The code you provided has several potential security issues:

1. **Use of HTTP GET Parameters for Sensitive Data**: It appears that the code is expecting the username and password to be sent as query parameters in the URL. This is a bad practice because:

   - URLs (and their parameters) can be logged by servers, proxies, and browsers.

   - URLs are displayed in the browser's address bar and can be shoulder-surfed.

   - They might be stored in browser history.

   Passwords should never be passed as URL parameters. Instead, they should be sent in the HTTP POST body or headers, where they're less exposed.

2. **Lack of HTTPS**: While this isn't directly visible from the code, always ensure that login endpoints are served over HTTPS to encrypt the data in transit. Otherwise, anyone snooping on the network can capture the credentials.

3. **Potential Lack of Password Hashing**: It's not clear from this code snippet how the `is_valid_login` function works, but it's vital that passwords are not stored in plaintext in the database. Typically, passwords should be hashed using a strong cryptographic hash function (e.g., bcrypt or Argon2) and then stored. When verifying user login attempts, the provided password should be hashed in the same way and compared to the stored hash.

4. **No Rate Limiting or Brute-force Protection**: The code does not seem to have any mechanism to prevent or slow down brute-force attacks. Implementing rate limiting or progressively increasing delays after failed login attempts can help protect against such attacks.

5. **No CSRF Protection**: If this endpoint is used in a web context, it should have protection against Cross-Site Request Forgery (CSRF) attacks. This typically involves sending a CSRF token with the login request that gets validated on the server.

6. **Potential for Session Fixation**: The `start_session` method is invoked without any context on how it's generating or managing sessions. If it uses a predictable session ID or does not regenerate session IDs upon successful login, it might be vulnerable to session fixation attacks.

7. **Lack of Input Validation**: There's no validation on the `username` and `password` before they are processed. Always validate and sanitize inputs to prevent issues like SQL injection, if the `is_valid_login` method interfaces with a database.

8. **Error Messages Provide Information**: The error message "Username or password incorrect" can be good from a usability perspective, but it can also inform attackers whether a username is valid or not. Some security experts recommend using a generic "Login failed" message to avoid giving away any information.

9. **Potential for Global Variable Overwrite**: The code sets `username` and `password` without any `let`, `const`, or `var` declaration. This could mean that they are global variables, which is bad for security and maintainability. Always use local variables to prevent unintended side effects.

Addressing these issues will make the login mechanism more robust and secure.