**Security dependability report**

The development of my system consisted of implementing and creating two password cracking tools to find a password with the use of its hash code. The two tools I created within my system is a dictionary attack and a brute force attack. I chose these attacks as I felt it would be a challenge to incorporate as well as I find the idea and design of them interesting to replicate. Additionally, these methods are more commonly used by attackers due to their effectiveness.

Diagram

Description automatically generatedMy system is broken down into three classes.

* **Main**: Parent Class holds user interaction and MD5 hash Function.
* **DicitionaryAttack**: Holds Dictionary attack algorithm.
* **BruteForceAttack**: Holds Brute force attack algorithm.

A UML diagram is provided to briefly visualize the design and components (**Figure 1**).

**Basic inputs and outputs**

Both tools each uses two text files as input and one for output (**Figure 2**).

Graphical user interface, text, application, chat or text message

Description automatically generatedDictionary attack algorithm text files:

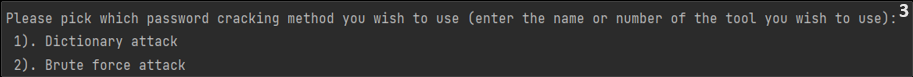
* **Dictionary.txt**: Contains plaintext words, used as input to provide functionality to cracking passwords.
* **DictionaryHashcode.txt**: Contains the hash codes of the elements from the Dicitionary.txt, used as input to generate ten random hash codes to be tested.
* **DictionaryOutput**.txt: Stores all words tested in plaintext with their hash codes.

Brute force algorithm text files:

* **CommonPasswords.txt**: Contains plaintext passwords commonly used. Used as input to create hash codes for CommonPasswordsHashcodes.txt.
* **CommonPasswordsHashcodes.txt**: Contains hash codes for the passwords in CommonPasswords.txt. Used as input to generate a random hash code for testing.
* **BruteForceOutput.txt**: Stores all strings tested in plaintext with their hash code.

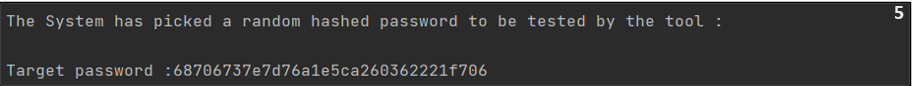
Additionally, the brute force algorithm uses an array called dataset as input which contains the characters used to create the strings that are tested. Once the target passwords have been found they will be printed to the console as output in plaintext along with their hash codes.

**How my system works?**

The System lets the user decide which attack method to use with console interaction (**Figure 3**).

Text

Description automatically generatedPicking the dictionary attack will crack ten random passwords with the use of their hash codes. These will be randomly selected from DictionaryHashcode.txt (**Figure 4**).

Picking the brute force attack will only crack one password with its hash code that is randomly selected from CommonPasswordsHashcodes.txt. Due to the time brute force attacks take, my system will only crack one target password per execution. As one password could potentially take seconds where another may take hours. The image below shows the hashed target password being tested (**figure 5**).

Graphical user interface, text

Description automatically generatedDuring execution both tools will display in console the iteration of strings tested against the password (**figure 6** & **7**). Additionally, once either attack has finished it will print all the iterated words and or strings to their own output text files.

**Real world problems this system solve?**

Two real world problems that either tool can be used for:

1. **Hacking**: To gain access to a device or account by uncovering the target systems password. This can be done from the perspective of a black hat hacker where no authorization has been given and is performed for personal gain or performed in a controlled environment where authorization has been given for penetration testing purposes.
2. **Encryption key cracking**: Encryption keys are used within a cipher to decrypt hashed strings or encrypted data. These Keys are made up of bits (1’s and 0’s) which determine the correct output. A brute force attack can be used to uncover the correct key for decryption of data.

Text

Description automatically generated**Details regarding algorithms**

Below are some important details implemented that helps my system function. Each image are different details I have introduced within my code. With the use of in code comments I have explained how each one functions. (**Figure 8**) shows the import libraries I used for development.

Text

Description automatically generatedAn important aspect is the use of arrays in both algorithms. These are used contribute to the iteration and the generation of passwords for testing (**Figures 9 & 10**).

**Text

Description automatically generated**This segment of code is the MD5 hash function found in class Main. This is used in hashing the plaintext files of either tool to create their hash code text files (**Figures 11**).

Text

Description automatically generatedThis block of code is the brute force iterator this creates different strings with the iteration of characters from the data set array (**Figure 12**).

Text

Description automatically generatedThis is the Dictionary attack iterator which compares every element within the plaintext text file hashed to the hashed passwords generated (**Figure 13**).

**Results and Successfulness**

I have measured the successfulness of my system by testing both tools and their ability to crack passwords. Due to their differences, I have had to apply different methods of testing. The successfulness of the brute force algorithm was determined by testing different target passwords which introduce different levels of security:

3

2

1

1. Basic lower-case password with a number (cat1) (**Figure 14**).
2. Capital and lower-case letters with a number (CaTs1)(**Figure 15**).
3. All possible characters (C@t$1)(**Figure 16**).

Text

Description automatically generatedDue to my hardware, I am limited to only test short passwords. Anything greater than five letters will take hours. The passwords tested demonstrates that my system is capable of cracking different security levelled passwords.

Text

Description automatically generatedTo confirm the hash code of the tested passwords I used an online MD5 decipher which confirms that they are correct (**Figure 17**). Site : [Md5 Decrypt & Encrypt - More than 15.000.000.000 hashes](https://md5decrypt.net/en/#answer).

**17**

Graphical user interface, text

Description automatically generatedThe successfulness of a dictionary attack is determined by its input data. The algorithm may adjust the speed it performs at however, if a password is not found it is because of the data set used. I tested the attack using different target passwords, each one was cracked successfully. Below are the results (**Figure 18**). Overall, 30 passwords were found by the dictionary attack which gives my tool a 100% success rate. If the dictionary attack fails an error warning is displayed next to the unfound hashed password which says to edit the data set to include more words (**Figure 19**).

I made sure that the hash codes were correct by comparing some to an online MD5 decipher (**Figure 20**) the hash codes match the plaintext word.

Graphical user interface, text, application, chat or text message

Description automatically generated

With the results from both tools, I can confidently conclude that my system as a whole is reliable at finding hashed passwords effectively with no problems. The only limitation my system faces is the lack of powerful hardware which prevents me from performing more intensive tests on my brute force algorithm. However, the brute force is capable of finding 10-character long passwords in given time.