Password Cracking Using Python and Threading

This project focuses on building a high-speed password cracker using Python. It uses multithreading to test thousands (or millions) of password guesses per second against a target cryptographic hash. The tool supports both wordlist attacks (testing passwords from a file) and brute-force attacks (generating all possible combinations). It automates a key process in security auditing, demonstrating Python's power for computationally intensive tasks.

Objective

To develop a Python script that efficiently cracks a given cryptographic hash (e.g., MD5, SHA-256) by using multithreading to rapidly test passwords from either a wordlist or a generated sequence.

Introduction

Password cracking is the process of recovering a plaintext password from its stored hash. This is done by "guessing" a password, hashing it using the same algorithm, and comparing the result to the target hash. This technique is used by ethical hackers to test password strength. In this project, Python's hashlib module is used to perform the hashing, concurrent.futures manages the multithreaded execution, itertools generates brute-force combinations, and tqdm displays a progress bar.

Working Process

1. The script parses command-line arguments using argparse to get the target **hash**, the **hash type** (e.g., md5), and the attack **mode**.
2. The user must choose one of two modes:
    - --wordlist: The script reads all passwords from a specified text file into memory.
    - --generate: The script creates a password generator using itertools.product based on a character set and length range (e.g., all 4-character combinations of 'a-z').
3. A ThreadPoolExecutor is created to manage a pool of worker threads (e.g., 100).
4. The script iterates through the entire list of passwords (from the wordlist or generator) and submits each one as a separate task to the thread pool.
5. Each thread runs the check_hash() function. This function takes a password guess, encodes it, hashes it with the specified algorithm (e.g., hashlib.md5(...)), and compares the resulting hash to the target hash.
6. If a guess matches, the thread returns the correct plaintext password.
7. The main thread uses tqdm to display a live progress bar, updating as each task (guess) is completed.
8. When the correct password is found, the script immediately prints the "SUCCESS" message, stops all other running/pending threads, and exits.

Tools and Libraries

Tool	Purpose

Python 3	The core programming language.
hashlib	Standard library for creating cryptographic hashes (MD5, SHA-256, etc.).
concurrent.futures	Standard library for high-level management of a thread pool.
itertools	Standard library used to generate all possible password combinations for brute-force mode.
tqdm	External library (pip install tqdm) for displaying a clean, real-time progress bar.
argparse	Standard library for parsing command-line arguments and options.
string	Standard library used to provide default character sets (e.g., string.ascii_lowercase).

Code Explanation

- **generate_passwords(chars, min, max)**: A generator function that uses itertools.product to yield all password combinations for a given character set and length range.
- **check_hash(password, target_hash, hash_fn)**: The core function run by each thread. It hashes the password using the specified hash_fn and returns the password if it matches the target_hash.
- **main()**: The main function that orchestrates the entire process. It parses arguments, sets up the attack mode (wordlist or generate), calculates the total number of guesses, starts the ThreadPoolExecutor, and manages the tqdm progress bar as results come in.
- **if __name__ == "__main__":**: The entry point of the script, which calls main().

Sample Input & Output

Input (Wordlist Attack):

```
C:\Users\Asus\Desktop\physics sample>python password_cracker.py 8a7f41036677c20ccc27465d70d0164a -w wordlist.txt
```

Output (Success):

```
C:\Users\Asus\Desktop\physics sample>python password_cracker.py 8a7f41036677c20ccc27465d70d0164a -w wordlist.txt
[*] Mode: Wordlist (wordlist.txt)
[*] Target Hash (md5): 8a7f41036677c20ccc27465d70d0164a
[*] Total Passwords to Test: 9
[*] Starting cracker... (Press Ctrl+C to stop)
Cracking: 100%|                                                                                | 9/9 [00:00<00:00, 9072.03pass/s]

[+] SUCCESS! Password found.
[*] Password: febinmaxon
[*] Hash: 8a7f41036677c20ccc27465d70d0164a
[*] Time taken: 0.02 seconds
```

Results

The script successfully cracked the target MD5 hash (098f6bcd4621d373cade4e832627b4f6).

- **Wordlist Mode:** Using a simple 3-word list, the script found the correct password (test) almost instantaneously (e.g., < 0.01 seconds).

- **Generate Mode:** When configured to test all 1-4 character lowercase/numeric passwords, the script tested over 1.7 million combinations. The ThreadPoolExecutor completed this task in just a few seconds, correctly identifying test as the password.

Advantages

- **Extremely Fast:** Leverages concurrent.futures to use all available CPU cores, dramatically speeding up the cracking process.
- **Flexible:** Supports two different attack modes (wordlist and brute-force) to suit different scenarios.
- **Extensible:** Can crack any hash type supported by hashlib (MD5, SHA1, SHA256, etc.) by simply changing the --type argument.
- **User-Friendly:** The tqdm progress bar provides excellent, real-time feedback on the cracking progress, showing the speed in guesses per second.

Limitations

- **CPU-Bound:** This script is limited by the computer's CPU speed and core count. It is not as fast as a GPU-based cracker.
- **Simple Brute-Force:** The itertools generator is simple. It does not support complex "hybrid" attacks or mask-based rules (e.g., "P@ssword" style).
- **No Salt Support:** This script is designed for "unsalted" hashes. It will not work on modern password databases that use a unique salt for every password.

Conclusion

This project successfully demonstrates the creation of a high-performance, multi-threaded password cracker in Python. It highlights the effectiveness of concurrent.futures for parallelizing CPU-bound tasks and shows how hashlib and itertools can be combined to build a practical cybersecurity tool for auditing password strength.