Alan Chiang
achiang31

Target 1

Since the account web page does not check request origins, attackers are able to arbitrarily send requests to the vulnerable web page using the permissions of the victim browser. Verifying the request origin through its headers would be an easy fix for this. Also, while a CSRF token seems to be set, it does not appear to be checked anywhere in the source code. Finally, the trivial hashing function and check does not seem to do much, especially when the result of the hash is printed to the page.

Target 2

The biggest vulnerability here is that the login web page writes form data directly to the DOM. This allows attackers to redirect victims to the vulnerable web page with malicious scripts (disguised as form data), giving them complete access to the victim's DOM environment, including the victim's cookies. Performing proper input validation and sanitization would help prevent this issue by blocking arbitrary code from being injected into the web page.

Target 3

While some form of input validation is used in user authentication, the input validation is very poor. For example, "#" is removed after "--", so by entering "-#-" as input, I am able to avoid the initial comment check while still commenting out the rest of the query once the pound symbol is removed. Also, the second query (which checks the username against its password) doesn't use the escaped username string for some reason. A better alternative, however, would be to use prepared statements, which forces a specific statement template to be used. Whitelisting inputs is another solution, but may not be as useful when there are so many usernames to check. Finally, the input validation could simply be improved to account for tricks like the one mentioned above, though that sounds a bit more challenging to do right.