Password Cracking Lab

Introduction

The goal of this lab is to familiarize you with password storage and cracking techniques using dictionary and brute-force attacks. You will create password files with different hashing algorithms, write scripts to perform dictionary attacks, and compare the performance of different hash functions.

Note: This lab can be performed using any Unix/Linux shell.

Task 1: Exploring Password Files

Objective:
Learn how passwords are stored in Linux and understand the difference between /etc/passwd and /etc/shadow.

Instructions:

1. View the /etc/passwd File:
This file lists all users on the system, but it does not store password hashes.

```
    more /etc/passwd
```

Observe that each line represents a user, but no passwords are visible.

2. Attempt to View the /etc/shadow File:
This file contains hashed passwords, but it is protected.

```
    more /etc/shadow
```

You will receive an error like `Permission denied`. Record this error.

3. View the /etc/shadow File with Root Privileges:
Use `sudo` to view the file:

```
   sudo more /etc/shadow
```

After entering your password, you will see entries that include username, hashed password, and other details like password expiration.

Task 2: Dictionary Attacks

Objective:
You will write a Python script to crack passwords using a dictionary attack. This task involves reading a password file and matching hashed passwords with hashed words from a dictionary.

Part 1: Creating the Password File

1. Install the htpasswd Tool:
If you don't have htpasswd installed, install it using:

```
sudo apt-get install apache2-utils
```

Mac users can install the httpd package using homebrew:

```
brew install httpd
```

2. Create a Password File Using SHA1:
Use the following command to create a password file with SHA1 hashing:

```
htpasswd -sc htpasswd-sha1 alice
```

This command will create a file named htpasswd-sha1 and prompt you to enter a password for the user `alice`. (enter simple password that is easy to crack)
Add more users without the `-c` flag (create flag):

```
htpasswd -s htpasswd-sha1 bob
htpasswd -s htpasswd-sha1 charlie
```

3. View the Password File using the cat command.
The file will look something like this:

```
alice:{SHA}hashedSHA1password
bob:{SHA}hashedSHA1password
charlie:{SHA}hashedSHA1password
```

Part 2: Writing a Python Script for Dictionary Attack

1. Write the Script:
The script should read the hashed passwords and attempt to crack them by comparing the hash values of dictionary words (from files below) with the stored password hashes.

2. Download Dictionary Files. Links below are just some samples for dictionary files. You can use any other dictionary files available.:

- Tinylist (small dictionary with common passwords): [Tinylist](https://github.com/danielmiessler/SecLists/blob/master/Passwords/darkweb2017-top1000.txt)
- Biglist (larger dictionary with millions of passwords): [Biglist (RockYou)](https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt)

3. Run the Script:
Attempt cracking the passwords using the tinylist. If it couldn't be cracked use the biglist instead.:

4. Record the Results:
- Record the passwords that were cracked
- The time taken to crack each password.

Task 3: Comparing Hash Function Performance

Objective: Modify your dictionary attack script to support multiple hash functions (MD5, SHA-256, SHA-512) and compare the performance of each hash algorithm.

Part 1: Modifying the Script for Multiple Hash Functions

1. Support for MD5, SHA1, and SHA512:
Modify the script to hash dictionary words using different algorithms.

2. Test on a password file containing different hash functions:

- MD5: Use `htpasswd -cm htpasswd-md5 alice` command. The -c flag creates a new file. The -m flag specifies MD5 hashing.
- SHA1: Use `htpasswd -s htpasswd-sha1 alice`
- SHA512: Use `openssl passwd -6` to generate a SHA256 hash. This will prompt you to enter a password and return the SHA256 hashed password. You can manually add this to the password file, like this:
  alice:$5$salt$hashedSHA256password

3. Measure the Time Taken:
Measure how long it takes to crack the passwords for each hash type (you can use Python's `time` module for this purpose)

4. Record the Results:
For each hash type, record:
- Record the passwords that were cracked
- The time taken to crack each password.
- The number of dictionary words attempted to crack each password.
- An analysis comparing the effectiveness of different hash algorithms (MD5, SHA1, SHA512).

Submission:

Submit a PDF Report containing:

First task: Screenshots of each command that was used, and a short description of the

Second and Third Task: Screenshot of your script, and the results which were asked in each task.