PASSWORD HASH CRACKER TOOL

A PROJECT REPORT

Submitted by

CHAITANYA REDDY
&
CHETHAN REDDY

CSE (CYBER SECURITY)

in

BRANCH OF STUDY

LOVELY PROFESSIONAL UNIVERSITY

TABLE OF CONTENTS

PASSWORD HASH CRACKER TOOL

1.ABSTRACT

This report details the development and functionality of a Python-based password hash cracker tool. The tool aims to recover original passwords from their hash values by using brute-force techniques with wordlists. It supports multiple hash algorithms, including MD5, SHA-1, SHA-256, and SHA-512. The tool provides a user-friendly command-line interface for input and displays real-time progress during the cracking process. This report also discusses the challenges faced during implementation and ethical considerations of using such a tool.

2.INTRODUCTION

i. Background

Password hashing is a common security practice used to protect passwords by transforming them into fixed-length strings using cryptographic hash functions. Despite its effectiveness, hashed passwords can still be vulnerable to brute-force attacks, where an attacker tries many possible passwords until the correct one is found. This tool is designed to demonstrate the brute-force technique in an ethical and controlled manner.

ii. Objectives

To develop a tool that can crack hashed passwords using various hash algorithms.
To provide an educational demonstration of the brute-force attack method.
To create a user-friendly command-line interface for ease of use.

3. Literature Review

In this section, we review existing password cracking tools and techniques, including their strengths and limitations. We discuss common hash algorithms, the concept of brute-force attacks, and the ethical implications of using such tools.

i. Existing Tools and Techniques

- John the Ripper: A fast password cracker that supports a variety of hash types.
- Hashcat: Advanced password recovery tool that utilizes GPUs for faster computation.
- Rainbow Tables: Precomputed tables for reversing cryptographic hash functions.

ii. Common Hash Algorithms

- MD5: Widely used hash function producing a 128-bit hash value.
- SHA-1: Produces a 160-bit hash value, now considered insecure.
- SHA-256: Part of the SHA-2 family, producing a 256-bit hash value.
- SHA-512: Produces a 512-bit hash value, offering higher security.

iii. Brute-Force Attacks

- Definition: Trying all possible combinations of passwords until the correct one is found.

- Complexity: Highly dependent on the password length and complexity.
- Mitigation: Use of strong passwords and hash salting.
-

v. Ethical Considerations

- Legal Issues: Unauthorized use of password cracking tools is illegal.
- Responsible Use: Tools should only be used for educational purposes or with explicit permission.

4. Methodology

i. Programming Language

The tool is developed using Python, a high-level programming language known for its simplicity and extensive library support.

ii. Libraries Used

- hashlib: A standard Python library for secure hash and message digest algorithms.
- sys: Provides access to some variables used or maintained by the Python interpreter.
- os: Provides a way of using operating system-dependent functionality.
- time: Used for tracking the duration of operations.

5. Design and Architecture

The design and architecture of the password hash cracker tool are crucial for its functionality and performance. This section highlights three main components: the hash cracker function, the hash algorithm guessing function, and the menu/main function.

i. Hash Cracker Function

The hash_cracker function is the core component responsible for attempting to crack the hash. It iterates through each wordlist and hashes each password using the specified algorithm, comparing the result to the target hash. If a match is found, it returns the original password.

```
def hash_cracker(wordlists, hash_to_decrypt, hash_algorithm):
    total_passwords = 0
    for wordlist in wordlists:
        if os.path.isfile(wordlist):
            total_passwords += sum(1 for _ in open(wordlist))

    passwords_tried = 0
    for wordlist_path in wordlists:
        if not os.path.isfile(wordlist_path):
            continue

        with open(wordlist_path, 'r') as file:
            for line in file:
                password = line.strip()
                if hash_algorithm == 'md5':
                    hash_object = hashlib.md5(password.encode())
                elif hash_algorithm == 'sha1':
                    hash_object = hashlib.sha1(password.encode())
                elif hash_algorithm == 'sha256':
                    hash_object = hashlib.sha256(password.encode())
                elif hash_algorithm == 'sha512':
                    hash_object = hashlib.sha512(password.encode())

                hashed_word = hash_object.hexdigest()
                passwords_tried += 1

                if hashed_word == hash_to_decrypt:
                    return password

    return None
```

ii. Hash Algorithm Guessing Function

The guess_hash_algorithm function helps in identifying the hash algorithm based on the length of the hash string. This feature is useful when the hash algorithm is unknown.

```
def guess_hash_algorithm(hash_str):
    hash_length = len(hash_str)

    if hash_length == 32:
        return 'md5'
    elif hash_length == 40:
        return 'sha1'
    elif hash_length == 64:
        return 'sha256'
    elif hash_length == 128:
        return 'sha512'
    else:
        return 'Unknown or unsupported hash length'
```

iii. Menu and Main Function

The main function displays a menu to the user and processes input. It handles user choices, prompts for necessary information, calls the hash cracker function, and displays the results.

```
def main():
    attention_message = "⚠🔒🛑 Ethical use only, please. Not for unauthorized access. 🛑🔒🛑"
    print(attention_message)
    print_header()

    while True:
        print_menu()
        choice = input("Enter your choice: ")
```

```
        if choice == '1':
            hash_to_crack = input("Enter the hash to crack: ")
            wordlist_input = input("Enter wordlist file paths (comma-separated): ")
            wordlists = wordlist_input.split(',')
            hash_algorithm = input("Enter hash algorithm (md5, sha1, sha256, sha512) or leave blank to guess: ")

            if not hash_algorithm:
                hash_algorithm = guess_hash_algorithm(hash_to_crack)
                print(f"Guessed hash algorithm: {hash_algorithm}")

            print("Cracking password, please wait...")
            start_time = time.time()
            cracked_password = hash_cracker(wordlists, hash_to_crack, hash_algorithm)
            end_time = time.time()

            if cracked_password:
                print(f"\nPassword found: {cracked_password}")
            else:
                print("\nPassword not found in the provided wordlists.")

            print(f"Time taken: {end_time - start_time:.2f} seconds")
        elif choice == '2':
            print("Exiting...")
            break
        else:
            print("Invalid choice, please try again.")
```

This main function:

- Displays an ethical use warning and prints the header.

- Shows a menu and processes user input in a loop.
- Prompts for the hash, wordlist file paths, and hash algorithm.
- Calls the hash cracking function and displays the results.
- Measures and displays the time taken for the cracking process.
- Allows the user to exit the program.

6. Implementation

i. Setting Up the Environment

To set up the environment for running the password hash cracker tool, follow these steps:

1. **Install Python in kali**: sudo apt install python (OR) sudo apt install python3.
2. **Prepare Wordlists**: Obtain or create wordlists containing potential passwords. These wordlists should be in plain text files with one password per line.

ii. Running the Tool

To run the tool:

Navigate to the Project Directory: Open a terminal and navigate to the directory where the tool's Python script is located.

Execute the Script: Run the Python script using the command:

python hash_cracker_tool.py

Follow the Prompts: The tool will display a menu. Choose the option to crack a password, enter the required details (hash, wordlists, hash algorithm), and wait for the results.

7. Results and Analysis

i. Test Cases

To validate the functionality of the password hash cracker tool, various test cases were conducted with known hashes and wordlists. Examples of test cases include:

1. **MD5 Hash Test**: Hash of 'password123' using MD5.
   - Hash: **482c811da5d5b4bc6d497ffa98491e38**
   - Result: Successfully cracked using the wordlist.
2. **SHA-1 Hash Test**: Hash of 'securepassword' using SHA-1.
   - Hash: **d033e22ae348aeb5660fc2140aec35850c4da997**
   - Result: Successfully cracked using the wordlist.
3. **SHA-256 Hash Test**: Hash of 'strongpassword' using SHA-256.
   - Hash: **5e884898da28047151d0e56f8dc6292773603d0d6aabbddf92982abfbafefead**
   - Result: Successfully cracked using the wordlist.

4. **SHA-512 Hash Test**: Hash of 'strongpassword' using SHA-512:
    - Hash: 71946a78935b4f8b9bce8ddf6ebd9d4be15d693de353b0711868827a68f52e5f7e9ff8b88abf12ff7aaf9d3db13ec0cb49d3a0eab0e869eaf1c7d0d5673b1907
    - Result: Successfully cracked using the wordlist.

ii. Performance Metrics

The performance of the tool was measured in terms of time taken to crack passwords and the number of passwords tried per second. The metrics for different test cases were recorded and analyzed.

iii. Challenges Faced

Several challenges were encountered during the development and testing of the tool:

- **Large Wordlists**: Handling large wordlists efficiently to avoid memory and performance issues.
- **Hash Algorithm Detection**: Accurately guessing the hash algorithm based on the hash length.
- **Error Handling**: Robustly managing file operations and unexpected inputs.

iv. Ethical Considerations

Using password hash cracker tools raises several ethical considerations:

- **Legal Issues**: Unauthorized use of password cracking tools is illegal and can lead to severe consequences.
- **Responsible Use**: The tool should only be used for educational purposes or with explicit permission from the owner of the data.
- **Awareness**: Understanding the potential misuse of such tools and promoting ethical behavior in cybersecurity.

8. Conclusion

The Python-based password hash cracker tool successfully demonstrates the brute-force attack technique for recovering hashed passwords. It supports multiple hash algorithms and provides a user-friendly command-line interface. While effective, the tool also highlights the importance of strong passwords and ethical considerations in cybersecurity practices.

9. Real World Applications

- Ethical Hacking: Find weak passwords in security assessments.
- Cybersecurity Training: Teach attack and defense techniques.
- Incident Response: Recover passwords during security breaches.
- Security Awareness: Show why strong passwords are essential

10. References

1. **Python Documentation:** https://docs.python.org/3/
2. **Hashlib Documentation:** https://docs.python.org/3/library/hashlib.html
3. **Owasp:** https://owasp.org/www-community/attacks/Brute_force_attack

11. Appendices

i. Full Source Code

(Include full source code here : Click Here)