Ruben Boero and Vanessa Heynes

Part 1: Cookies

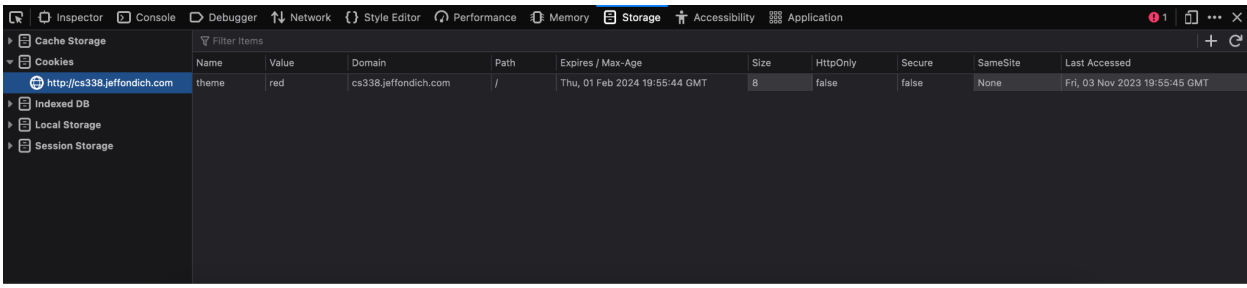
A. Go to FDF and use your browser's Inspector to take a look at your cookies for cs338.jeffondich.com. Are there cookies for that domain? What are their names and values?

a. Yes.

It's called theme and its value is default (theme=default)

B. 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?

a. Yes, the value changed from default to red:



C. 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?

Response

Pretty Raw Hex \n

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Fri, 03 Nov 2023 20:33:20
  GMT
4 Content-Type: text/html;
  charset=utf-8
5 Connection: close
6 Set-Cookie: theme=default;
  Expires=Thu, 01 Feb 2024 20:33:20
   GMT; Path=/
7 Vary: Cookie
8 Content-Length: 5839
9
10 <!DOCTYPE html>
11 <html lang="en">
12   <head>
13     <meta charset="utf-8">
14     <meta name="viewport" content
       ="width=device-width,
```

Search... 0 matches

a. Yes: (theme=default)

Response

Pretty Raw \n

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.18.0
  (Ubuntu)
3 Date: Fri, 03 Nov 2023
  20:37:03 GMT
4 Content-Type: text/html;
  charset=utf-8
5 Connection: close
6 Set-Cookie: theme=red;
  Expires=Thu, 01 Feb 2024
  20:37:03 GMT; Path=/
7 Vary: Cookie
8 Content-Length: 5843
9
10 <!DOCTYPE html>
11 <html lang="en">
12   <head>
13     <meta charset="utf-8">
14     <meta name="viewport"
```

b. Yes: (theme=red)

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

a. yes

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

a. The browser sends the cookie to the server when it accesses the site as part of the GET request:

```
1 GET /fdf/?theme=red HTTP/1.1
2 Host: cs338.jeffondich.com
3 Upgrade-Insecure-Requests: 1
```

b. The server sends the cookie to the browser via an HTTP Set-Cookie header (see line 6 in part a and b from question C).

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

a. It gets sent over as part of the HTML GET request (see part a above)

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

a. We can manually change the value of the cookie.

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

a. We can edit the packet before we forward it.

I. 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.)

a. /home/kali/.BurpSuite/pre-wired-browser
b. Generally, the cookies are stored in the same directory as the browser.
    i. For example, Firefox on Mac OS, stores my cookies at the following location:
    ii. /Users/rubenboero/Library/Application Support/Firefox/Profiles/umgwxhr4.default-release

Part 2: Cross-Site Scripting (XSS)

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

a. Mal creates a post on FDF that contains JS within script tags (<script> … </script>)
b. Later, a user clicks on Mal's post:
c. Browser asks web server for HTML of the page
d. Web server sends the HTML of the page
e. Browser begins to parse the HTML of the requested web page
f. The browser continues to parse the HTML until it encounters a JS script tag. At this point it will stop parsing HTML, load in the script, and execute it.
    i. The execution of the JS script is when Mal's attack on the users of FDF will be carried out
g. The browser will continue to parse the remaining HTML (and execute any other JS scripts that are included by Mal).
h. Once all of the HTML has been parsed, the web page is displayed to the user.

B. 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.

a. JS has access to cookies on a user's computer. Mal could gain access to that user's session ID (for a different, more important website), and therefore access one of the user's accounts without needing their password.

C. **Do it again: describe a second attack that is more virulent than Moriarty's, but that's substantially different from your first idea.**
   a. Mal can use JS to rewrite the HTML of a page. From there, Mal can direct the user to a website controlled by Mal. (Mal can phish the user.) If this website impersonates a site such as amazon or ebay, it's reasonable that an unaware user could mistake the fake site for the real amazon/ebay and provide Mal with their credit card information.

D. **What techniques can the server or the browser use to prevent what Moriarty is doing?**
   a. The server/browser can parse out the <> from the body of the user's post to the forum, and decide to keep them as functional elements of the page, or to convert them to be aesthetic only.
      i. The problems arise when JS can be executed from a user's machine, so being able to parse out the <script></script> tags and display them only as plain text would do a lot to mitigate the attack.
      ii. It may be nice if the user could change some HTML/css properties of their post, so leaving HTML tags would be okay.
   b. PortSwigger describes some additional prevention mechanisms:

How to prevent XSS attacks

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- **Filter input on arrival.** At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- **Encode data on output.** At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- **Use appropriate response headers.** To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the `Content-Type` and `X-Content-Type-Options` headers to ensure that browsers interpret the responses in the way you intend.
- **Content Security Policy.** As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.