Assignment 8 Extra Credit: Rachel Rockenfeller

URL: kermitthefish.c1.biz/login.php

1. The first attack I did on my system was to go in and alter the password format. I did not use JS to do this, but I did have a HTML input pattern attribute. By switching this in the Inspect page, I could get around the issue.

   This shows that you should not put checks like this in the HTML of your page if you really care about format since anyone can go in and change it.

   Steps: Click on Signup. Enter an email [tea@gmail.com] and incorrect formatted password [RRR] (correct format is one uppercase letter, one lowercase letter, and one number). Select Submit. An error occurs. Go into the Inspect page for the website, and locate the area in the code where the input is accepted and alter the pattern attribute.
   - Original: pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{3,}"
   - Change: pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"

   Then enter a password matching what you desire [RRRrrr33]. You should be logged in with that password. Check the database to verify that your password is entered correctly.

   How I chose to exploit this isn't really a bad attack that would cause much harm to my system. It merely exploits the formatting of the password. However, with this flaw in my code, I can enter a large amount of characters into my database. As such, I could come up with many more exploits.

2. The second attack I did on this website was more serious. For this one, I utilized a XSS attack.

   Steps: In the blog page in the textfield, I wrote a [“<script>alert(“hello world”)</script>”] and hit send. The result was that this Javascript line of code was stored in the database. Therefore, everytime you go to that page a “hello world” message appears and you have to click out of it.

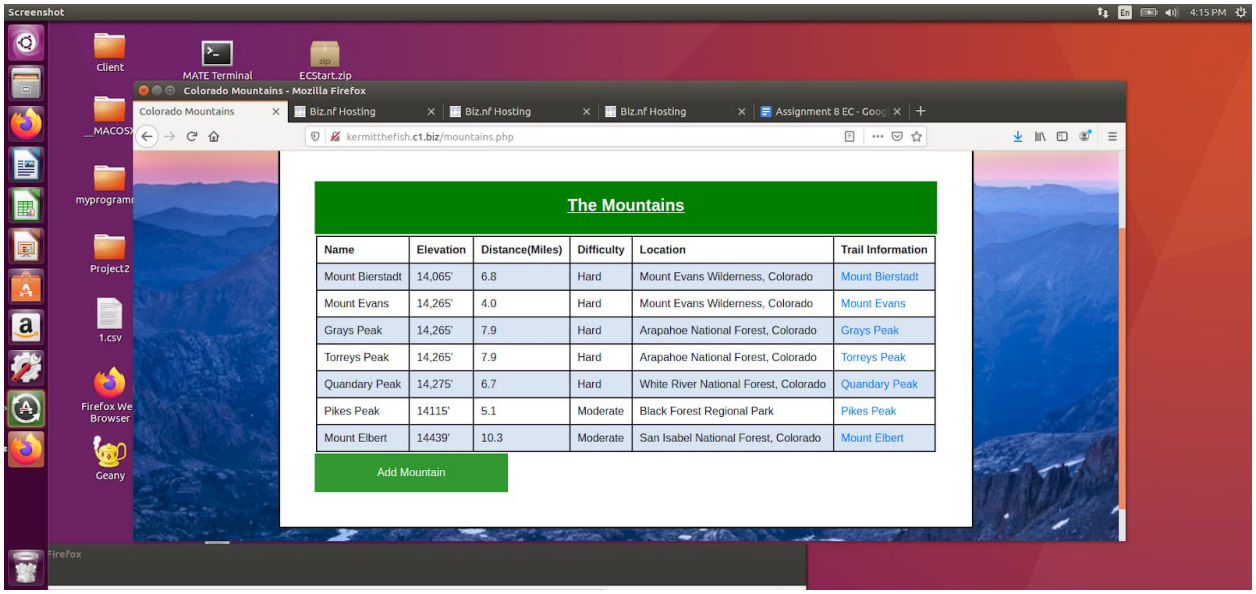
   While this particular example of an XSS attack is very minor, it could be used to harvest the credentials of the user, hijack the session cookies of the user, or instead of “hello world” it could send a user to an insecure and dangerous webpage.

   Note: you can leave the level of difficulty empty for this form.

   Source:
   https://cyware.com/news/drawing-the-contrast-between-xss-and-csrf-attacks-a646270a

3. The third attack I did on this website was like the 2nd attack except I entered [“<img src=”Boooo”)>”] into the blog page text field and hit send. The line of Javascript was stored in the database and appeared in the resulting table as trying to put an image into the comments column.
   This shows a different way I could cause issues for my website. For instance, if I were to block <script> tags in my code, I could still get around it by using the <img> tag. It could do the same/similar harm to a user but it would just be a different method of writing that code. To ensure this would not be an issue, I would have to make all Javascript tags be treated as strings in my code or just not permit them.

4. The next attack I did was in the Mountains page of my website. In this part, you can add a mountain to the database. To do so, you must click the “Add Mountain” button on the table and fill out a form. The final input field creates a link to a website with more information about that particular hike. I filled out each of the fields with correct information until I got to the last link field (see below image for Mount Elbert) . At that point, I entered the http://sharedslides.com/signup link instead of a link for information about the actual trail.



This is an example of a CSRF attack because you must be “logged in” to access the Mountain page and enter a new mountain into the database. Furthermore, because my code names the link as the mountain name given in the first column of the table, it is not easy to see what website you will be sent to. Users would be tempted to click on the link because all the other entries in the database are legitimate links. They would assume this would also lead to a legitimate site. However, it could instead lead to a very dangerous site.

5. The final attack I did was similar to the first attack. However, instead of going in and altering the format of the passwords in the Signup page, I bypassed the login on the Login page. To do this, I went into the Inspect page. I then found the login inputs in the HTML code and removed the [required=""] part of the code from the following lines:

   <input type="email" name="email" required="">
   <input type="pwd" ,="" id="pwd" name="pwd" pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{3,}" title="Must contain one number and one uppercase and one lowercase letter." required="">

   I then hit the Submit button and I was logged in. Because I could bypass this, I was never given a user cookie.

Things that didn't work:

1. The way I implemented the discussion forum, any user with an account can see the discussion and the email id's of everyone who posted anything. I tried to go into the Inspect page and alter the cookie so if you were logged in as Sandy@gmail.com you could alter the cookie to say you were Zippy@gmail.com. My idea was you could then post things to the discussion page as Zippy even though you were Sandy. This was unsuccessful and showed the true user in the database.

2. I tried doing a SQL Injection attack on the Login page of my website by entering another user's email and then ["or=""="] as the password to see if something would be returned to me. My thought was that if I could use this sql injection to get the password of another user I could login and create a legitimate session as that user.

   This did not succeed because I had placed a HTML input pattern attribute that was able to block the original attack. I tried taking that piece out of my HTML code but was still unable to get past the php check I implemented.
   Resource: https://www.w3schools.com/sql/sql_injection.asp

3. I tried to do other SQL injections similar to those discussed in the Robert'); DROP TABLE example. However, where I added addslashes in my code prevented this from working unless I could find a way to bypass the addslashes adding a slash before the single quote. I found sources on how to do this, but I could not seem to get it to work.