Hash Cracker Project

Spring 2024

Under the Supervision of:

Dr. Nashwa AbdelBaki

By:

Name	***ID***
Nadeen Mohamed Farid	202002561
Nadine Walid	202000607
Sama Ahmed Okasha	202000452
Nouran Hady Shaaban	202001903

Table of Contents

Table of Figures

1) Introduction

Hash Cracker is a based tool designed to automate the identification and cracking of various hash types. Supporting MD5, SHA1, SHA256, SHA384, and SHA512, the tool leverages multi-threading for efficient performance. Users can extract and crack hashes from files and directories Recursively, making Hash Cracker a versatile addition to the toolkit of security professionals and enthusiasts.

2) Project Description

Background: Hash functions are a cornerstone of modern cybersecurity, providing integrity and authenticity to data. However, malicious actors often exploit weak hashes, necessitating tools that can identify and crack these hashes for security audits and forensic investigations.

Objective: This project aims to develop an automatic hash type identification and cracking tool, named HashCracker, which simplifies the process of hash analysis and improves the efficiency of security assessments.

Scope: The tool supports five common hash types and offers features like file and directory hash extraction, multi-threading for performance optimization, and API integrations for hash lookups. While comprehensive, the tool focuses on general-purpose hash cracking and may not cover all advanced hash types or cryptographic scenarios.

2) Methodology

Overview:

The HashCracker web service is a Flask-based application designed to crack various types of hash values (MD5, SHA1, SHA256, SHA384, and SHA512) using multiple online APIs. The service supports single hash cracking, batch hash cracking from files, and hash extraction and cracking from entire directories.

Steps Involved

1. Importing Libraries:

   - Import necessary libraries for handling HTTP requests, file operations, regular expressions, concurrent execution, and Flask for building the web service.

2. Setting Up Flask Application:

- Initialize the Flask application:

```
app = Flask(__name__)
```

3. Defining Hash Cracking APIs:

- Define functions to interact with various online hash cracking APIs (alpha, beta, gamma, delta, theta):

```
def alpha(hashvalue, hashtype): return False
def beta(hashvalue, hashtype):
    # API interaction code
def gamma(hashvalue, hashtype):
    # API interaction code
def delta(hashvalue, hashtype): return False
def theta(hashvalue, hashtype):
    # API interaction code
```

4. Configuring Hash Types and API Functions:

- Map hash types to their corresponding API functions:

```
md5 = [gamma, alpha, beta, theta, delta]
sha1 = [alpha, beta, theta, delta]
sha256 = [alpha, beta, theta]
sha384 = [alpha, beta, theta]
```

sha512 = [alpha, beta, theta]

‘‘‘

5. Identifying Hash Types:

- Define a function to identify the hash type based on its length:

6. Hash Cracking Logic:

- Implement the ‘crack’ function to determine the hash type and attempt cracking using the defined API functions

7. Multi-threaded Hash Cracking:

- Define a function for multi-threaded hash cracking

8. Hash Extraction from Directory:

- Implement a function to search and extract hashes from all files within a directory

9. Batch Hash Cracking from File:

- Implement a function to extract hashes from a file and perform multi-threaded cracking

10. Flask Routes for API Endpoints:

- Define routes to handle API requests for single hash cracking, file-based hash cracking, and directory-based hash extraction and cracking

11. Running the Flask Application:

- Start the Flask application to listen for incoming requests:

4) Implementation

Programming Language: Python 3 was chosen for its simplicity, extensive library support, and strong community. Its built-in modules and third-party packages facilitate rapid development and robust performance.

Dependencies: Key dependencies include `requests` for API interactions, `re` for regular expressions, and `argparse` for command-line argument parsing. These libraries are standard in most Python environments and provide the necessary functionality for the tool.

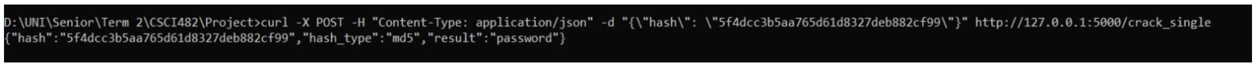
5) Conclusion

The HashCracker web service provides an efficient and versatile solution for cracking a variety of hash types using multiple online APIs. By leveraging the Flask framework, the service offers a user-friendly interface for performing single hash cracking, batch cracking from files, and directory-based hash extraction and cracking. The implementation of multi-threading ensures optimal performance, allowing the service to handle numerous hash values concurrently.

This methodology showcases a robust approach to hash cracking, incorporating various steps from identifying hash types to utilizing multiple APIs for enhanced success rates. The inclusion of regex-based hash extraction from directories and files further extends the service's functionality, making it a comprehensive tool for security researchers and enthusiasts.

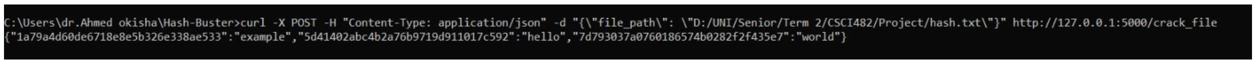
Overall, the HashCracker web service exemplifies a practical application of Python programming and web development in the domain of cybersecurity, demonstrating how various technologies and techniques can be integrated to solve real-world problems effectively. This project not only highlights the importance of efficient hash cracking but also serves as a foundational framework that can be expanded with additional features and improved API integrations in the future.

6) Results



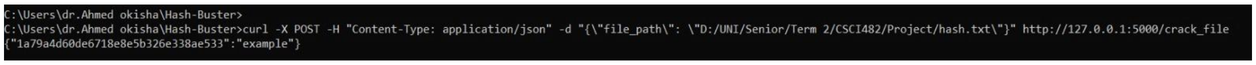
```
D:\UNI\Senior\Term 2\CSCI482\Project>curl -X POST -H "Content-Type: application/json" -d "{\"hash\": \"5f4dcc3b5aa765d61d8327deb882cf99\"}" http://127.0.0.1:5000/crack_single
{"hash":"5f4dcc3b5aa765d61d8327deb882cf99","hash_type":"md5","result":"password"}
```

Fig[1]



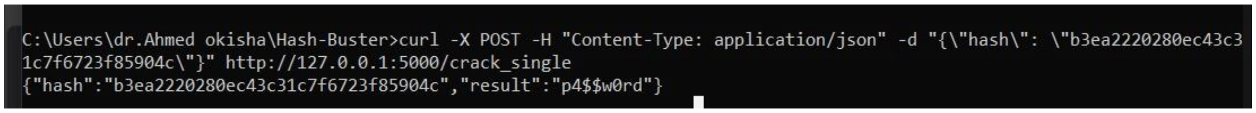
```
C:\Users\dr.Ahmed okisha\Hash-Buster>curl -X POST -H "Content-Type: application/json" -d "{\"file_path\": \"D:/UNI/Senior/Term 2/CSCI482/Project/hash.txt\"}" http://127.0.0.1:5000/crack_file
{"1a79a4d60de6718e8e5b326e338ae533":"example","5d41402abc4b2a76b9719d911017c592":"hello","7d793037a0760186574b0282f2f435e7":"world"}
```

Fig[2]



```
C:\Users\dr.Ahmed okisha\Hash-Buster>
C:\Users\dr.Ahmed okisha\Hash-Buster>curl -X POST -H "Content-Type: application/json" -d "{\"file_path\": \"D:/UNI/Senior/Term 2/CSCI482/Project/hash.txt\"}" http://127.0.0.1:5000/crack_file
{"1a79a4d60de6718e8e5b326e338ae533":"example"}
```

Fig[3]



```
C:\Users\dr.Ahmed okisha\Hash-Buster>curl -X POST -H "Content-Type: application/json" -d "{\"hash\": \"b3ea2220280ec43c3
1c7f6723f85904c\"}" http://127.0.0.1:5000/crack_single
{"hash":"b3ea2220280ec43c31c7f6723f85904c","result":"p4$$w0rd"}
```

Fig[4]

7) References

Tatlı, E. I. (2015). Cracking more password hashes with patterns. *IEEE Transactions on Information Forensics and Security*, *10*(8), 1656-1665.

Zonenberg, A. (2009). Distributed hash cracker: A cross-platform gpu-accelerated password recovery system. *Rensselaer Polytechnic Institute*, *27*(395-399), 42.

https://github.com/s0md3v/Hash-Buster.