Doug Pham

=====Part 1=====

A. There is one cookie for this domain and it's name is theme and has a value of default.

B. Now the value for the cookie says red or blue and we also see the color of the page has changed.

C. Now using Burpsuite we see in the GET request a header that says Cookie: theme=default and in the response we see Set-Cookie: theme=default. Once we change the theme and forward it in burpsuite we see now this time a GET /fdf/?theme=red and in the response we see Set-Cookie: theme=red.

D. Yes, the red theme is still selected after relaunching.

E. The cookie is sent between the browser and FDF server which contains the current theme. When the GET request is sent, the browser saves the theme as a cookie so when a new GET request is sent, the browser communicates to the server what the current theme is.

F. The change is transmitted between the browser and FDF server within the GET request which looks like: GET /fdf/?theme=red HTTP/1.1. The server then responds by utilizing the Set-Cookie header to tell the browser what the theme is and the browser then makes this change and once again stores it for the next time a request is made.

G. The cookie's value is editable via the browser's Inspector, so you could change the value of the theme and that will make the change.

H. You could use Burpsuite's Proxy tool to edit the GET request and change it to GET /fdf/?theme=red which will also change the theme.

I. On my Windows 10 OS, the cookies for Google Chrome are stored in the following directory: C:\Users\Your_User_Name\AppData\Local\Google\Chrome\User Data\Default\Network.

=====Part 2=====

A. User's browser parses the javascripts and executes it

- Prof. Moriarty creates a new post on FDF which contains HTML and JavaScript code inside of <p> tags.
- A user is then able to view all the posts on FDF since they are all public so any user could potentially click on Prof. Moriarty post
- Once the user clicks on the post they will enable Moriarty's attack and the user's browser then parses the embedded code and runs it as JavaScript code when the post is viewed.
- JavaScript code includes variety of things such as getting elements, listening for events, and an alert
- Then the user gets a pop-up message which says "Mwah-ha-ha-ha!"

Doug Pham

B. An XSS attack that is more virulent than what Moriarty posted could be utilizing some Javascript that uses the function getElementbyId which could allow the attacker to get control of a value of the input field. This could lead to there being malicious or unwanted content being shown to the user.

C. Another possible attack would be if when the user clicked on the post, a pop up appeared which is another login screen and tells the user that their previous session expired and to re-login. From the user perspective this seems pretty legitimate as websites often to have session management and log you out after a certain time. So if the user were to put in their login credentials into the pop-up, this would give the attacker access to this sensitive information from which they could do whatever they want with the user's account.

D. Some techniques to prevent cross-site scripting is to filter the input from the user on both the browser and server sides and check for an expected format or look for any malicious content. Another way to prevent this is by encoding any user-generated data before it appears on the browser which can help the browser from interpreting the content as executable code. Another method is to implement and enforce CSP (Content Security Policy) which is a security standard to help mitigate attacks like these.