blue), then the value of the cookie is updated as well.

So, yes, the cookie values are the same as what I saw with the Inspector.

4. Quit your browser, relaunch it, and go back to the FDF. Is your red or blue theme (wherever you last left it) still selected?

```
GET /fdf/ HTTP/1.1
Host: cs338.jeffondich.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/119.0.6045.105 Safari/537.36
Sec-Purpose: prefetch;prerender
Purpose: prefetch
Accept:
text/html,application/xhtml+xml,application/xml;q
=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,a
pplication/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: theme=red
Connection: close
```

Yes. I had selected the red theme before quitting the browser.

5. How is the current theme transmitted between the browser and the FDF server?

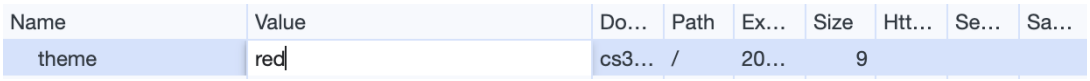
   In the cookie! Every HTTP request that the browser makes, it sends the theme cookie along with it (under the Cookie header) to the FDF server.

6. When you change the theme, how is the change transmitted between the browser and the FDF server?

```
GET /fdf/?theme=red HTTP/1.1
Host: cs338.jeffondich.com
Cookie: theme=blue
```

The new color is specified in the GET request. Then the server response includes a new Set-Cookie header with instructions for the browser to store the updated theme cookie.

7. How could you use your browser's Inspector to change the FDF theme without using the FDF's Theme menu?



Name	Value	Do…	Path	Ex…	Size	Htt…	Se…	Sa…
theme	red	cs3…	/	20…	9			

Change the cookie's text value directly. Then when you reload the page, the updated value you implemented will be the theme.

8. How could you use Burpsuite's Proxy tool to change the FDF theme without using the FDF's Theme menu? `

You could edit any HTTP request. That is, you could edit the Cookie header to have whatever value you want. Or you could change the HTTP request itself such that GET /fdf/?theme=* HTTP/1.1, where * is your updated theme value.

9. Where does your OS (the OS where you're running your browser and Burpsuite, that is) store cookies? (This will require some internet searching, most likely.)

According to this source, the OS stores cookies in a sqlite file on your computer. (Temporary cookies–such as session cookies–are stored in memory, but more permanent cookies are added to the cookies file.)

CROSS-SITE SCRIPTING (XSS)

1. Provide a diagram and/or a step-by-step description of the nature and timing of Moriarty's attack on users of the FDF.

   Moriarty writes some malicious Javascript and posts it on the FDF.

   Any amount of time later, an unsuspecting Alice clicks on the post and opens it. (This means that Moriarty cannot really control who he attacks–anyone could click on his post, and anyone could just as well not.)

   The Javascript in the body of the post is executed on Alice's browser. (The input was not "sanitized," and so key characters like <> are not stripped of their power and read as benign text.)

Moriarty's attack on Alice does whatever it does. (His code has the potential to access everything that Javascript can on Alice's browser.)

2. Describe an XSS attack that is more virulent than Moriarty's "turn something red" and "pop up a message" attacks. Think about what kinds of things the Javascript might have access to via Alice's browser when Alice views the attacker's post.

   Moriarty could access the cookies stored on Alice's browser. This means that he could send himself Alice's session cookie (via setting up a server to receive the information) and "join" her session (effectively logging himself in as Alice). Then Moriarty could make any number of nefarious posts. He could also send himself any other cookie data that he so desired and could access. There may be things stored in those cookies that Alice would really rather not have exposed.

3. Do it again: describe a second attack that is more virulent than Moriarty's, but that's substantially different from your first idea.

   Moriarty could implement a denial of service attack where random browser tabs are opened in some sort of continuous loop. (Actually, any infinite loop that Moriarty puts in his attack could be a real pain.) Maybe these tabs opened are just annoying, but they also could be to sites that Alice would not really want on her browser history.

   Or, Moriarty could cause some login chaos where he switches around session cookies between other users. I imagine this attack would go somewhat like this. First, Alice clicks on Moriarty's post. Her session cookie is recorded somewhere (probably on a server). Then, Bob clicks on Moriarty's post too. Then the server retrieves Alice's session cookie and replaces Bob's session cookie with it. Now, Alice and Bob are both logged into Alice's session. Bob's session cookie gets sent to the server. Then when Eve clicks on the post, she gets Bob's session cookie instead, and so on. This would cause a lot of issues, given that people would have access to accounts that they shouldn't have! And it is also denial of service, as one is no longer logged in as intended. However, this would only be successful if the timing worked out–two or more people need to have an active session cookie at the same time and also click on Moriarty's post–but it could happen!

4. What techniques can the server or the browser use to prevent what Moriarty is doing?

   The text needs to be sanitized so that key characters like <> are not recognized as such. (This could require putting escape characters around them.) It's super important to be careful with user input–especially something that will/can be executed. While this could cause many other issues, the browser could disallow users from using specific

characters in their input (even before the code is posted). Or it could clean up the text immediately afterward, and replace any key characters with nullified versions.