1. Broken Access Control
   1. There is no check in the function for get\_account() to verify whether the user\_id is valid or is non-fradulent. While input validation isn’t a concern, without a proper check on the user id the case that a call for this function could return user info to someone who is trying to access something they shouldn’t. There should be an additional check in this function for whether the current user is the user of the account they’re trying to access.
2. Cryptographic Failures
   1. While hashing is important, it’s only as secure as the level of hashing you use and what is getting hashed. The flaw in the python code was that it only used SHA1 to hash the password, but in my version, I have encrypted the password before hashing it with SHA256 instead of SHA1. This makes it far more ambiguous to malicious actors that try to intercept the password.
3. Injection
   1. This fails to perform proper query parameterization to prevent injection attacks. It simply passes the username into the premade query without taking additional efforts. Concatenating the given input to the string is what creates this vulnerability, and this must be changed to use something such as the PreparedStatement function to insert the input values into the statement as strings instead of their raw forms.
4. Insecure Design
   1. There seems to be zero implementation of a “strong” password during the reset\_password() function. It does request the new password, but there is little additional context to indicate that the password will be adequate to avoid it being too simple or even being the same as the previous password. The solution would be to implement these checks as well as making sure the current password is stored securely.
5. Software and Data Integrity Failures
   1. This script container has the fault of allowing a potential XSS attack. To prevent these, usage of Subresource Integrity checks could prove very useful. An SRI hash can be generated from the script using a specific sha algorithm and then the hashed string will be used to compare with the specified script to make sure the hashes match. This circumvents a modified script and prevents the potential XSS attack.
6. Server-Side Request Forgery
   1. The problem with this snippet of code is that it is asking for a full URL from the user and then passing that to the .get(), which ideally wouldn’t happen at all. It far better to avoid taking a full URL from the user. However, measures could be made to avoid certain URLs, such as only accepting HTTP or HTTPS URLs. This is done with a type of URL validation, which can get quite complex, but allows for more strict requirements on what is accepted by the application before the URL is sent to the parser.
7. Identification and Authentication Failures
   1. Immediately a problem arises with how the passwords are compared and stored. The passwords may be hashed or encrypted but could also lack proper salting or peppering to prevent a malicious user from cracking the protected password. If the passwords aren’t hashed, they can be hashed with one of several algorithms to further prevent vulnerabilities. The solution here is to add hashing, salting, and/or peppering to the stored password to prevent it from being cracked and making it more secure.