Multi-Threaded Hash Cracker in Python

By Inlighn Tech (Anjana Saiju)

Multi-Threaded Hash Cracker in Python
By Inlighn Tech (Anjana Saiju)

Abstract
This project presents an educational Python tool that demonstrates dictionary and brute-force password cracking against hashed values. The tool supports common hash algorithms (MD5, SHA-1, SHA-256, SHA-3, SHA-512), uses multithreading to accelerate attempts, and provides progress feedback via tqdm. The goal is to help students understand hash functions, cracking techniques, and performance trade-offs in a controlled and ethical environment.

1. Introduction & Objective Passwords are typically stored as cryptographic hashes. This project demonstrates how attackers attempt to recover passwords, and how defenders can better protect systems. Objective: implement a documented, safe Python script to demonstrate dictionary and brute-force attacks.

2. Background - Cryptographic hash functions (MD5, SHA family) - Dictionary vs brute-force attacks - Importance of salts and strong hashing

3. Design & Implementation - Language: Python 3.x - Libraries: hashlib, itertools, concurrent.futures, tqdm, argparse, string - Key components: - generate_passwords(min_length, max_length, characters) - compute_hash(hash_fn, password) - crack_with_wordlist(target_hash, hash_fn, wordlist_path, max_workers) - crack_with_bruteforce(target_hash, hash_fn, min_length, max_length, characters, max_workers) - Safety: chunking to avoid memory blowups, conservative default threads

4. Usage - Dictionary example: python3 cracker.py 5f4dcc3b5aa765d61d8327deb882cf99 -w wordlists/small.txt --hash_type md5 - Brute-force example: python3 cracker.py 098f6bcd4621d373cade4e832627b4f6 --hash_type md5 --min_length 1 --max_length 4 --characters alnum --max_workers 4

5. Results & Observations - Dictionary attacks are fast if correct password in list. - Brute-force scales exponentially; CPU-limited on Python. - Multithreading gives improvements but is bounded by GIL

for pure-Python work and by CPU cores.

6. Limitations & Ethics - Educational demo only; not optimized for production cracking. - No salt support or real password store parsing. - Must obtain written permission before real testing.

7. Future Work - GPU acceleration (hashcat / PyOpenCL) - Salt handling and small rainbow table demo - GUI / Streamlit for safe lab demos

8. References - Python hashlib docs - itertools docs - tqdm docs

Appendix: Key snippets (see cracker.py in repo)