Distributed Hash Cracker

Jaidev Chittoria
Information Technology Department
National Institute of Technology Karnataka
Surathkal, India
jaidev.chittoria02@gmail.com

Shivamani Patil
Information Technology Department
National Institute of Technology Karnataka
Surathkal, India
shivamanipatil10@gmail.com

Ayush Bhandari
Information Technology Department
National Institute of Technology Karnataka
Surathkal, India
ayushbhandari8909@gmail.com

***Abstract*—Password cracking is an entirely parallel application. It consists of a set of operations on small chunks of data, and there's no data that needs to be shared between different password cracking nodes as they crack. This work focuses on distributed password cracking through nodes in a network.**

***Keywords*—Hash, Password Cracking, Multithreading, socket.**

I. Introduction

For a few decades, UNIX-based systems (and now OS X and windows) all store only the hash of a user's password. When you type your password to login, the system hashes the password you type and compares it against the stored hash value. In this way, an attacker who takes over a computer doesn't immediately learn the cleartext passwords belonging to the system's users. Recall that a cryptographic hash function is a hopefully one-way transformation from an input string to an output string, where it's impossible to go backwards from the output to an input that will generate the output: Hash(cleartext) → hash value This system is a great start, but it's not unbreakable. The original, and still somewhat commonly used, way of hashing passwords uses a hash function called crypt(). When it was initially designed in 1976, crypt() could only hash four passwords per second. With today's (drastically) faster computers, incredibly optimized password cracking code can attempt over three million hashes per second on a single core. Still, three million per second isn't all that fast. Using just the upper, lowercase, and numeric characters, there are 62^8 possible eight-character passwords. That would take 842 days to crack using a single core. But why stop there? There must be a few hundred idle cores sitting around campus... Many hands make light work. This Project we will create a distributed password cracker. We will use the concepts of multithreading, and client/server communication protocols to solve this task.

A. Problem Statement

To make a distributed password cracking system that can run across a local network.

B. Objectives

- Create application
- Distribute searching space for brute force hash cracking amongst.
- Reduce workload on single client for hash cracking.
- Make hash cracking faster.

II. Literature Survey

The following papers were an inspiration to this project, and have therefore been listed here:

1. Distributed Password Cracker by CMU CS
2. Distributed Password Cracker by Randy Bryant

III. Design of Algorithm

Server Architecture:
As per problem definition the server will generate a password with 5 characters.
This password may contain numbers (0-9) or Capital letters (A-Z). It will then generate a hash of combining the password and the system date. Now the server is ready to distribute this hash among the worker clients for break. The Server should be multi-threaded to facilitate communication with the client. Each time a new request for connection from client comes the server should create a new thread to communicate with that server. The client should be independent and should be able to find out everything it needs to know from the server. The server should recognize the two different clients as they contact it. It should keep track of the clients and which jobs they're currently working on or what job they've requested.

Client Architecture:
The Worker client after successfully get connected to the server it will get a range at which it will try to break the actual password. In order to making password within given range following mapping is done
Total possible alpha-numeric characters: 36 (numbers (0-9) and capital letters (A-Z)).
Password length: 5
Possible combinations: 36^5 (60466176)
In order to map passwords within one million ranges for each client we can divide the range with the Base 36 system.

In Base 36 system all possible combinations: start 00000 to ZZZZZ.
Some conversion from decimal to base 36 is given in following table: -

Range No.	1,000,000 range in decimal	1,000,000 range in base 36
0	000,000 to 999,999	00000 to 0LFLR
1	1,000,000 to 1,999,999	0LFLS to 16V7J
2	2,000,000 to 2,999,999	16V7K to 1SATB
3	3,000,000 to 3,999,999	1SATC to 2DQF3
4	4,000,000 to 4,999,999	2DQF4 to 2Z60V
5	5,000,000 to 5,999,999	2Z60W to 3KLMN
6	6,000,000 to 6,999,9992	3KLMO to 4618F

IV. Hardware and Software Requirements

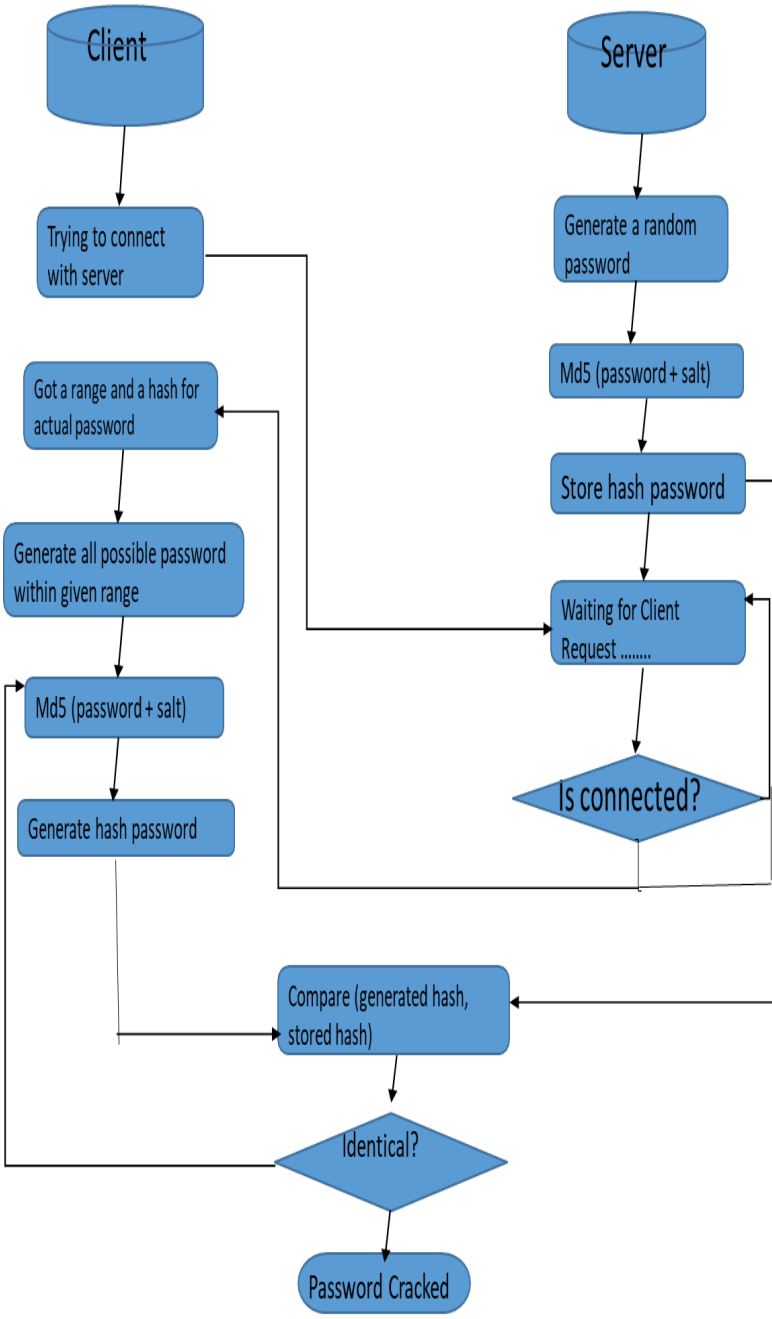
Since our password cracking system is a software program, no particular hardware requirements exist.

The program is compiled using Linux compilers (G++, GDB), because of which the program would run only on Linux based operating systems like Ubuntu, Fedora, etc.

V. Methodology

- Creation of two programs i.e. worker client and server client
- Server
- Multithreading
- Password cracking job and dividing job into parts and allocation to worker clients
- Client program
- Receiving of hash and generate password and comparing with key

VI. Implementation of Algorithm



VII. Result and Discussion

The following is a rundown of how the application would operate:

This project consists of two programs named Server.cpp and Client.cpp These two C++ programs are compiled using g++ compiler and corresponding bat files are given. This project is built in C++ language and hence to run the project G++ Compiler is first needed to install. For windows pc simply run the bat file. For Linux the two programs should

first compile and then run from Terminal. The Server Program should run first and then the client program. To break the password the server should always live. Please DO NOT CLOSE the command prompt of the server while any client is connecting with the server or make the server in “select command prompt” state.

VIII. Conclusion

When the server starts running with a hash provided by the user . Then it waits for an incoming client request. The client received a range from the server. If it fails to get the password within the range, it sends a RETRY packet to the server. When a client successfully gets the password, it prints the password, and also notifies the server about it. But if it does not, it terminates with a failure message.

IX. Acknowledgment

We would like to thank our professors in our department, particularly Dr. Geetha V, Ms. Thanmayee who helped us shape the idea of our software.

X. Authors and Affiliations

This report is a part of a mini-project for Computer Communications and Networking. Following are the members who contributed:

- Jaidev Chittoria, Roll No. 181IT119
- Shivamani Patil, Roll No. 181IT144
- Ayush Bhandari, Roll No. 181IT209

XI. References

[1] Florian Mendel; Christian Rechberger; Martin Schläffer. "MD5 is Weaker than Weak: Attacks on Concatenated Combiners". "Advances in Cryptology – ASIACRYPT 2009". p. 145. quote: 'Concatenating ... is often used by implementers to "hedge bets" on hash functions. A combiner of the form MD5||SHA-1 as used in SSL3.0/TLS1.0 ... is an example of such a strategy.'

[2] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. LNCS 3152/2004, pages 306–316 Full text.Sang-Keun Gil.

[3] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman; “An Introduction to Mathematical Cryptography”

[4] Wikipedia, Schneier, Bruce. "Cryptanalysis of MD5 and SHA: Time for a New Standard". Computerworld. Retrieved 2016-04-20. Much more than encryption algorithms, one-way hash functions are the workhorses of modern cryptography.