GENI MINI PROJECT
MD5 PASSWORD CRACKER

FULL REPORT

December 10, 2020

Meghna Sengupta
CS655 Computer Networks
Boston University

INTRODUCTION

Passwords are an indispensable part of our daily lives - from the major communication means we use these days (email accounts) to shopping for bare necessities (online retail), we constantly need to validate our identities online for security. Passwords are one of the most fundamental mechanisms used for this purpose.

Almost all password protected systems use hashes of passwords to store them in their databases these days. MD5 is one such hashing algorithm that generates 128-bit message digests. Even though vulnerabilities have been discovered in the security of the algorithm, it is still widely used all over the internet.

In this project, we aim to design a way to crack an MD5 password hash by utilizing a distributed system in GENI. The system is going to have a central server which will also have a web interface using which a user will be able to submit an MD5 hash. We assume, for the scope of this project, that the hash is that of a 5-character password that only consists of alphabets (either lowercase or upercase). The user inputs this message digest to the system using a web interface. The web interface with the help of worker nodes cracks the password by a brute force approach. This means that the system, using the worker nodes, calculates the hash of all possible passwords and checks the given password against each possibility, until a match is found.

We will be implementing the web-interface as well as the management service that will dedicate jobs to worker nodes. The web-interface and the management service are going to be on the same central machine and the worker nodes are going to be on different machines. We will be aiming to make the systems scalable - to add/remove workers on the fly according to the requirements. In order for the server to communicate with the worker nodes and vice versa, we will be using socket programming.

EXPERIMENTAL METHODOLOGY

The Resource Specification

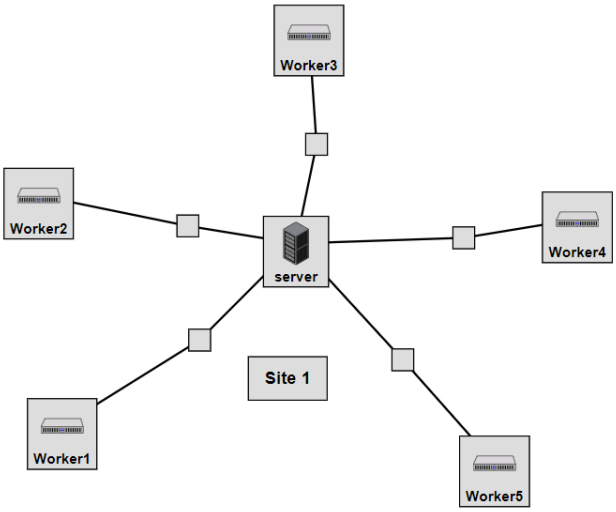

Figure 1: RSpec in GENI Slice

The password cracker will function as a distributed system with one central server. The server will communicate with 5 worker nodes that will be doing all the MD5 hash computations. We explain the function of these two components in detail in the next two sections.

All the nodes are taken to be "deafult-vm" and the Site chosen is the "Colorado InstaGeni". Since we build a web interface for the Server, we require the server VM to be publicly routable.

The server code and the client codes are not platform-specific. They are python codes and have been tested on both Windows (Windows 10) and Linux (Ubuntu 18.04). The codes can be run on any computer having Python (tested with Python3).

The Server

- Firstly, the server will be running the web interface with which the user will interact with the system. This web interface will provide a way for the user to submit a hash digest to the system. The system will then search for the password that matches this hash and if found, will return the password as an output on the web interface. If the password is not found in the search space, an error will be returned.

- The main function of the server, however, will be to act as the brain of the system and manage the work that needs to be done by each of the worker nodes.

In the scope of this project, we assume that the hash digest submitted by the user corresponds to a 5-character password made up of either lowercase or uppercase characters. This results in the password space having the size of $(2 \times 26)^5 = 380204032$ possible passwords.

We use an integer-to-password function that converts any integer in the range 0 to 380204031 into a 5-letter string consisting of lowercase and uppercase alphabets only. In our program, we use the the following mapping of digits - $a = 0$, $b = 1$, ..., $z = 25$, $A = 26$, ..., $Z = 51$.

The server identifies and assigns search space ranges (from password a to password b) of a specific size (say n) to each of the worker nodes. If the worker does not find the password in the given search space, the server allocates a new search space to that worker. If any of the workers find the matching password, the server immediately stops the search and outputs the password to the user.

In the worst case scenario, the server will need to allocate search spaces $\frac{380204032}{n}$ number of times to some worker node. We have not yet decided on the value of n that will be optimal for this project. As an example, if the value of n is taken to be 100000, there might need to be $\frac{380204032}{100000} = 3803$ worker node search sessions in the worst case scenario.

The Workers

Each worker node communicates with the server using an exclusive link. The server sends each worker a range of passwords that will constitute as the search space for that worker node in that search session.

Once the worker node receives the search space from the server, it computes the MD5 hashes of each password in the delegated search space. If any one of these computed hashes match the input digest, the worker node sends back a message to the server saying the password has been found along with the password itself. If it goes through the entire search space and does not find a match for the input digest, it sends back a message to the server saying that the password has not been found in that particular search space.

If computing the MD5 hash digest of one password takes t time and the search space consists of n passwords, then each search session in a worker node is going to take tn time. Practically, t should be of the order of some micro seconds. So, taking n to be something like 10^6 should be achievable. However, we need to do more experiments in order to arrive at an optimal value.

The Web Interface

We will be using the Apache HTTP Server in order to build the web interface at the central server of our system. We will be making use of the CGI-bin folder of the server to store the scripts that we will need to run to provide the required functionality for the Web interface of our password cracking system. Common Gateway Interface (CGI) is a resource for accommodating the use of scripts in Web design. As scripts are sent from a server to a Web browser, the CGI-bin often gets referenced in a url.

RESULTS

Usage Instructions

1. SSH into the server and all the worker nodes in order to open the sockets in each of the nodes. We do this using GENI and the SSH Keys that are provided by GENI.

2. We run the server code first by using the command -

   ```
   python3 mserver.py
   ```

   The user is prompted to enter the MD5 hash digest of the password that they want to crack.

   The server code then opens a socket and begins listening.

3. We now run the client code on the worker nodes by using the following command on each of the nodes -

   ```
   python3 mclient.py
   ```

4. The program runs until the password is either found by one of the workers and gets reported by the server or the password is not found in the range of 5-length passwords consisting of lowercase or uppercase alphabets.

Analysis

We provide the times required by the password-cracker system to crack the passwords when we have 3, 4 and 5 worker nodes respectively. We do 3 case studies for this - one for cracking

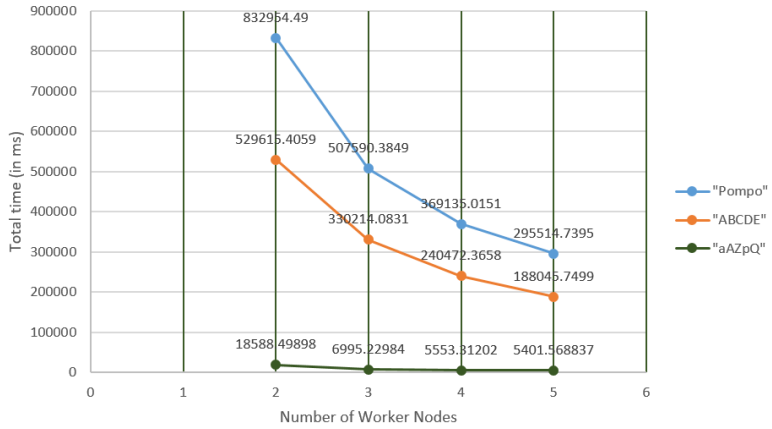

Figure 2: Graph showing the variation of Cracking Time with Number of Nodes

the password 'ABCDE', one for cracking the password 'aAZpQ' and a third case for cracking the password 'Pompo'.

The password 'ABCDE' corresponds to the integer 193975682 and should take that many tries for one of the workers to find the matching password to the digest. Similarly, the password 'Pompo' corresponds to the integer 301778010 while the password 'aAZpQ' corresponds to 3794508.

Evidently, in our implementation of the program, the passwords that correspond to integers of lesser value will be found faster than the passwords that correspond to integers of higher value. This is because we search in an increasing order from 0 ('aaaaa'). Thus, password 'aAZpQ' will be found faster than 'ABCDE' which in turn will be found faster than 'Pompo'.

We represent the observed data in Figure 2. The performance is better the more worker nodes are employed. The improvement is more pronounced in case of passwords that correspond to lower integers.

Additionally, we perform a separate analysis on how the time taken to crack a password varies when we vary the size of the search ranges. We provide the data for the password 'ABCDE'. The size of each search space is set at 10000, 20000, 50000, 100000. the results are shown in Figure 3.

As we increase the size of the search space, time required follows a general decreasing trend. However, the amount of decrease seems to vary on the specific password selected.

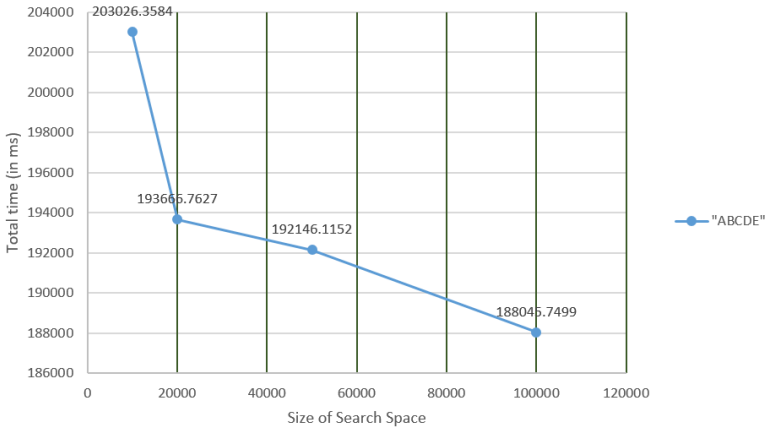

Figure 3: Graph showing the variation of Cracking Time with the Size of the search space

Possible Extensions

The password cracker system currently only tries to crack a single password at a time. however, it can be extended to parallely process multiple password digests.

Aside from this, the performance of the system can definitely be improved by increasing the number of worker nodes. This is quite easy to do by adding additional VMs on GENI. The present system is however a good representation of the general framework.

REPOSITORY

The Git Repository for the project can be found at *https://github.com/meg816/CS655-Password-Cracker*. It contains the following files -

- The server code, **mserver.py** - in Python, works on all platforms.
- The worker code, **mclient.py** - in Python, works on all platforms.
- The resource specification, **RSpec** - an XML file
- The full report, **Report.pdf**
- The demo video, **Demo Video.mp4**

REFERENCES

[1] Wikipedia - MD5 <https://en.wikipedia.org/wiki/MD5>

[2] Distributed Password Cracking - John R. Crumpacker <https://calhoun.nps.edu/bitstream/handle/10945/4461/09Dec_Crumpacker.pdf?sequence=1&isAllowed=y>

[3] APACHE - HTTP Server Project <https://httpd.apache.org/>

[4] Wikipedia - Common Gateway Interface <https://en.wikipedia.org/wiki/Common_Gateway_Interface>

[5] Configuring Apache to permit CGI <https://httpd.apache.org/docs/2.2/howto/cgi.html>

[6] Socket Programming with Multi-threading in Python - <https://www.geeksforgeeks.org/socket-programming-multi-threading-python/>