Nicholas Vlahos-Sten

Jeff Ondich

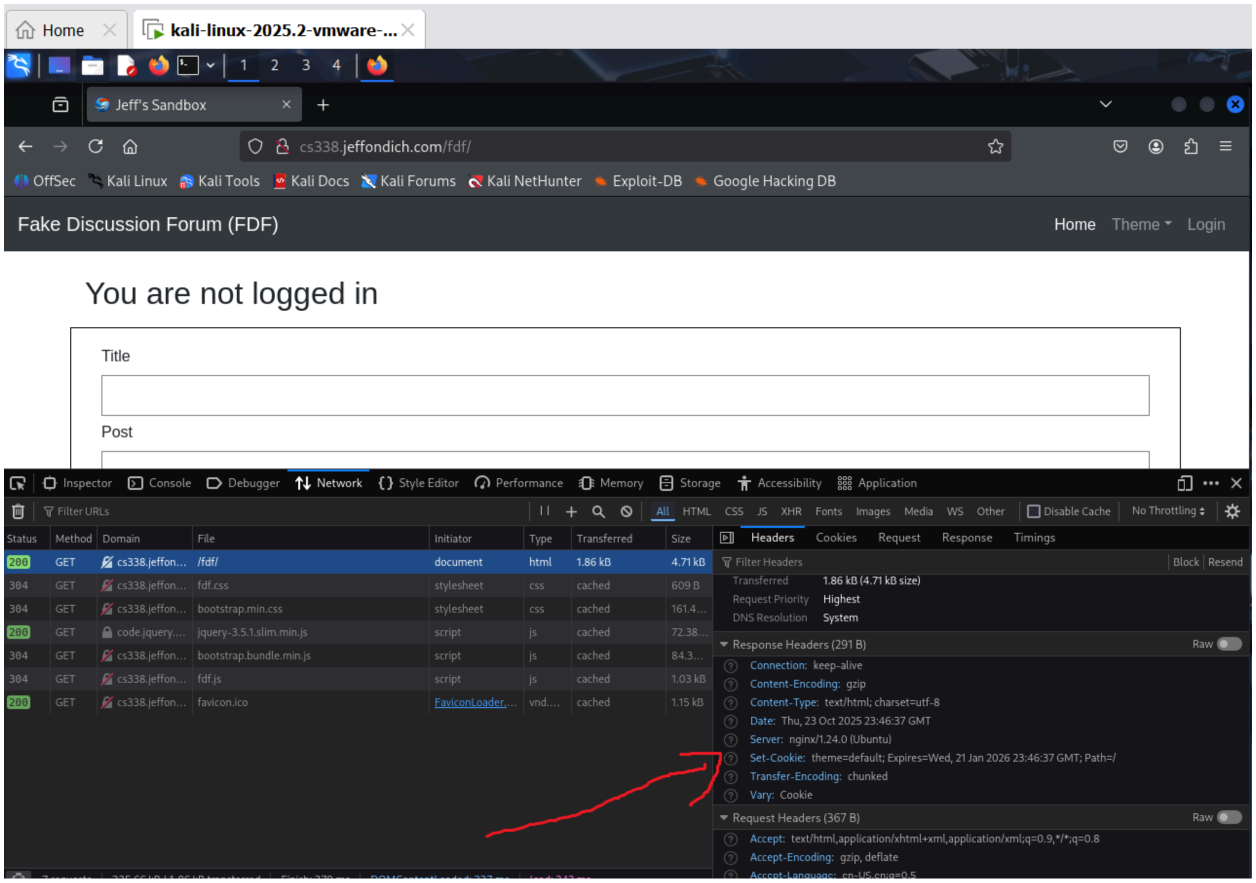
Computer Security

25 October 2025

Cookies and Cross Site Scripting

Part 1: Cookies

a. Yes. There are 3 cookies for the site which are “theme” which = “default,” “Expires” which equals “Jan 21, 2026 23:46:37 GMT” and “Path” which is “/”



You are not logged in

Title

Post

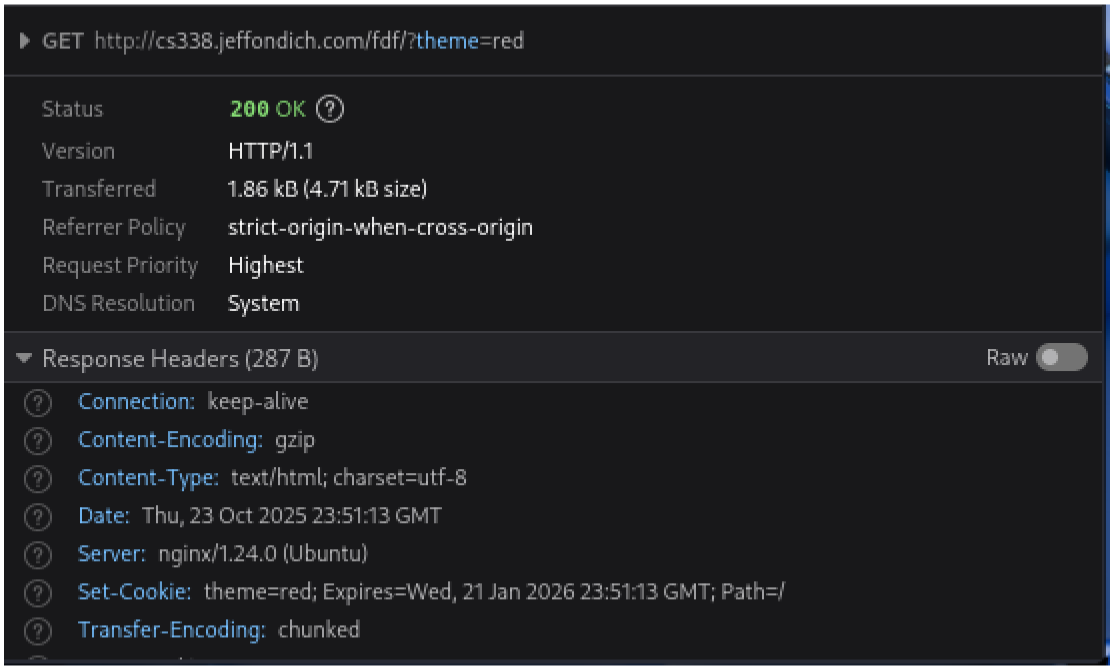
Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	cs338.jeffon...	/fdf/	document	html	1.86 kB	4.71 kB
304	GET	cs338.jeffon...	fdf.css	stylesheet	css	cached	609 B
304	GET	cs338.jeffon...	bootstrap.min.css	stylesheet	css	cached	161.4...
200	GET	code.jquery....	jquery-3.5.1.slim.min.js	script	js	cached	72.38...
304	GET	cs338.jeffon...	bootstrap.bundle.min.js	script	js	cached	84.3...
304	GET	cs338.jeffon...	fdf.js	script	js	cached	1.03 kB
200	GET	cs338.jeffon...	favicon.ico	FaviconLoader....	vnd....	cached	1.15 kB

```
Transferred          1.86 kB (4.71 kB size)
Request Priority     Highest
DNS Resolution       System

Response Headers (291 B)
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Thu, 23 Oct 2025 23:46:37 GMT
Server: nginx/1.24.0 (Ubuntu)
Set-Cookie: theme=default; Expires=Wed, 21 Jan 2026 23:46:37 GMT; Path=/
Transfer-Encoding: chunked
Vary: Cookie

Request Headers (367 B)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
```

b. Yes, the “theme” cookie changed to “red” while the “Expires” cookie went five minutes later

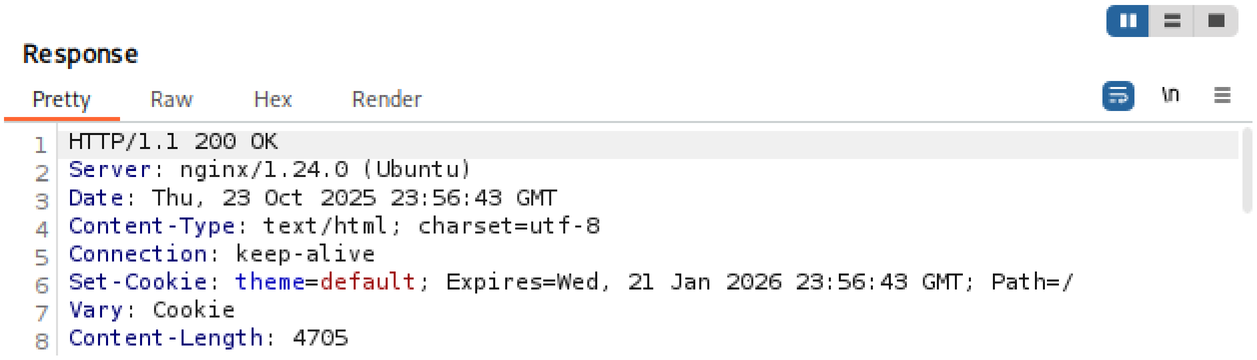


```
GET http://cs338.jeffondich.com/fdf/?theme=red

Status             200 OK
Version            HTTP/1.1
Transferred        1.86 kB (4.71 kB size)
Referrer Policy    strict-origin-when-cross-origin
Request Priority   Highest
DNS Resolution     System

Response Headers (287 B)                                Raw
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Thu, 23 Oct 2025 23:51:13 GMT
Server: nginx/1.24.0 (Ubuntu)
Set-Cookie: theme=red; Expires=Wed, 21 Jan 2026 23:51:13 GMT; Path=/
Transfer-Encoding: chunked
```

c. When you first make the GET request the browser does not send any cookies, so Jeffondich.come in its response sends a Set-Cookie header setting the three cookies to their aforementioned values, the same I said in part a.



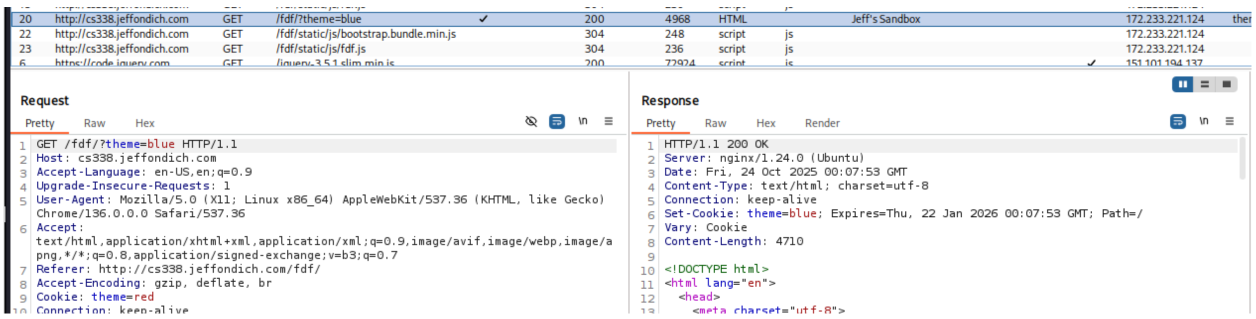
```
HTTP/1.1 200 OK
Server: nginx/1.24.0 (Ubuntu)
Date: Thu, 23 Oct 2025 23:56:43 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Set-Cookie: theme=default; Expires=Wed, 21 Jan 2026 23:56:43 GMT; Path=/
Vary: Cookie
Content-Length: 4705
```

Now, in every subsequent GET request sent by the browser, the browser sends theme=default in its response header. I have no idea why it doesn’t bother to send the other cookies (maybe because expires is just rewritten every time we go back to the website so its just an internal cookie?)

d. Yes sir it is! Good old red

e. The browser keeps the cookies stored in its system even when it closes the connection to the server. And it remembers that the next time it reconnects to the server, it needs to include a Cookie header with the saved cookies

f. The server is the one that tells the browser to change its damn cookies through another Set-Cookies header, as you can see in the image below



```
20  http://cs338.jeffondich.com  GET  /fdf/?theme=blue                   ✓  200  4968  HTML    Jeff's Sandbox  172.233.221.124  ther
22  http://cs338.jeffondich.com  GET  /fdf/static/js/bootstrap.bundle.min.js  304  248   script  js              172.233.221.124
23  http://cs338.jeffondich.com  GET  /fdf/static/js/fdf.js                 304  236   script  js              172.233.221.124
```

Request

```
GET /fdf/?theme=blue 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/136.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a
png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://cs338.jeffondich.com/fdf/
Accept-Encoding: gzip, deflate, br
Cookie: theme=red
Connection: keep-alive
```

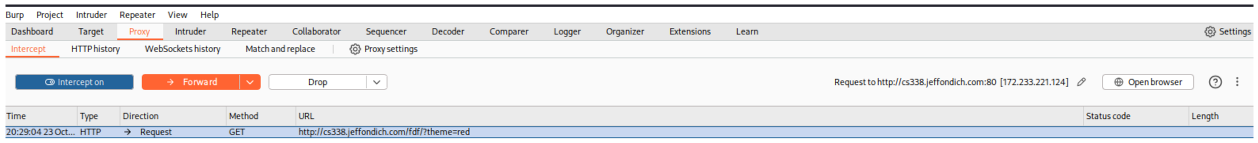
Response

```
HTTP/1.1 200 OK
Server: nginx/1.24.0 (Ubuntu)
Date: Fri, 24 Oct 2025 00:07:53 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Set-Cookie: theme=blue; Expires=Thu, 22 Jan 2026 00:07:53 GMT; Path=/
Vary: Cookie
Content-Length: 4710

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
```

g. Well I did try, I went on Kali and I went to Firefox's browser inspector and I clicked on the storage tab to see all the cookies, and under the theme cookie I manually changed the value from "blue" to "red" but when I reloaded the page, it just refused to ever load the page again. Like even when I exited out and tried to reopen the page, it would just never open again. So I broke it. I tried.

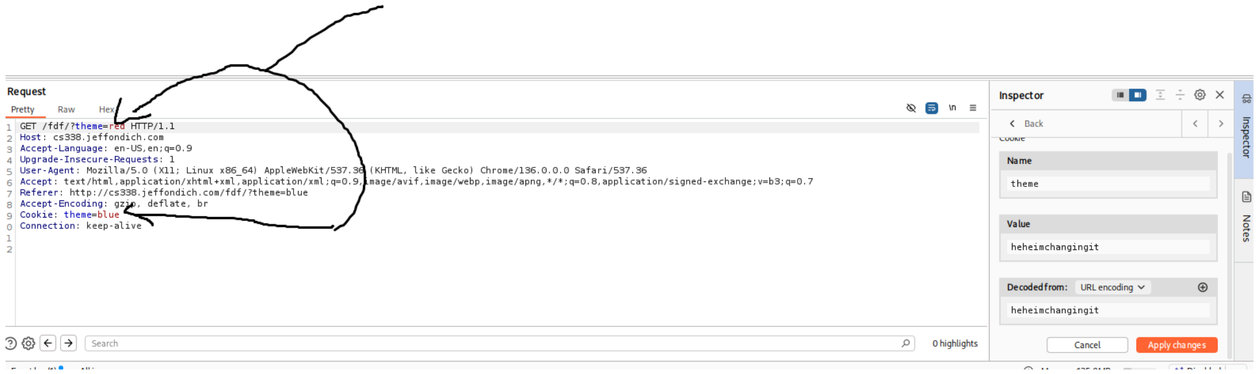
h. Intercept the GET request from the browser and then manually change the Cookie header and the GET url to make the theme whatever you want it to be before sending the modified response to the server


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Open browser

Request to http://cs338.jeffondich.com:80 [172.233.221.124]

Time	Type	Direction	Method	URL	Status code	Length
20:29:04 23 Oct...	HTTP	→ Request	GET	http://cs338.jeffondich.com/fdf/?theme=red		

You need to change these two parameters to whatever you want it to be

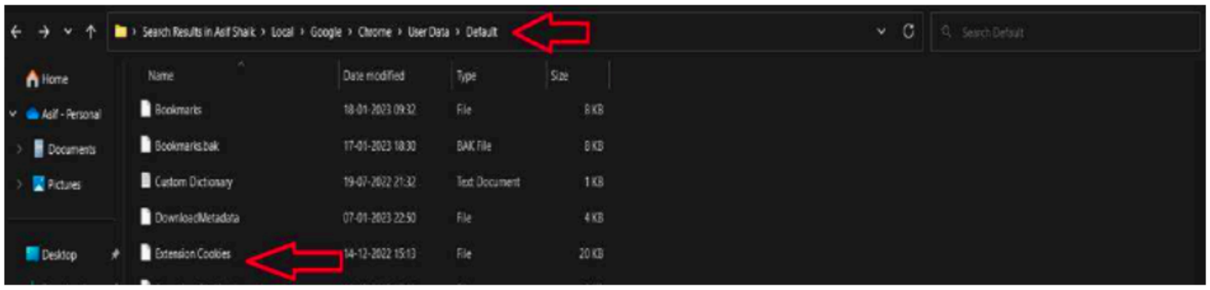


```
GET /fdf/?theme=red HTTP/1.1
Host: cs338.jeffondich.com
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://cs338.jeffondich.com/fdf/?theme=blue
Accept-Encoding: gzip, deflate, br
Cookie: theme=blue
Connection: keep-alive
```

Name: theme
Value: heheimchangingit
Decoded from: URL encoding
heheimchangingit

i. According to GeeksForGeeks, every individual browser (Chrome, Edge, etc) stores all the cookies for every website in a single file underneath the browser's corresponding folder in your computer's OS

File Path(Google):
C:\Users\Your_User_Name\AppData\Local\Google\Chrome\User Data\Default.


Search Results in Asif Shaik › Local › Google › Chrome › User Data › Default
Search Default
Home
Asif - Personal
Documents
Pictures
Desktop

Name	Date modified	Type	Size
Bookmarks	18-01-2023 09:32	File	8 KB
Bookmarks.bak	17-01-2023 18:30	BAK File	8 KB
Custom Dictionary	19-07-2022 21:32	Text Document	1 KB
DownloadMetadata	07-01-2023 22:50	File	4 KB
Extension Cookies	14-12-2022 15:13	File	20 KB

https://www.geeksforgeeks.org/javascript/in-which-location-cookies-are-stored-on-the-hard-disk/#

XSS Attacks

a. I used this link: https://www.acunetix.com/websitesecurity/xss/ Seems there are three types of XSS attacks:
    i. Stored XSS. The attack injects most likely javascript somewhere in the website where their malicious code becomes permanently part of ("stored" in) the website's code. For example, if you have edit access to a forum post or database entry that's embedded within the overall html code, you can edit code into that that can edit really anything else on the website
    ii. Reflected XSS. The malicious code is part of something like a GET request, that tricks the web server into sending a response that contains the malicious code ("reflecting" it). Then if the user interacts with the response, the malicious code executes in the user's browser. It's non-persistant–you have to resend the malicious code in every new payload
    iii. DOM-based XSS. Some servers write user data to a Document Object Model that the web application reads from and sends back to the user. If you can write malicious code to the Document Object Model that the web application sends to the user, you can execute that code on the user's browser without the server necessarily even realizing something's going on.

b. Moriarty's is definitely a Stored XSS attack. When Moriarty wrote his forum post, he permanently added his malicious code (in this case `<script>alert('Mwah-ha-ha-ha!');</script>`) to the server's html code. This is because the server permanently stores every forum post so that anyone can later access it (the whole point of a forum website). Then, perhaps a long time after he originally sent the stored code, someone else clicks on his forum post, and when the server outputs the text of his message it also outputs/runs his malicious code.

c. That dastardly man Moriarty knows that the code behind the website takes the user's unique cookies to properly function, so Moriarty can build a cookie scrapper that secretly and silently records the cookie values of whoever clicks on his post, so that he can later sell that data to companies at a profit. Mwahaha!

d. Inspired by someone else's post that I really liked, Moriarty can redirect the user to his own evil website, linking them there as soon as they click on the post, a website where he has even more control over the code to do more evil things. Perhaps I wonder if he can also just directly download a virus onto your computer, download a file when you click on the post.

e. I mean the main way is you need to input some sort of input checking system. After any user writes anything, be it to a post or even a request, the server should before just blindly accepting it, read it to check if the user's payload contains any code they want to run on the server or a browser. This article https://www.acunetix.com/blog/articles/preventing-xss-attacks/ describes one implementation, a "filter" that combs through a message and deletes things it deems potentially dangerous like <span> or any javascript code.