MULTI-THREADED BRUTE-FORCER

Anirudh Kaushal, Pratyush Jaiswal, Shreeraj Bansal, Shubham Sharma

Jaypee Institute of Information Technology, Noida

**Abstract.** In this project, we have tried to generate a word list using brute force in order to try all the possible combinations to crack a password but the catch is that the list is dynamic. In other words, the random sequence of letters generated in the list gets created and deleted after a span of seconds to reduce the amount of memory utilized. It is found that creating a word list with 8 characters using all possible combinations of letters, numbers and special characters is several Petabytes. In our model the process of creation, checking and deletion runs parallelly with some time delay between them to keep the working synchronized. This project is built so that brute force attacks for cracking password gets more efficient in terms of time and space.

**Keyword:** brute-force**,** parallel programming, multi-threading

# Introduction

Brute-forcing is a technique of trying all possible combinations in a given key-space to crack a given password or to find a given hash.

The focus of our project is to reduce the time and space required for a brute force algorithm to generate a word-list with all possible combinations of available characters. Further details are given under the detailed study and the implementation. We have used the concept of multi-threading to implement parallel programming. Multi-threading enables us to use multiple cores of the CPU simultaneously to enhance the speed of execution. Therefore more than one process can run parallely on different cores.

For data structures, we have used linked list where the length of the list is already defined and the nodes are connected in a way that makes it a circular linked list. For the functioning of the model, it is important that the CPU has multiple cores, other system requirements are listed under the requirement analysis. In this project, we have tackled the reader-writer problem that occurs when writing data onto the nodes and reading the data to check if it matches the password.

1. **Previous methodologies**

**Naive Brute-force attacks[3]** – this attack includes trying all the combinations of alphanumeric strings of given length. That is, it will check all the strings from aaaa1… to zzzz10 to find the correct password. But the main problem with the brute-force attacks is that they are very time consuming as they have to create a very long list of strings and check each one of them if they match the correct password. Also since they create such long list of strings, a large amount of memory is needed to store the generated strings. It is found that creating a word list with 8 characters using all possible combinations of letters, numbers and special characters is several Petabytes. And the time it takes to crack a password could range from a few hours to a few weeks depending upon the strength of the password and the speed of the processor. But the thing is, brute-force will find the correct password as it will check every combination of given set of characters. To make the brute force faster parallel brute-forcing has been implemented before in which there were two processes that were creating and checking half of the list i.e. one process checked the first half and the second process checked the second half and they were run parallelly. This reduced the time of creating and checking by half but still it did not solve the problem related with the huge amount of memory that is consumed by the generated list.

**Pipe-lined Brute-force[Pipelined wala paper]:** The pipe-lined brute-forcing method uses a similar approach as in [Pipelined wala paper] To save space while brute-forcing, a method was introduced, named pipelined brute-force[Pipelined wala paper], that has working similar to created, checked and deleted the word in a single variable. Using a pipelined structure gave us freedom to do brute force

All the above methodologies were naive approach and has not been changed since their very introduction. The only way to improve the timing involved in brute- forcing process was said to be to enhance the hardware used, i.e. to use a better CPU, GPU or even a set of parallel GPUs.

Our proposed solution is using a similar approach to pipelined brute-force but is a lot more efficient.

1. **Proposed solution**

The two downsides of Brute-forcing are, it consumes a lot of time, and secondly, it consumes a lot of space. We here have proposed a solution for both the above said issues. We have used a technique that uses multi-threading and process synchronization to decrease the time taken and the space now needed becomes comparatively negligible because of the circular “Staircase” implementation of the linked list.

To reduce memory usage, the program creates the string and after checking the string for password it deletes the string so that a new string can be written in its place. This way we are able to limit the amount of memory used when the program runs. Now if the program creates, checks and deletes strings in this order and then create another string, this will take more time than it normally would. So to fix this problem we run the creation function and the checking function parallelly. We store the generated strings in circular linked list of string array type, this way we don’t actually have to delete the strings but just over-write the previous one. So this way the program generates the strings and check them parallelly and once they have been checked they are overwritten. But this is not achieved just yet. When the two functions are running parallelly, there will come a time when the speed of creation will decrease due to increase in length of words but checking function will keep checking at the same speed and at one point of time the checking function will run ahead of the creation which will cause problems. To get rid of this we need to synchronize the two functions in such a way that the checking function is always behind the creation function. When the creation function is acting on one node, the checking function should not be allowed to check that same node. This needed to be solved with the reader-writer solution.

# Implementation

The goal is to create a wordlist model that when created, simultaneously   
 checks itself with target and deletes, hence saving memory space.

1) We divided the model into three major parts that would run in parallel   
 synchronous. The three parts are Creation, Checking, and Deletion.

* Creation() used to create a wordlist. This creation function, at its maximum potential on the above specified system [System requirement deni thi, kaha di h?, Agar do ab toh Implementation k upar dena] has a rate little less than 4,000,000 words/second (~3.87 Million words/second).
* Checking() to check the created wordlist with the target.
* Deletion() to delete the created wordlist, *(this function was not used, explained later)*

2) All three of the above said parts need to run parallelly, to achieve that we   
 used multi-threading[1] that divides the processes into threads and helps   
 them execute parallelly.

3) All the processes had to be applied to the same memory space, which caused   
 a problem, referred to as Reader-Writer problem [2], due to it we were unable   
 to execute the processes on same memory space [3]. As its mitigation, we   
 instead of using a single data structure, i.e. an array or a vector, using a   
 circular linked-list,

* The linked list consisting of three nodes, that are node1, node2 and node3
* Each node of the linked list consists of an array of size 26\*n, where n is decided according to the length of the target to be brute-forced.
* Due to the use of a circular wordlist, we were able to implement creation and deletion in a single process, as when the control was passed from the last node to the first node, the first node was over-written, hence deleting the previous value.

1. We are using binary semaphore with “symmetric signaling synchronization technique”. A symmetric synchronization technique is when all the threads use a similar pattern of wait and signal functions. A symmetric technique is used as it is much more efficient and takes less time during context switching between thread. The synchronization works as follows,

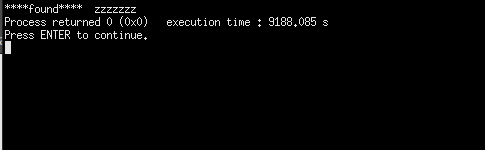
* The two functions creation() and checking() are executed parallely.
* Initially Creation() is called, followed by checking(). figure x.x shows a series of wait and signal functions along with A, B and C.
* A, B and C are referred to as the three nodes of linked-list, marked sequentially.
* The control initially goes to creation(), hence locking linked-list node-A, so when Checking() is called, it can not execute till creation() releases the lock.
* Then when creation() releases lock on Node A, then checking() locks the node-A for checking, and simultaneously creation() has started creating word-list in node-B. From this moment, both the functions starts running parallely.
* A more better understanding can be seen through a rough dry-run in figure x.y.

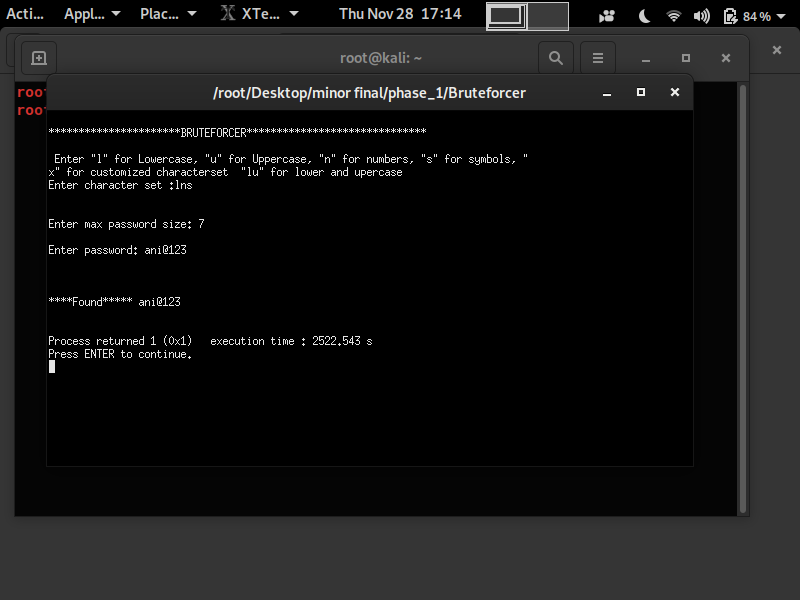
# Result analysis

Table ki numbering

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S No. | Password | Character Set | Avg Time for Naive BF  (seconds) | Avg Time for Multi-Threaded BF  (seconds) |
| 1. | zzzz | a-z | 0.546 | 0.151 |
| 2. | zzzzz | a-z | 14.213 | 3.046 |
| 3. | zzzzzz | a-z | 312.438 | 84.120 |
| 4. | zzzzzzz | a-z | 9187.324 | 2525.471 |
| 5. | ani@123 | a-z, A-Z, 0-9,( 22 special chars) | 8124.786 | 2252.543 |

# Fig 6.4. Time taken to find password using threading

  
 Fig 6.5. Time taken to find password without threading

 Here we can see that the time taken by the threaded program to reach the   
 last word of 7 characters consisting of small case alphabets (2524.411 s) is   
 considerably less than the time taken by the program which is not   
 implemented using threading(9188.085 s). Here we can see that the time   
 required to process every word possible with length of 7 characters took   
 2524.411 s.

# Conclusion To crack a password sometimes it is most sensible to try all the combinations of characters but creating a word-list using brute force takes up too much time and space and it is not feasible to carry software that has already created word-list as it will be as big as many hundred petabytes. In this project, we have successfully used the brute force algorithm to generate all the combinations and tried to make the model more efficient in terms of time and space. To achieve this, we created the list which is dynamic that is, it creates the list and continuously overwrites the checked combinations with a new one, and this happens parallelly with some lag between both the functions just so that it works in a synchronized manner and the checking function does not get ahead of the function responsible for generating the list, this way we were able to limit the usage of space and the parallel processing enabled us in reducing the time of execution.

1. **Future Scope**  
   Brute-forcing in todays era is mostly used in block-chain mining. The miners need to use

**8. Refrences** [1] J.B. Dennings and G.R. Gao, “Multi-threaded Architectures: Principles,   
 Projects and Issues”, Laboratory of Computer Science, MIT, 1994.

[2] W.C. Hsieh, and W.E. Weihl, “Scalable Reader-Writer Locks for Parallel   
 systems”, Laboratory of Computer Science, Massachusetts Institute of   
 Technology, Cambridge, 1992.

[3] L.Bosnjak, J.sres and B. Brumen, “Brute-force and dictionary attacks on   
 hashed and real-world password”, University of Maribor, Faculty of,   
 Slovenia, May 2018.

[4] A. Alexandrescu, H. Boehm, K. Henney, B. Hutchings, D. Lea and B.   
 Pugh, “Memory model for Multithreading”, March 2005.

# 