Author: Kyle Machalec

**Part 1: Cookies**

a. There is one cookie called “theme”. Its value is “default”.
b. Changing the theme to red or blue changes the value of the “theme” cookie to “red” or “blue.
c. After changing the theme from default to blue, I see an HTTP request GET /fdf/?theme=blue with a cookie header "Cookie: theme=default". The HTTP response had a Set-Cookie header “Set-Cookie: theme=blue; Expires=Sat, 03 Feb 2024 21:34:52 GMT; Path=/”. I see the same cookie values in Burpsuite as I see in the Inspector.
d. Yes, the theme is still blue.
e. The current theme is transmitted between the browser and the server through the cookie. When the GET request is sent for a new theme, the server tells the browser to save the new value as a cookie. When a new GET request is sent to the server, the browser will send the server the value of the theme cookie, indicating the theme color.
f. The change is transmitted between the browser and the server when the server sends GET /fdf/?theme=_____, with the value of the new theme in the blank. The server responds by using the set-cookie header to tell the browser to store the new theme color as the theme cookie value. The next time the browser sends a GET request to the server, it will also send the new theme color stored in the theme cookie.
g. You can edit the value of the theme cookie in the Inspector. Therefore, you can change the value to “red”, “blue”, or “default”, and when the browser sends its next HTTP request, the server will change the theme color.
h. Using the proxy tool, you can edit the GET /fdf/?theme=____ value to whatever theme color you want, and the server will change the theme.
i. On my Windows computer using Chrome, cookies are stored in C:\Users\kylem\AppData\Local\Google\Chrome\User Data\Default\Network

**Part 2: Cross-Site Scripting (XSS)**

a.
    i. Browser asks the web server for HTML of the page
    ii. Web server send the browser the HTML of the page
    iii. Browser starts to parse the HTML of the page
    iv. The browser comes across the JavaScript script tag <>, so the browser chooses to execute the script. This executes Mal’s attack on users of the FDF.
    v. The browser finishes parsing the HTML and displays the page to the user.
b. Mal could use JS to put a link into their forum post. Mal could claim the link is from Amazon or some other legitimate company, but in actuality, the link directs Alice to a website controlled by Mal that imitates a legitimate company. From there, Mal could get personal information about Alice (i.e. credit card number) if she chooses to enter the information into Mal’s website.
c. Mal could get Alice to run JS that sends the value of her session ID cookie to a website controlled by Mal. This could be done by creating a new image object and setting the src attribute to Mal’s website. As a result, Alice's browser makes an HTTP request to Mal’s

website with the URL containing the session cookie. Once Mal obtains Alice's session ID cookie, Mal could impersonate Alice on FDF.

d. The server could parse out the JS script tag <> upon post and store them as plain text. The browser could also parse out the <> after receiving the HTML from the server but before executing any JS. The server could have certain criteria for valid input that it uses to filter out anything unexpected when a user makes a post. Additionally, any user-controllable data in HTTP responses from the server could be encoded so that it is not interpreted as executable content.