**CODE ALPHA CYBERSECURITY INTERNSHIP**

**TASK-3 : SECURE CODING REVIEW**

By : Navjot Singh

Let's consider a simpler Python application that handles user authentication using a basic username and password system. We'll review the code for security vulnerabilities and provide recommendations for secure coding practices.

# Simple User Authentication System

users = {'admin': 'password123'} # Simulated user database

def login(username, password):

if username in users and users[username] == password:

return True

else:

return False

def main():

username = input("Enter your username: ")

password = input("Enter your password: ")

if login(username, password):

print("Login successful!")

else:

print("Invalid username or password.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Code Review :**

* Password Storage: passwords are stored in plain text, which is insecure. Always hash passwords using a strong hashing algorithm like bcrypt before storing them.
* Input Validation: User inputs (username and password) are accepted directly without any validation. Always validate and sanitize user inputs to prevent injection attacks and other security vulnerabilities.
* No Rate Limiting: There's no rate limiting mechanism implemented for login attempts, making the application susceptible to brute-force attacks. Implement rate limiting to prevent such attacks.

**Recommendations for Secure Coding Practices :**

* Use Secure Password Storage: Hash passwords using a strong hashing algorithm like bcrypt before storing them in the database.
* Implement Input Validation: Validate and sanitize all user inputs to prevent injection attacks. For instance, use libraries like passlib for password validation and ensure username validation checks for allowed characters only.
* Add Rate Limiting: Implement rate limiting mechanisms to prevent brute-force attacks on the authentication system. Limit the number of login attempts within a specific time frame.
* Enforce Strong Passwords: Implement password policies such as minimum length, complexity requirements, and expiration periods to enhance password security.
* Use HTTPS: If this application is deployed over the web, ensure it runs over HTTPS to encrypt data transmitted between the client and server.
* Error Handling: Implement proper error handling to avoid leaking sensitive information through error messages.
* Regular Security Audits: Perform regular security audits and code reviews to identify and address any security vulnerabilities.

In addition to manual review, tools like Bandit and pylint can be used to identify security vulnerabilities and coding errors. Moreover, incorporating third-party authentication libraries or frameworks can enhance security and reduce the risk of vulnerabilities.