Samuel Lopez

Application Security

Assignment 2: When a Wreck Reaches the World Wide Web

Part1

XSS Vulnerability

For the XSS vulnerability, I found a vulnerability in the buy_card_view function with the director variable. If you follow this director variable into the templates it's used in, gift.html and item-single.html, you see that the director variable has the Safe tag attached to it. Normally Django automatically escapes a string to prevent wrongdoing, but because the Safe tag was attached to the director variable, it didn't escape the string and this allowed me to use the string "127.0.0.1:8000/gift.html?director= <script> alert('Vulnerable to XSS!') </script>" to force an alert to show up on the screen once typed into the URL box of the web browser. This vulnerability can be mitigated by removing the Safe tag from the director variable

CSRF Vulnerability

For the CSRF vulnerability I used the same vulnerability that was exploited for the XSS above. The use of the Safe tag on the director variable allow me to use the string " 127.0.0.1:8000/gift.html?director=<script id="CSRFscript"> let csrfvul = new XMLHttpRequest(); csrfvul.open("POST", '/gift/0', true); var formData = new FormData(); formData.append("username", "sl4506@nyu.edu"); formData.append("amount", "1000"); csrfvul.send(formData); </script>" This script presents a link to the user that, if clicked, automatically gifts a card to the user with username sl4506@nyu.edu with a value of 1000. This is also mitigated by removing the Safe tag from the director variable.

SQL Injection Vulnerability

For the SQL injection vulnerability, I found that in the views.py file, under the use_card_view function, there is a SQL query being performed with an unescaped single quote for the signature of the gift card being passed in. This allows me to pass in the signature "sl4506@nyu.edu' UNION SELECT password FROM LegacySite_user WHERE LegacySite_user.username='admin'--" Here, the single quote after .edu allows me to run the SQL query that will return the password to the site. This vulnerability can be mitigated via removing the superfluous single quote in the function.

Password Salting

This vulnerability is more so with the configuration of the database and the way the passwords are salted when the user registers a new login. The current way the salting is setup, the program will save the same password string with the same salt leading to the same hash output being stored on the database. Now this isn’t a direct problem but if let’s say the database of the site was dumped by a bad actor due to a vulnerability in the site, they’d be able to see that perhaps some users have the same hash, and therefore it would be easier for them to figure out the passwords for multiple users. This vulnerability is caused in the extras.py file, under the generate_salt function. Here we see that we're using the random.seed() function and passing in the SEED variable, which is created in the settings.py file with a random string of characters. The issue with this, is that since we're passing a variable to the random.seed() function, it'll always use the same string to hash, which is good for testing, but not good for production. The fix is to remove the SEED variable from the random.seed() function and this will allow it to actually create random hashes by using the current system time since no argument is passed in.