Password manager

Descriere proiect

Proiectul meu consta in pastrarea sigura a parolelor pe computerul personal. Aceasta necesitate si inspiratie a mea a aparut din restrictiile si obligatiunile cat mai agresive a diferitelor servicii ce necesita autentificare. In ziua de azi, ni se cer parole cat mai lungi si cat mai complexe, care eventual sa nu semene cu numele de utilizator sau data nasterii. Toate acestea fac dificila memorarea parolelor, mai ales cand acestea nu sunt folosite pentru un timp indelungat.

Tot prin acest proiect, voi incerca sa remedies si problema reciclarea parelelor pentru diferite servicii. Astfel, daca un serviciu va avea baza de date compromisa, acesta nu ne va compromite si celelalte conturi.

Modulul folosit

Limbajul de programare ales de mine este Python, iar modulul cryptografic de care m-am folosit este “cryptography”.

Am ales limbajul Python pentru inlesnirea crearii programului, deoarece eventualul timp castigat la fiecare rulare prin folosirea altui limbaj de programare este neglijabil.

Din cate am observant, modulul “cryptography” inca are suport tehnic fata de alte librarii mai usor de folosit precum “PyCrypto” care sunt inactive de mai mult de doi ani. O data cu inactivitatea, apar si vulnerabilitati care sunt descoperite si nerezolvate, fapt pentru care am decis sa folosesc un modul mai putin user-friendly, dar care e tinut up-to-date.

Site-ul acestui modul: https://pypi.org/project/cryptography/. Dupa cum se poate observa si pe acest site, modulul se poate instala direct cu comanda “pip install cryptography”.

Mod de gandire si implementare

Modalitatea uzuala de a pastra parolele este sub forma hash a lor, dar acest manager de parole trebuie sa ne afisaze parola salvata in text clar. Nu are sens sa salvam parolele prin aceasta metoda uzuala deoarece noi incercam sa obtinem o parola uitata de care presupunem ca nu ne mai aducem aminte nimic. O implementare uzuala cu funtii hash nu ne-ar putea spune decat daca am nimerit sau nu parola, lucru aproape inutil daca ne asumam cele spuse mai sus.

Toate acestea fiind spuse, managerul de parole va functiona pe un sistem de useri cu o parola “Master” care va fi salvata cu ajutorul functiilor hash intr-un mod usual, iar celelalte parole vor fi salvate prin criptarea prin algoritmul AES, folosind drept cheie parola “Master”.

Mai jos este captura de ecran cu o parte din fisierul exportat pe spatiul de stocare al calculatorului. Cum acest fisier poate fi accesat de oricine, este foarte important ca datele sensibile sa ramana private.

```
{
  "users": [
    {
      "name": "Cosmin",
      "salt": "4A9qzdy9TaKl6iV5ISL6lzrDPkgeE64mnfDQGThqwB4=",
      "master_password": "uP+7vmeEuL4mUpT26CqjleVynerh4uVW4k7gSdIGZ/42XHxqjC1J6GWByXcCaUuImKVdoZlHJJ8/pVXsd+tx7w==",
      "seif": {
        "Microsoft": {
          "parola": "kns5+ZFlZYSnV+F6HVNRKQ==",
          "iv": "4vXpv64PrBklyzr5xVnAzQ=="
        },
        "Yahoo": {
          "parola": "AeUtGT3c8Udf4G4hI5IUfQ==",
          "iv": "1nypnymWt+Lewzx8JmG4Gw=="
        },
        "Facebook": {
          "parola": "63uHGxQiYRWGkhvVJ0RpMw==",
          "iv": "VJPw2bKTvDJDiEcrzkTLaQ=="
        }
      }
    },
    {
      "name": "Paul",
      "salt": "bLk5H9P82yM59MNXkks73ecUVZ1OcOGGVMaIeWGjUIc=",
      "master_password": "aeR8px+Jnw6gov3bmoqt4x5hOonvEueNQMNZhIAGFTgbzOSKlTjnWlJIcDlZ/COTSTTyZM5mS/sF8UgTiMR34g==",
      "seif": ""
```


Pentru parola master am folosit algoritmul SHA3_512 din modulul “cryptography”. Cum am invatat ca este un obicei bun de a concatena un salt parolei, am implementat aceasta concatenare si salvare a saltului si de abia apoi am folosit functia hash.

Crearea salt-ului a fost facuta prin functia de sistem os.urandom() deoarece este universal considerata a fi buna din punct de vedere criptografic. Aceasta dant numere random acceptabile.

Crearea salt-ului:

```
salt = os.urandom(32)
```

Functia pentru hash:

```
def encrypt_password_master(parola, salt):

    parola = parola.encode('ascii')
    parola = b''.join([parola,salt])

    digest = hashes.Hash(hashes.SHA3_512())
    digest.update(parola)

    parola_hashed = digest.finalize()

    parola_hashed = byte_to_string(parola_hashed)
    return str(parola_hashed)[2:-1]
```

Pentru criptarea parolelor diferitelor servicii ale userului am folosit AES pe 256 de biti, parola fiind parola master, iar IV-ul fiind generat asemanator salt-ului de la master prin apelul os.urandom().

Functia de criptare:

```
def encrypt_password(parola, key):

    if len(key) * 8 > 256:
        raise ValueError("key e prea mare")

    else:

        while len(key) * 8 < 256:
            key += ' '

    while len(parola) % 16 != 0:
        parola += ' '


    parola = parola.encode('ascii')
    key = key.encode('ascii')


    IV = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CBC(IV))
    encryptor = cipher.encryptor()

    parola_criptata = encryptor.update(parola) + encryptor.finalize()

    return parola_criptata, IV
```

Functia de decriptare(desigur, foarte asemanatoare cu functia de criptare):

```
def decrypt_password(parola, key, IV):

    if len(key) * 8 > 256:
        raise ValueError("key e prea mare")

    else:

        while len(key) * 8 < 256:
            key += ' '
    key = key.encode('ascii')

    cipher = Cipher(algorithms.AES(key), modes.CBC(IV))
    decryptor = cipher.decryptor()

    parola_decriptata =  decryptor.update(parola) + decryptor.finalize()

    parola_decriptata = parola_decriptata.decode('ascii')
    parola_decriptata = parola_decriptata.rstrip()

    return parola_decriptata
```

Aceastea au fost inima algoritmului creat de mine, restul fiind doar unelte pentru comunicarea cu utilizatorul si scrierea si citirea din fisier.

Concluzie

In concluzie, am ales o modalitate de implementare care incearca a spori securitatea cat se poate mai mult si am ales un modul care imi poate oferi toti algoritmii criptografici necesari si care inca are suport tehnic.