Jeffrey Qiu

HW2 Writeup

To create this website, we needed to create a registration, log in, and a spell check page. I also created a home page that allowed for navigation between these pages. The home page also gave the ability to log off if the user is logged in. However, none of the sites redirects back to the home page. So the easiest way to navigate is by typing in the url that you're looking for instead of navigating back to home every time.

Because we are giving the user certain abilities once they log in, we are using sessions instead of cookies to hold onto this data. This is because it is more secure for the user information to be stored in sessions where it is harder to read than if they were stored in cookies. So sessions are created when the user logs in and the session ends when the user logs off. These sessions persist so even though the user might close out of the webpage, as long as the server remains running, they can reopen the website and remain logged in.

CSRF is an attack which can involve using the victim's credentials to access a site. Vulnerabilities include any forms that allow users to POST data. In order to protect from these attacks, we used CSRFProtect from a python module. This creates a token when rendering forms. These tokens need to be included in requests otherwise the request is rejected. Since the attacker doesn't know what needs to be included in the token it is difficult to forge this token allowing requests to be more secure.

XSS can occur because we are allowing the user to input various data. Jinja does an ok job at mitigating this attack because they escape all values. However, jinja does not protect against attribute insertion which was avoided in this scenario by completely avoiding placing jinja in html attributes. However when we did have to do this with csrf tokens, quotations were added around the jinja portion. This makes it harder to escape into the html. We also used wtforms to handle all the form data.

In order to protect to create a more general protection against more XSS or other attacks that may come from different addresses, we implemented a content security policy where the website will only accept scripts from valid sources. So in this case, we are only accepting scripts from our own website. This was done by adding a content security policy http header that defines where to accept scripts from.

When attacking my site, I first started by inputting various things in the forms because I wanted to see if I can escape some part of the forms. But this didn't get me anywhere. So then I tried to add various scripts in the url to see if I can trigger anything but that didn't work either.

Using inspect element, I was able to find values for csrf token, the session id, and the session. However, I wasn't really sure what to do with those values to create an attack. You can potentially try to create an http request with those tokens and see if the website will accept that request. You might also be able to hijack the session with the session id. However, the website should be set up to block traditional XSS methods.