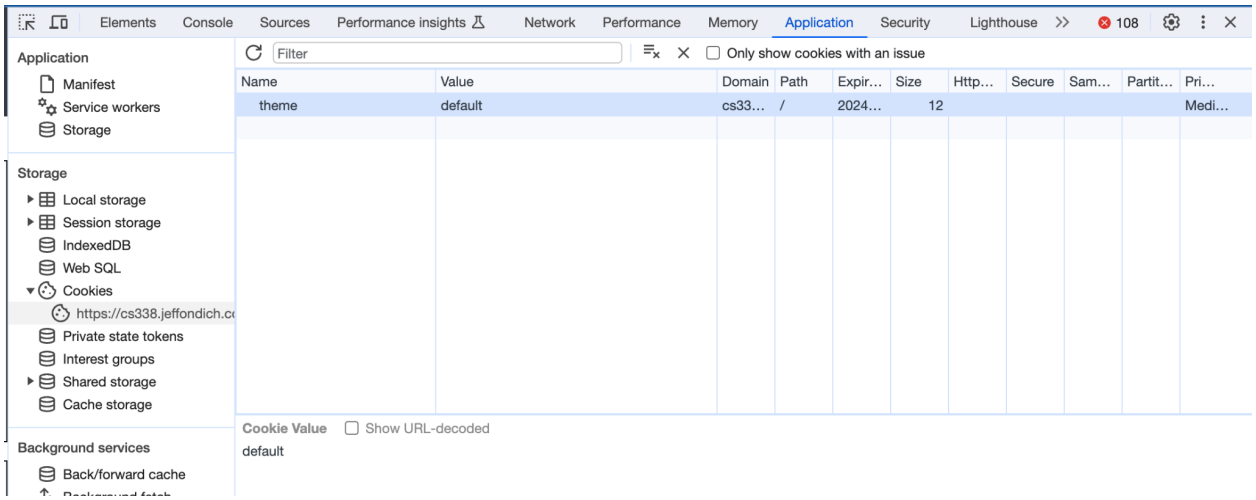
Arthur Viegas Eguia

Cookies:

a)



Yes, there are cookies for that domain. The only cookie for cs338.jeffondich.com is named theme. It has value default and expires on 2024-02-03, which is in roughly three months. So the cookie name is theme and its value is default.

b) They did, now the value is blue. The name is still theme, and it still expires on 2024-02-03 (however, at a slightly later time). The only change was in the value (it is now blue). Therefore, we have theme=blue. Updating the theme by clicking one of the buttons updates one of the cookies of the site.

c) I installed Burp Suit for the first time in my Kali today. So it is safe to say that it had no cookies stored in its browser when I first accessed the cs338.jeffondich.com/fdf/ website.

My first request was a rather common HTTP request.

```
1  GET /fdf/ HTTP/1.1
2  Host: cs338.jeffondich.com
3  Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"
4  Sec-Ch-Ua-Mobile: ?0
5  Sec-Ch-Ua-Platform: "Linux"
6  Upgrade-Insecure-Requests: 1
7  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.90
   Safari/537.36
8  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
9  Sec-Fetch-Site: none
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Accept-Encoding: gzip, deflate, br
14 Accept-Language: en-US,en;q=0.9
15 Connection: close
16
```

No cookies whatsoever were mentioned in it. However, the request included the hostname, that is cs338.jeffondich.com (who the browser thinks it is talking to), and it included the user agent,

that we read about (it is the identification of the browser). Apparently, this one is Mozilla/5.0 AppleWebkit/537.36/
It also indicates the type of files it is accepting (HTML, XHTML, images, among others), the languages it is expecting (english-US), and indicates that the client wants to close the connection, which according to developer.mozilla is the default on HTTP/1.0 requests.
So the website saw no mentions of cookies in this request. Therefore, one of the first things it did was sending the cookies.

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Sun, 05 Nov 2023 22:35:43 GMT
Content-Type: text/html; charset=utf-8
Connection: close
Set-Cookie: theme=default; Expires=Sat, 03 Feb 2024 22:35:42 GMT;
Path=/
Vary: Cookie
Content-Length: 27176
```

The request started with code 200 OK, which according to developer.mozilla tells the user that the request succeeded and the resource requested has been fetched and will be included in this message.
The server also identifies itself, saying that it is nginx/1.18.0 (Ubuntu), it tells the user the date it thinks it is, and says that it also wants to close the connection.
After that, it finally sets the cookie with Set-Cookie, the first argument to that is nameOfCookie=value, in this case, theme=default, it also includes an expiration date, which is 03 Feb 2024 (which is in roughly 3 months), says the path that is path=/
Based on this request, I believe that to set a cookie you do attribute=value; Expires=Expiration date;Path = /;
expiration date contains multiple words, as it is a date.
The next header of the request is Vary, which according to mozilla.developer is used to create a cache key when content negotiation is in use. In this case, the cache will be used to save cookies on the user's computer.
So there is a Set-Cookie header in this request, that creates a cookie with name theme, value default and expiration date in 3 months, which is identical to the one I saw in inspect.
The user made yet another request, this time a bit simpler. However, it contained very similar information. Just like the first request sent by the user, it did not contain any mentions of cookies. It contains the get command GET /fdf/ HTTP/1.1
It also contains host-name which is the host the user thinks they are talking to, User-Agent which is the browser version (we have read about it at the beginning of the term), Accept which is the expected file type, and Accept-Language which is the language that we are expecting (in this case english-US). However, this request was much simpler, it did not include any of the headers that start with sec. I assume those headers are somehow related to security, and as we are using HTTP and not HTTPS, the connection is not secure (all information is broadcasted in

plain text), so there is no need for the sec headers. Then, the browser created a request without them.

```
GET /fdf/ 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/118.0.5993.90
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
f,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v
=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: close
```

The response from the browser was identical to the one described above. No differences whatsoever. The exact same Set-Cookie was sent.
The browser then proceeded to send three requests to the server. All of which request JS files. All of the requests sent to the server include the Cookie header now. For example, the first request was:

```
GET /fdf/static/js/fdf.js HTTP/1.1
Host: cs338.jeffondich.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.90
Safari/537.36
Accept: */*
Referer: http://cs338.jeffondich.com/fdf/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: theme=default
Connection: close
```

Observe that the request has header Cookie, and its value is theme=default. The format followed is cookieName=cookieValue. This was sent to the server in the HTTP request.
The rest of the request is very usual. It included the GET /fdf/static/js/fdf.js HTTP/1.1 that indicates what JS file the user is requesting, host (cs338.jeffondich.com) that is who the user

thinks they are talking to, user-agent that represents the user's browser version, accept (in this case we accept everything). Referer (http://cs338.jeffondich.com) is who sent the browser there, or who website prompted the browser to make that request.
The only mention of cookies in this request is under the Cookie header, which sends the pair name=value. However, the response of the server did not mention cookies at all.
The second request for a script is identical to the first one. The only difference is that the file being requested is bootstrap.bundle.min.js. The request includes the same Cookie header with the same value as described above (theme=default).

```
GET /fdf/static/js/bootstrap.bundle.min.js HTTP/1.1
Host: cs338.jeffondich.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.90
Safari/537.36
Accept: */*
Referer: http://cs338.jeffondich.com/fdf/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: theme=default
Connection: close
```

The last request was a bit different, it requested code from code.jquery.com. I see this because this was the server under Host. The GET command requests /jquery-3.5.1.slim.min.js HTTP/1.1. The headers with the sec prefix are back, I assume that the jQuery website uses HTTPS and the browser requests it in HTTPS by default. Just like the previous request it accepts anything (as the value of the accept header is */*), the Sec-Fetch-Site value is cross-site which means that the script is being requested from another server. The Connection header value is close, just like the previous ones. Observe that the header Cookie is not in this request. I assume this is the case as the browser is requesting data from an external server. As it is not requesting from cs338.jeffondich.com, the cookies are not valid on the other server.

```
GET /jquery-3.5.1.slim.min.js HTTP/1.1
Host: code.jquery.com
Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"
Origin: http://cs338.jeffondich.com
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.90
Safari/537.36
Sec-Ch-Ua-Platform: "Linux"
Accept: */*
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: script
Referer: http://cs338.jeffondich.com/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: close
```

So far, the cookie set is identical to the one I have inspected in part a). Now I am going to change the color of the theme to Blue.
In order to do this, the browser sent the server a request. The GET request was GET /fdf/?theme=blue HTTP/1.1
Observe that right beside the address of the file we are trying to get there is a parameter, that is separate from the address by the ? sign. The parameter name is theme and the value is blue as we have ?theme=blue. The Host still has value cs338.jeffondich.com, the accept-language is en-US, and Cookie is theme=default. Observe that the value of Cookie when the server made this request was the same. This means that the server did not change the value of the cookies= just yet. It was not prompted by the server to do so.

Arthur Viegas Eguia

```
GET /fdf/?theme=blue 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/118.0.5993.90
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
f,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v
=b3;q=0.7
Referer: http://cs338.jeffondich.com/fdf/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: theme=default
Connection: close
```

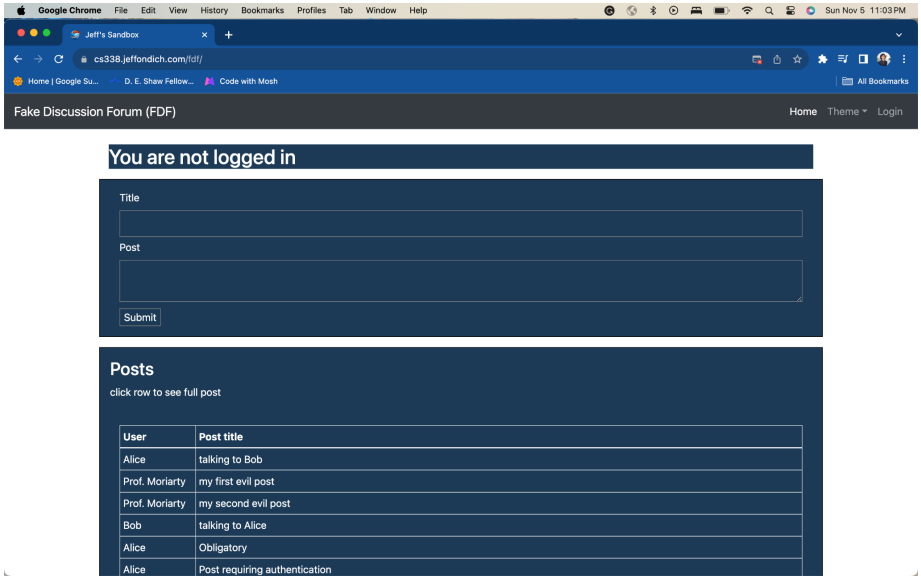
The response was HTTP / 1.1 200 OK, which means that the request was successful and is being attached to the response. The Server value is nginx/1.18.0 (Ubuntu). The content type is text/html and the connection value is close. The server once again included the header Set-Cookie but this time the value of the header is theme=blue; Expires=Sun 04 Feb 2024; Path=/. Observe that it is setting a cookie with the same name but with a different value. Instead of being theme-default, it is theme=blue.
The vary header value is Cookie.
This means that the only thing that was updated was the value of the cooke (and its expiration date). The value of the cookie now is no longer default, it is blue. So yes, the cookie has changed.
I do see the same values as I did in the inspector. The difference is that now instead of seeing them in a table I see them as a set cookie followed by some parameters. But yes, the cookie

has the same values, but the interface is different.

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 06 Nov 2023 02:54:23
 GMT
Content-Type: text/html;
charset=utf-8
Connection: close
Set-Cookie: theme=blue;
Expires=Sun, 04 Feb 2024
02:54:23 GMT; Path=/
Vary: Cookie
Content-Length: 28349
```

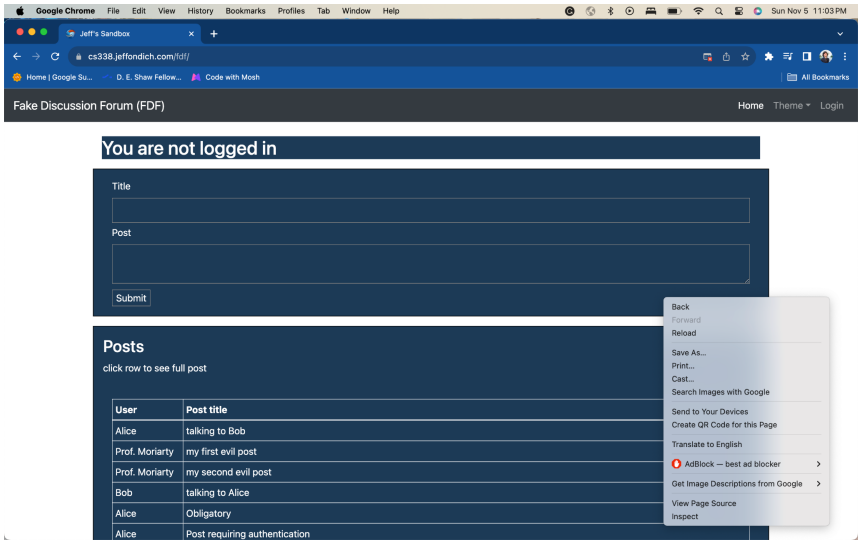
d) Yes, it is still blue
e) When the user accesses the website for the first time, there are no mentions of cookies in the HTTP request. As the browser has never accessed the website, it cannot have cookies for it.
In the HTTP response from the website, there is header Set-Cookie. In my interaction with the website, the Set-Cookie value is theme=default; Expires=04 Feb 2024 02:54:23 GMT; Path=/
This means that the name of the cookie is theme, the value of the cookie is default, and that the cookie is valid for 90 days. path=/ means that the cookie is valid on the entire website. For different, interactions the cookie will have the name theme, the value default, it will be valid for about three months and have path /.
After receiving this cookie, every time the user makes a request to the server the browser will add the value of the cookie in the request under the Cookie header. The value of the Cookie header, in this example, will be theme=default. When the server receives the request, it will read the cookie, and know what the theme should be.
Now, suppose that the user wants to change the theme of the website. To do so, they will click theme, and select blue. This will prompt the browser to make another GET request with theme=blue as a parameter. I know this as the request is GET /fdf/?theme=blue. And theme=blue is after the ? in the request. theme is the name of the parameter and the value of the request is blue. The Cookie header sent in this request has value theme=default, as the cookie was not yet updated by the server.
Upon receiving the request, the server will see that the user did a GET request with parameter theme=blue, so it will use header Set-Cookie in its response. But this time the value is going to be theme=blue; Expires=04 Feb 2024 02:54:23 GMT; Path=/, this way it will update the value of

the cookie saved in the user's computer. The value of Set-Cookie is almost identical to the previous one, the only difference is that the value of the cookie is blue instead of default (the Expires value is also a bit different, as the server creating a new cookie that is valid for around three months).
What the browser is effectively doing is creating a new cookie in the browser with the same name and a different value. This seems to update the cookie, by overwriting its previous value.
Suppose the user closes the browser and a few days later tries to access the Fake Discussion Forum again. Before sending the request to the website, the browser will see if it has cookies saved from cs338.jeffondich.com. As it does, the browser will attach them to the HTTP request. So, the HTTP request will have the header Cookie, which has value theme=blue. This way, when the server receives the request, it will read the cookie and know that the theme of the website should be blue. An interesting thing about cookies is that the webserver does not need to store any values in its database, the cookie values are stored in the user's computer.
f) The user will click theme, and will be presented with a menu. Suppose they choose blue. This will prompt the browser to send the server an HTTP request with parameter theme=blue. Parameters have format name=value and, in an HTTP request, they are separated from the page address by the ? sign. So the request will be formatted like GET /fdf/?theme=blue.
Upon receiving a request with this parameter, the server knows it is time to update the user's cookie. To do so, it will simply create a new cookie. So, the HTTP response sent by the server has header Set-Cookie with value theme=blue; Expires=04 Feb 2024 02:54:23 GMT; Path=/ (the Expires will be different, as it is 90 days after the time the response was sent). In other words, the browser creates a new cookie with a different value but with the same name. So the browser will overwrite the old cookie. As a collateral effect of creating a new cookie, the expiration date of the cookie will also be updated, it will be valid for an extra three months.
g) Here is a quick tutorial.
Go to the website:

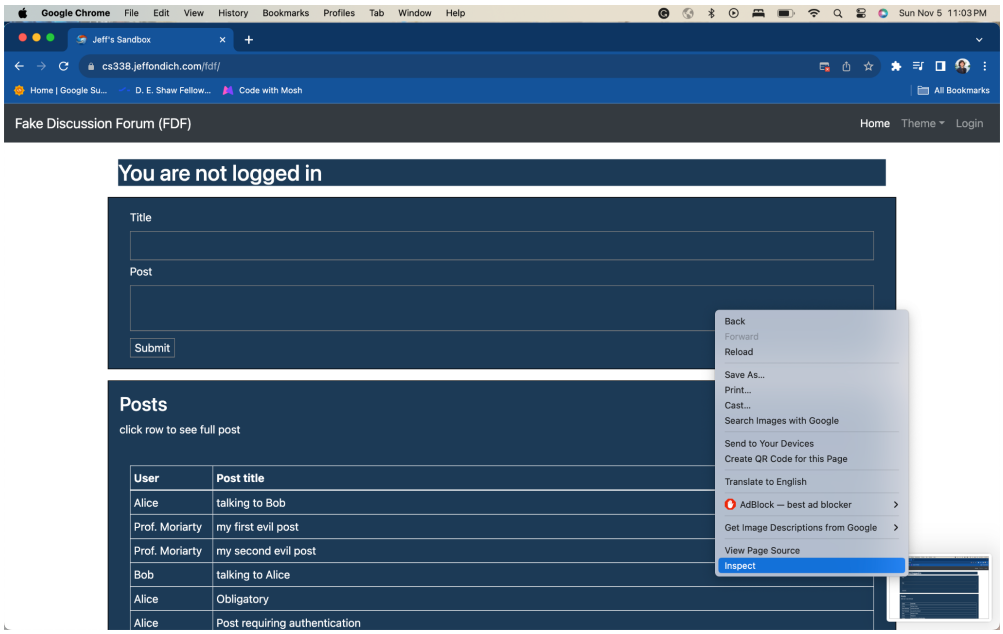


Do a right-click:

Arthur Viegas Eguia

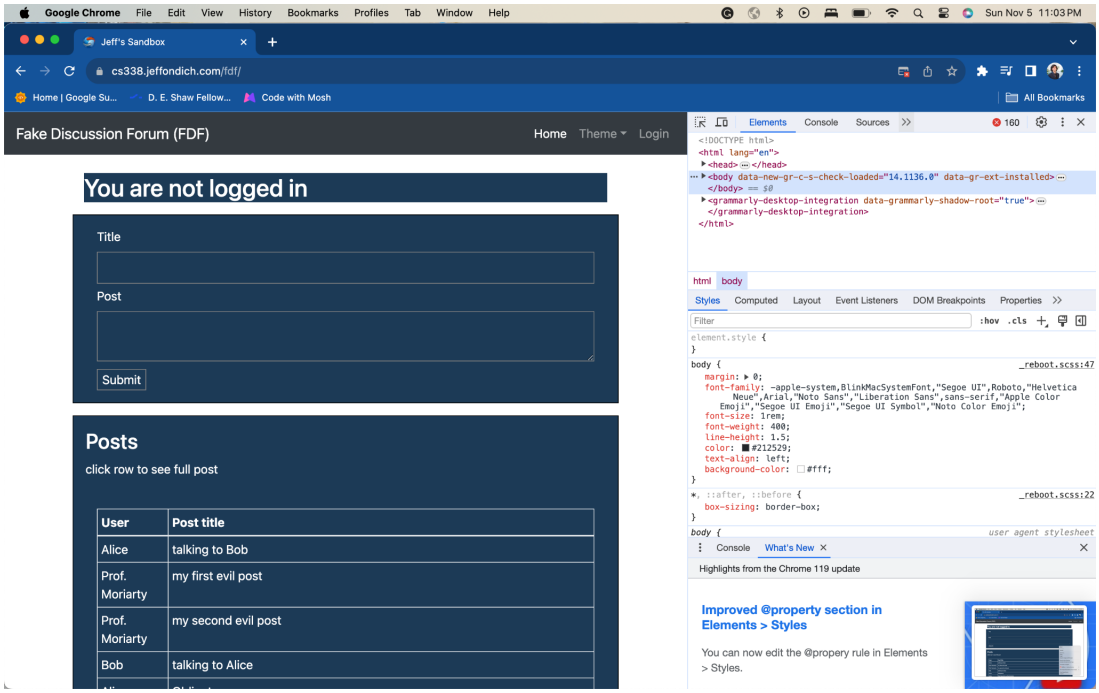


Click on inspect:

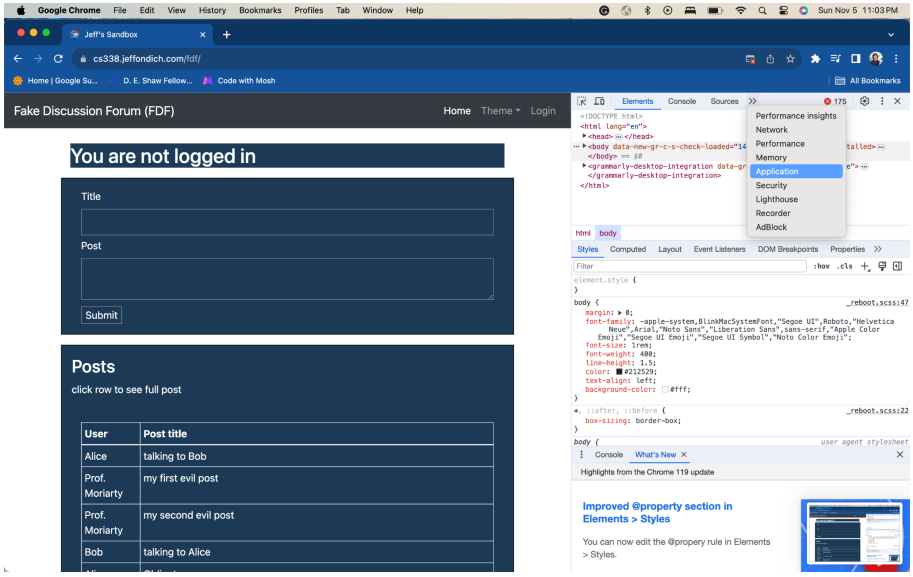


Arthur Viegas Eguia

You will be led to the inspect page.

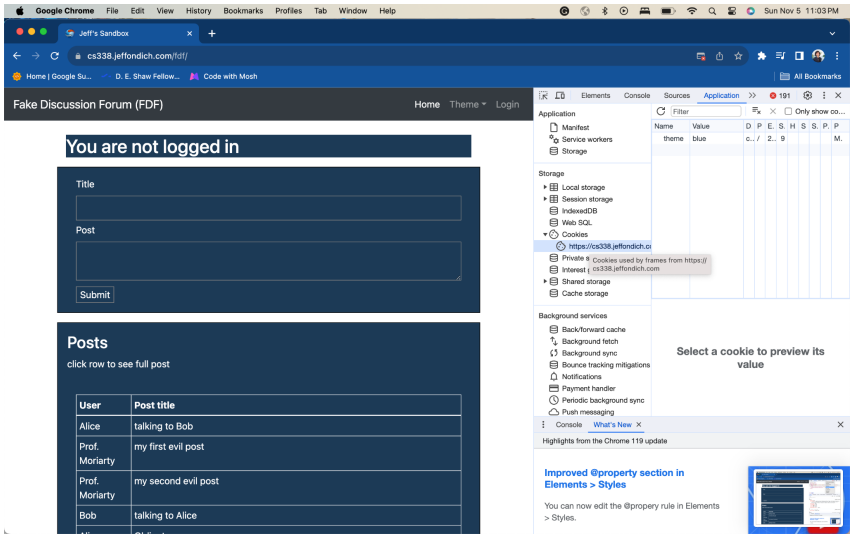


Click on the two arrows:

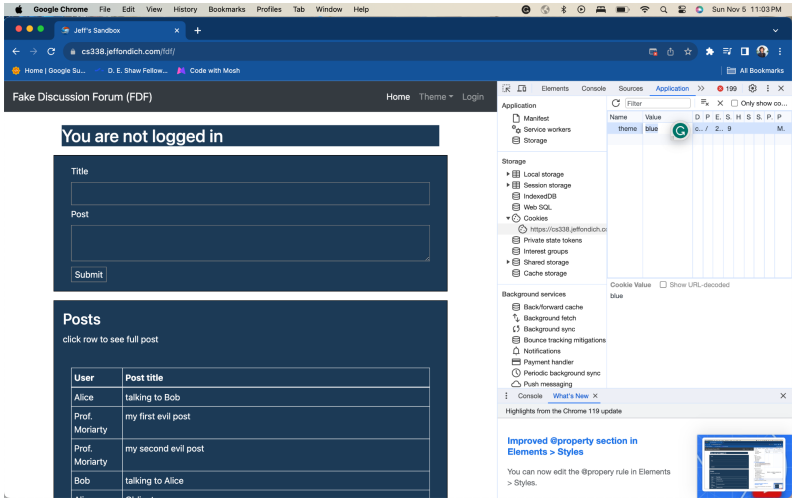


Then click on application. This way, you will be led to the applications tab:

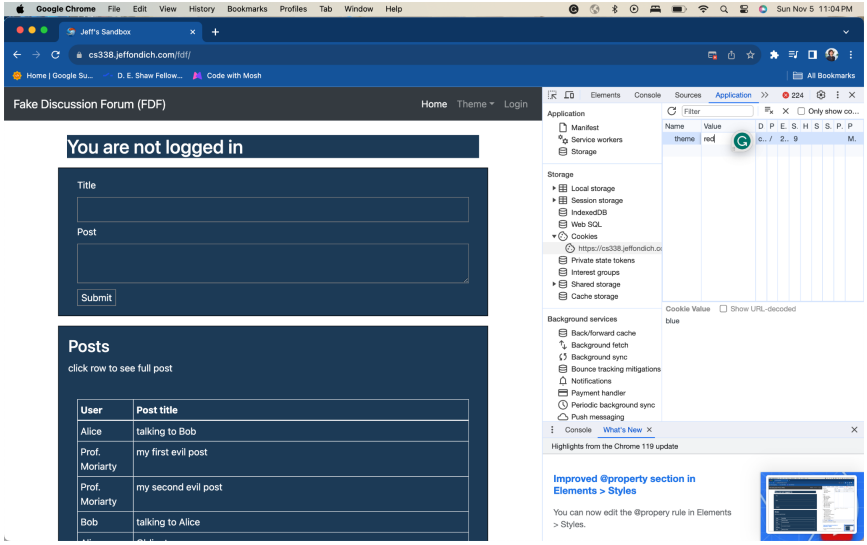
Arthur Viegas Eguia



Under cookies, click on https://cs338.jeffondich.com/, and then click on the value of the cookie you want to update:

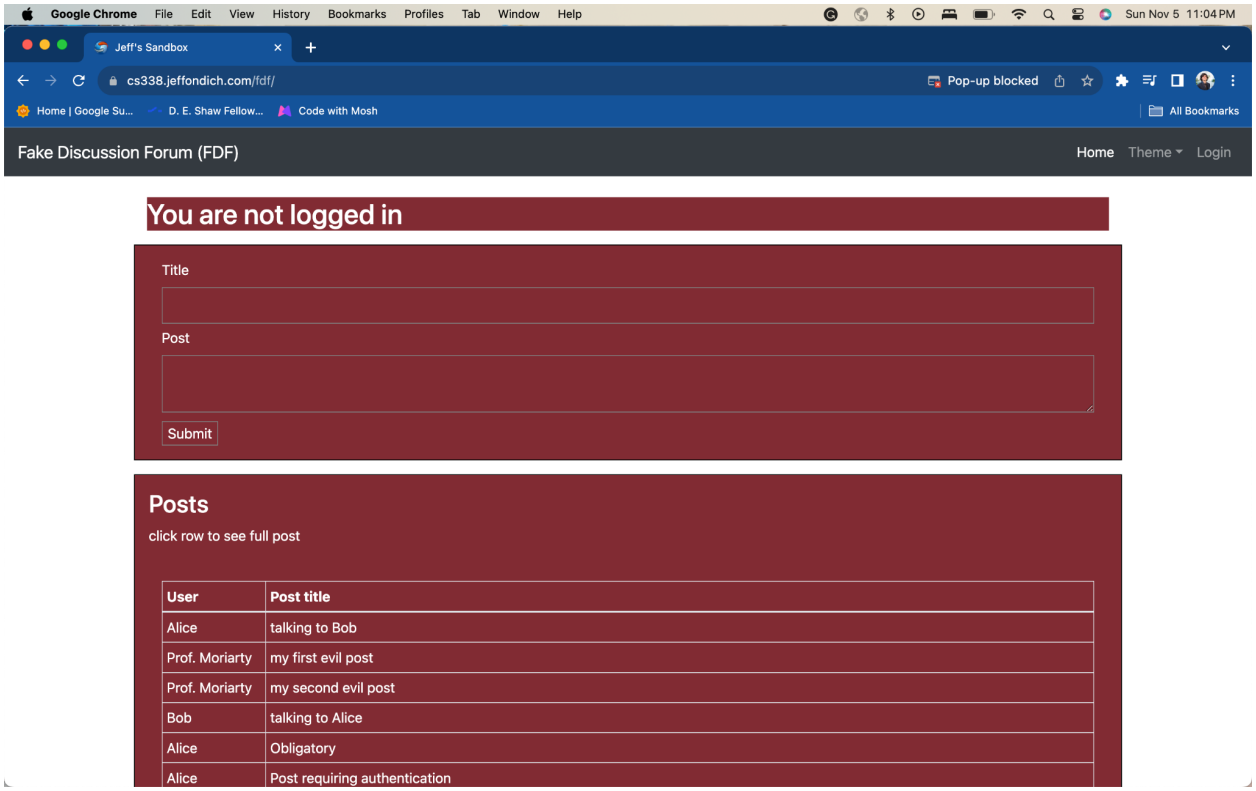


Change its value:

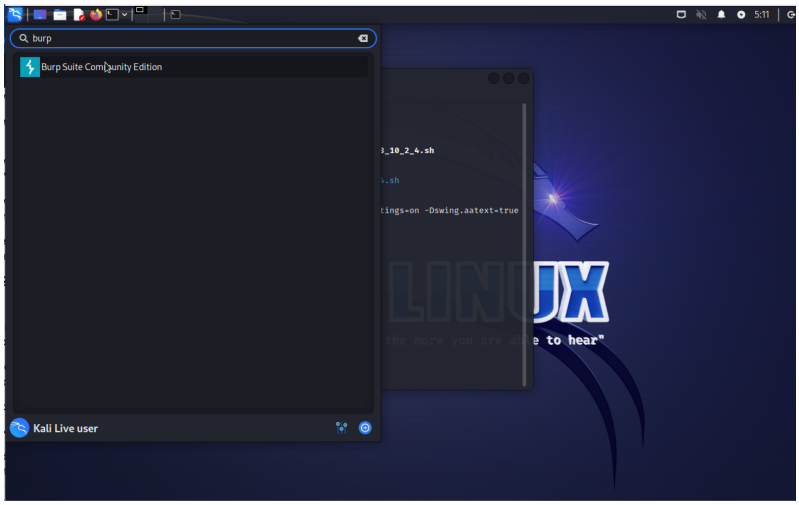


Arthur Viegas Eguia

Reload the page, and you will see the results:

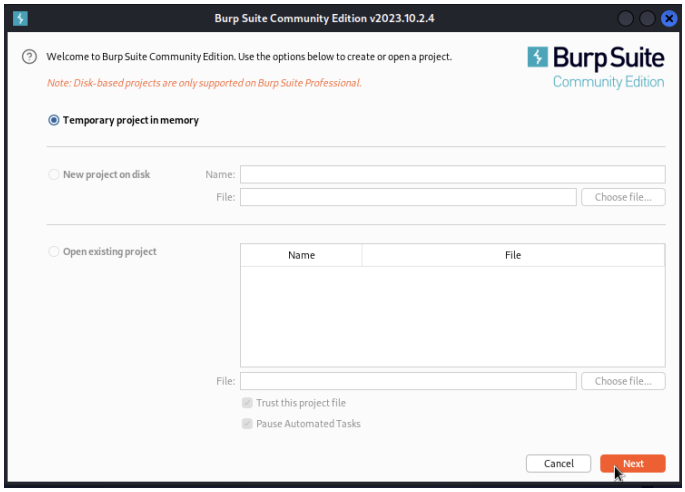


h) Open burpsuite:

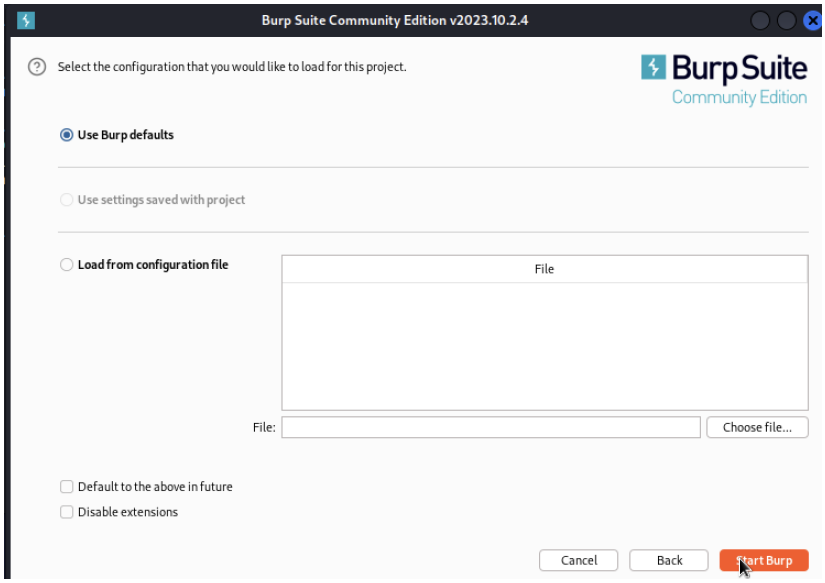


Click next:

Arthur Viegas Eguia

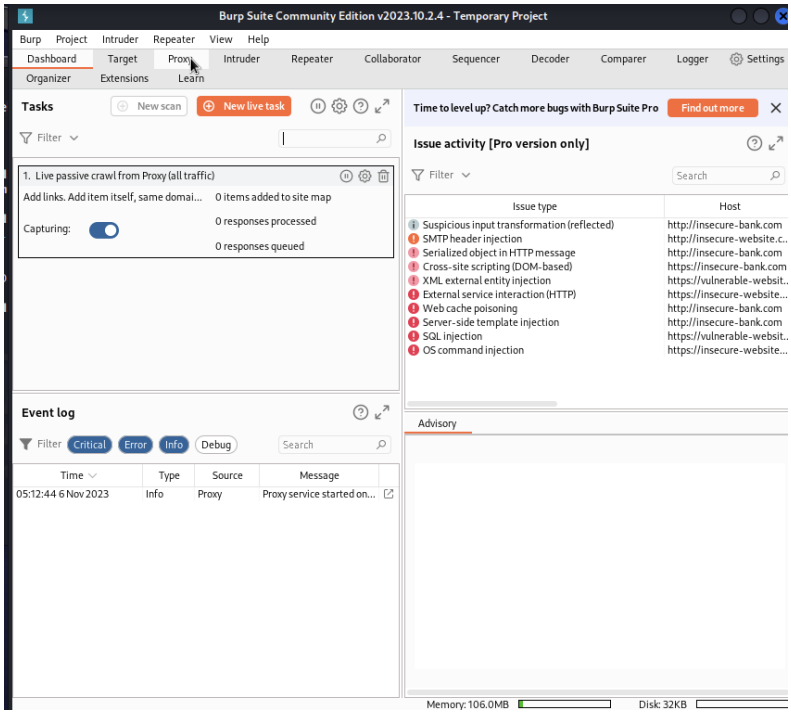


Click start Burp:

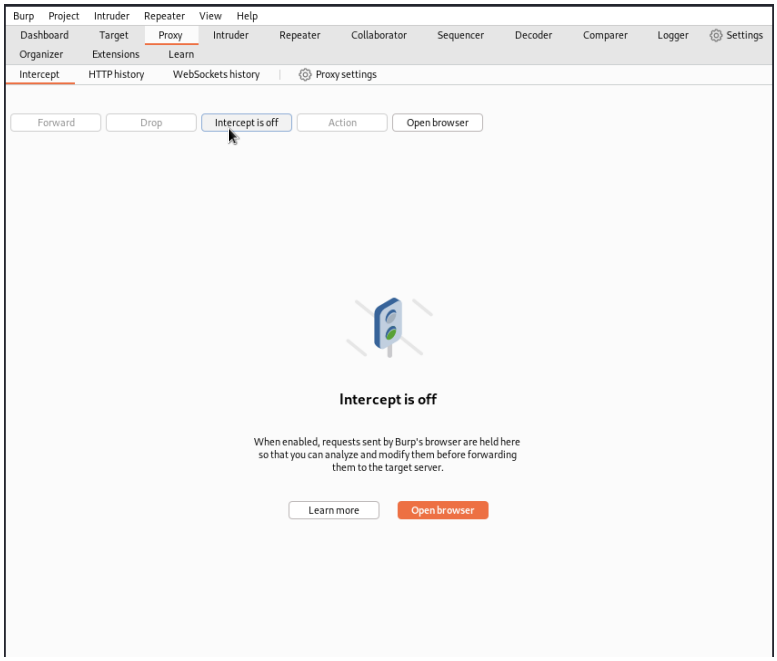


Click on Proxy:

Arthur Viegas Eguia

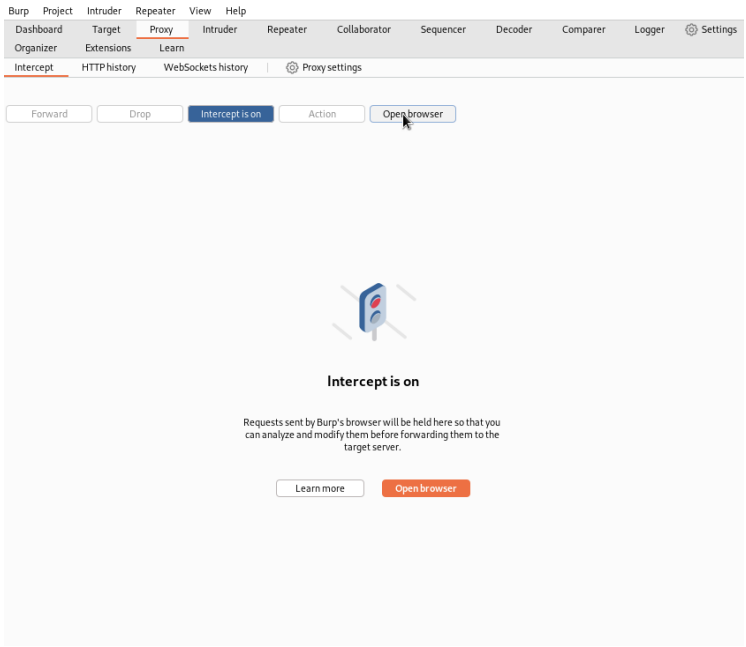


Click on intercept is off:

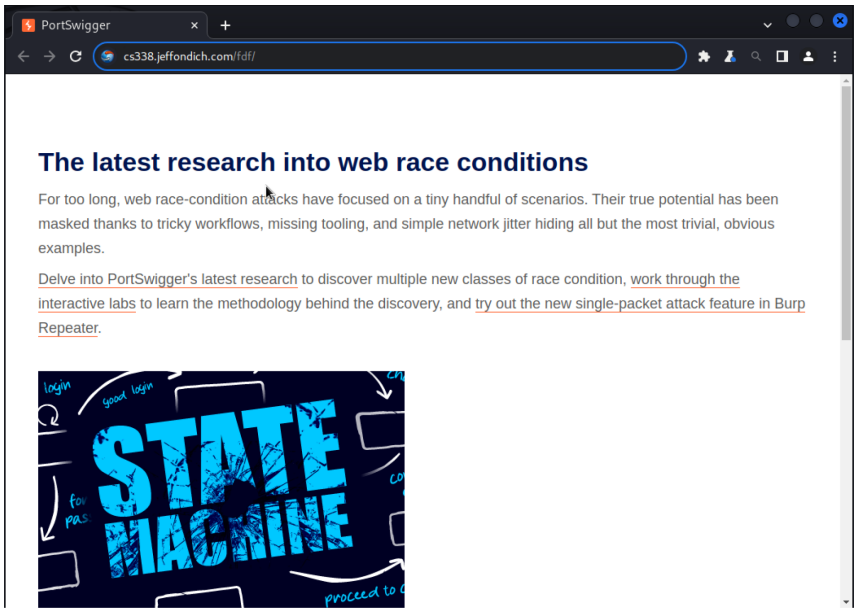


This will turn intercept on. So, whenever we try to access a webpage on Burp suite's browser, we can see and modify the request before the browser it. To open the Burp Suite browser, click on Open Browser:

Arthur Viegas Eguia

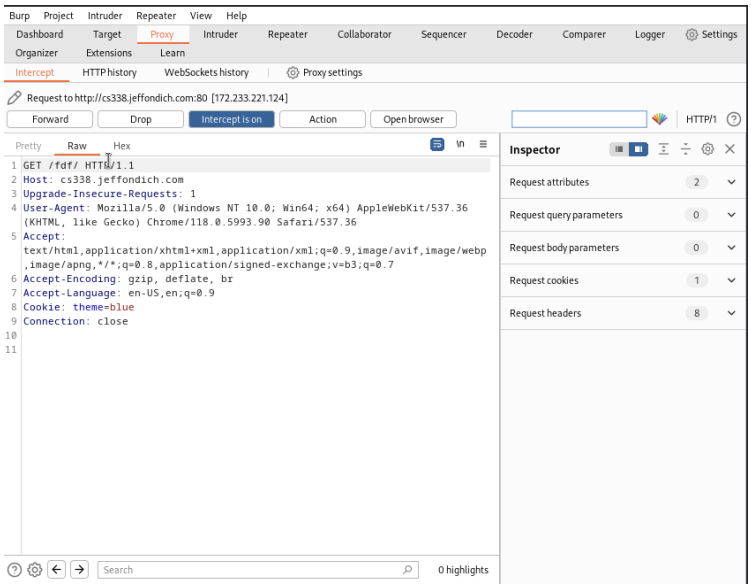


Type the domain of the website:

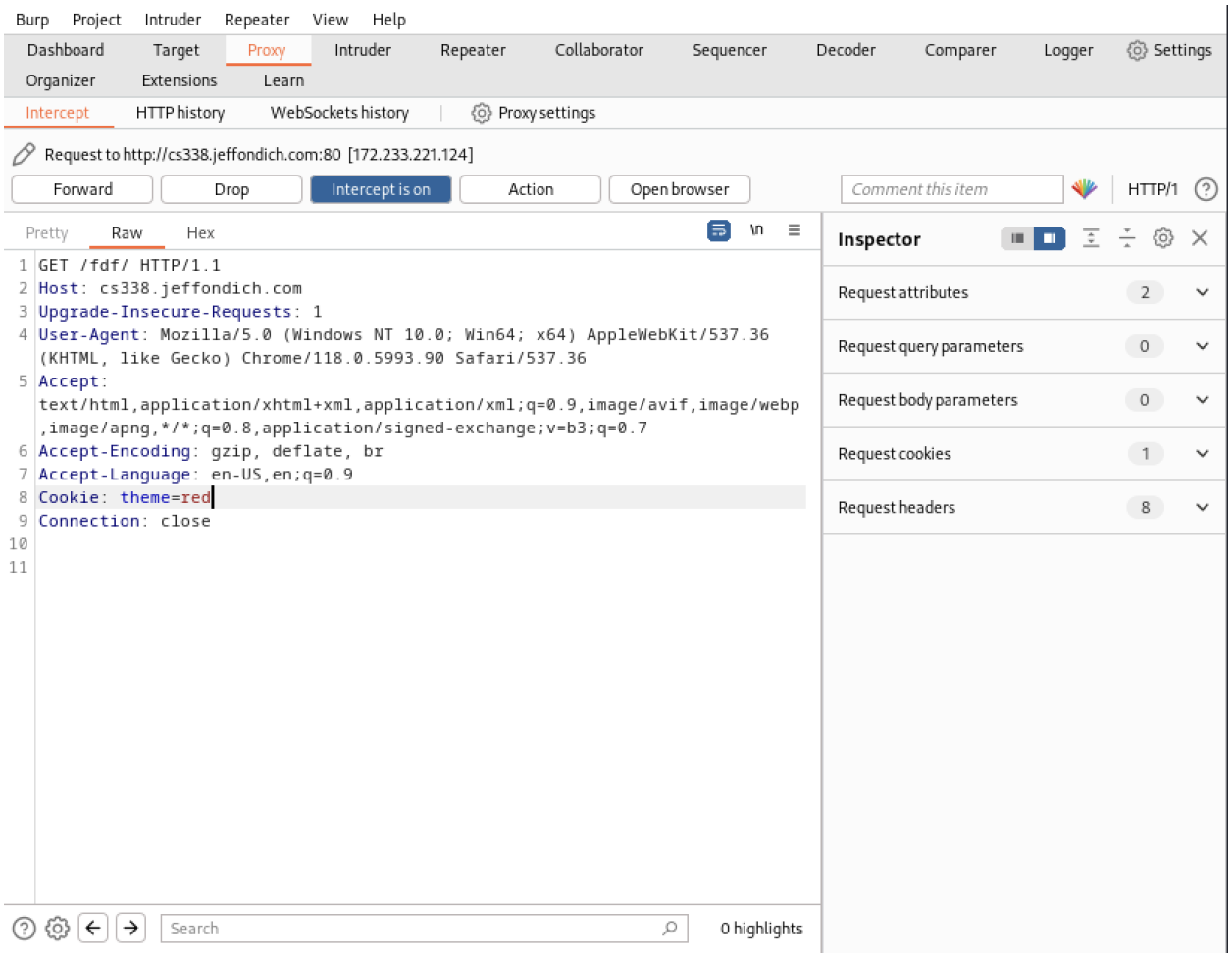


This will open Burp suite for you, and show you the request that is being sent to the website:

Arthur Viegas Eguia

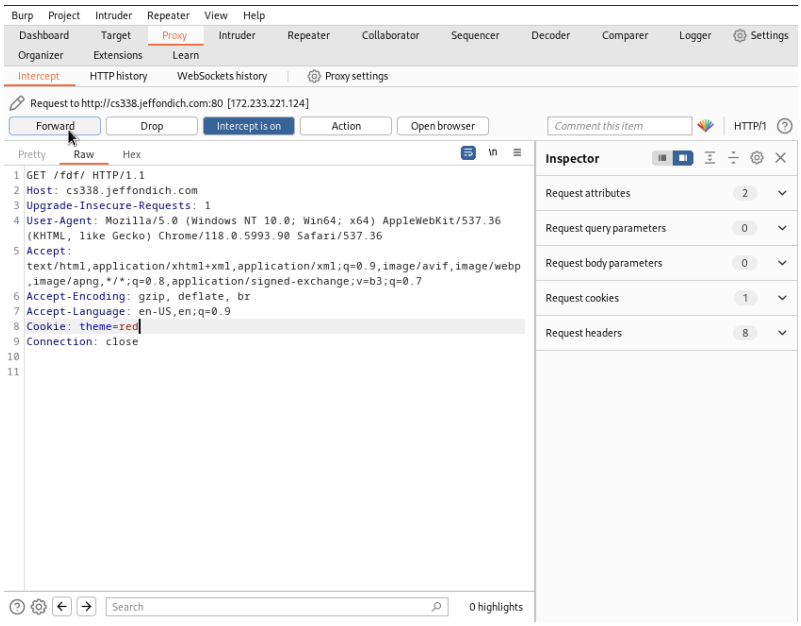


Now, go to the header Cookie, that has value theme=blue. And change it from blue to red:

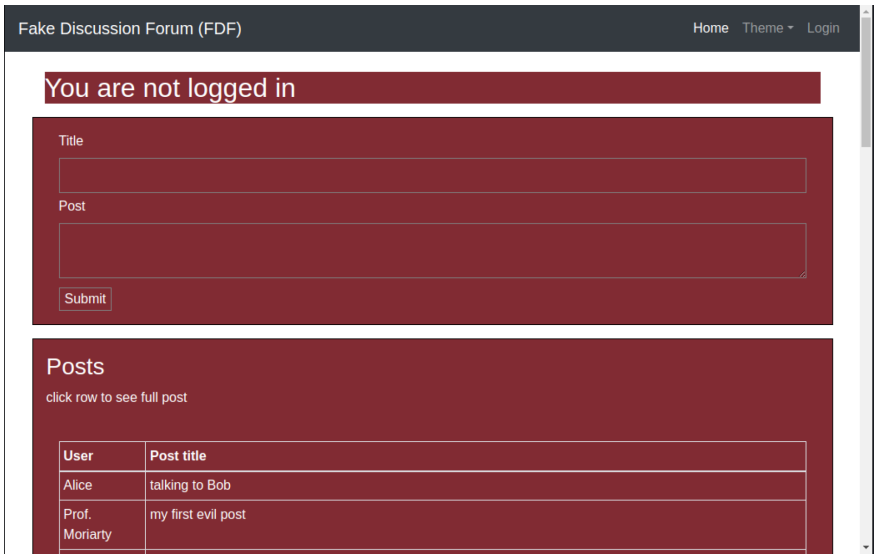


Click forward:

Arthur Viegas Eguia



And now you will see that the website is red and not blue:



h) The cookies of Firefox, on Kali are on:
home/kali/firefox/qdfvt7mh.default-est/cookies-sqlite
Found using the AskUbuntu Forum
For my Mac they are on:
Library/Application\ Support/Google/Chrome/Profile\ 1/Cookies
Used Stack Exchange to answer this question

Cross-Site Scripting

a) Moriarty writes his post. Somewhere in the text of the title/content of his post, he will add HTML tags. For example, he might add tags to turn some of his text red. Or he might use the script tag, and inside their scope, embed (potentially malicious) JavaScript. He clicks submit and waits.

His code will be processed/parsed by the server, and stored in the database. I know this is the case because the raw code on the forum is

But when I inspect the page source it is

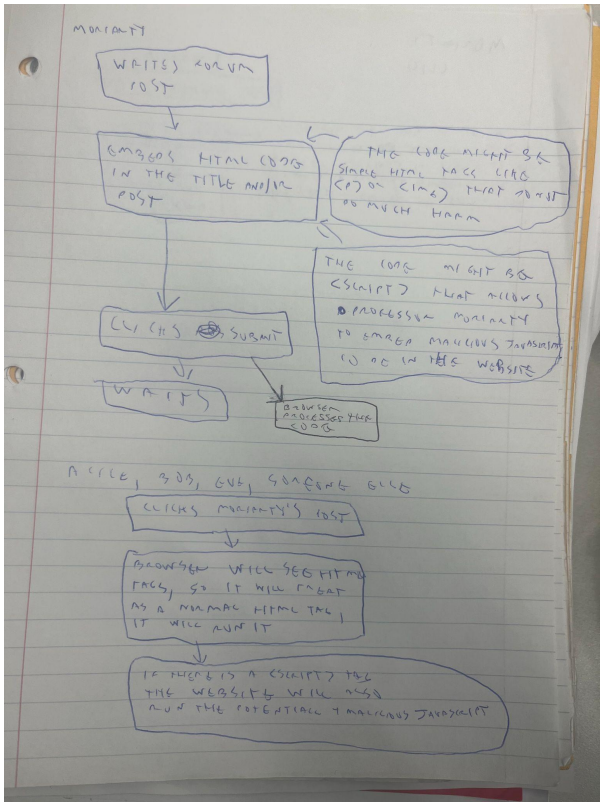
```
<p><strong>Post</strong>: Look at me with my fancy Javascript. <script>document.getElementsByClassName("w-100")[0].innerHTML = "<strong>Post by Prof. Moriarty</strong>";
document.getElementById('posts').children[0].innerHTML="<strong>Title</strong>: my second evil post"; window.addEventListener("load", (event) => {
  document.getElementsByClassName("col-12 mt-4")[1].children[0].innerHTML = "Look at me with my fancy Javascript. &lt;script&gt;alert('Mwah-ha-ha-ha!');&lt;/script&gt;";
  setTimeout(() => { alert('Mwah-ha-ha-ha!\n(Brendan was here)'); }, 50);
}); </script></p>
```

This means that the site parsed it. Instead of leaving the script tag in the middle of the post, it added code to wait for the page to fully load before executing the Javascript code.

Eventually, Alice, Bob, or Eve might click a forum post made by Professor Moriarty. Their browser will see the HTML code in the title/content of the post, think it is legitimate HTML code, and run it. All the browser sees are tags, and the tags inserted by Professor Moriarty look the same as the legitimate tags used by the website creators. Therefore, the browser will run them and execute the code inside them.

So, for the first evil post, the browser will see that there is a tag that makes the text written by Moriarty red. The browser will execute that code and that part of the text will be red. If Alice, Bob, or Eve click on the second evil post, the browser will see a script tag, think that the server is telling it to run JavaScript (as there is a script tag) and it will run the JavaScript code. It is important to remember that the script code will have been parsed by the server, so it will wait for the page to load before running. Luckily, in this case, it is just an alert, but if the JavaScript code is malicious, then problems might arise.

Arthur Viegas Eguia



b) Moriarty creates a forum post that writes more forum posts. If the forum post writes posts in a loop, this might add a lot of spam to the home page of the Fake Discussion Forum. In the worst case, it might cause Denial of Service.
Jeff mentioned this attack in class. The idea is simple, it is easy to get the name of the form fields with Burp Suite. One might even get the entire post request from Burp Suite, you can literally copy it all. We can create a post request with JavaScript and send it to a server, as described in this StackOverflow post.
Get the entire request sent to the server on Burp Suite (or get the basic fields and create your own post request), and write JavaScript code that sends it to the server via a post request millions and millions of times. This way, there will be plenty of spam on the home page, and in the worst case, shut the server down. Some people in class managed to crash the server.

c) Read the user's cookies, and send potentially sensitive information to Moriarty.
JavaScript can read cookies. To do so, all you have to do, according to w3schools, is use document.cookie. JavaScript can also make post requests, just as described by this StackOverflow post. So, in this attack, the hacker might read the user's cookies and send them to a server owned by the hacker through a POST request.
The user clicks the post made by the hacker, and the javascript in the post steals their cookies and sends them to the hacker's server via POST. The hacker will have all the data from the cookies. While they might not steal login information (as session cookies are HTTP only), the hacker gets all the other cookies. If the server uses cookies to track how the user uses the website, the hacker can steal all of that information.

The hacker can also insert other malicious cookies into the user's browser, or update the value of their existing cookies. Chrome does not allow to update of the value of session cookies, nor let the hacker read session cookies (or any other httponly cookies). However, they might update any other cookie, to break the user experience.
Yes, I do feel that this example is not good enough. So there's another attack that the hacker might execute is a Web Worm.
According to Veracode one of the impacts of XSS is spreading webworms, that might give hackers access to the user's browsing history, and even control the browser remotely. Other worms might simply add code that consumes bandwidth and overloads the user's computer. Just like the code might steal the user's cookies, it might also steal their browsing history, or do operations that might slow the user's computer down.

d) The easiest way to do this is to check for HTML tags on the input by the user. Whenever the server sees < or > it should replace it with a special character, so the user's browser does not think it is an actual tag. For < I think the character code is &lt. It would also be good to prevent users from adding the Unicode characters for < > as an additional way to prevent cross-site scripting. However, I still feel that the most effective way to prevent it is by looking for the < and > character in plain text and replacing it.

Sources:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Connection
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Vary
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
https://www.linuxquestions.org/questions/linux-general-1/where-cookies-are-store-in-linux-516811/
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer
https://askubuntu.com/questions/1066986/how-can-i-find-the-cookies-files-to-specific-websites
https://apple.stackexchange.com/questions/232433/where-are-google-chrome-cookies-stored-on-a-mac#:~:text=Session%20cookies%20are%20only%20stored,Cookies%20%2C%20it's%20an%20sqlite3%20database.
https://www.w3schools.com/js/js_cookies.asp
https://stackoverflow.com/questions/6396101/pure-javascript-send-post-data-without-a-form
https://www.veracode.com/security/xss
https://www.veracode.com/security/computer-worm