Computer Security
Dept. of Computer Engineering,
Chulalongkorn University.

Activity I : Hacking Password

Created by : Krerk Piromsopa, Ph.D

Objectives

To understand the concepts of hashing and salting.

Overviews

This activity demonstrates the fundamentals of password security. Several hacking techniques will be demonstrated throughout the exercises. In particular, we will learn: brute-force attack, rainbow-table attack, and password analysis.

We will use a free password dictionary from the given url as our dictionary.
https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10k-most-common.txt

Prerequisite

Please prepare a computer with Python installed. Please also install hashlib and bcrypt (e.g pip install hashlib bcrypt).

Here is a sample code that might be useful in this activity.

```
import hashlib
import bcrypt
# SHA1
m=hashlib.sha1(b"Chulalongkorn").hexdigest()
print(m)
# MD5
m=hashlib.md5(b"Chulalongkorn").hexdigest()
print(m)
# BCRYPT
salt = bcrypt.gensalt()
m=bcrypt.hashpw(b"Chulalongkorn", salt).hexdigest()
print(m)
```

Exercises

1. **Objective:** Understand how attackers use pre-built word lists (dictionaries) to crack hashes of common passwords.

   Scenario: You have discovered a SHA-1 hash in a compromised system:
   d54cc1fe76f5186380a0939d2fc1723c44e8a5f7.

   You suspect the password is a simple, common word, possibly with some character substitutions.

   Task: Write a Python program that reads a list of words, applies common substitutions, hashes the result, and checks if it matches the target hash.
   Note that you might want to include substitution in your code (lowercase, uppercase, number for letter [‘o’ => 0 , ‘l’ => 1, ‘i’ => 1]).

2. **Objective:** To understand *why* modern password hashing algorithms like `bcrypt` are more secure than older ones like `MD5` and `SHA-1`.

   Task: Design and run an experiment to measure how many hashes each algorithm can compute in a fixed amount of time. The code must test atleast **MD5** , **SHA-1** , and **bcrypt** .
   (You may also try additional algorithms like SHA256, SHA512, scrypt, Argon2.)

   Hint: Use time function in python.

3. **Objective:** To apply the performance measurements to understand the importance of password length and algorithm choice.

   Task: Based on the measurements from Exercise 2, estimate how long it would take an attacker to brute-force a password of a given length
   You may assume that the password contains only upper-case, lower-case, numbers and symbols.

   What does it suggest about the length of a proper password. (ie. Use more than a year to brute force.)

4. If a given hash value is from a bcrypt algorithm, is it practical to do a brute-force attack?

5. If a given hash value is from a bcrypt algorithm, is it practical to perform a rainbow table attack?

6. You have to store a password in a database. Please explain your design/strategy for securely storing it.
   (Hints: Proper hash function, Salting, Cost factor, Database Security)