**Unit Tests**

The unit tests below give an idea of how the system should and shouldn’t behave in various conditions. It is important that the programs only return “Accepted” and “Rejected”. These messages are used as to not betray the inner workings of the program to an adversary that is trying to brute force the system. For example, they are unable to learn if they got the username or password wrong if the only output they receive is “Rejected”.

**1 – Two users with the same password should have their passwords hash to different values**

**Description:** This is testing the salting of the hashes. If the salting is done correctly, the identical passwords should hash to separate values. If the passwords hashed to the same value, an adversary would be able to use the information to perform attacks on the database. For example, the hacker could hash the password “cat” and instantly find all the usernames with that password if I did not use a salt correctly. This test is protecting against a bug where passwords would hash to the same value.

**Input:**

**1)** "python .\enroll.py testUser1 testPassword"

**2)** "python .\enroll.py testUser2 testPassword"

**Expected Outputs:**

**1)** Accepted

**2)** Accepted

**3)** Unencrypted database file should have entries with usernames "testUser1 " and "testUser2 " that each have unique hashed passwords.

**2 – Password of only numbers should return “Rejected”**

**Description:** This tests to see if the system rejects a type of “weak” password. Number only passwords reduces the amount of space that an adversary needs to consider when brute forcing a password. This test is protecting against a bug where weak passwords are accepted.

**Input:**

**1)** "python .\enroll.py testUser 123123123"

**Expected Outputs:**

**1)** “Rejected” with exit code -1

**3 – Password that is a word should return “Rejected”**

**Description:** This tests to see if the system rejects a type of “weak” password. A password that is only a single word reduces the amount of space that an adversary needs to consider when brute forcing a password. Single word passwords are also commonly found in the top 100 password list. For example, words like “cat”, “dog”, and “password” are common used as passwords. This test is protecting against a bug where weak passwords are accepted.

**Input:**

**1)** "python .\enroll.py testUser word"

**Expected Outputs:**

1. “Rejected” with exit code -1

**4 – Password of the form [WordNum] should return “Rejected”**

**Description:** This tests to see if the system rejects a type of “weak” password. A password with numbers at the end is similarly weak since it is easier for adversaries to brute force. This test is protecting against a bug where weak passwords are accepted.

**Input:**

**1)** "python .\enroll.py testUser word123"

**Expected Outputs:**

**1)** “Rejected” with exit code -1

**5 – Password of the form [NumWord] should return “Rejected”**

**Description:** This tests to see if the system rejects a type of “weak” password. A password with numbers at the start is similarly weak since it is easier for adversaries to brute force. This test is protecting against a bug where weak passwords are accepted.

**Input:**

**1)** "python .\enroll.py testUser 123word"

**Expected Outputs:**

**1)** “Rejected” with exit code -1

**6 – Trying to enroll with a username that is already in the database should return “Rejected”**

**Description:** This test makes sure that the data being added to the database is valid. If multiple users were enrolled with the same username, the ‘authenticate’ program would be unable to determine which user is trying to login. This test is protecting against a bug where a user can use a username that has already been claimed.

**Input:**

1. "python .\enroll.py validUser goodPassword"
2. "python .\enroll.py validUser someOtherGoodPassword"

**Expected Outputs:**

1. “Accepted” with exit code 0
2. “Rejected” with exit code -1

**7 – Enroll with valid username and strong password should return “Accepted”**

**Description:** This test makes sure that the happy path of the program works correctly. A user that has entered an unclaimed username and strong password should be able to enroll. This test is protecting against a bug where a valid user can’t enroll even though they did everything right.

**Input:**

**1)** "python .\enroll.py validUserName goodPassword"

**Expected Outputs:**

1. “Accepted” with exit code 0

**8 – Authenticating with enrolled username and the right password should return “Access Granted”**

**Description:** This test makes sure that happy path of the program works correctly when authenticating. A user that has entered an enrolled username and matching password should be able to authenticate. This test is protecting against a bug where a valid user can’t authenticate even though they did everything right.

**Input:**

**1)** "python .\authenticate.py validUserName goodPassword"

**Expected Outputs:**

1. “Access Granted” with exit code 0

**9 – Authenticating with enrolled username and a bad password should return “Access Denied”**

**Description:** This test makes sure that a user can’t authenticate without knowing the correct password. It is important that this test returns “Access Denied” and not “Bad Password”. This keeps the adversary in the dark and makes it harder for him to guess a valid username/password combination. This test is protecting against a bug where an adversary can authenticate with a bad password.

**Input:**

1. "python .\enroll.py validUserName goodPassword"
2. "python .\authenticate.py validUserName badPassword"

**Expected Outputs:**

1. “Accepted” with exit code 0
2. “Access Denied” with exit code -1

**10 – Authenticating with bad username and a good password should return “Access Denied”**

**Description:** This test makes sure that a user can’t authenticate without knowing the correct username. It is important that this test returns “Access Denied” and not “Bad Username”. This keeps the adversary in the dark and makes it harder for him to guess a valid username/password combination. This test is protecting against a bug where an adversary knows a valid password and can authenticate without a matching username.

**Input:**

1. "python .\enroll.py validUserName goodPassword"
2. "python .\authenticate.py invalidUserName goodPassword"

**Expected Outputs:**

1. “Accepted” with exit code 0
2. “Access Denied” with exit code -1

**11 – Authenticating with bad username and a bad password should return “Access Denied”**

**Description:** This test makes sure that a user can’t authenticate without knowing the correct password and a correct username. This test is protecting against a bug where an adversary can authenticate with a bad password and username.

**Input:**

**1)** "python .\authenticate.py invalidUserName badPassword"

**Expected Outputs:**

1. “Access Denied” with exit code -1

**Other Tests:**

* **Cant authenticate if rejected in first step**
* **Database should be encrypted and stay encrypted**
* **Edge cases should be handled (Max length strings for username and password)**