Secure Coding Review

Task Number 2 : Prajwal Kunjekar

Let's consider a simpler Python application that handles utilizer authentication utilizing a

rudimentary username and password system. We'll review the code for security

susceptibilities and provide recommendations for secure coding practices

#Simple User Authentication System

users = {'admin': 'password123'}

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 utilizing a vigorous hashing algorithm like bcrypt afore storing them.

\* Input Validation: Utilizer inputs (username and password) are accepted directly

without any validation. Always validate and sanitize utilizer inputs to obviate injection

attacks and other security susceptibilities.

\* No Rate Constraining: There's no rate constraining mechanism implemented for

authenticate endeavors, making the application susceptible to brute-force attacks.

Implement rate circumscribing to obviate such attacks.

Recommendations for Secure Coding Practices :

\* Use Secure Password Storage: Hash passwords utilizing a vigorous hashing

algorithm like bcrypt afore storing them in the database.

\* Implement Input Validation: Validate and sanitize all utilizer inputs to obviate

injection attacks. For instance, utilize libraries like passlib for password validation and

ascertain username validation checks for sanctioned characters only.

\* Add Rate Circumscribing: Implement rate inhibiting mechanisms to obviate brute-

force attacks on the authentication system. Limit the number of authenticate endeavors

within a concrete time frame.

\* Enforce Vigorous Passwords: Implement password policies such as minimum length,

intricacy requisites, and expiration periods to enhance password security.

\* Use HTTPS: If this application is deployed over the web, ascertain it runs over

HTTPS to encrypt data transmitted between the client and server.

\* Error Handling: Implement felicitous error handling to evade leaking sensitive

information through error messages.

\* Regular Security Audits: Perform customary security audits and code reviews to

identify and address any security susceptibilities.

In additament to manual review, implements like Bandit and pylint can be habituated to

identify security susceptibilities and coding errors. Moreover, incorporating third-party

authentication libraries or frameworks can enhance security and minimize the peril of

susceptibilities.