Introduction & Overview
ClatsCracker is a cutting-edge password cracking tool designed for cybersecurity professionals, penetration testers, and security auditors. It enables users to quickly identify and address weak passwords in their systems by attempting to recover passwords from hashed values. Through its extensive feature set, ClatsCracker helps assess password strength, verify security measures, or recover authorized passwords. Importantly, this tool is intended for ethical use only, on systems and passwords that you have explicit legal permission to test. Unauthorized use may be illegal.

Key Capabilities & Features

- **Auto-Detection of Hash Types:**
  ClatsCracker can automatically identify the type of hash provided, streamlining the workflow and eliminating guesswork.
- **Real-Time Metrics & Progress Reporting:**
  The tool provides Estimated Time of Arrival (ETA) and Attempts Per Second (APS) metrics, giving users insight into current performance and how long the cracking process might take.
- **Logging Session Details:**
  All attempts, results, and settings can be logged for future reference, making it easy to revisit previous tests, compare results, or perform audits.
- **Graceful Interruption Handling:**
  Should the user need to stop the process (e.g., pressing Ctrl+C), ClatsCracker handles interruptions cleanly, preserving partial progress data and preventing corrupted output.
- **Dynamic Resource Management:**
  Users can adjust resource usage on-the-fly, including changing the number of threads mid-process, ensuring optimal use of system resources without restarting the entire cracking task.

What the Tool Does

- **Hash Cracking:**
  Given a password hash, ClatsCracker 1.0 attempts multiple passwords—either from dictionary files or via brute force. If a match is found, the recovered password is displayed.
- **Algorithms Supported:**
  - **Unsalted:** MD5, SHA1, SHA256, SHA512, SHA3-256, Scrypt
  - **Salted:** Bcrypt, Argon2id

  Supporting eight popular hashing algorithms, ClatsCracker is a one-stop solution for multiple hashing methods, including legacy and modern forms.

- **Cracking Methods:**
  1. **Dictionary-Based Attacks:** Quickly test large lists of known or common passwords, removing duplicates for efficiency.
  2. **Brute Force Attacks:** Systematically try every possible password of a given length and character set, ideal for unknown or very strong passwords.
- **Resource Management:**
  ClatsCracker leverages multithreading to speed up the cracking process. Users can select low, medium, high, or custom thread counts (1 to 1000 threads) and can dynamically adjust resources mid-operation. This flexibility ensures users can optimize performance as needed.

Correct Verification of Salted Hashes
Salted hashes like Bcrypt and Argon2id require precise handling. Many tools fail by rehashing passwords with new salts. ClatsCracker uses:

- `bcrypt.checkpw()` for Bcrypt
- `argon2-cffi`'s `PasswordHasher.verify()` for Argon2id

This ensures proper verification with the original salts and parameters, increasing the likelihood of successfully cracking more complex hashes.

User-Friendly Interface & Controls
A menu-driven interface, intuitive prompts, and real-time progress updates (including APS and ETA) simplify the process, even for non-experts. Users can start with a low number of threads and scale up as needed, while also benefiting from clean logging and graceful interruption handling.

Threaded Performance & Resource Management
Using Python's `ThreadPoolExecutor`, ClatsCracker runs multiple password checks in parallel. Thread-safe locks ensure accurate progress reporting. Dynamic resource management allows you to adjust threads and other settings on-the-fly, maintaining the best balance between speed, stability, and system resource usage.

Validation & Error Handling
ClatsCracker validates hash lengths and formats before attempting a crack. If the input is invalid, it prompts users to correct it, saving time and reducing confusion.

Ethical Use & Compliance
ClatsCracker includes disclaimers and encourages responsible usage. It aligns with legal and ethical standards for penetration testing, incident response, and password recovery. Compliance with applicable laws and regulations is essential.

In Practice

- **Penetration Testing & Auditing:** Quickly identify weak passwords and strengthen organizational security.
- **Research & Development:** Benchmark and compare different hashing algorithms' resistance to cracking.
- **Incident Response & Forensics:** Recover critical passwords after security breaches.

Future Updates
User-requested feature additions, bug fixes, and timely updates will be provided when possible

Technical Requirements

- **Environment:** Python 3.x
- **Libraries:**
    - Built-in: `hashlib`, `itertools`, `string`, `sys`, `os`, `time`, `threading`, `concurrent.futures`
    - External: `bcrypt`, `argon2-cffi` (Install with: `pip install bcrypt argon2-cffi`)
- **Compatibility:** Works on Windows, macOS, and Linux with Python 3.x and required libraries.

Limitations & Future Considerations

- Cracking speed depends on hash complexity and system resources.
- Algorithms like Argon2id and Scrypt are intentionally slow, resisting brute force attempts.
- Ongoing changes in hashing standards may require future tool modifications.

Step-by-Step Usage Guide

1. **Run the Tool:** Start from your command line or IDE.
2. **Legal Notice:** The tool displays a disclaimer reminding you to use it ethically and legally.
3. **Choose Resource Usage (Threads):**
    - Option 1: Low (1 thread)
    - Option 2: Medium (4 threads)
    - Option 3: High (8 threads)
    - Option 4: Custom (1–1000 threads)

      Begin with Low or Medium if you're new. You can adjust resources later if needed.
4. **Main Menu:**
    - Option 1: Crack Password
    - Option 2: Exit

      Select Option 1 to proceed.
5. **Hash Detection:**
   Either let the tool auto-detect the hash type or specify the algorithm if prompted.
6. **Enter the Hash Value:** Paste or type the hash. If invalid, you'll be asked to re-enter.
7. **Choose the Cracking Method:**
    - Dictionary-Based: Provide dictionary file paths. The tool deduplicates and tests each password.
    - Brute Force: Specify password length. The tool tries all character combinations of that length.
8. **Run the Cracking Process:**
   Observe real-time metrics (ETA, APS). If a match is found, it's displayed; if not, you'll be informed. Session details are logged.
9. **Interruption & Adjustment:**
   If you need to stop or adjust threads mid-process, do so gracefully. The tool handles interruptions and dynamic changes without losing state.
10. **Timing & Feedback:**
    After completion, the tool reports total runtime and logs the session, aiding in analysis and performance optimization.

Example Usage

- **Dictionary Attack (MD5):**
    1. Select resource usage.
    2. Choose "Crack Password."
    3. Allow auto-detect or specify `md5`.
    4. Enter the MD5 hash.
    5. Select dictionary-based attack, provide dictionary file(s). The tool runs until it finds a match or exhausts options.
- **Brute Force Attack (SHA1):**
    1. Select resource usage.
    2. Choose "Crack Password."
    3. Allow auto-detect or specify `sha1`.
    4. Enter the SHA1 hash.
    5. Choose brute force, specify length (e.g., 4). The tool attempts all 4-character passwords, reporting APS and ETA.

FAQ

- **Q: Can I crack any hash I find?**
  A: No. Only use this tool on hashes you’re authorized to test. Unauthorized use is illegal.
- **Q: Why is it slow?**
  A: Some algorithms are designed to be slow. Increase threads, use more powerful hardware, or accept that certain algorithms resist quick cracking.
- **Q: Hash not recognized?**
  A: Ensure the hash is correctly formatted or let the tool auto-detect the type.
- **Q: Can I add more algorithms?**
  A: Yes, with code modifications, but this is not recommended for beginners.

Legal & Ethical Disclaimer
This tool is for educational, authorized security testing only. Always act lawfully and ethically. Compliance with all applicable laws and regulations is your responsibility.