Computer Security
Dept. of Computer Engineering, Chulalongkorn
University.

Activity I : Hacking Password

Created by : Krerk Piromsopa, Ph.D

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

Exercises

1. Write a simple python program to use the word from the dictionary to find the original value of **d54cc1fe76f5186380a0939d2fc1723c44e8a5f7**. Note that you might want to include substitution in your code (lowercase, uppercase, number for letter [‘o’ => 0 , ‘l’ => 1, ‘i’ => 1]). Hint: Here is a snippet for sha1 and md5 functions.

```
import hashlib
m=hashlib.sha1(b"Chulalongkorn").hexdigest()
print(m)
m=hashlib.md5(b"Chulalongkorn").hexdigest()
print(m)
```

2. For the given dictionary, create a rainbow table (including the substituted strings) using the sha1 algorithm. Measure the time for creating such a table. Measure the size of the table.
3. Based on your code, how long does it take to perform a hash (sha1) on a password string? Please analyze the performance of your system.
4. If you were a hacker obtaining a password file from a system, estimate how long it takes to break a password with brute force using your computer.
(Please based the answer on your measurement from exercise #3.)
5. Base on your analysis in exercise #4, what should be the proper length of a password. (e.g. Take at least a year to break).
6. What is salt? Please explain its role in protecting a password hash.

Peerawit Chariyawongsiri

6238151921

1. The target hash “d54cc1fe76f5186380a0939d2fc1723c44e8a5f7” decryption result is the word “ThaiLanD” by using SHA1 function.

```
❯ python3 bruteForceCrack.py
The original value is: ThaiLanD
The runtime is 0.8836004734039307 seconds.
```

Code:

```python
import hashlib
import itertools
import time

def brute_force_attack(dictionary_file, target_hash):
    with open(dictionary_file) as f:
        words = f.read().splitlines()
    for word in words:
        # Create all possible combinations of characters and number substitutions
        all_combinations = [''.join(combo) for combo in itertools.product(*((c.upper(), c.lower(), '0' if c == 'o' else '1' if c in 'il' else c) for c in word))]
        for c in all_combinations:
            # Hash the combination
            hashed_word = hashlib.sha1(c.encode()).hexdigest()
            # Compare the resulting hash value with the target hash value
            if hashed_word == target_hash:
                return c
    return None

# Example usage
start_time = time.time()
target_hash = 'd54cc1fe76f5186380a0939d2fc1723c44e8a5f7'
original_value = brute_force_attack('dictionary.txt', target_hash)
end_time = time.time()

if original_value:
    print(f'The original value is: {original_value}')
else:
    print('The original value could not be found in the dictionary.')
print(f'The runtime is {end_time - start_time} seconds.')
```

2.

Output:

```
❯ python3 rainbowTableCrack.py
The tablesize is 2650956.
The runtime is 3.890115261077881 seconds.
The file size is 145.966 MB
```

Code:

```
import hashlib
import itertools
import json
import time
from concurrent.futures import ThreadPoolExecutor
import os

def get_combination_character(letter):
    characters = [letter.upper(), letter.lower()]
    if letter in 'il':
        characters.append('1')
        return characters
    elif letter in 'o':
        characters.append('0')
        return characters
    else:
        return characters

def create_rainbow_table_part(words):
    rainbow_table = {}
    for word in words:
        # Create a copy of the word to make modifications
        modified_word = word
        # Create all possible combinations of characters and number substitutions
        all_combinations = [''.join(combo) for combo in itertools.product(*((get_combination_character(c)) for c in modified_word))]
        for c in all_combinations:
            # Hash the combination
            hashed_word = hashlib.sha1(c.encode()).hexdigest()
            # Add the combination and its hash to the dictionary
            rainbow_table[hashed_word] = c
    return rainbow_table

def create_rainbow_table(dictionary_file):
    rainbow_table = {}
    with open(dictionary_file) as f:
        words = f.read().splitlines()
    with ThreadPoolExecutor() as executor:
        # Divide the words into chunks and submit each chunk to a separate process
        chunks = [words[i:i+10] for i in range(0, len(words), 10)]
        results = [executor.submit(create_rainbow_table_part, chunk) for chunk in chunks]
        for future in results:
            rainbow_table.update(future.result())
    print(f'The tablesize is {len(rainbow_table)}.')
    return rainbow_table

def write_to_file(rainbow_table, filename):
    with open(filename, 'w') as f:
        json.dump(rainbow_table, f)

def find_original_value(hash_value, rainbow_table_file):
    with open(rainbow_table_file) as f:
        rainbow_table = json.load(f)
    return rainbow_table.get(hash_value)
```

```python
def get_size(file_path, unit='bytes'):
    file_size = os.path.getsize(file_path)
    exponents_map = {'bytes': 0, 'kb': 1, 'mb': 2, 'gb': 3}
    if unit not in exponents_map:
        raise ValueError("Must select from \
        ['bytes', 'kb', 'mb', 'gb']")
    else:
        size = file_size / 1024 ** exponents_map[unit]
        return round(size, 3)

# Example Usage
start_time = time.time()
rainbow_table = create_rainbow_table('dictionary.txt')
with ThreadPoolExecutor() as executor:
    executor.submit(write_to_file, rainbow_table, 'rainbow_table.json')

hash_value = 'd54cc1fe76f5186380a0939d2fc1723c44e8a5f7'
original_value = find_original_value(hash_value, 'rainbow_table.json')
end_time = time.time()

if original_value:
    print(f'The original value is: {original_value}')
else:
    print('The original value could not be found in the rainbow table.')
print(f'The runtime is {end_time - start_time} seconds.')
filePath = './rainbow_table.json'
fileSize = get_size(filePath, 'mb')
print(f'The file size is {fileSize} MB')
```

Part of Rainbow Table:

3. Rainbow Table: 2650956 hashes

   Time taken: 3.890115261077881 seconds

   The hash rate would be 681,459.4998055322 hashes per second. To perform hashing with SHA1 it would average to 1.4674386376378 49 microseconds.

4. Average password length would be between 8 to 11 characters. Let me assume that the password that I am cracking has a length of 8 characters. The requirements of a password are uppercase letters, lowercase letters, numbers and symbols(~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/). Therefore, it will have 94^8 combinations which is 6,095,689,385,410,816. Which the total time to hash will average to 8,945,050,127.19074489489269557478 4 seconds and can be converted to 283.6 years.

5. 7-characters password length will have the crack time closest to 1 year. The password will have 94^7 combinations which is 64,847,759,419,264. To crack the password it will take 95,160,107.73607175420098612313 6 seconds which equivalent to approximately 3 years.

6. Salt is a random string of data that is added to a password before it is hashed. The purpose of a salt is to protect against dictionary attacks and precomputed rainbow table attacks.

   When a password is hashed without a salt, an attacker could precompute the hash of every possible password and store them in a "rainbow table." This would allow the attacker to quickly lookup the hash of a given password and determine if it matches the hash of a stored password.

   By adding a unique salt to each password before it is hashed, the attacker would need to precompute a separate rainbow table for each salt, making the attack infeasible. Additionally, it makes the same password hashed with a different salt, resulting in a different hash, making it difficult for an attacker to use a precomputed list of common password hashes.