Nanako Chung

Brendan Dolan-Gavitt

Application Security

October 27th, 2019

Assignment 2 – Report

**WEEK 4**

*GitHub repo link*: <https://github.com/nanako-chung/assign-2>

**WEEK 5**

*Design decisions for the Web service*:

One of the most secure ways to build a Web service is through Python’s flask module. We also did not have to worry about data-based attacks because all of the data was stored in memory and databases were not used.  
 It has three pages: the register, login, and spell\_check. The spell\_check page can only be accessed upon login, and this is checked using Flask. The user registers and inputs a username, password, and 2fa phone number (which are all verified) and are all stored into a dict with this structure:

actual\_username: {  
 password: hashed\_password,

2fa: phone\_number

}

The password is hashed using md5 for the reasons discussed in class. Again, this dict disappears after the application restarts because it is stored in memory. After the user logs in with these credentials, a session is set up using Flask to verify that they are logged in and directs them to the page accordingly. With the use of sessions and without the use of JavaScript, we do not need to worry about cookies or session hijacking (see below for how I mitigated this). After login, we get HTTP/1.1 Responses, with 200 code status for redirecting of links and 302 code status after the user logs in.

*How I mitigated different categories of common Web vulnerabilities*:

Using Flask is extremely helpful because it automatically protects against both the XSS attacks we discussed in class, and using Flask-Paranoid and login\_manager.session\_protection = ‘strong’ can further help with this. Flask also easily integrates CSRF tokens to prevent CSRF attacks, as used in my code. Flask also prevents session hijacking by using Flask’s session attribute, as used in my code. Further, Flask/Jinja under the hood help prevent command line injection through its built in curly-brace placeholders in the templates, also as seen in my code.

For even more safe practices, as the OWASP Cross Site Scripting Prevention sheet mentions, I did not put untrusted data except in allowed locations. Every time the input was taken from the text input field when given by the user, we store it directly into memory (hashed if necessary) or just used it to keep us logged in using Flask’s session. Additionally, we did not use JavaScript for URLs; we utilized Flask’s url\_for method to redirect our user (which is authenticated before being redirected).

**WEEK 6**

*The vulnerabilities you found, why they occurred, why you didn’t catch them during the first part of the assignment, and how you patched them*:

I did not find any of the vulnerabilities when testing my application for vulnerabilities, and this is most likely because of Flask’s built-in security measures and the overall design of my project as mentioned above.