GRS CS 655: GENI Mini Project Final Report: Password Cracker

Team Members:
1) Rhythm Somaiya – U84158310
2) Aman Ahmed – U01171262
3) Nirbhay Malhotra – U42321017

GitHub link: https://github.com/amanahmed97/java-networks-password-cracker
Project on GENI: Slice name: cracker-2 (slice by Aman Ahmed, username: amana)

1. Introduction/Problem Statement

Whether it is accessing our email account or shopping online to purchase the essentials, passwords are an integral part of our everyday lives. Passwords are among the most fundamental mechanisms for securing data. It is common today for password-protected systems to store passwords in their databases in the form of hashes. The MD5 hashing algorithm generates digests that are 128 bits in length. However, even though vulnerabilities have been discovered in the algorithm's security, it continues to be widely used on the Internet.

According to the project title, this project will address the "Password Cracker" problem, where we intend to decrypt passwords based on their MD5 hash using Brute force. Given such an MD5 hash made up of 5 alphabetic characters, the objective is to be able to extract the password from the hash provided by the user.

Since MD5 works only in one direction, it is impossible to decrypt a hashed password. This can only be accomplished by brute-forcing every possible 5-character alphabetic password to find its MD5 hash and then comparing it with the submitted hash. Since brute-force algorithms can take a considerable amount of time, one of the most efficient ways to accomplish this would be to use a distributed or multi-threaded approach. In the case of uppercase and lowercase characters, each of the five characters in the input has 52 possible combinations, which means that the system can iterate through 52^5 = 380204032 possible password combinations. Regarding the architecture of this project, we intend to develop a method for cracking an MD5 password hash by utilizing a distributed system in GENI. There will be a central server and a web interface where the MD5 hash will be passed. The hash will be based on a five-character password consisting only of alphabets (either lowercase or uppercase) for the purpose of this project. This message digest is input by the user via a web interface, and once it has been processed on the back end by the nodes in a distributed manner, the original password should be returned. The objective of this assignment is to determine how to configure the connections between a web page, a server node, and a set of multiple worker nodes so that they will be able to communicate together to crack the MD5 hash. The worker nodes must be able to cooperate so that they can try different passwords and see if they match the input, as well as stop their processes once one of them has completed the process. It is also necessary for the web page and the server node to communicate to receive the input and return the cracked password. The server node must also communicate with the worker nodes to properly distribute work among them.

2. Experimental Methodology

The following project consist of multithreaded servers and multithreaded workers communicating to crack the respective password where API web interface client sends hash to the server which will then distribute it to the number of workers specified by the web interface and the workers will try to find the matching hash of some password found from the word list after chains of looping.

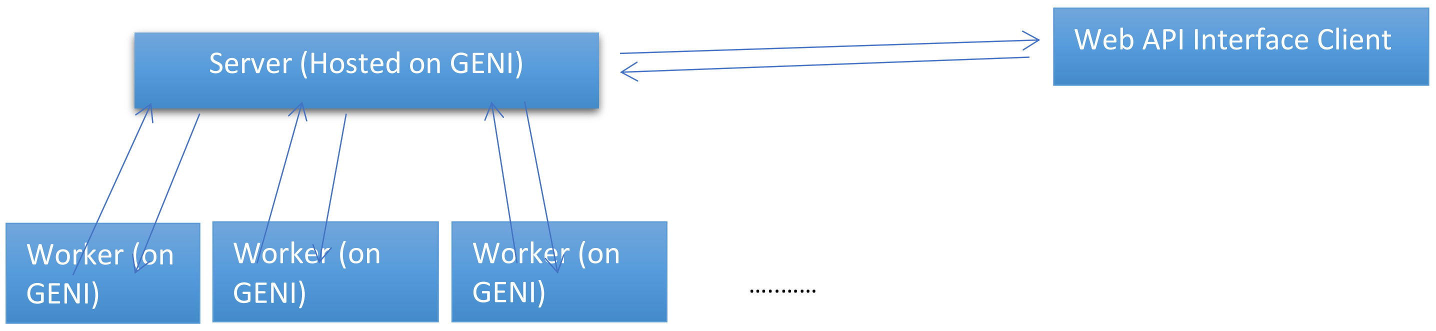


Figure 1: Project Architecture

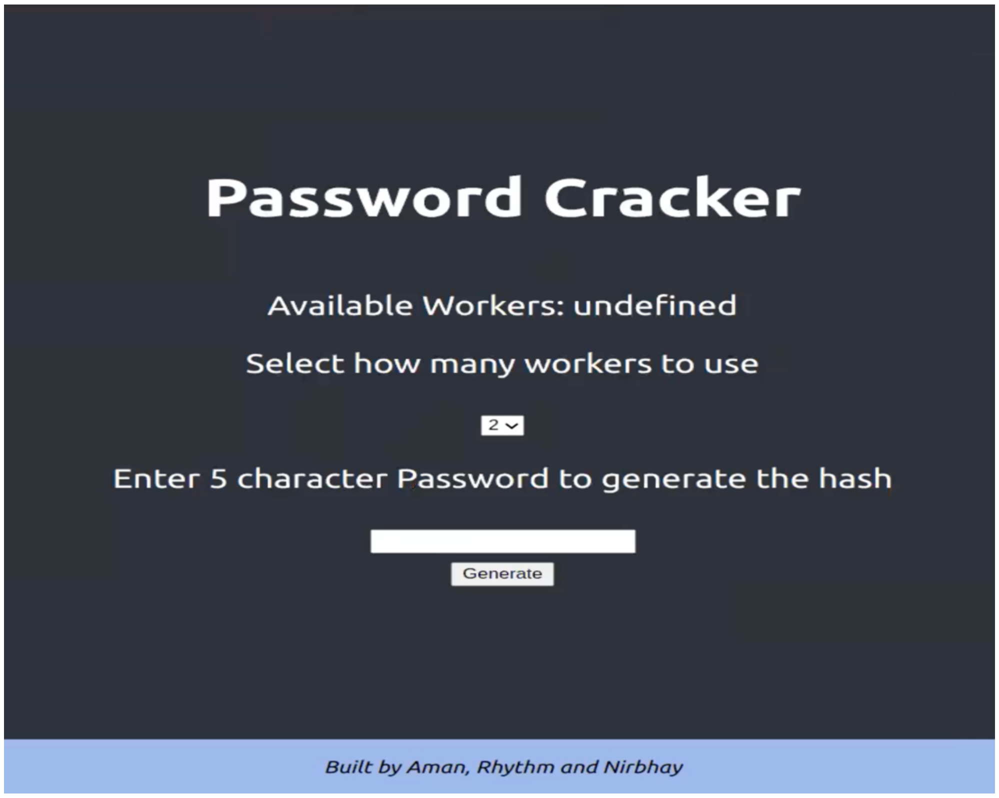
- Worker nodes on GENI:
  Each of the worker nodes is multithreaded, they keep listening infinitely, and they wait for the hash to be delivered to them from the server so that they can start working cracking the password by generating various hashes from the wordlist (consisting of all upper case and lower case alphabetic letters) in order to find the valid five letter word that represents the hash matching to the one provided by the server. If only one of the workers is chosen for the following task to be performed on, the single worker node will loop through all the possible combinations of five letter word to find the appropriate matching hash. Also, if we have various nodes that are selected by the client, then we can distribute the work of looping through each wordlist among the workers so that it will maximize the performance and minimize the delay for the distributed system. In the following project, if we had multiple worker nodes assigned for a particular hash, then each individual worker will be assigned to loop through wordlist in a fashion that it will use the first alphabetic letter for the password formed for evaluation depending upon the number of workers chosen and it's worker Id and will move on to another first alphabetic letter after the first chain of looping ends for the password depending upon the number of workers chosen and it's worker Id and, this helps in distribution of work among the workers so that the procedure of brute force attacking on the password is time efficient. For example, if we have two workers chosen – Server will provide the hash to both workers where the first worker will loop through all the combination of words that start with A and the second worker will loop through all the combination of words that start with B. And if still the password Is not found, the first worker will continue to find all the combination of words that start with C and the second worker will continue to final the combinations of words that start with D to find the matching hash and it will go on till the matching hash (password) isn't found.

- Server node on GENI:
  Server node here is a multithreaded server that handles all the workers. The task of the server is to first display to the client the number of workers it has access to so that the API client can choose the appropriate workers of it's liking to send the hash to the workers through the server. It works accordingly to assign the job to each worker by providing it the hash and indicating the number of workers chosen and worker Id to each worker for the appropriate hash.

- Web API Interface Client:
  This client will choose the number of workers it wants to assign for a particular hash and will inform that to the server and it will also provide the hash to the server on which the brute force attacking can happen to find the password and it will wait to retrieve the following hash's password. A further extension of the following API client will be that it will be able to push in several hashes at the same time. Also assigning more workers for a particular hash while the processing happens, and making the API client as a multithreaded client who will always listen will help in an easier system that won't close and collect the password for different hashes concurrently.
  We are assuming that we can pass any number of hashes into the queue which will result in a decryption of password.

3. Results

3.1 Usage Instructions –



Password Cracker

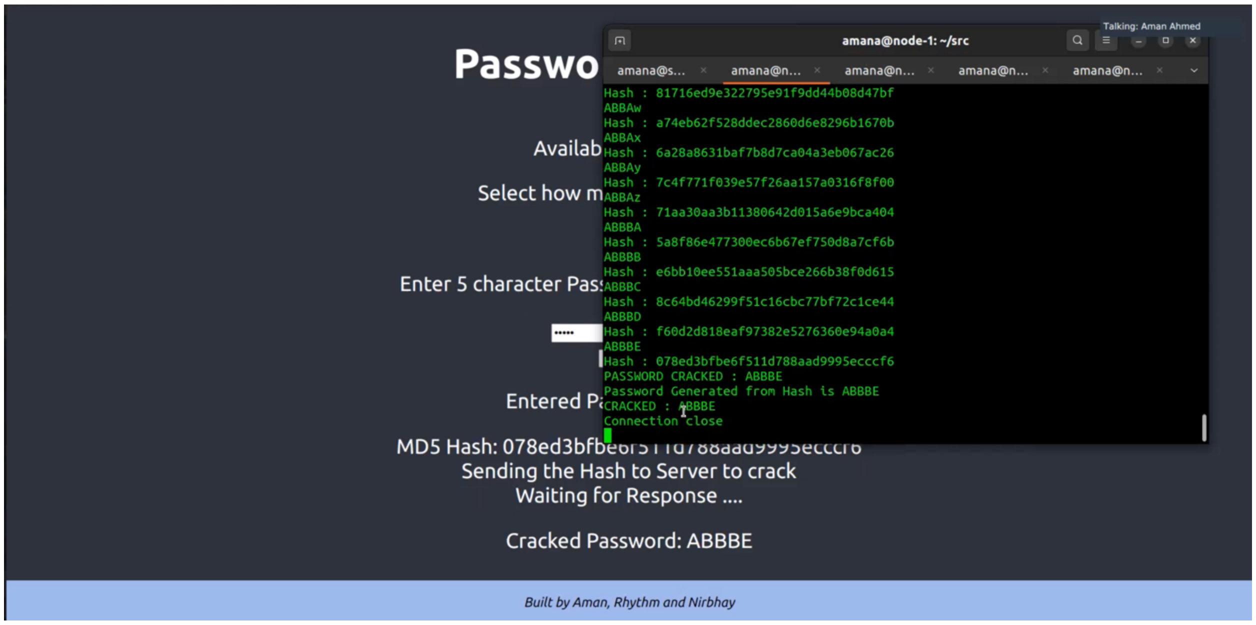
Available Workers: undefined

Select how many workers to use

Enter 5 character Password to generate the hash

Built by Aman, Rhythm and Nirbhay

Figure 2: Web Interface (before decryption)



```
Hash : 81716ed9e322795e91f9dd44b08d47bf
ABBAw
Hash : a74eb62f528ddec2860d6e8296b1670b
ABBAx
Hash : 6a28a8631baf7b8d7ca04a3eb067ac26
ABBAy
Hash : 7c4f771f039e57f26aa157a0316f8f00
ABBAz
Hash : 71aa30aa3b11380642d015a6e9bca404
ABBBA
Hash : 5a8f86e477300ec6b67ef750d8a7cf6b
ABBBB
Hash : e6bb10ee551aaa505bce266b38f0d615
ABBBC
Hash : 8c64bd46299f51c16cbc77bf72c1ce44
ABBBD
Hash : f60d2d818eaf97382e5276360e94a0a4
ABBBE
Hash : 078ed3bfbe6f511d788aad9995ecccf6
PASSWORD CRACKED : ABBBE
Password Generated from Hash is ABBBE
CRACKED : ABBBE
Connection close
```

MD5 Hash: 078ed3bfbe6f511d788aad9995ecccf6
Sending the Hash to Server to crack
Waiting for Response ....

Cracked Password: ABBBE

Built by Aman, Rhythm and Nirbhay

Figure 3: Web Interface (after decryption)

1. Reserve resources on GENI
Use the Rspec "rspec.xml" to reserve the resources on GENI.

2. 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. And you can initiate the process for sending the hash by running the API client locally and connecting it to the server on GENI since the server is allocated a public routable IP.

3. We run the server code after the ssh on GENI by using the command -
```
javac Server.java
java Server
```
- Now using the built web interface, the user enters the 5-character password (which they want to crack) which further gets converted into a MD5 hash and it passes the API request to the server. The user also has the option to assign the number of workers they want to find the password for that particular hash.
- The server code then opens a socket and begins listening.

4. For running the API Client locally, we can use the command –
```
javac Client.java
java Client
```

5. For running the Worker node on GENI by using the command –
```
javac Worker.java
java Worker
```

The program runs until the password is either found by one of the workers and gets reported by the server. The cracked password is then shown in the web interface to the user.

3.2 Analysis –
To calculate the performance of our workers with respect to the time it takes to crack the password, we tested the following 5-character passwords to get the proper idea on the analysis our program works on:
1. DABBY
2. BATSY
3. BOssC
4. HELLO
5. adddx

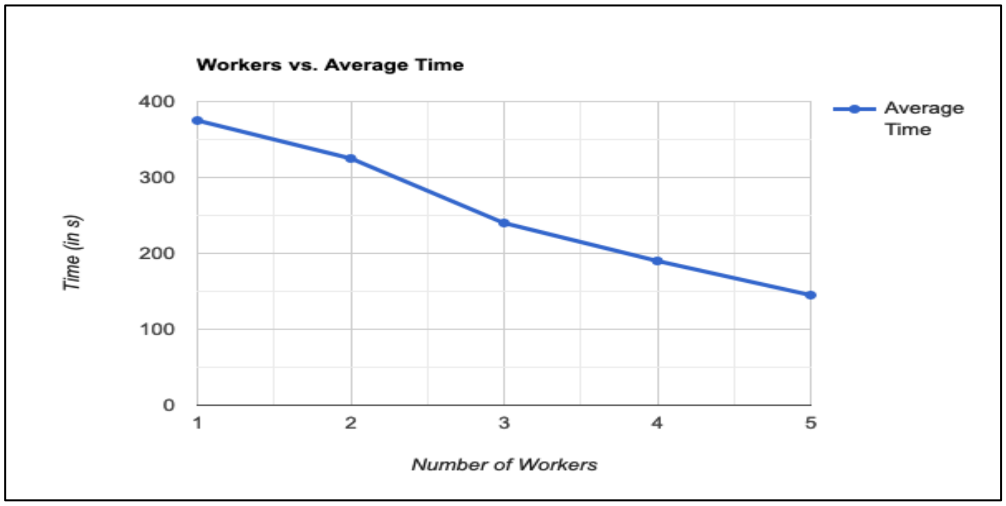


Figure 4: Line Graph to show the variation of Average time with Number of Workers

From the above graph, we get the inference that as the number of workers increase, we know that an individual worker will hop depending upon the number of workers the system has. So, if there are more number of workers, then the possibility of decrypting the password by finding the matchable hash increases rapidly. For example, if we have 5 workers, the first worker will skip the next 4 characters after processing the character it has access to and meanwhile the other workers will process on those 4 characters. Hence, these results will help in increasing the efficiency of performing the brute-force algorithm depending on the number of workers. After performing the respective algorithm on the hashes of the above passwords, we get the average time vs number of workers graph denoted by figure 4.

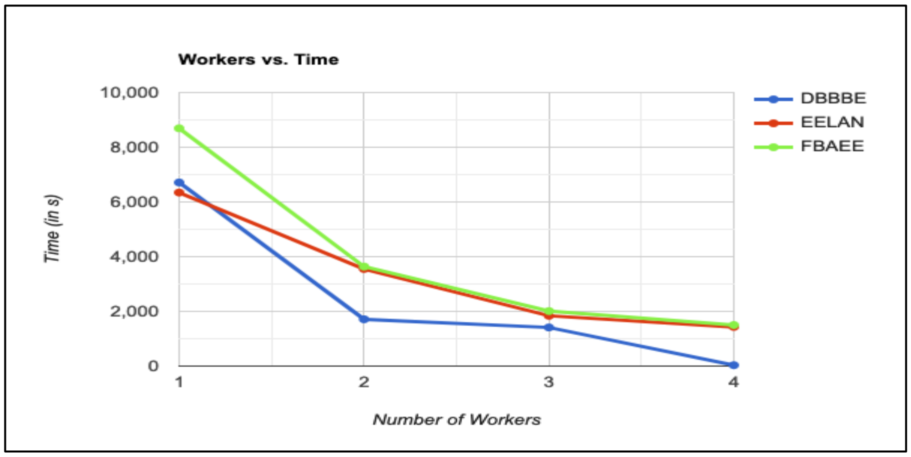


Figure 5: Line Graph to decrypt the password using four workers

In Figure 5, we can see for three passwords – "DBBBE", "EELAN", and "FBAEE", how much time does it take to decrypt the respective password using four workers.

4. Conclusion

Overall, the main findings of this project illustrate the importance of distributing such a workload. With one node, we would start iterating at the letter "A" but with multiple worker nodes, we can start with the next alphabet in an alternating way thus, not having too much of a concern about passwords starting with letters later in the alphabet. In terms of scalability and workload distribution, it was the architecture of the network and how the jobs were distributed that had a greater effect than the TCP network conditions. In its current state, the project could be extended to enable the number of worker nodes to be set up on the spot for each user. Additionally, this project could be extended to include longer passwords, alphanumeric passwords, allowing the user to specify parameters such as bandwidth and delay, caching previously found MD5 hashes to prevent re-compiling them in future rounds or iterations, and others.

5. Division of Labor

- Aman and Rhythm focused on the frontend implementation using React and worked on the backend using a Java framework – Spring to make a RESTful web service and handle the API calling procedure.
- Nirbhay worked on the backend architecture part of this project with all three of us contributing on the testing and other implementation tasks required.

REFERENCES

[1] ReactJS, "https://reactjs.org/tutorial/tutorial.html"

[2] Java Spring, "https://spring.io/guides"