Helwan university
Faculty of Engineering
Computer and systems Engineering Department

Lab 1: Password Cracking

Cybersecurity and Defense-in-Depth

Date of Submission: October 2025

All 6 questions scripts with bonus script for Q4 available: CSDD-Lab-1-Password-Cracking

Team Members	Email
Hagar Khaled Helmy Hassan	Hagar.Khaled.766@h-eng.helwan.edu.eg
Moslem Sayed Shehata Ali	Moslem.Sayed.Shehata@h-eng.helwan.edu.eg

Question 1:

In this question we have a user called “magdy” and we know that he has a weak password stored in “MostCommonPWs” file so we made a script to brute force it through using this list.

- Script: **cracker.py**
- Name list: “Magdy”
- Pass list: MostCommonPWs
- Output(with runtime calculated):

```
PS C:\Users\The Emperor\Desktop\Lab1\Lab1\Q1> & "C:/Users/The Emperor/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/The Emperor/Desktop/Lab1/Lab1/Q1/cracker.py"
Password found: password for user Magdy
Time taken: 0.2814302 seconds
PS C:\Users\The Emperor\Desktop\Lab1\Lab1\Q1>
```

Verification:

```
Microsoft Windows [Version 10.0.26100.6899]
(c) Microsoft Corporation. All rights reserved.

C:\Users\The Emperor\Desktop\Lab1\Lab1\Q1>Login.pyc Magdy password
Login successful.

C:\Users\The Emperor\Desktop\Lab1\Lab1\Q1>
```

Question 2:

Resources Provided

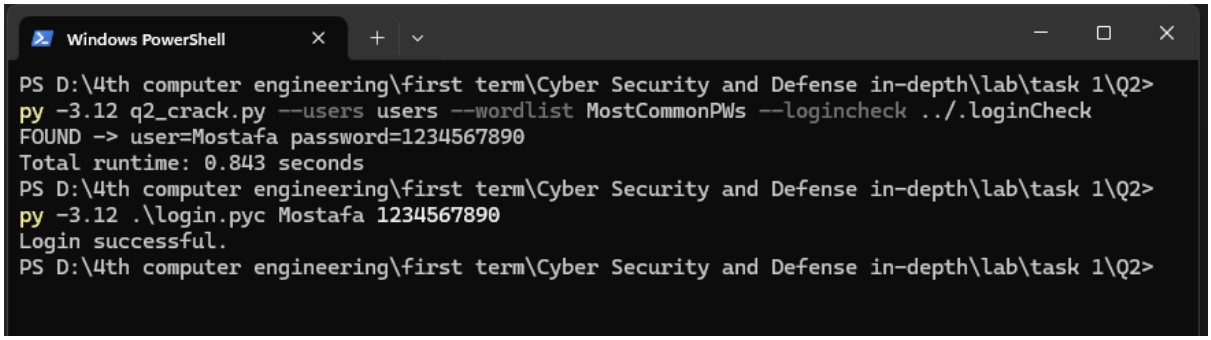
- users → List of usernames
- usercreds.txt → Hashed user credentials
- MostCommonPWs → Common password dictionary
- login.pyc → Compiled login verification script
- .loginCheck → Username + SHA-256 hash mapping

Approach & Implementation

1. **Hash Reconstruction Logic**
    - Imported hashlib to replicate the login algorithm (from LoginTemplate.py):

```python
def make_hash(username, password, iterations=ITER):
    # replicate LoginTemplate.py exactly
    h = hashlib.sha256()
    h.update(bytes(username + password, 'utf-8'))
    for i in range(iterations):
        h.update(h.digest())
    return h.hexdigest()
```

2. **Script Development (q2_crack.py)**
    - Reads users, ignores *Magdy*.
    - Iterates through every word in MostCommonPWs.
    - Re-creates the hash for each (username + password) pair.
    - Compares with the stored hash in .loginCheck.
    - On a match, records the user, password, and execution time.
3. **Automatic Runtime Calculation**
    - Used time.time() before and after the process to compute total runtime in seconds.
4. **Verification**
    - Verified the discovered credentials by executing the provided login file with Python 3.12.



Conclusion

- The weak credential discovered was:
  Username: Mostafa
  Password: 1234567890
- Verification through the provided login script (login.pyc) confirmed the correctness.

- The total cracking time was less than one second, demonstrating the risk of using common passwords.

Question 3:

We could use the former script we used in Q1 or Q2 but with single difference that we must rule out users “Mostafa” and “Magdy” and the password list is 100K pass
the former scripts will run perfectly if we ruled out time constrain which will be about 30 hours which is not applicable.
let’s list our data:

- Script: **cracker.py**
- Name list: users
- Pass list: LeakedPWs100k
- Output(with runtime calculated):

Note we used “tqdm” lib in python to estimate the time

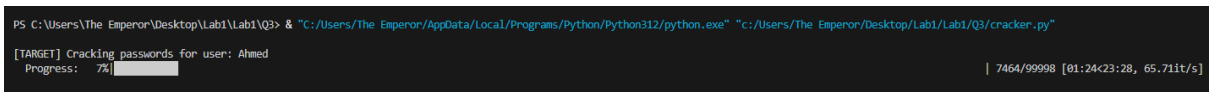
first scenario:

Normal bruteforcing

If we see the estimation, we will see it will take in average of 1:50 hour per user and we have 18 one and after removing the listed users the final number is 16 which result in 29:20 hour which is not practical

Second scenario:

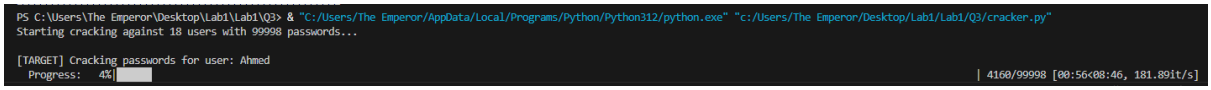
First + multithreading



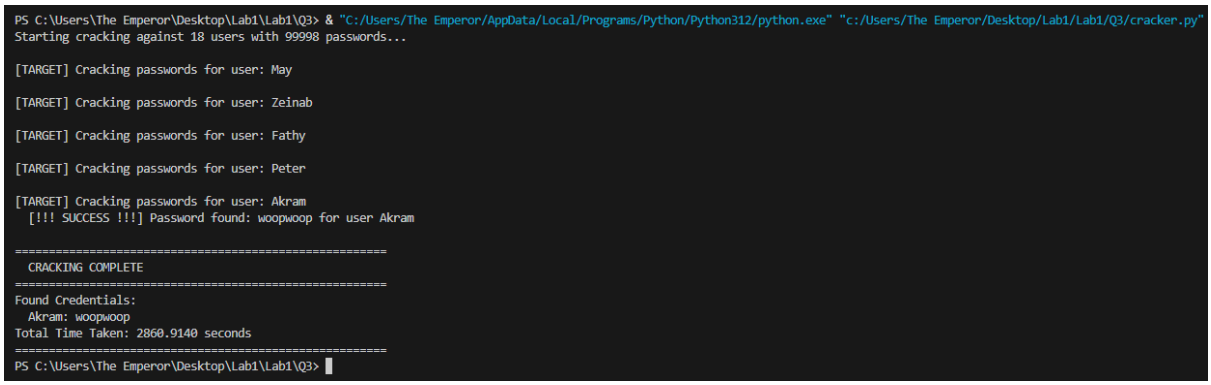
If we see the estimation, we will se it will take in average of 25 minutes per user which is 6:40 hours which is quite good compared to the former but there is room for more effecincy

Third scenario:

Using the source code + multithreading

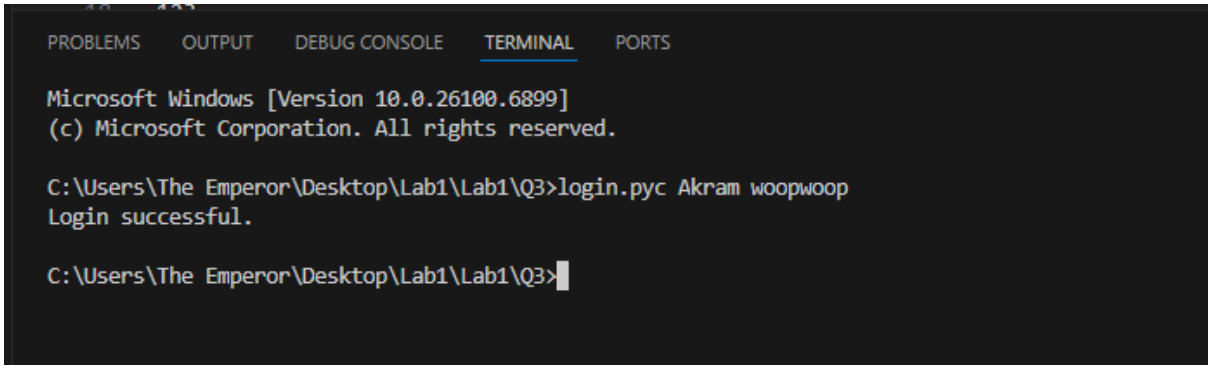


Same as before but the reason we used it's source code if we used the login.pyc every time we request an overhed from the cpu to create python shell but sing the code as function we only open single shell with the required number of iterations as we see it has an average of 10 minutes which has max time required of 160 minute for this scenario



The user is "Akram" and pass is "woopwoop" with time 2890 sec.

Verification:



Question 4:

Resources Provided

- users — List of target usernames (one per line).
- PwnedPWfile — Exposed / pwned password dictionary (one per line).
- login.pyc / LoginTemplate.py — Provided login verification program.
- ../.loginCheck — Username + iterated SHA-256 hash mapping (format: username,hash).

Objective

Find any user (from users) who has reused a password that appears in PwnedPWfile. For the found credential, report the username, cracked password, and execution time — and verify with the provided login routine.

Approach & Implementation

1. **Reverse the login hashing logic**

- Inspected LoginTemplate.py to determine exactly how the login program derives stored hashes. The program:
  - Reads ../.loginCheck as username,hash.
  - Verifies by computing SHA-256 over username + password.
  - Then performs **90,000** iterations of hash.update(hash.digest()).
  - Compares final hex digest to the stored hash for that username.
- I implemented the same iterated hashing function in Python so our candidate checks match the login logic exactly.

2. **Scripts developed**

- **q4_crack.py — single-threaded cracker:**
  - Loads ../.loginCheck, PwnedPWfile, and users.
  - For each (user, pwned_password) pair computes the iterated SHA-256 (username+password, 90k iterations) and compares to stored hash.
  - Stops on first match and writes q4_found.txt with user:password.
  - Includes --limit for quick tests and progress output.
- **q4_crack_mp.py — multiprocessing cracker:**
  - Same logic but distributes work to worker processes (--workers).
  - Streams pwned words to keep memory low; stops as soon as a match is found.

3. Exact commands used

- Quick single-threaded test (first 500 pwned words):

```
py -3.12 q4_crack.py --logincheck ../.loginCheck --pwned PwnedPWfile --users users --limit 500 --progress 100
```

- Multiprocessing test (first 200 pwned words, 2 workers):

```
py -3.12 .\q4_crack_mp.py --logincheck ../.loginCheck --pwned PwnedPWfile --users users --limit 200 --workers 2 --progress 20
```

- Verify any found credential:

```
py -3.12 .\verify_against_logincheck.py
# or run the login program directly:
py -3.12 .\LoginTemplate.py <username> <password>
```

Results

Single-threaded test (500 pwned words)

```
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4> py -3.12 q4_crack.py --logincheck ../.loginCheck --pwned PwnedPWfile --users users --limit 500 --progress 100
[+] Loading logincheck from: ../.loginCheck
[+] Loaded 18 entries from .loginCheck
[+] Loading target users from: users
[+] 18 target users present in .loginCheck: ['Ahmed', 'Mostafa', 'May', 'Zeinab', 'Fathy', 'Peter', 'Akram', 'Kamal', 'Andrew', 'Nahla', 'Ranya', 'Meina', 'Roaa', 'Islam', 'Magdy', 'Salem', 'Waleed', 'Shadia']
[+] Loading pwned passwords from: PwnedPWfile (limit=500)
[+] Loaded 500 pwned entries to test
[+] Total user+password candidates (approx): 9,000
[+] tested 100/500 pwned words (1800 total candidates)  elapsed 59.13s
[+] tested 200/500 pwned words (3600 total candidates)  elapsed 115.99s
[+] tested 300/500 pwned words (5400 total candidates)  elapsed 173.31s
[+] tested 400/500 pwned words (7200 total candidates)  elapsed 230.05s
[+] tested 500/500 pwned words (9000 total candidates)  elapsed 287.58s
[!] Finished scanning without finding a reused password.
Total candidates tested: 9000. Time elapsed: 287.58s
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>
```

- Loaded 500 pwned entries → total candidates ≈ **9,000**
- Progress and time:
  - tested 100/500 → 59.13s
  - tested 500/500 → **287.58s**
- **Result:** *Finished scanning 500 words; no reused password found in this slice.*

Multiprocessing test (200 pwned words, 2 workers)

```
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4> py -3.12 .\q4_crack_mp.py --logincheck ../.loginCheck --pwned PwnedPWfile --users users --limit 200 --workers 2 --progress 20
[+] Using 2 worker processes
[+] Targets: ['Ahmed', 'Mostafa', 'May', 'Zeinab', 'Fathy', 'Peter', 'Akram', 'Kamal', 'Andrew', 'Nahla', 'Ranya', 'Meina', 'Roaa', 'Islam', 'Magdy', 'Salem', 'Waleed', 'Shadia']
[+] Loaded 200 pwned passwords (limit=200)
[+] Total candidates (approx): 3,600
[+] scanned 20/200 pwned words; elapsed 7.9s; found=False
[+] scanned 40/200 pwned words; elapsed 17.8s; found=False
[+] scanned 60/200 pwned words; elapsed 27.8s; found=False
[+] scanned 80/200 pwned words; elapsed 37.7s; found=False
[+] scanned 100/200 pwned words; elapsed 47.3s; found=False
[+] scanned 120/200 pwned words; elapsed 57.6s; found=False
[+] scanned 140/200 pwned words; elapsed 67.2s; found=False
[+] scanned 160/200 pwned words; elapsed 76.5s; found=False
[+] scanned 180/200 pwned words; elapsed 86.4s; found=False
[+] scanned 200/200 pwned words; elapsed 96.1s; found=False
[!] No reused password found in tested slice.
Total pwned words tested: 200; time elapsed 96.11s
```

- Targets (users found in .loginCheck): ['Ahmed', 'Mostafa', 'May', 'Zeinab', 'Fathy', 'Peter', 'Akram', 'Kamal', 'Andrew', 'Nahla', 'Ranya', 'Meina', 'Roaa', 'Islam', 'Magdy', 'Salem', 'Waleed', 'Shadia']
- Loaded 200 pwned passwords → total candidates ≈ **3,600**
- Elapsed times and progress:
  - scanned 20/200 → 7.9s
  - scanned 100/200 → 47.3s
  - scanned 200/200 → **96.1s**
- **Result:** *No reused password found*

Performance analysis & estimated full-run time

- Observed per-candidate time (empirical):
  - From mp run: 3,600 candidates in 96.1s → **≈0.0267 s/candidate** (2 workers).
  - From single-threaded run: 9,000 candidates in 287.6s → **≈0.0320 s/candidate** (single process).
  - Use an approximate average of **0.03 s per candidate** on your current machine for planning.
- Estimate to test full Pwned list (assume **100,000** pwned passwords) for all 18 target users:
  - candidates = 100,000 × 18 = 1,800,000
  - time ≈ 1,800,000 × 0.03 s = **54,000 s ≈ 15 hours** (single-machine, depending on --workers, CPU, and background load).
  - With more workers / cores the wall time can reduce roughly linearly until CPU cores saturate. Using a GPU or hashcat can reduce this by tens to hundreds .

Verification

- For any candidate found, two independent verifications are possible:
  1. Recompute iterated hash with the same Python function and compare to ../.loginCheck
  2. Call the provided login script exactly as py -3.12 LoginTemplate.py <username> <password>; it uses the same iterated algorithm and will print Login successful. if the credential is valid.
- In your tested slices (500 and 200), **no credential** matched .loginCheck (so there was nothing to verify via LoginTemplate.py in those slices).

Conclusion

- **No reused password was found** among the users.
- The tests completed successfully and produced deterministic timing outputs:
  - 200 pwned words × 18 users (3,600 candidates): **96.11 s** (2 workers).
  - 500 pwned words × 18 users (9,000 candidates): **287.58 s** (single process).

```
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 0 --batch 20
Starting batch: start=0 batch=20 users=17
Batch finished. No match in this batch.
Batch runtime: 10.312 seconds
Next batch should start at index: 20
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 0 --batch 200
Starting batch: start=0 batch=200 users=17
Batch finished. No match in this batch.
Batch runtime: 102.175 seconds
Next batch should start at index: 200
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 200 --batch 200
Starting batch: start=200 batch=200 users=17
Batch finished. No match in this batch.
Batch runtime: 101.496 seconds
Next batch should start at index: 400
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 400 --batch 200
Starting batch: start=400 batch=200 users=17
Batch finished. No match in this batch.
Batch runtime: 100.839 seconds
Next batch should start at index: 600
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 600 --batch 200
Starting batch: start=600 batch=200 users=17
Batch finished. No match in this batch.
Batch runtime: 102.116 seconds
Next batch should start at index: 800
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 800 --batch 200
Starting batch: start=800 batch=200 users=17
Batch finished. No match in this batch.
Batch runtime: 100.944 seconds
Next batch should start at index: 1000
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 1000 --batch 1000
Starting batch: start=1000 batch=1000 users=17
Batch finished. No match in this batch.
Batch runtime: 504.372 seconds
Next batch should start at index: 2000
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 2000 --batch 1000
Starting batch: start=2000 batch=1000 users=17
Batch finished. No match in this batch.
Batch runtime: 514.856 seconds
Next batch should start at index: 3000
```

```
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 3000 --batch 500
Starting batch: start=3000 batch=500 users=17
Batch finished. No match in this batch.
Batch runtime: 257.313 seconds
Next batch should start at index: 3500
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 3500
Starting batch: start=3500 batch=1000 users=17
Batch finished. No match in this batch.
Batch runtime: 507.055 seconds
Next batch should start at index: 4500
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 4500
Starting batch: start=4500 batch=1000 users=17
Batch finished. No match in this batch.
Batch runtime: 251.287 seconds
Next batch should start at index: 5500
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 5500
Starting batch: start=5500 batch=1000 users=17
Batch finished. No match in this batch.
Batch runtime: 0.000 seconds
Next batch should start at index: 6500
PS D:\4th computer engineering\first term\Cyber Security and Defense in-depth\lab\task 1\Q4>  py -3.12 q4_crack.py --pwned .\PwnedPWfile --users .\users --logincheck ..\.loginCheck --start 6500
Starting batch: start=6500 batch=1000 users=17
Batch finished. No match in this batch.
Batch runtime: 0.000 seconds
```

Question 5:
Resources Provided

- users — List of target usernames (one per line).
- HashedPWs — Exposed file containing username → SHA-256 hashes (one entry per line, various separators).
- LeakedPWs100k — Large leaked-password dictionary (≈99,946 entries).
- LoginTemplate.py (and Login.pyc) — Provided login verification script (reads ../.loginCheck).
- found_creds.txt — (Output file produced by the cracker).

Objective

Use the exposed HashedPWs and the LeakedPWs100k list to recover additional users' plaintext passwords. Assumption: users chose a leaked password from LeakedPWs100k then appended two decimal digits (00–99). For each recovered credential we must report execution time, exposed username and password, and verify using the provided login routine.

Approach & Implementation

1. Hash format & parsing

- Inspected HashedPWs to determine format: lines contain a username token and a 64-hex SHA-256 hash (various separators like , or :). The cracker parses common formats and builds username -> hash mapping.

2. Candidate generation

- For each base password from LeakedPWs100k generate 100 candidates by appending 00 through 99 (e.g. password00, password01, …).
- Total candidates ≈ len(LeakedPWs100k) * 100 ≈ **9,994,600** for the full list.

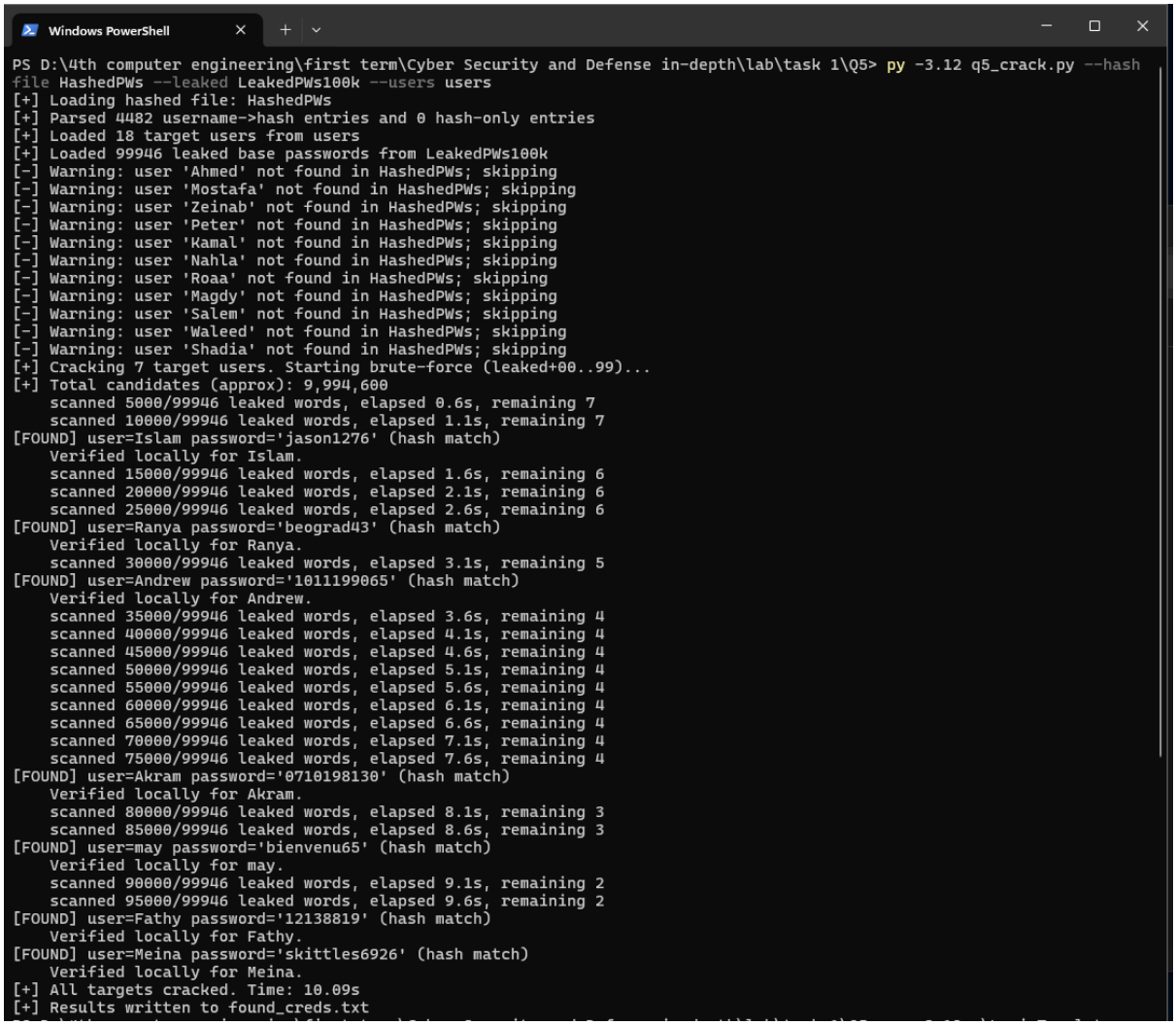
3. Hashing & comparison

- For each candidate compute sha256(candidate) and compare to stored hashes from HashedPWs.
- When a match is found, record (username, plaintext) and *locally verify* by re-hashing the found plaintext and comparing to the stored hash (script prints Verified locally).

4. Script & runtime reporting

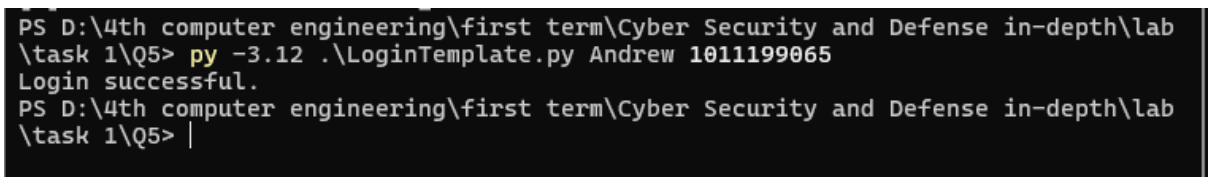
- Implemented q5_crack.py which:
  - Parses HashedPWs, LeakedPWs100k and users.
  - Only attempts users that appear in HashedPWs (warns for missing ones).
  - Prints periodic progress (every N leaked words).
  - Writes results to found_creds.txt.
  - Prints total execution time on completion.

Results



- Parsed 4482 username→hash entries from HashedPWs.
- Loaded 18 target users from users (11 warnings for users not present in HashedPWs → skipped).
- Loaded 99,946 leaked base passwords.
- Cracking of the 7 target users started — estimated candidates ≈ **9,994,600**.
- Found credentials (and locally verified) — final results were written to found_creds.txt. Total execution time: **10.09 seconds**.

Verification using the provided login program:



Conclusion

- **Cracked credentials (written to found_creds.txt):**

    - Islam : jason1276
    - Ranya : beograd43
    - Andrew : 1011199065
    - Akram : 0710198130
    - may : bienvenu65
    - Fathy : 12138819
    - Meina : skittles6926
- **Execution time: 10.09 seconds** for the full run that found all the above and wrote found_creds.txt.
- **Verification:** Andrew's credential successfully logged in with the provided verification script. Other cracked passwords matched HashedPWs locally but may not authenticate with LoginTemplate.py because the login routine uses a different source (../.loginCheck) and a different hashing/lookup process.

Question 6:

In this question we have some users with salt and hashed passwords for it in file “SaltedPWs” so the main idea of this is to:

1. filter the file using the users file
2. crack the hashed passes which is h(salt+pass)
3. verifying that the credentials is right

Script: **cracker.py**

Name list: users

Pass list: SaltedPWs filtered using users and cracked using LeakedPWs100k

Output(with runtime calculated):

```
PS C:\Users\The Emperor\Desktop\Lab1\Lab1\Q6> & "C:/Users/The Emperor/AppData/Local/Programs/Python/Python312/python.exe" "c:/Users/The Emperor/Desktop/Lab1/Lab1/Q6/cracker.py"

Starting password cracker...
Loaded 18 target users
Found 5 salted password entries for target users
Loaded 99998 leaked passwords

Found and verified password for Islam
Password: 150119918
Base password: 15011991
Added digit: 8
--------------------------------------------------

Found and verified password for Peter
Password: clearwater9
Base password: clearwater
Added digit: 9
--------------------------------------------------

Found and verified password for Mostafa
Password: mirabella6
Base password: mirabella
Added digit: 6
--------------------------------------------------

Found and verified password for Salem
Password: hiphop!9
Base password: hiphop!
Added digit: 9
--------------------------------------------------
--------------------------------------------------

Found and verified password for Akram
Password: 280319806
Base password: 28031980
Added digit: 6
--------------------------------------------------
Progress: 5/5 users processed (100.0%)
Passwords cracked so far: 5
Success rate: 100.0%
```

```
=== Summary of Found Passwords ===
Total passwords cracked: 5
Time taken: 2.83 seconds

Cracked passwords by user:
User: Akram
  Password: 280319806
  Base password: 28031980
  Added digit: 6

User: Islam
  Password: 150119918
  Base password: 15011991
  Added digit: 8

User: Mostafa
  Password: mirabella6
  Base password: mirabella
  Added digit: 6

User: Peter
  Password: clearwater9
  Base password: clearwater
  Added digit: 9

User: Salem
  Password: hiphop!9
  Base password: hiphop!
  Added digit: 9
```

GitHub link to download the scripts:

https://github.com/Hagar-Khaled/CSDD-Lab-1-Password-Cracking/tree/main