Bryce Bernstein

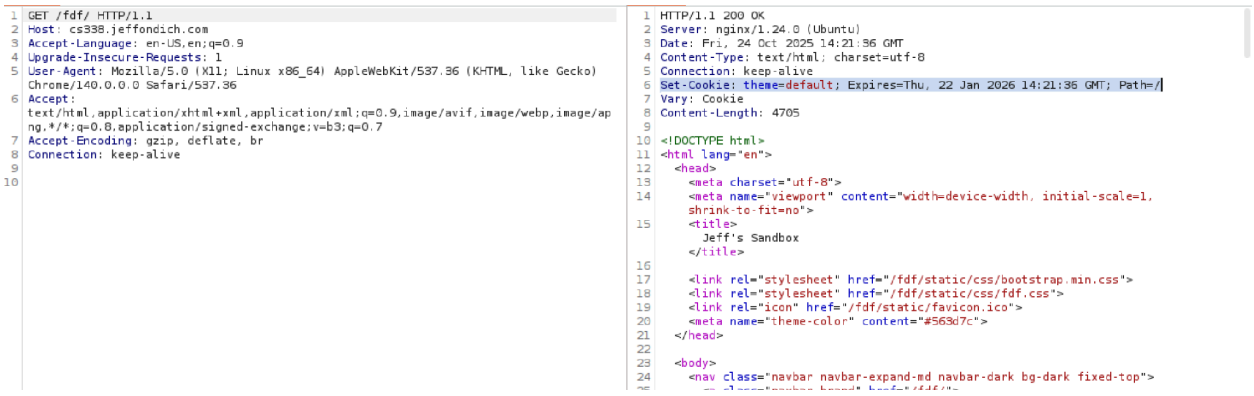
Part 1

a. Looking at the cookies on Firefox, it would seem that there is one cookie with the name theme and the value being whatever theme you have set (which is either blue, red, or default). When I log in, one more cookie shows up, which is a session cookie that stores your session ID so the browser knows you have logged in. This is just some random sequence of characters as best I can tell. It might be Base64. Looking at others' forum posts, I suspect some have a remember_token token, but I don't.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
session	.eJwlzjsOwjA...	cs338.jeffo...	/	Session	218	false	false	
theme	blue	cs338.jeffo...	/	Thu, 22 Jan 2026 ...	9	false	false	

b. Yes, the cookies change to whatever you have it set to between red, blue, and default.

c. Using Burpsuite, I can see the Set-Cookie header is used in a response to GET requests I make to FDF. This happens regardless of whether the Cookie header is included in the HTTP request or not. You can see it highlighted in the below image. On future requests where I have changed the theme, the response again has the header for Set-Cookie, but it will differs from what I have saved (see the second picture where the request says Cookie: theme=default, but the response says to set the cookie to red. Finally, in the third picture, we can see that when the regular page is called with no arguments, the Set-Cookie theme will return whatever the cookie is currently set to. If there is no cookie (like in the first example) it just sets it to default.



```
GET /fdf/ HTTP/1.1
Host: cs338.jeffondich.com
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/140.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Server: nginx/1.24.0 (Ubuntu)
Date: Fri, 24 Oct 2025 14:21:36 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Set-Cookie: theme=default; Expires=Thu, 22 Jan 2026 14:21:36 GMT; Path=/
Vary: Cookie
Content-Length: 4705

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
    shrink-to-fit=no">
    <title>
      Jeff's Sandbox
    </title>

    <link rel="stylesheet" href="/fdf/static/css/bootstrap.min.css">
    <link rel="stylesheet" href="/fdf/static/css/fdf.css">
    <link rel="icon" href="/fdf/static/favicon.ico">
    <meta name="theme-color" content="#563d7c">
  </head>

  <body>
    <nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
```

d. The theme persists because the cookie persists. If I opened it sometime after January 22, 2026, when the cookie expires, the theme would be gone.
e. It is transmitted through the Cookie request header for HTTP. It is just transmitted as plain text from the way Burpsuite presents it.
f. For specifics, you can reference my above pictures, but it looks at the arguments and tells the browser to set the theme cookie based on those arguments. After that, the cookie is set so the browser will remember.
g. Just by double clicking the value, I can change the text. For example, if I double click on “red” and type “blue,” the cookie changes so that it will be blue when I refresh.
h. I can turn on intercept and change the HTTP request header before it is ever sent to the server. Then the server will see it as if I sent a theme that asks for red, even though the cookie still stores blue. Because the server responds by telling the browser to set the cookie to red regardless of if the cookie is already set, this change persists.
i. This will depend some on different IDs (my profile has a weird ID), but the path to the cookies.sqlite file, which is where cookies are stored, is /home/kali/.mozilla/firefox/ybjm6ac6.default-esr/. In this, there is the file “cookies.sqlite,” which has all the data.

Part 2

a. OWASP provides a list of types of XSS attacks here. https://owasp.org/www-community/Types_of_Cross-Site_Scripting. I also found the StackOverflow thread here to be helpful: https://stackoverflow.com/questions/28392997/server-xss-vs-client-xss. They explain that, over time, the broad consensus on types of XSS has shifted. In modern terms, this is an example of Server XSS because the server is allowing for untrusted code to execute. There are also Client XSS attacks, which basically means the server does not send over any untrusted code, but they are still able to execute bad code through something like a vulnerable search bar.
b. First, a user clicks one of the posts in question. The trusted javascript changes the URL to open that post. Then, as the browser is parsing and rendering HTML (which it does from the top down), it will eventually encounter a script tag and treat it as any other script

tag. This means it executes the script at the step where it parses and renders that script. This is an example of Server XSS because the actual script is stored on the server and served to the client when it is requested. In older terms, this is Stored XSS because the attack comes from a database.

c. You could use it to have the client download a file that has something malicious. For a broad example, when Wannacry was a huge issue, a XSS attack could have users downloading Wannacry onto their computer.
d. For an effective DDoS attack, some servers will simply reject requests when too many come from the same spot. You could use an XSS script to execute requests from many, many computers and have them all send requests to your target server, effectively allowing a workaround to the issue of a site blocking requests from one IP. If enough people open your post, you now have a good DDoS attack.
e. The most effective technique is to clean all inputs before they are stored in the database. This means that even if they are called improperly down the road, the data should be clean. You could also do something similar to how the source code is shown when data is retrieved from storage. I see this idea as more vulnerable because if it is implemented incorrectly down the line, the attack is still there.