Files

main

Go to file t

.idea
__pycache__
templates
.DS_Store
GUI.md
README.md
app.py
login-info.txt
main.py
pyqtgui.py

Login-Password-Infrastructure / README.md

mattsel README.md f7c77aa · now History

Preview Code Blame 53 lines (19 loc) · 3.91 KB Code 55% faster with GitHub Copilot Raw

Login-Password-Infrastructure

Overview: Create a program that can run several tests to ensure not only a valid password is entered, but also grade the strength of the password in regards to its content. The overall goal for this project to to create a system that can help the user ensure that their password is safe from being vulnerable to brute force from common passwords such as. (i.e. birthdays, first/last name, etc.) Another feature of this program is that it can also allow the user to encrypt their password to further ensure its strength.

Ceaser Cypher Encryption - Gives the user the ability the option to receive their password in Ceaser Cypher encrypted version to pad more security for the user's password.

```
#Takes the users password and will utilize the Ceaser Cypher Shift formula to make their password more secure.
def ceaser_encryption(password, n):
    global ciphertext
    ciphertext = ""
    for char in password:
        if char.isalpha():
            if char.isupper():
                ciphertext += chr((ord(char) - ord('A') + n) % 26 + ord('A'))
            else:
                ciphertext += chr((ord(char) - ord('a') + n) % 26 + ord('a'))
        else:
            ciphertext += char
    return ciphertext
```

Stores Hashed Information - Something neat that this program does is that it will store your password locally to your machine via login-info.txt in sha1 hashed text. **THIS IS A TEST DO NOT** store serious information in this system because it is not designed for practical use, simply a simulation of storing hashed information in a database.

```
#The function to has the users email and password using sha1.
def data_storing(email_initial, password):
    email_hash = hashlib.sha1(email_initial.encode()).hexdigest()
    password_hash = hashlib.sha1(password.encode()).hexdigest()
    return email_hash, password_hash
```

Test: Some of the tests that the user's password will go through will be the following ---->

Similarity to email - This test will utilize the Levenshtein Distance Test to discover the similarity of the user's email and password. If they are too similar to one another, the program will prompt the user to enter a new unique password.

```
#Test to see if the password is similar to the user's email.
def similarity_test(password, email):
    distance = Levenshtein.distance(password, email)
    max_length = max(len(password), len(email))
    similarity = 1 - (distance / max_length)
    return similarity
```

Special Characters - This test will use a range of valid special characters that can be used in a password. When the test is run, it will check the user's password for a special character.

```
#Test to see if the function contains any special characters.
def special_test(password, strength_meter, special_character):
    if any(c in special_character for c in password):
        strength_meter[0] += 1
```

Capitalization - This test will check the user's input to see if it includes capitalization throughout the input to make the password more unique.

```
#Test for capitilization in the user's password for more security.
def capital_test(password, strength_meter):
    if any(char.isupper() for char in password):
        strength_meter[0] += 1
```

Character Length - This test will ensure that the user's password has at least 12 characters in the string. This is important to make the password more complex and harder to guess.

```
#This will ensure that the user has at leaast 12 characters to prevent vulnerability.
def length_test(password, strength_meter):
    if len(password) >= 12:
        strength_meter[0] += 1
```

Numerical - This will test to see if the user has included a numerical value in the string. The more complex the password, the harder it is to brute force the password.

```
#Tests to see if the password has any numerical values.
def numerical_test(password, strength_meter):
    if any(c.isdigit() for c in password):
        strength_meter[0] += 1
```

Password Analysis - After each of these tests are completed, the system will increment the password's grade by one each time. To have a valid password, the user must have passed at least one of these tests. If the user fails to pass any of the tests, it will then remind the user with some helpful tips to encourage the user to include in their password to make it not only valid but safer password.

```
#The funciton that will determine the output for the user's password, depending on the tests that it passed.
def meter(strength_meter):
    if strength_meter[0] == 4:
        return "\nYour password is considered EXCELLENT and unique"
    elif strength_meter[0] == 3:
        return "\nYour Password is considered strong"
    elif strength_meter[0] == 2:
        return "\nYour password is weak. Please consider making it more complex and diversified in order to make for a safer password."
    elif strength_meter[0] == 1:
        return "\nVery Weak Password. This password is very simple and can be easily guessed."
```