Safety Scouts - Durden, Johnson, Trinity, Casella

Mini-Project 1 Report

Exploit: IDOR

The user's page is directly accessible by manipulating the URL to be '/users'. Were the site larger, weaknesses like this could expose a lot of exclusive (if not private) information! This was mitigated using authentication tokens (cookies) to keep track of user sessions and limiting access to only those with valid cookies. These cookies contain a random 64 character base 64 id and expire after a short time (preventing guessing and stealing).

Exploit: CSRF

The change password feature is vulnerable here. By using the user that has recently accessed the site, the attacker can make a post request to '/changepassword' and change that user's password and giving the attacker control of the user's account (example found at '/csrf'). To mitigate this, sensitive actions, like password resets, require re authentication with the user's old password as well (csrf tokens and CORS protection were considered, but not implemented).

Exploit: XSS

The users page (via the registration page) is vulnerable to XSS attacks. Registering with a name such as <svg onload=alert("Gotcha")> will successfully leave your attack for the next person to access the users page. This is mitigated by sanitizing any HTML in usernames that could be potentially threatening using 'sanitize-html' which escapes any potentially threatening html elements.

Exploit: SQL Injection

The username and password section of the login page is vulnerable to SQL injection. Using the username admin and the password 'OR'1'='1, you can gain access to the admin account without knowing the password (it's admin if you wanted to know) This form allows attackers to manipulate the database as they wish (though they cannot get any information returned directly). The primary mitigation is to sanitize any SQL inputs using the node library 'sqlstring' so that all input is properly escaped.

Hashing vs Encrypting

Encrypting passwords hides the content from everyone that doesn't have a key while hashing hides the content from *everyone*. This reduces the vulnerabilities that come from internal breaches and losses of the private key. The safest way to store a password is where NO ONE knows the password. Hashing accomplishes this, encryption does not. Though, encrypting a hash is a relatively inexpensive additional wall of security. In addition, hashes built for protection are slow and expensive which makes brute forcing slow and expensive likewise.

Salt and Pepper

Salting passwords precludes the use of rainbow tables for the hash function used. The protection is because the hashes of common passwords are dependent upon the specific salt used. Peppering is very similar to salting, but instead of keeping it public or at least in the database, the pepper is stored in the code. This means a breach of the database AND the source code are required to even attempt to crack the passwords. However, once the pepper is compromised, it becomes identical to a salt and is difficult to change without rebuilding the database.