# CS 161 - Project 3 Write Up

Adam Osborn: cs161-aqy
Emanuel Lucban: cs161-acp

## Weaponize Vulnerability

Out of all the vulnerabilities found, we concluded that the SQL injection vulnerability by modifying the cookie's SESSION_ID provided the best means of crafting an exploit that allows us to post as the user dirks. We were able to modify the cookie's SESSION_ID by first logging in as the test user xoxogg, which is a test user whose credentials are found in a HTML comment in the login page (http://127.0.0.1/login). Using the console in the developer's tools in Google Chrome, we can alter the cookie directly and changing the SESSION_ID to include SQL code. We were then able to use a sub-query to query the sessions table and return the session id of the user dirks.

The console command used to modify the SESSION_ID:
document.cookie='SESSION_ID=sfadfs\' OR id=(SELECT id from sessions WHERE username="dirks") OR id=\''

As you can see, the sub-query will return the SESSION_ID of the user dirks when the get_username_by_session() function is called giving us full access to the user dirk's session. Once we have gained access to user dirk's session, we can now post as dirks or modify anything in his account. This attack requires no interaction with the user dirks.

# Vulnerability Writeup

**Reflected XSS**
Upon reviewing the file server.py, we found a vulnerability in the code pertaining to displaying the wall page. The app pulls the username for the wall from the URL (ex: /wall/<username>). In line 100 of server.py, the function escape_sql() is called to escape SQL in the username, however HTML is not escaped. A malicious link can be crafted where HTML is injected into the username portion of the URL to be reflected back to the user.

URL exploit used:
http://127.0.0.1:5000/wall/%3Cimg%20src=%22fasdf%22%20onerror=%22window.alert()%22%3E

Vulnerability Fix: In the file server.py, line 96 other_user needs to either escape HTML or only allow alpha-numeric characters to prevent HTML from being accepted as a username.

**Code Injection Vulnerability (Profile page - Avatar URL logout)**
Another vulnerability found in the profile page involves the types of URL's we can input as an avatar image. The only checks of the code involves validating if the input starts with http:// or https://. We were able to add the URL http://127.0.0.1/logout as the avatar URL. When viewing the wall page this essentially loads the logout URL when the site tries to display the avatar image, invalidating the user's session ID and denying them use of the site.

Code Injection used:
In avatar field: http://127.0.0.1:5000/logout

Vulnerability Fix: One of the possible fixes is to check that URL ends with a valid image extension such as .jpg, .gif, etc.

## Other Issues

**Session ID's are not truly random.**
In auth_helper.py (line 8) the generate_session_id() function uses the python function random(). However, random() is only a pseudo random number generator. This creates a security flaw because it limits the number of possible session ids. An attacker may also be able to use the source code of python random number generator to predict the next session id that the server will generate.

Proposed fix: Use a cryptographically secure python pseudo-random number generator library such as os.urandom() or SystemRandom.

**Logout does not actually log you out from the server.**
In server.py (line 60) the logout() method replaces the session id in the users browser cookie with an empty string. Server.py should instead wipe the session id from its own table to invalidate it. If the user replaces the session id with the one that was valid before the logout, he can be logged back in without providing a password.

Proposed fix: When logout is called, clear the SESSION_ID from the cookie and delete the session record from the sessions table in the database.

**Passwords are hashed but not salted.**
In auth_helper.py (line 11) it uses a SHA-256 hash algorithm. However, it only uses the password for the input. If an attacker could get access to the hashed password, he could use a rainbow table to reverse look up the user's password. (provided that the password was not high entropy).

Proposed fix: Use Password Based Key Derivation such as Argon2 to salt and hash passwords.

**Sessions ID's never expire.**
The server.py code does not have a method for invalidating or removing old session ids. So even after the user has logged out, the session id will still give access to anyone stores it in a browser cookie. This would still work to gain access even if the user changed their password. With no expiration date, this will still give access years into the future.

Proposed fix: Create a date column on the sessions table and add the current date/time whenever a session is added. Then add code that checks the session date/time and invalidates a session after a set amount of time.

**Clickjacking Attack.**
When posting an HTML iframe (as any user) you can enlarge the frame size beyond the size of the post. This can cover some or all of the users "wall" page. You can also make the iframe

invisible by setting the opacity to equal one.  It is possible to make part or all of the iframe a link to a malicious site. Then when any visitor to the users wall clicks on any part of the page covered by the invisible iframe, they will be taken to the malicious site.

Proposed fix: Use the frame-ancestor directive from Content Security Policy to disallow embedding of potentially malicious pages.

## Mandatory Feedback

We feel that the this project was not only interesting and challenging but it allowed us to put into practice many of the website attacks that we had learned about in class.  The project was definitely worth while and was also fun.
Overall the class has been great and I am very glad I took it.

My one complaint so far is that it has been difficult to do project write ups get full credit.  I feel that it is often the case that the grading rubric does not match what the project instructions asked for.  This was especially true with project one. One classmate described it as “trying to check invisible checkboxes”.  This statement captured the feeling I had where I had thought that I had properly answered some of the questions, only to find out after grading that the rubric was looking for a very specific and different answer.