

# Lab 5: Password-hashing (iterative hashing, salt, memory-hard functions)

Cilj vježbe: upoznati se pobliže sa osnovnim konceptima relevantnim za sigurnu pohranu lozinki te usporediti klasične (*brze*) kriptografske *hash* funkcije sa specijaliziranim (*sporim* i *memorijski zahtjevnim*) kriptografskim funkcijama za sigurnu pohranu zaporki i izvođenje enkripcijskih ključeva (*key derivation function (KDF)*).

Prvo smo napravili tekstualnu datoteku requirements.txt u kojoj smo zalijepili dani tekst koji je potrebno instalirati a to je popis paketa porebnih za pokretanje budućeg koda. Instalirali smo te pakete koristeći naredbu pip install -r requirements.txt nakon što smo ušli u našu mapu (cd 05-labov). Zatim smo otvorili novu datoteku show\_hash.py u koju smo zalijepili isto dani kod te je pokrenili kako bi provjerili jesmo li sve dobro izvršili. Te smo dobili ovo →

Prikaz usporedbe prosječnog vremena hashiranja pri čemu su dani rezultati dobiveni na temelju 100 ponavljanja jer se na temelju 1 nemože donijeti zaključak. Te smo kasnije dodali i računanje vremena koristeći:

linux\_crypt\_6 (100 puta sporiji od hash\_sha256)

linux\_crypt 100k (1000 puta sporiji)

scrypt\_n\_2\_18 (puno sporiji prosjek je bio 1.66sek)

Te je zaključak da je na prvi pogled vrijeme hashiranja kod sporih hash funkcija djelovalo malo ali kad se usporedi s brzima i s velikim brojem pokušaja hashiranja koje napadač najčešće mora izvesti tada ako je potrebno duže vrijeme to će potencijalno odvratiti napadača od pokušaja napada. Te ako koristeći hash\_sha256 je potrebno 1 dan koristeći linux\_crypt 100k koji je 1000 puta sporiji znaci da

Function	Avg. Time (100 runs)
AES	0.002364

  

Function	Avg. Time (100 runs)
HASH_MD5	0.00028
AES	0.002364

  

Function	Avg. Time (100 runs)
HASH_SHA256	0.000183
HASH_MD5	0.00028
AES	0.002364

je potrebno 1000 dana sto malo manje od 3 godine → neisplativo.

Drugi zadatak je bio implementirati proces inicijalne registracije i login korisnika korištenjem sigurne Argon2 *password hashing* funkcije.

Unijeli smo 3 razlicite vrijednosti te smo uvidjeli da prilikom registracije vrijednost zaporki i ako su iste se hash-ira u različitu vrijednost.

Te kod provjere unesene zaporkke argon2 uz pomoć salta generira hash vrijednost koju onda uspoređuje s pohranjenom vrijednosti.

U funkciji `do_sign_in_user()` od korisnika tražimo i `username` i `password` jer ako bi mu za krivi `username` javili da je neispravan olakšali bi napadaču pokušaje pogađanja. Ovako ako samo javimo grešku u prijavi, napadač ne može zaključiti je li unesen krivi `username` ili lozinka. Ali on i na temelju vremena proteklog može nekako zaključiti što je krivo `username` ili lozinka stoga je potrebno je prije svega prvo provjeravati lozinku pa zatim `username`.

```
import sqlite3
from sqlite3 import Error
from passlib.hash import argon2
import getpass

import sys
from InquirerPy import inquirer
from InquirerPy.separator import Separator

def verify_password(password: str, hashed_password: str) -> bool:
    # Verify that the password matches the hashed password
    return argon2.verify(password, hashed_password)

def get_user(username):
    try:
        conn = sqlite3.connect("users.db")
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE username = ?", (username,))
        user = cursor.fetchone()
        conn.close()
        return user
    except Error:
        return None
```

```

def register_user(username: str, password: str):
    # Hash the password using Argon2
    hashed_password = argon2.hash(password)

    # Connect to the database
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()

    # Create the table if it doesn't exist
    cursor.execute(
        "CREATE TABLE IF NOT EXISTS users (username TEXT PRIMARY KEY UNIQUE, password TEXT)"
    )

    try:
        # Insert the new user into the table
        cursor.execute("INSERT INTO users VALUES (?, ?)", (username, hashed_password))

        # Commit the changes and close the connection
        conn.commit()
    except Error as err:
        print(err)
    conn.close()

def do_register_user():
    username = input("Enter your username: ")

    # Check if username taken
    user = get_user(username)
    if user:
        print(f'Username "{username}" not available. Please select a different name.')
        return

    password = getpass.getpass("Enter your password: ")
    register_user(username, password)
    print(f'User "{username}" successfully created.')

def do_sign_in_user():
    username = input("Enter your username: ")
    password = getpass.getpass("Enter your password: ")
    user = get_user(username)

    if user is None:
        print("Invalid username or password.")
        return

    password_correct = verify_password(password=password, hashed_password=user[-1])

    if not password_correct:
        print("Invalid username or password.")
        return
    print(f'Welcome "{username}"')

```

```
if __name__ == "__main__":
    REGISTER_USER = "Register a new user"
    SIGN_IN_USER = "Login"
    EXIT = "Exit"

    while True:
        selected_action = inquirer.select(
            message="Select an action:",
            choices=[Separator(), REGISTER_USER, SIGN_IN_USER, EXIT],
        ).execute()

        if selected_action == REGISTER_USER:
            do_register_user()
        elif selected_action == SIGN_IN_USER:
            do_sign_in_user()
        elif selected_action == EXIT:
            sys.exit(0)
```