Varun Saini

Cookies and Cross-Site Scripting (XSS)

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?

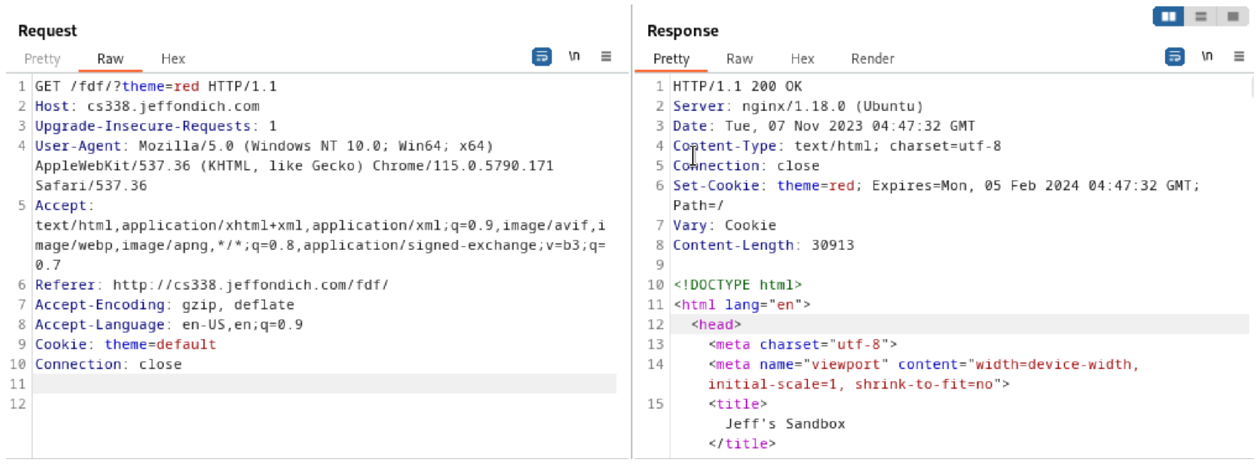
There is a single cookie with the name *theme* and the value *default*.

(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?

Yes, the *theme* cookie's value changed to *red* and *blue* when I changed the theme to red and blue, respectively.

(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?

On the right of this image is the HTTP response after setting the theme to red. As we can see, the *Set-Cookie* header contains *theme=red.*

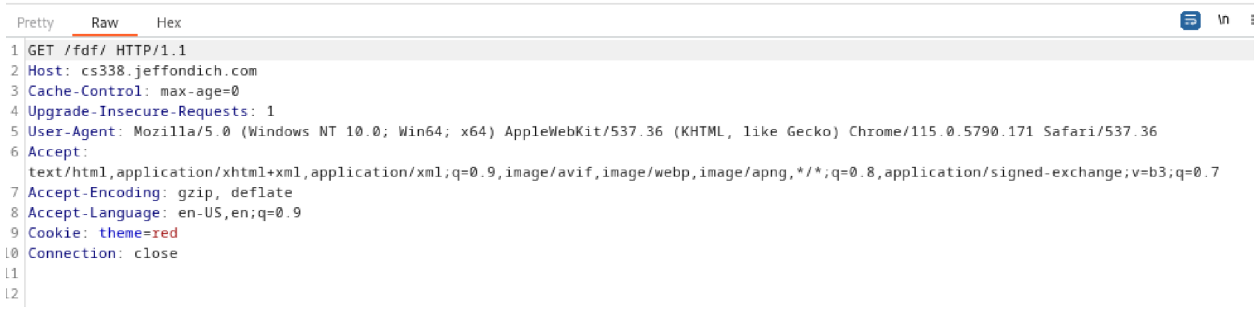


```
GET /fdf/?theme=red HTTP/1.1
Host: cs338.jeffondich.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.171
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,i
mage/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=
0.7
Referer: http://cs338.jeffondich.com/fdf/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: theme=default
Connection: close
```

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 07 Nov 2023 04:47:32 GMT
Content-Type: text/html; charset=utf-8
Connection: close
Set-Cookie: theme=red; Expires=Mon, 05 Feb 2024 04:47:32 GMT;
Path=/
Vary: Cookie
Content-Length: 30913

<!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>
```

After setting this theme and receiving the new *theme* cookie value, we can see that the *Cookie* header contains *theme=red* for future HTTP requests.



```
GET /fdf/ HTTP/1.1
Host: cs338.jeffondich.com
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.171 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
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: theme=red
Connection: close
```

(Yes, I see the same cookie values as I saw with the Inspector.)

(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?

Yes, the same theme is active.

(e) How is the current theme transmitted between the browser and the FDF server?

The current theme is transmitted between the browser and FDF by two methods:

- URL parameter: https://cs338.jeffondich.com/fdf/?theme=red
- Cookies: sending *theme=red* in the *Cookie* and *Set-Cookie* headers.

(f) When you change the theme, how is the change transmitted between the browser and the FDF server?

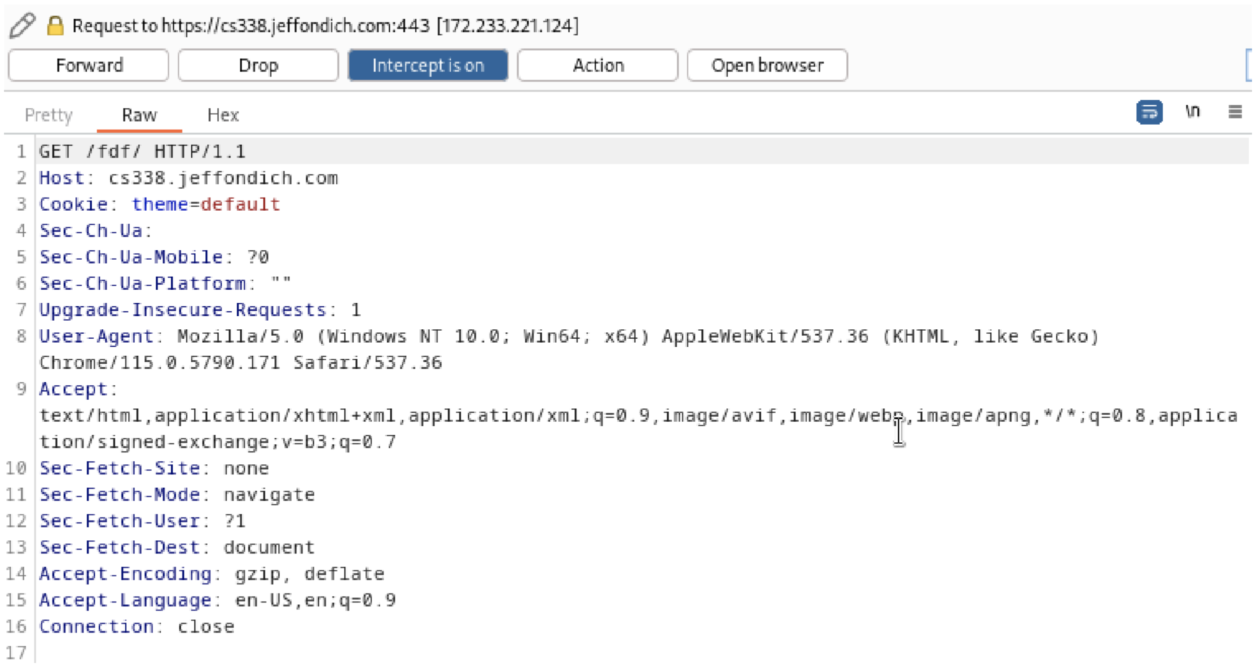
When the theme is changed, the client sends a new HTTP GET request to the server with a URL parameter containing the new theme. The server's HTTP response contains a cookie (*Set-Cookie*) that the client stores. For future communications, the client sends this cookie (*Cookie*) to the server in its HTTP request headers.

(g) How could you use your browser's Inspector to change the FDF theme without using the FDF's Theme menu?

By going to Cookie-Editor in Chrome's Inspector, I can set the *theme* cookie's value to *default*/*red*/*blue*. Refreshing the page reflects this change.

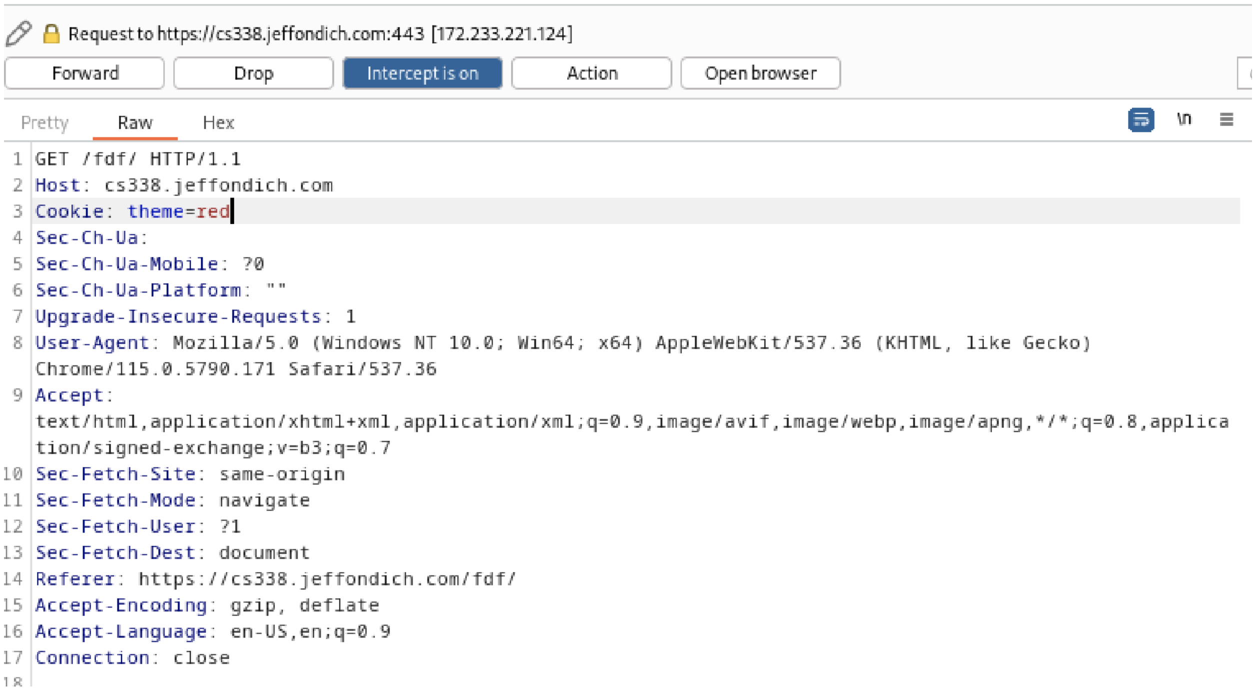
(h) How could you use Burpsuite's Proxy tool to change the FDF theme without using the FDF's Theme menu?

When visiting the main website, I could intercept the HTTP request being sent to the server, which looks like this.

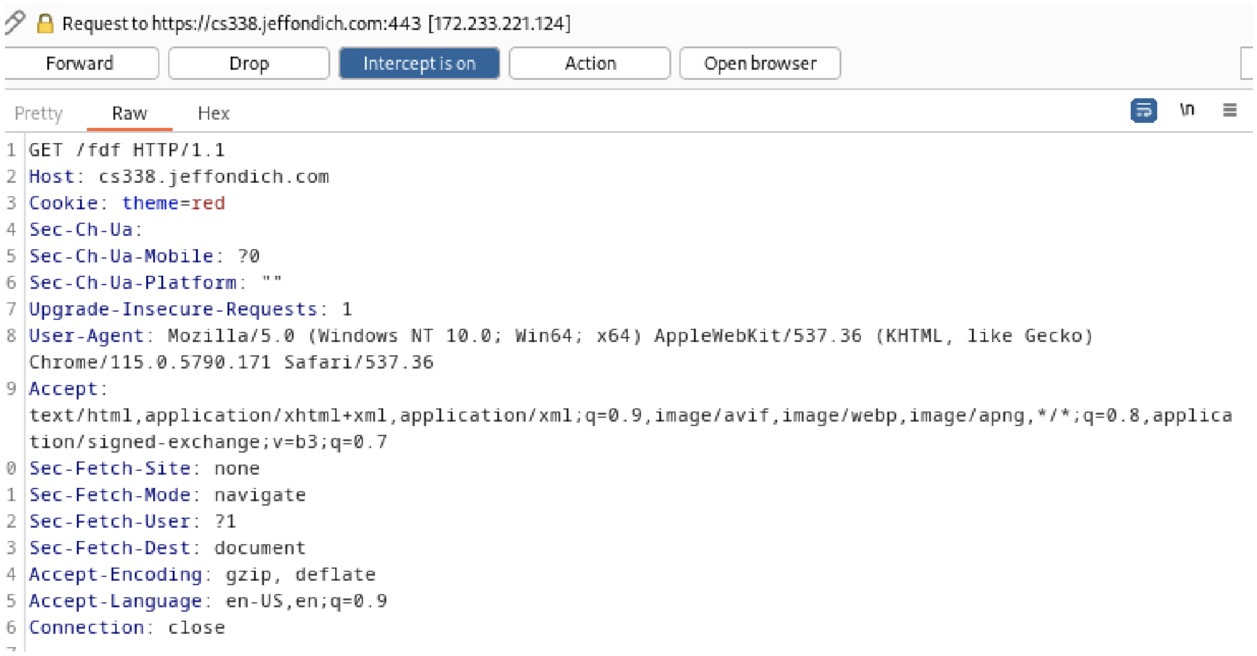

Request to https://cs338.jeffondich.com:443 [172.233.221.124]

```
GET /fdf/ HTTP/1.1
Host: cs338.jeffondich.com
Cookie: theme=default
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/115.0.5790.171 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
tion/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

I have two options: I could either change “GET /fdf HTTP/1.1” to “GET /fdf/?theme=red HTTP/1.1”, or I could change “Cookie: theme=default” to “Cookie: theme=red”. These are pictured below.

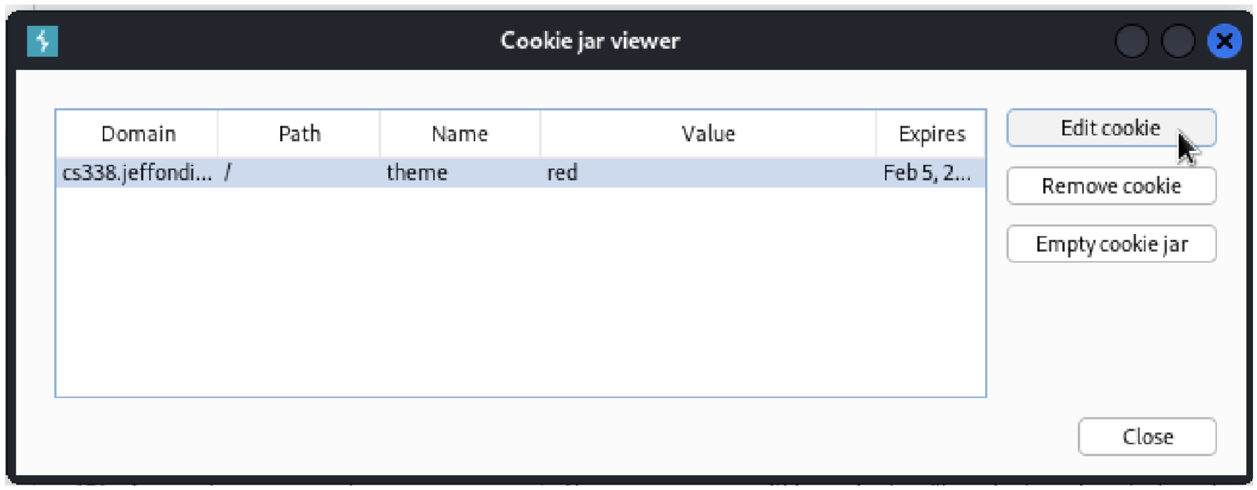

Request to https://cs338.jeffondich.com:443 [172.233.221.124]

```
GET /fdf/ HTTP/1.1
Host: cs338.jeffondich.com
Cookie: theme=red
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/115.0.5790.171 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
tion/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://cs338.jeffondich.com/fdf/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```


Request to https://cs338.jeffondich.com:443 [172.233.221.124]
Forward
Drop
Intercept is on
Action
Open browser
Pretty
Raw
Hex

```
GET /fdf HTTP/1.1
Host: cs338.jeffondich.com
Cookie: theme=red
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/115.0.5790.171 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
tion/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

I could also directly edit BurpSuite's cookie jar, which stores cookies. This can be found at Proxy Settings > Sessions > Cookie jar > Open cookie jar > Edit cookie.



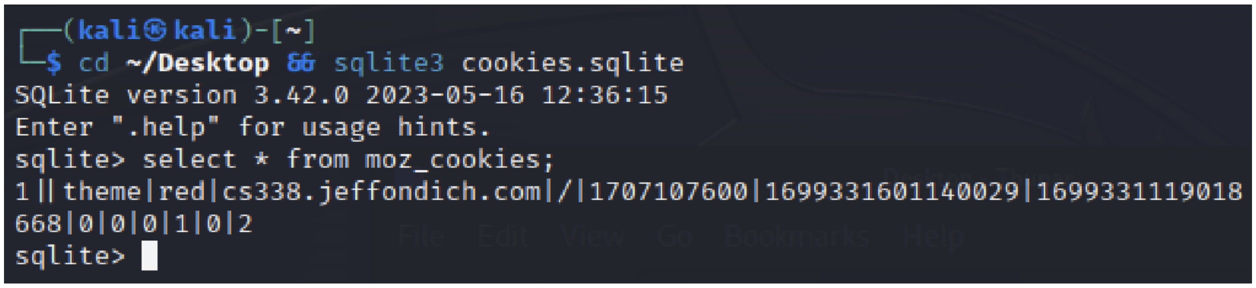
Domain	Path	Name	Value	Expires
cs338.jeffondi...	/	theme	red	Feb 5, 2...

(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.)

On Kali, Mozilla Firefox cookies are stored in the following file:

- /home/kali/.mozilla/firefox/5qr9vehe.default-esr/cookies.sqlite

I copied this file onto my Desktop and used the following commands to inspect its contents, revealing the *theme* cookie.



```
┌──(kali㉿kali)-[~]
└─$ cd ~/Desktop && sqlite3 cookies.sqlite
SQLite version 3.42.0 2023-05-16 12:36:15
Enter ".help" for usage hints.
sqlite> select * from moz_cookies;
1||theme|red|cs338.jeffondich.com|/|1707107600|1699331601140029|1699331119018
668|0|0|0|1|0|2
sqlite> 
```

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.

- Moriarty's attack occurs asynchronously. In other words, he doesn't need to be actively monitoring anything when users fall prey to his attacks.
- The attack begins with Moriarty making a post with a title or post body that contains dynamic (<script></script>) code.
- Then, any time a user views the compromised post's title (on the main page) or the post body (by clicking the post), the JavaScript code (contained in <script></script> tags) will execute in the user's browser.
- When the attack happens for a user, Moriarty himself is fairly uninvolved. Each user experiences their own version of the attack.

(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.

We could write a malicious script that does the following:

- Gets the user's cookies (document.cookie)
- Extracts Alice's *session* cookie by parsing document.cookie
- Sends an HTTP request to Moriarty's database server to store the cookie

As long as Alice doesn't log out (invalidating her session cookie), Moriarty could make posts while pretending to be Alice.

(c) Do it again: describe a second attack that is more virulent than Moriarty's, but that's substantially different from your first idea.

Here's a DDOS attack:

- Create a post with a title that contains a <script> tag
- In this script, set the values of the post creation form (title and post) to anything, and submit this form
- Repeat the previous step in an infinite loop

Thus, each user that opens the FDF webpage will immediately become a node in this distributed system. Each node adds posts in an infinite loop, which will likely crash the server or slow it to a crawl.

(d) What techniques can the server or the browser use to prevent what Moriarty is doing?

The best way to avoid these types of attacks is to filter out "suspicious-looking" titles and post bodies. From searching the web, it seems like there are blacklist regex strings that can be used for this purpose. A heavier-duty solution would be to pre-render the post bodies in an isolated environment and then display the rendered text. Similarly, we could perform simple string replacements like "<" to "&lt;" and ">" to "&gt;" to avoid inputted text from being interpreted dynamically.