SYSC 4810

Ali Abdollahian #101229396

User authentication system

December 5, 2024

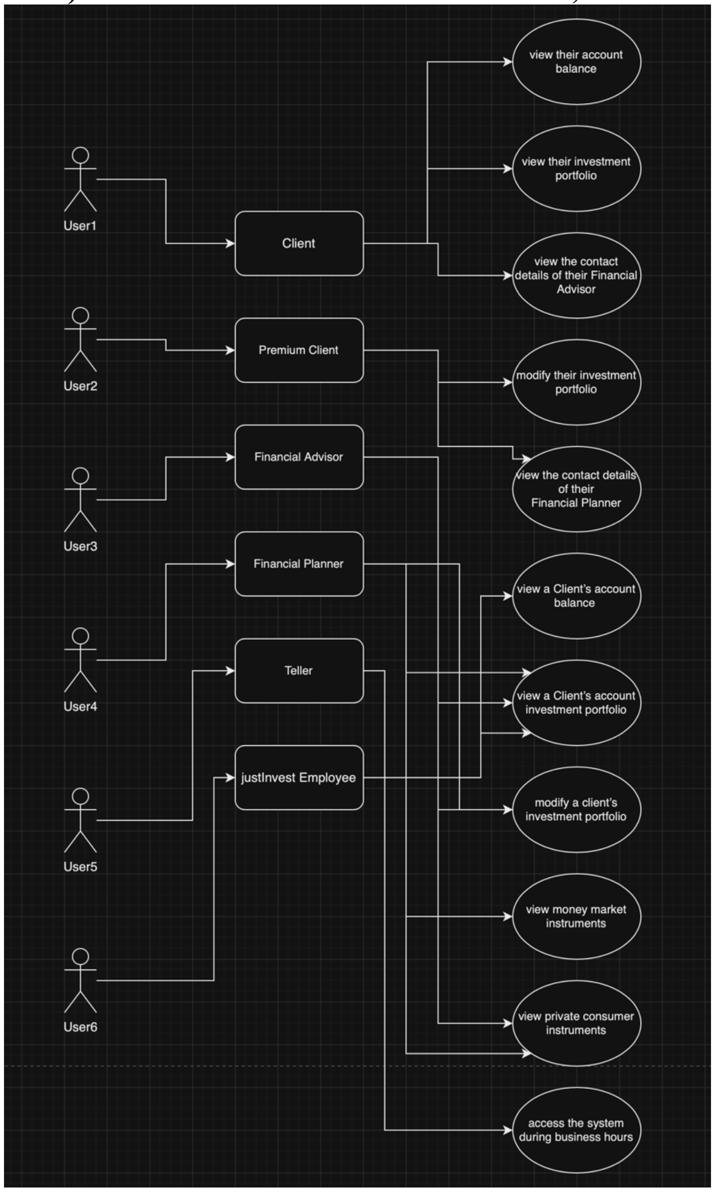
Note: the default password for the provided sample list of employees and clients is "defaultPassword123@"

Problem 1

1) **List the chosen access control model and justify your answer:**

There is multiple access control such as DAC, MAC, RBAC. However, I have decided to use RBAC which permissions are assigned based on user roles and a user is represented as a subject and can be assigned one or more role since in this company we have a set of pre-assigned permissions and subject's role determines its permissions it is best to use RBAC.

2) **Include the sketch and annotate it, if needed.**


User1
User2
User3
User4
User5
User6
Client
Premium Client
Financial Advisor
Financial Planner
Teller
justInvest Employee
view their account balance
view their investment portfolio
view the contact details of their Financial Advisor
modify their investment portfolio
view the contact details of their Financial Planner
view a Client's account balance
view a Client's account investment portfolio
modify a client's investment portfolio
view money market instruments
view private consumer instruments
access the system during business hours

3) **Describe the test cases you used and show that they provide adequate coverage of the access control policy.**

User Role Validation:

1. Valid Users: Tests verify that users (Sasha Kim, Noor Abbasi, Mikael Chen, Alex Hayes) are correctly assigned their respective roles (CLIENT, PREMIUM_CLIENT, FINANCIAL_ADVISOR, TELLER).
2. Invalid Users: Verifies that invalid or empty usernames return null and handles case insensitivity.

Operation Authorization:

1. Role-Based Operations: Tests check that roles like CLIENT and PREMIUM_CLIENT have the correct permissions, while TELLER has limited access or all justInvest EMPLOYEE.

Problem 2

1) **Select the hash function.**

Argon2id is my choice of password hashing due to the following reasons:
Argon2id is highly secure, resistant to side-channel and GPU attacks, and is the winner of the 2015 Password Hashing Competition. It allows to adjust memory, iterations, and helping balance security based on system's needs.
I also can easily adjust its settings as needed without changing the algorithm and it stays secure over time.
Why Not Others?

- bcrypt: Secure, but slower and less customizable compared to Argon2id.
- scrypt: Also, secure but not as flexible as Argon2id.
- PBKDF2: Good for compliance but less resistant to GPU attacks.[1]

Based on the recommendations from the Argon2 paper, a balanced approach for hashing passwords would use a 128-bit hash length, 128-bit salt length. [2][3] and the salt must be unique for each password to avoid the risks of salt reuse. Using a cryptographically secure RNG ensures that the salt is truly random [4]

2) **Provide an example records in the password file, and briefly describe its components and the reason for their inclusion.**

Username	Salt	Password Hash
Ali Abdollahian	vcsBtZatoWijokLf	$argon2i$v=19$m=16,t=2,p=1$dmNzQnRaYXRvV2lqb2tMZg$rJIU/+zxJqJDJNtbi4ZNkg

(The password is just the word "password" for example purpose)
Username: Identifies the user during authentication.
Salt: improves password security by making each password hash unique.
Password Hash: In case of the password file is exposed.

3) **Provide a description of the test cases you used to test the password file and explain how these provide adequate coverage.**

Add User Test: Verifies that a new user is added to the password file in the correct format (username:salt:hash) with non-empty fields.

Retrieve User Test: Checks that the getUser method retrieves the correct record for a given username.
Password Verification Test: Tests if the correct password matches the stored hash and salt, and ensures incorrect passwords fail.
File Integrity Test: Adds multiple users and ensures all records remain intact and accessible.
Adequacy:
These tests cover all critical functionalities: adding users, retrieving records, verifying passwords.

Problem 3

I selected the list of 10,000 weak passwords from the GitHub repository because it is associated with OWASP, a well-regarded organization in web application security. OWASP provides widely used wordlists of weak and commonly used passwords. The repository has 174 stars, indicating some community interest, I find it to be a solid starting point for my analysis. [5]

1) **Provide a description of the test cases that you used to test both the enrolment mechanism and proactive password checker and explain how these provide adequate coverage.**

The testVerifyPassword checks if valid passwords are correctly authenticated and invalid ones are rejected. Tests like testPasswordFileIntegrity confirm multiple user records are stored and retrieved accurately. Additionally, the proActivePasswordChecker method is tested extensively for compliance with password strength rules, including length, character types, and restrictions on spaces, punctuation, and weak passwords. The final test confirms that a valid password not in the "worst passwords" list passes the validation.
The EnrolUsersTest class tests the functionality of the EnrolUsers class, which handles user enrollment and role assignment. The tests verify the correct enrollment process with a valid password and role, checking that invalid password trigger error messages, and an invalid role selection results in an error message.
testEnrolUser: Verifies successful enrollment when a user provides a valid password and selects a valid role. It checks that the user is assigned the correct role, and the enrollment is properly written to a CSV file.
testEnrolUserInvalidPassword: Simulates entering a weak password followed by a valid one, that the program prompts for a valid password and then successfully enrolls the user with the correct role.
testEnrolUserInvalidRole: Tests the scenario where a user selects an invalid role, confirms that the system prompts the user to reselect a valid role.

Problem 4

1) **Provide a description of the test cases that you used and explain how these provide adequate coverage.**

The testUserLoginPassword test verifies that the UIManager's login function works as expected when the user provides the correct username and password. It simulates user input for the username (test_user12) and the correct password (ValidPassword123@), then adds the user to the PasswordManager. The test attempts to log in by calling the login method and checks if the login succeeds with the correct password using an assertion. This ensures the login process grants access when valid credentials are entered.

References:
[1] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
[2] https://www.ory.sh/choose-recommended-argon2-parameters-password-hashing/#:~:text=Key%20Length%20(i.e.%20Hash%20Length,key%20of%20the%20required%20length.
[3] https://www.password-hashing.net/argon2-specs.pdf
[4] https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/
[5] https://github.com/OWASP/passfault/blob/master/wordlists/wordlists/10k-worst-passwords.txt