COMP.SEC.300

Secure Programming

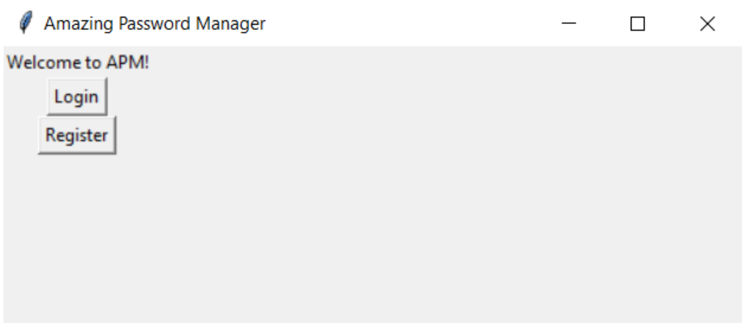
Password manager Documentation

General description:

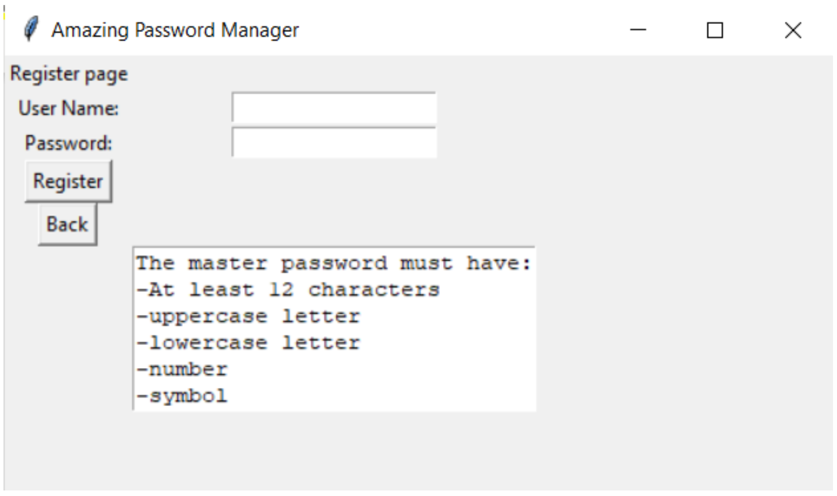
The main purpose of this program is to act as one stop solution for all password storage needs. It supports multiple users, each with their own login credentials and unique encryption keys. New cryptographically secure passwords can be generated quickly and the passwords will be stored behind strong encryption. The program was implemented following OWASP cheat sheet series on password storage and authentication.

The dependencies of the program are listed in the dependencies.txt file. The program was built mainly with Python standard libraries, but some additional libraries were required especially for cryptographical functions. To install the required dependencies the system must have pip installed first. The rest can be easily installed by running the command “pip install -r dependencies.txt”. If this does not work for some reason refer to the text file and try to install them individually.

The program can be started by running the command “python GUI.py”. The following window should now open:

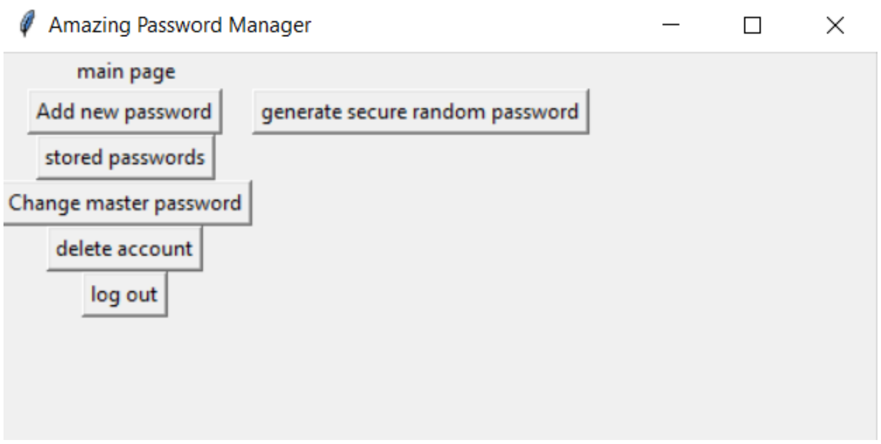


This is the start page. The database file was created upon starting the program, but it is currently empty. To get further one must click “register” to create the master user in the database. After clicking register the view will change to this:



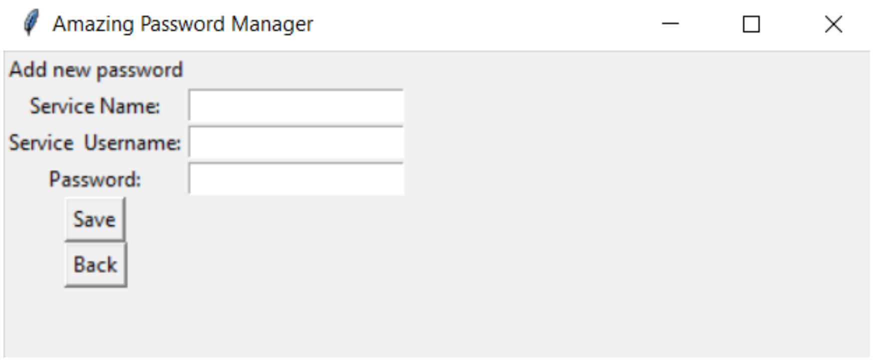
```
The master password must have:
-At least 12 characters
-uppercase letter
-lowercase letter
-number
-symbol
```

The username can have variable length, but password must adhere to the rules presented. The program will check the credentials and warn the user if they are faulty. After agreeable credentials are provided, they are saved to the database and the user is brought back to the start page. Now the login can be chosen, where upon providing the created login credentials user can log in. The main page is then rendered:

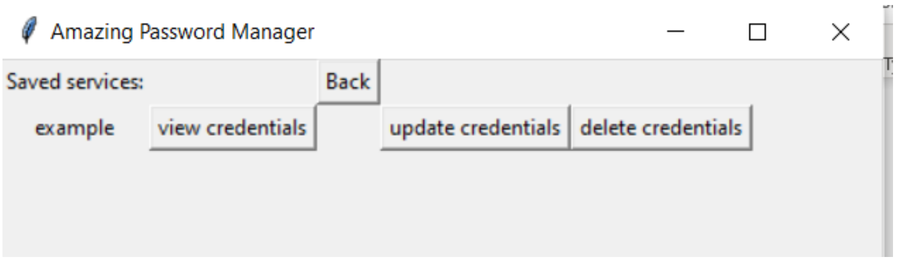


Here are the main functions of the program. By clicking “generate secure random password” the program will generate a secure password and paste it to the clipboard.

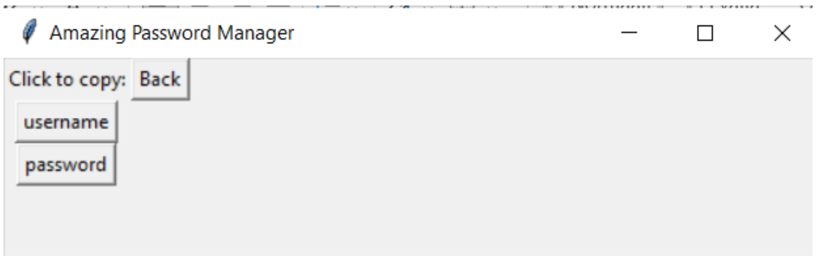
By clicking add new password the user can add credentials to the database for a new service. The view looks like this:



All fields must be filled to proceed. After save is clicked, the credentials are saved to the database. They can then be viewed in the “stored passwords” tab. Which looks like this:



By clicking view credentials one can examine specific credentials:



Click either username or password to copy their content to clipboard. The service credentials have full CRUD implementation so they can be updated or deleted by pressing the appropriate button in “stored passwords”

In the main view pressing “Change master password” provides a form to change the master password of the account. Pressing “delete account” deletes the user credentials and all stored service credentials from the database. The user is prompted before the delete and if yes option is chosen, the deletion is performed. afterwards user is logged out.

Structure of the program

The program consists of four modules (or files).

GUI.py: The main module of the program. Uses tkinter library to build a user interface with functions to render the different views and functions that support the rendering. All functions are commented on what they do in the source code. Creates the main window and calls the “startPage” function to begin execution.

UILogic.py: Houses the logicHandler class that is used to keep track of the user’s id and their unique password that is used to encrypt and decrypt their stored service credentials. Routes the GUI’s database access, providing info on whether the command was successfully carried out. Also handles the more complex database operations.

databaseHandler.py: Module that houses the basic database operations. Connects to database, fetches user and service credentials, deletes them, and updates them. The functions are smaller pieces that UILogic module can chain together to perform bigger tasks.

encryptionHandler.py: Module that houses the cryptographic operations of the program. Uses argon2 to hash and verify the master passwords, PBKDF2 to generate symmetric encryption keys, and Fernet to perform encryption and decryption on the stored credentials.

Secure programming solutions:

The OWASP cheat sheets for password storage:

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

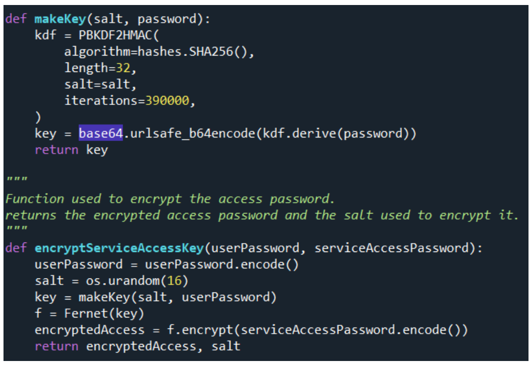
and for authentication:

https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

were referenced for the functionality. From there I implemented the following items.

Salting

For both argon2 and PBKDF2, I generate the salt using the cryptographically secure os.urandom() function. Refer to encryptionHandler.py.



```python
def makeKey(salt, password):
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=390000,
    )
    key = base64.urlsafe_b64encode(kdf.derive(password))
    return key

"""
Function used to encrypt the access password.
returns the encrypted access password and the salt used to encrypt it.
"""
def encryptServiceAccessKey(userPassword, serviceAccessPassword):
    userPassword = userPassword.encode()
    salt = os.urandom(16)
    key = makeKey(salt, userPassword)
    f = Fernet(key)
    encryptedAccess = f.encrypt(serviceAccessPassword.encode())
    return encryptedAccess, salt
```

Upgrading the work factors

Every time the user's master password is re-hashed. Found in databaseHandler.py rehashUserPassword function.

password hashing algorithm

Master password is hashed using argon2.

argon2

The winner of the 2015 password hashing competition. The master password is hashed using this algorithm when registering and re-hashed every time they authenticate. See encryptionHandler.py and databaseHandler.py's storeNewUser function.

PBKDF2

The password-based key derivation function is used to symmetrically encrypt stored service credentials. See encryptionHandler.py.

Implement proper password strength control

Passwords of length 12 and up are mandated for the master user. This is not enforced on the stored credentials as some services may allow weaker passwords. The cheat sheet also recommended a strength checker, but this is not currently implemented.

compare password hashes using secure functions

The password hashes are always compared using the verify function. See encryptionHandler.py verifyUserPassword function.

change password feature

The user is provided the option to change their master password. The credentials stored in the database can be updated too. When changing the master password they must be logged in and they have to provide their old password for verification. See GUI.py userUpdateHelper and UILogic passwordUpdateCheck functions.

authentication responses

When authentication fails the message provided is always "incorrect username or password".

login throttling

We do not directly give maximum number of attempts, but every time the user fails to log in the, the longer the login button will be disabled between attempts. See GUI.py loginUser function.

Testing

The initial plan was to implement at least some basic automated testing of the program, the time limit combined with the trouble of getting the test suite working with the tkinter foiled these plans. Instead there were a lot of manual exploratory testing performed. A lot of small bugs were found this way, especially in user inputs. With the most egregious one being that the system happily accepted a master user without a name or a password into the system. This was of course immediately fixed.

The text fields received the most attention due to the type of the program. Empty field checking and their values received extensive testing, because with encryption a wrong value might encrypt weirdly.

Speaking of encryption, the symmetric encryption functions were written again three times before the final version, due to the encoding breaking at some point of the development. Overwhelmingly most bugs were of this nature. Many times, the encryption would produce a encrypted value that could not decrypted due to an encoding issue. In the final version the key generation and encryption seem to perform as expected.

Further work

The user interface of the program is still a bit rough. If I were to develop the program further, this would be the firs stop. The original plan was to use this exercise as a learning opportunity for the python's graphical library, but the encryption and database side of the implementation ended up swallowing all the time I had available.

The other thing things I thought about but was forced to abandon were multi-factor authentication and password reset functionalities.

Known vulnerabilities

Nothing more was found during the final testing. Bug reports are welcome.

Improvements

As stated before, the user interface could use a facelift. Another small improvement could be to add the password generation button to the service credential storing page too.