CST1510 Programming for data communication and networks

Week 7: Building a Secure Authentication System

LAB

Lab Objectives

1. Environment Setup
2. Implementing Core Security Functions
3. Implementing User Registration
4. Implementing User Login
5. Testing Your Implementation
6. GitHub Submission

Environment Setup

PULL your project repo from github

Step 1: Install Required Library

In [ ]:
```
pip3 install bcrypt
```

Implementing Core Security Functions

Step 2. Create the Main Python File

Create a new file named `auth.py` in your project directory:

Step 3. Import Required Modules

In [ ]:
```
import bcrypt
import os
```

Step 4. Implement the Password Hashing Function

```
    Hashes a password using bcrypt with automatic salt
generation.

    Args:
```

```
            plain_text_password (str): The plaintext password
    to hash

        Returns:
            str: The hashed password as a UTF-8 string
```

In [ ]:
```
def hash_password(plain_text_password):

    # TODO: Encode the password to bytes (bcrypt requires byte strings)


    # TODO: Generate a salt using bcrypt.gensalt()


    # TODO: Hash the password using bcrypt.hashpw()


    # TODO: Decode the hash back to a string to store in a text file

    return
```

Step 5. Implement the Password Verification Function

```
        Verifies a plaintext password against a stored bcrypt
    hash.

        Args:
            plain_text_password (str): The password to verify
            hashed_password (str): The stored hash to check
    against

        Returns:
            bool: True if the password matches, False
    otherwise
```

In [ ]:
```
def verify_password():

    # TODO: Encode both the plaintext password and the stored hash to byt


    # TODO: Use bcrypt.checkpw() to verify the password
    # This function extracts the salt from the hash and compares

    return
```

Step 6. Test Your Hashing Functions

Before proceeding, test your functions by adding this temporary code at the bottom of auth.py:

In [ ]:
```
# TEMPORARY TEST CODE - Remove after testing
test_password = "SecurePassword123"
```

```
# Test hashing
hashed = hash_password(test_password)
print(f"Original password: {test_password}")
print(f"Hashed password: {hashed}")
print(f"Hash length: {len(hashed)} characters")
```

In [ ]:
```
# Test verification with correct password
is_valid = verify_password(test_password, hashed)
print(f"\nVerification with correct password: {is_valid}")
```

In [ ]:
```
# Test verification with incorrect password
is_invalid = verify_password("WrongPassword", hashed)
print(f"Verification with incorrect password: {is_invalid}")
```

Implementing User Registration

Step 6. Define the User Data File

Add a constant at the top of your file (after imports):

In [ ]:
```
USER_DATA_FILE = "users.txt"
```

Step 7. Implement the Registration Function

```
    Registers a new user by hashing their password and
storing credentials.

    Args:
        username (str): The username for the new account
        password (str): The plaintext password to hash and
store

    Returns:
        bool: True if registration successful, False if
username already exists
```

In [ ]:
```
def register_user(username, password):

    # TODO: Check if the username already exists


    # TODO: Hash the password


    # TODO: Append the new user to the file
    # Format: username,hashed_password

    return True
```

Step 8. Implement the User Existence Check

```
    Checks if a username already exists in the user
database.
```

```
        Args:
            username (str): The username to check

        Returns:
            bool: True if the user exists, False otherwise
```

In [ ]:
```python
def user_exists(username):
    # TODO: Handle the case where the file doesn't exist yet


    # TODO: Read the file and check each line for the username


    return False
```

Implementing User Login

Step 9. Implement the Login Function

```
    Authenticates a user by verifying their username and
    password.

        Args:
            username (str): The username to authenticate
            password (str): The plaintext password to verify

        Returns:
            bool: True if authentication successful, False
    otherwise
```

In [5]:
```python
def login_user(username, password):

    # TODO: Handle the case where no users are registered yet


    # TODO: Search for the username in the file



    # TODO: If username matches, verify the password


    # TODO: If we reach here, the username was not found
```

Building the Interactive Interface

Step10. Implement Input Validation

Add these helper functions for input validation:

```
    Validates username format.
```

```
        Args:
            username (str): The username to validate

        Returns:
            tuple: (bool, str) - (is_valid, error_message)
```

In [ ]:
```
def validate_username(username):
    pass
```

```
    Validates password strength.

        Args:
            password (str): The password to validate

        Returns:
            tuple: (bool, str) - (is_valid, error_message)
```

In [ ]:
```
def validate_password(password):
    pass
```

Step 11. Implement the Main Menu

Add the main program logic:

In [ ]:
```
def display_menu():
    """Displays the main menu options."""
    print("\n" + "="*50)
    print("  MULTI-DOMAIN INTELLIGENCE PLATFORM")
    print("  Secure Authentication System")
    print("="*50)
    print("\n[1] Register a new user")
    print("[2] Login")
    print("[3] Exit")
    print("-"*50)


def main():
    """Main program loop."""
    print("\nWelcome to the Week 7 Authentication System!")

    while True:
        display_menu()
        choice = input("\nPlease select an option (1-3): ").strip()

        if choice == '1':
            # Registration flow
            print("\n--- USER REGISTRATION ---")
            username = input("Enter a username: ").strip()

            # Validate username
            is_valid, error_msg = validate_username(username)
            if not is_valid:
                print(f"Error: {error_msg}")
                continue

            password = input("Enter a password: ").strip()
```

```
            # Validate password
            is_valid, error_msg = validate_password(password)
            if not is_valid:
                print(f"Error: {error_msg}")
                continue

            # Confirm password
            password_confirm = input("Confirm password: ").strip()
            if password != password_confirm:
                print("Error: Passwords do not match.")
                continue

            # Register the user
            register_user(username, password)

        elif choice == '2':
            # Login flow
            print("\n--- USER LOGIN ---")
            username = input("Enter your username: ").strip()
            password = input("Enter your password: ").strip()

            # Attempt login
            if login_user(username, password):
                print("\nYou are now logged in.")
                print("(In a real application, you would now access the d

                # Optional: Ask if they want to logout or exit
                input("\nPress Enter to return to main menu...")

        elif choice == '3':
            # Exit
            print("\nThank you for using the authentication system.")
            print("Exiting...")
            break

        else:
            print("\nError: Invalid option. Please select 1, 2, or 3.")


if __name__ == "__main__":
    main()
```

Testing Your Implementation

Step 12: Basic Functionality Tests

Run your program and perform the following tests:

est 1: Register a New User

1. Select option [1] Register a new user
2. Enter username: alice
3. Enter password: SecurePass123

4. Confirm password: SecurePass123
5. Expected: "Success: User 'alice' registered successfully!"

Test 2: Attempt Duplicate Registration

1. Select option [1] again
2. Enter username: alice
3. Enter any password
4. Expected: "Error: Username 'alice' already exists."

Test 3: Successful Login

1. Select option [2] Login
2. Enter username: alice
3. Enter password: SecurePass123
4. Expected: "Success: Welcome, alice!"

Test 4: Failed Login - Wrong Password

1. Select option [2]
2. Enter username: alice
3. Enter password: WrongPassword
4. Expected: "Error: Invalid password."

Test 5: Failed Login - Non-existent User

1. Select option [2]
2. Enter username: bob
3. Enter any password
4. Expected: "Error: Username not found."

GitHub Submission

Step 13. Update Your README

Edit README.md to include project documentation:

In [ ]:
```
# Week 7: Secure Authentication System

Student Name: [Your Name]
```

```
Student ID: [Your Student_ID]
Course: CST1510 -CW2 -  Multi-Domain Intelligence Platform

## Project Description

A command-line authentication system implementing secure password hashing
This system allows users to register accounts and log in with proper pass

## Features

- Secure password hashing using bcrypt with automatic salt generation
- User registration with duplicate username prevention
- User login with password verification
- Input validation for usernames and passwords
- File-based user data persistence

## Technical Implementation

- Hashing Algorithm: bcrypt with automatic salting
- Data Storage: Plain text file (`users.txt`) with comma-separated values
- Password Security: One-way hashing, no plaintext storage
- Validation: Username (3-20 alphanumeric characters), Password (6-50 cha
```

Step 14. Update Your requirements.txt

Edit requirements.txt to include project modules to install

In [ ]:
```
bcrypt==4.2.0
```

Step 15. Commit Your Work

Stage and commit all your files:

In [ ]:
```
git add auth.py README.md .gitignore users.txt
git commit -m "Complete Week 7: Secure authentication system with bcrypt"
```

Push to GitHub

Push your work to GitHub:

In [ ]:
```
git push origin main
```

Verify Your Submission

1.Visit your GitHub repository in a web browser

2.Verify that all files are present:

- auth.py
- users.txt
- README.md

```
•.gitignore
```

3.Check that your README displays correctly

4.Verify the commit message is descriptive

Extension Challenges (Optional)

If you finish early, try implementing these additional features:

Challenge 1: Password Strength Indicator

Add a function that rates password strength:

```
    Evaluates password strength.

    Returns:
        str: "Weak", "Medium", or "Strong"
```

In [ ]:
```python
def check_password_strength(password):

    # Implement logic based on:
    # - Length
    # - Presence of uppercase, lowercase, digits, special characters
    # - Common password patterns
    pass
```

Challenge 2: User Role System

Extend registration to support user roles:

```
    Register a user with a specific role (user, admin,
analyst).
```

In [ ]:
```python
def register_user(username, password, role="user"):

    # Modify the file format to: username,hashed_password,role
    pass
```

Challenge 3: Account Lockout

Implement a system that locks accounts after 3 failed login attempts:

In [ ]:
```python
# Track failed attempts in a separate file or data structure
# Lock account for 5 minutes after 3 failures
```

Challenge 4: Session Management

After successful login, generate a session token and store it:

```
    Creates a session token for a logged-in user.
```

In [ ]:
```python
import secrets

def create_session(username):

    token = secrets.token_hex(16)
    # Store token with timestamp
    return token
```