

Assignment 2

Link to project: <https://github.com/klunghamer/Application-Security-Project>

The primary way that I made this web app resilient against common web vulnerabilities is through the user text input and file upload.

I used MongoDB for this project, which is not vulnerable to SQL injections, but is still vulnerable to other injections. To prevent attackers from inputting {\$ne:null} in the username and password fields or similar attacks, I configured Mongoose to only accept strings as input.

```
4 var UserSchema = new mongoose.Schema({
5   username: { type: String, required: true},
6   password: { type: String, required: true},
7 })
```

Another preventative measure before Mongoose switched all input to strings was to use express-sanitizer middleware to remove all script tags and other javascript DOM methods, such as alert(). This should prevent any cross-site scripting and also any attempts for cookie theft. Here is an example on the login route:

```
22 app.use(expressSanitizer());

49 router.post('/login', passport.authenticate('local'), function (req,res) {
50   req.body.username = req.sanitize(req.body.username)
51   req.body.password = req.sanitize(req.body.password)
52   req.session.save(function (err) {
```

To prevent cross-site request forgery, I used express-session and csrf to create tokens for the GET methods which valid POST requests on all forms. For example when the user is at the home screen, a token is created using req.csrfToken() and placed as the hidden value on the login form:

```
14 router.get('/', function (req,res) {
15   console.log(req.csrfToken());
16   res.render('home', {
17     title: 'Spellchecker',
18     token: req.csrfToken()
19   })
20 });
```

```
17     <form action="/login" method="POST">
18       <div>
19         <label for="text">Log in: </label>
20         <input type="hidden" name="_csrf" value="{{token}}">
21         <input type="text" name="username" placeholder="username">
22         <input type="password" name="password" placeholder="password">
23         <button type="submit" name="submit"> Log in</button>
24       </div>
25     </form>
```

Lastly, I validated the user input of the text file for the spellchecker. If the user inputs a file without the .txt extension, they are unable to upload.

```
73 router.post('/upload', function(req, res) {
74   if (req.files.newFile.name.slice(-4) !== '.txt') {
75     res.send("Invalid input!")
76   }
```

I also check the file size of the text file in case the user changes the extensions to trick the previous validation. I only allowed files less than 100kB to be processed.

```
101   if (getFileSizeInMBytes('output.txt') > 0.1) {
102     res.send("Error in file. Please upload text file less than 100kB.")
103   }
```