Rebecca Donohoe
CSCI 3010
10/02/18

Individual Project Proposal

1. Project Introduction

The goal of this project is to design a password cracker that will attempt to gain access to a password protected system. This password cracker is intended solely for testing the security of a system that requires password authentication to determine if there are any potential vulnerabilities to the overall security of the system. This tool should only be used on systems where one has been given explicit permission to use it on a system by the owner of that system. It is not intended for illegal purposes or malicious intents.

This program will focus on the cracking of passwords designed using hashing algorithms. It will be designed to implement at least two of the three most common password-cracking attacks; a brute-force attack and a dictionary attack. If time allows, after the successful implementation of a brute-force attack and a dictionary attack, the implementation of a rainbow table attack would be added. The program will be designed to attempt to crack the MD5 message-digest algorithm and the secure hash algorithms (SHA1 and SHA2). **This program will be implemented in C++.**

2. Technologies That Will Be Used`

Table 1. Technologies that will be used to aid in the generation of wordlist and dictionaries

Technologies	Familiarity	Purpose
Crunch	Somewhat familiar	Tool used to generate a wordlist
PAX	Not familiar	Tool used to create a dictionary or wordlist with simple passwords
Cupp	Not familiar	Tool used to create wordlist, with a focus on specific target
Pydictor	Not familiar	Tool that creates wordlist with both normal words and with base64 encryption
Dymerge	Not familiar	Merges previously made dictionaries into one dictionary

Table 2. Technologies specific to C++

Technologies	Familiarity	Purpose
C++	Familiar	Main Program
Crypto++	Not familiar	Allows use of cryptographic schemes
Boost C++ Libraries	Not familiar	Extends the standard library
LibTomcrypt	Not familiar	Comprehensive, modular and portable cryptographic toolkit that provides access to well-known block ciphers, one-way hash functions, chaining modes, pseudo-random number generators and public key cryptography
Botan	Not familiar	A C++ cryptography library
RxTerm	Not familiar	A C++ library for functional-reactive terminals
TUI Library	Not familiar	A terminal user interface library built on top of ncurses, that allows for focus on user interface design

These tables are not necessarily all inclusive as it may be discovered through the coding process that another library or technology might be needed, and many of the libraries listed here replicate each other, and only one may be needed rather than having multiple overlapping libraries. Including these extra resources helps to ensure that there are several options to get this program successfully working and leaves several backup options if one ends up not working as intended or necessary.

3. Outside Resources

This program will not require much in the way of outside resources. Most of the resources this program requires to work is in the form of the wordlist and dictionaries as discussed in the technology section above. Without these resources, this program would not work as all the password cracking methods depend on these wordlist and dictionaries to crack a password and as a result there is not a backup plan if this does not work out. The best way of supporting this issue is to have multiple options on how to create these wordlists and dictionaries, with many of these options listed in Table 1 in section 2. Having multiple options on how to create these will allow some fallback options in case one of the resources doesn't work as expected.

The other resources that will be needed for this program are not directly for the program's use, but rather resources that can be used throughout the creation of the program as reference or learning material for the programmer. Without these valuable resources, it is very likely this program will not work at all, and there would not be any other plan for this as access to such resources is essential for the creation of this program. Currently, several resources of this type have been obtained for use alongside the creation of this program. A brief list of these resources that have been obtained and granted access to include:

- Books
    - Hacking: The Art of Exploitation
    - Serious Cryptography A Practical Introduction to Modern Encryption

- Introduction to Algorithms
- Black Hat Python
- Object Oriented and Classical Software Engineering
- Secure Programming Cookbook

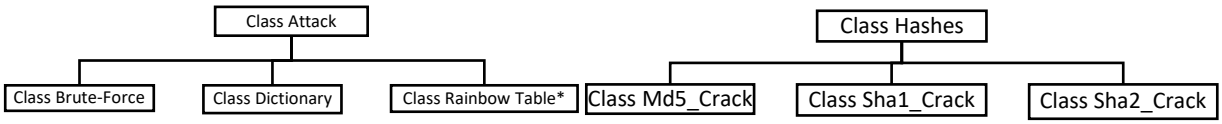
4. Architecture

Throughout the length of this project, all work on it will be maintained in a Github repository to aid in version control and continuous integration. This program will have a back end with the user interface being implemented as a text UI. While a GUI has its advantages for ease of use, this program is intended for use by people who understand running programs from a terminal-based interface, thus the need for an ease of use interface is eliminated. Additionally, since this program could easily be misused outside of its intended purpose for security/penetration testing, reducing the ease of use will help maintain the intents of this program and prevent misuse. Furthermore, since this program is intended to be used in cases such as penetration testing, many tools utilized for this purpose, are implemented in a similar format with the user interface being text UI based.

Since the user interface will be implemented through a text UI, this program does not need to have a front end. The idea for the design of the text UI is to display the name of the program, information about the program (i.e., version number, usage/disclaimer about proper usage and description about what the program does), and a menu where the user can select an option for what they would like to do. This menu will have the option for the user to view more information about what each option does, and then they will be able to go back to the main menu and select the option they would like to perform. The menu will likely be set up using switch statements, where depending on the user's choice, it will utilize the objects appropriate for the case. For example, if the user wants to perform a brute force attack on a SHA1 hash, selecting this option would construct the appropriate object(s) to interact with the classes that will perform this (i.e., the class for brute force attack and the class for SHA1). The UI will display feedback throughout the process telling the user what it is doing (i.e., if it is working), and provide the results of the program back to the user. There are some C++ libraries that can be used to help create a more interactive terminal/text-based UI, which have been listed in Table 2 above. Additionally, while the program is running, logging of the processes such as time taken, the steps its taking throughout the process or errors that occur as this information can be useful in helping figure out errors in the program itself, or why exactly a hash might not be able to be broken. Travis CI will be utilized as the testing framework for this program and will also aid in the continuous integration and version control provided through Github.

This program will have at least two major classes; brute-force attack (i.e., BruteForceAttack) and dictionary attack (DictionaryAttack). There will be additional classes that will inherit from both of these classes; an md5 class, a sha1 class, and a sha2 class. The sha2 class may be further broken down into two separate classes since the SHA2 algorithm consists of six different hash functions. For this program, the intent is to focus on two of these six hash functions, which is why the sha2 class may be split further into sha256 class and a sha512 class. The base classes will be responsible for the different types of attacks, and their derived classes will responsible for the information about a specific type of hash function that the attacks can be used on. **This program intends to utilize the Iterator design pattern. Additionally, this program will utilize inheritance amongst it's various classes. Since this program is going to heavily rely on inheritance, the utilization of a second design pattern is not beneficial to this program. While initially,**

the Factory method was chosen because of the ability to let subclasses decide which class to instantiate which would have helped in adding the ability to have the specific hash type class decide which base class it wants to use based off what attack is being performed, it was decided that inheritance would be a more appropriate way of implementing the classes in this function rather than using the factory method. The plan for the class inheritance is as follows:



*****If rainbow table ends up being added to the program
The two separate classes; Attack and Hashes would likely connect to each other either through the user interface class, or another class designed merely for this purpose.

The Iterator design pattern was chosen because of its ability to access all elements sequentially in many different containers, which will be useful when the attacks are needing to move through the wordlists or dictionaries while attempting to crack the password. **Since the wordlist and the dictionaries will likely be quite large, and will need to be searched throughout the process of the program, so having the ability to be able to store the contents of these lists into a container and iterate through them or having the ability to write custom traversals for these containers will likely help to increase the speed of the overall program as it has to search through these lists and will allow for larger list, thus helping to increase the likelihood that this program will be able to successfully crack a password.**

5. Detailed Plan

Week	Plan	Things needed to accomplish plan	Things needed to learn to accomplish plan	What will be turned in
Oct 7	Setup GitHub Repository; set up **Travis CI**; begin creation of wordlist, hash table(s); continue researching hashing algorithms	Wordlist and dictionary generator tools as described in section 2	Need to gain a better understanding in the specifics behind md5, SHA1 and SHA2 Hashes; **get a better understanding of how Travis CI works and how to use it correctly**	Nothing to turn in. Goal for the end of the week is to have the wordlist generated.
Oct 14	Begin writing the brute-force attack class; begin implementation of brute-force attack	Nothing needed aside from materials to learn from	Need to understand how a brute force attack works specifically	The files containing the wordlists, and any functions written or used to create them. Any other parts of program started. **Due on Oct.17**
Oct 21	Continue implementation of brute-force class; **writing test cases for**	Library documentation; found online	Need to understand how to use the hashing libraries and how to	Nothing to turn in. Goal for the end of the week is to have brute-force class

	brute-force class and testing with these cases		implement them into the code	finished and implemented **and have begun testing**
Oct 28	**Finish any testing/working on correcting any failing test cases**; begin writing of the dictionary attack class	Nothing needed aside from materials to learn from	Need to understand how a dictionary attack works specifically	The files containing the brute-force attack implementation; **the test cases and results/solutions to them** and anything started for the dictionary attack class **Due on Oct.31**
Nov 4	Continue implementation of dictionary attack class; **writing test cases for dictionary attack class and testing with these cases**	Library documentation; found online	Need to understand how to use the hashing libraries and how to implement them into the code	Nothing to turn in. Goal for the end of the week is to have the dictionary attack class finished and implemented **and have begun testing.**
Nov 11	**Finishing testing/correcting any failing test cases**; begin writing of the derived classes (md5, sha1, sha2)	Nothing needed aside from materials to learn from	Need to understand how the hashing of md5, SHA1 and SHA2 work	The files containing the dictionary attack class, **the test cases and results/solutions to them** and anything started for the derived classes **Due on Nov.14**
Nov 18*	**Designing user interface**	Library documentation; found online	**Need to understand any libraries using for designing a UI**	Nothing to turn in. Goal for the end of the week is to **have the user interface mostly finished**
Nov 25	**Finishing writing of the derived classes (md5, sh1, sha2); writing test cases for derived classes and begin testing with these cases**	Documentation for testing framework; found online	Any unresolved questions about the hash functions; how the hashing libraries work with these functions	Nothing to turn in. Goal for the end of the week is to **have derived classes finished and implements and have begun testing on them**
Dec 2	**Finishing user interface; finishing up any remaining tasks from prior week (most likely any issues that need resolving from testing)**		**Any questions or problems that haven't been resolved at this point**	Nothing to turn in. **Goal for the end of the week is to have program mostly finished**
Dec 9	Finalizing any other details of the program, making sure the project is complete.	Should already have the things needed to accomplish the goals for this week.	N/A	Finished Project **Due on Dec. 12**

*The week of Nov. 18 is Thanksgiving week and realistically, spending a lot of time working on this is likely not going to happen which is why the plan for this week is a bit lighter