Parallelization of John the Ripper (JtR) using MPI

Ryan Lim
Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln, NE 68588-0115
rlim@cse.unl.edu

January 22, 2004

Abstract

JtR is a utility used commonly used to detect weak passwords. It employs a dictionary attack and brute-force cracking mechanisms to perform its purpose. While JtR is probably not designed specifically to crack passwords of strength, it uses a brute-force strategy. Brute-force is considered infallible, but is a time consuming and computationally expensive approach, and thus, is an ideal program to parallelize.

1 Introduction

A password is an arbitrary string of characters chosen by a user and is used to authenticate the user when he or she attempts to use a particular resource such as computing resources.

Once a password is chosen, it is usually encrypted using a one-way function, $f(x) = y$, which is easy to compute. However, since it is a one-way function, there is no tractable function $f^{-1}(y) = x$ that can reverse the encryption. The only way to crack the encrypted string is by using brute-force, that is to exhaustively try all possible combinations and permutations of characters.

Given that a password has the following properties:

- At least 8 characters.

- Has a good mix of upper and lowercase letters, numbers, punctuations and other symbols.

there are 95 distinct choice of characters. With this assumption, a n-character password has 95^n permutations (for a 8-character password, there about 6000 trillion)!

While cracking is not impossible, it would definitely take a lot of computational time. The goal of this paper is to address these issues by parallelizing the popular password cracker, John the Ripper (JtR) [Des03] .

This paper discusses related work in Section 2, my implementation in Section 3. Section 4 describes how I evaluated my code and presents the results. Section 5 presents my conclusions and describes potential future work.

2 Related Work

The San Diego Supercomputer Center (SDSC) presented its results [PK03] from applying High Performance Computing (HPC) resources such as a parallel supercomputer, abundant disk and large tape archive systems to precompute and store `crypt()` based passwords. SDSC's project, *Tablecrack* is able to quickly determine easily guessed passwords for UNIX accounts. *Teracrack*, another project, applies modern HPC capabilities.

A similar project, "dkbf" [B0r] is a distributed, keyboard, brute-force program for Linux clusters using the message-passing interface (MPI). However, this program is specifically tailored to crack only Windows NT Lanman and NT hashes.

JtR, on the other hand, supports various encryption formats such as standard and double-length DES, BSDI's extended DES, FreeBSD's MD5, OpenBSD's Blowfish, AFS and Windows NT Lanman formats.

3 Implementation

This section covers some of the details of the work involved in the strategy, planning and implementation of the parallelization of the John the Ripper source code.

3.1 Strategy

I initially approached this problem using the *bag-of-tasks* approach where a single master processor hands out sets of tasks (portions of keyspaces) to slave processors to conduct brute-force search. Using this approach, a master may hand out a keyspace ranging from `0x000000` to `0x888888` to slave #1 and keyspace ranging from `0x888889` to `0xFFFFFF` to slave

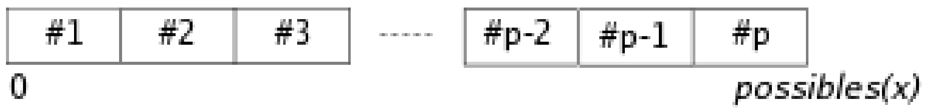

Figure 1: Each processor p gets an equal chunk ($possibles(x)/p$) of the keyspace from 0 to $possibles(x)$, where $possibles(x)$ is the number of possible permutations for x.

#2. Once slaves #1 and #2 complete their task, they may return to the master processor for more work.

However, after analyzing the JtR source code, I noticed that there would be no need for using the *bag-of-tasks* approach as each processor could compute and work on separate chunks of keyspaces simultaneously (Figure 1). This is because the keyspace range is always constant, and thus, can be divided equally among all processors.

3.2 Planning

In order to find out possible code sections which can be parallelized, I profiled JtR's code using the GNU `prof` utility. Part of the profile output can be viewed in Appendix A.

Besides code profiling, I did some substantial analysis on the function calls listed in the gprof output.

3.3 Implementation

In order to parallelize JtR, I chose to use the message-passing interface (MPI).The reason for this is because MPI is relatively portable and it is installed on many of multi-computers at the University of Nebraska - Lincoln such as `rcf` and `prairiefire`.

Once code profiling and analysis was done, I added auxiliary functions to enable JtR to run in parallel. Most of the work was done in the file locking mechanisms, and the `do_incremental_crack()` function.

In order to handle file locking, each process will 'own' its files exclusively. These files are the character set files (`all.chr, alpha.chr, digits.chr`), recovery files (`john_rec`) and log files (`john_log`). Each process' files are denoted by the processor identifier suffix. For example, processor 0's files are `all.chr.0, alpha.chr.0, digits.chr.0, john_rec.0` and `john_log.0`

Modifications to `do_incremental_crack()` enable it to splits the keyspace into chunks. Processors pick up interleaved chunks and leave the rest to the other processors.

4 Evaluation

Since work is evenly split between processors, the total number of crypts per second should increase linearly with the number of processors. As each processor gets different sets of keyspaces, the total of crypt operations per second is approximately $\sum_{i=0}^{P} C_i$ where C_i is the crypt operations per second for processor i and P is the number of processors.

Using one processor on `rcf`:

```
Benchmarking: FreeBSD MD5 [32/64 X2]... DONE
Raw:    1124.00 c/s real, 1127.00 c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/64]... DONE
Raw:    74.02 c/s real, 74.01 c/s virtual
```

Using two processors on `rcf`:

```
Benchmarking: FreeBSD MD5 [32/64 X2]... DONE
Raw:    2242.00 c/s real, 2248.00 c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/64]... DONE
Raw:    148.02 c/s real, 148.06 c/s virtual
```

Using eight processors on `rcf`:

```
Benchmarking: FreeBSD MD5 [32/64 X2]... DONE
Raw:    8757.00 c/s real, 8990.00 c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/64]... DONE
Raw:    571.31 c/s real, 592.25 c/s virtual
```

This shows impressive benchmark results. One of the reasons for this is that there is rarely, if not, no communication between processors. Each processor calculates its own keyspace ranges and then performs operations on it.

The complete set of results are in Appendix B.

5 Conclusions and Future Work

Currently, this JtR code is only parallelized for the incremental cracking mode. This is because incremental cracking consumes the most time as it is searches the entire keyspace.

This parallelized JtR does not do parallel dictionary-attacks. Adding parallel dictionary-attacks remains work to be done.

In addition to that, JtR could be further parallelized on shared memory architectures using OpenMP or PThreads.

Acknowledgments

A big thanks to "Solar Designer" for writing the wonderful program, John the Ripper which is already optimized for many platforms. Ironically, I required help from a person called John, more specifically, John Lim Eng Hooi, to help me debug portions of my code. I would also like to thank Dr. David Swanson for helping, advising and encouraging me thorough the semester.

References

[B0r] D4 B0rg. dkbf. URL source: http://dkbf.sourceforge.net/.

[Des03] Solar Designer. John the ripper, 2003. URL source: http://www.openwall.com/john/.

[PK03] Tom Perrine and Devin Kowatch. Teracrack: Password cracking using teraflop and petabyte resources. 2003.

A gprof output

Only functions with more than 10 calls are listed.

```
Flat profile:

Each sample counts as 0.000999001 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  ms/call  ms/call  name    
 97.56     31.03    31.03                             MD5_body
  1.18     31.40     0.37  6983883     0.00     0.00  expand
  0.99     31.72     0.32   176497     0.00     0.00  MD5_std_crypt
  0.12     31.76     0.04      261     0.15     1.59  inc_new_count
  0.05     31.77     0.02      170     0.10     0.10  inc_new_length
  0.02     31.78     0.00    88251     0.00     0.00  MD5_std_set_key
  0.02     31.78     0.00    88247     0.00     0.00  crk_salt_loop
  0.01     31.79     0.00    88247     0.00     0.00  fix_state
  0.01     31.79     0.00      306     0.01     1.12  inc_key_loop
  0.01     31.79     0.00   176498     0.00     0.00  MD5_std_set_salt
  0.01     31.79     0.00   176494     0.00     0.00  crk_password_loop
  0.01     31.80     0.00   176494     0.00     0.00  idle_yield
  0.01     31.80     0.00   176493     0.00     0.00  status_update_crypts
  0.01     31.80     0.00      749     0.00     0.00  status_get_time
  0.00     31.80     0.00   176493     0.00     0.00  add32to64
  0.00     31.80     0.00    88247     0.00     0.00  crk_process_key
  0.00     31.80     0.00    88246     0.00     0.00  fmt_default_clear_keys
  0.00     31.80     0.00      745     0.00     0.00  log_time
  0.00     31.80     0.00        1     1.00   777.19  do_incremental_crack
  0.00     31.80     0.00   176501     0.00     0.00  cmp_all
  0.00     31.80     0.00    88251     0.00     0.00  set_key
  0.00     31.80     0.00    88251     0.00     0.00  strnfcpy
  0.00     31.80     0.00      927     0.00     0.00  mem_alloc_tiny
  0.00     31.80     0.00      753     0.00     0.00  get_time
  0.00     31.80     0.00      746     0.00     0.00  log_event
  0.00     31.80     0.00      745     0.00     0.00  log_file_write
  0.00     31.80     0.00      630     0.00     0.00  trim
  0.00     31.80     0.00      577     0.00     0.00  fgetl
  0.00     31.80     0.00      576     0.00     0.00  cfg_process_line
  0.00     31.80     0.00      463     0.00     0.00  str_alloc_copy
  0.00     31.80     0.00      402     0.00     0.00  cfg_add_line
  0.00     31.80     0.00       44     0.00     0.00  tty_getchar
  0.00     31.80     0.00       34     0.00     0.00  strlwr
  0.00     31.80     0.00       24     0.00     0.00  add32to64m
  0.00     31.80     0.00       24     0.00     0.00  log_file_flush
  0.00     31.80     0.00       21     0.00     0.00  init_line
  0.00     31.80     0.00       20     0.00     0.00  cfg_add_param
  0.00     31.80     0.00       15     0.00     0.00  mem_alloc
  0.00     31.80     0.00       14     0.00     0.00  cfg_add_section
  0.00     31.80     0.00       14     0.00     0.00  mem_free
  0.00     31.80     0.00       13     0.00     0.00  write_loop
  0.00     31.80     0.00       12     0.00     0.00  MD5_std_get_binary
  0.00     31.80     0.00       12     0.00     0.00  mul32by32
  0.00     31.80     0.00       10     0.00     0.00  ldr_get_field
  0.00     31.80     0.00       10     0.00     0.00  path_expand
```

B John the Ripper (MPI) results

```
### 1 PROCESSOR ###
rlim@rcf ~/john-1.6.36/run% mpirun -np 1 ./john -test
Benchmarking: Traditional DES [64/64 BS]... DONE
Many salts:     172019.00 c/s real, 172019.00 c/s virtual
Only one salt:  155584.00 c/s real, 155895.00 c/s virtual

Benchmarking: BSDI DES (x725) [64/64 BS]... DONE
Many salts:     5620.00 c/s real, 5632.00 c/s virtual
Only one salt:  5544.00 c/s real, 5544.00 c/s virtual

Benchmarking: FreeBSD MD5 [32/64 X2]... DONE
Raw:    1024.00 c/s real, 1024.00 c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/64]... DONE
Raw:    72.02 c/s real, 72.02 c/s virtual

Benchmarking: Kerberos AFS DES [48/64 4K]... DONE
Short:  57344.00 c/s real, 57344.00 c/s virtual
Long:   157235.00 c/s real, 157235.00 c/s virtual

Benchmarking: NT LM DES [64/64 BS]... DONE
Raw:    1333644.00 c/s real, 1333644.00 c/s virtual


### 2 PROCESSORS ###
rlim@rcf ~/john-1.6.36/run% mpirun -np 2 ./john -test
Benchmarking: Traditional DES [64/64 BS]... DONE
Many salts:     327909.00 c/s real, 344081.00 c/s virtual
Only one salt:  308774.00 c/s real, 311576.00 c/s virtual

Benchmarking: BSDI DES (x725) [64/64 BS]... DONE
Many salts:     11239.00 c/s real, 11262.00 c/s virtual
Only one salt:  11110.00 c/s real, 11110.00 c/s virtual

Benchmarking: FreeBSD MD5 [32/64 X2]... DONE
Raw:    2042.00 c/s real, 2042.00 c/s virtual
```

```
Benchmarking: OpenBSD Blowfish (x32) [32/64]... DONE
Raw:    144.00 c/s real, 144.04 c/s virtual

Benchmarking: Kerberos AFS DES [48/64 4K]... DONE
Short:  114636.00 c/s real, 114750.00 c/s virtual
Long:   314368.00 c/s real, 314682.00 c/s virtual

Benchmarking: NT LM DES [64/64 BS]... DONE
Raw:    2666725.00 c/s real, 2664064.00 c/s virtual


### 8 PROCESSORS ###
rlim@rcf ~/john-1.6.36/run% mpirun -np 8 ./john -test
Benchmarking: Traditional DES [64/64 BS]... DONE
Many salts:     1319779.00 c/s real, 1377273.00 c/s virtual
Only one salt:  1222771.00 c/s real, 1245203.00 c/s virtual

Benchmarking: BSDI DES (x725) [64/64 BS]... DONE
Many salts:     44915.00 c/s real, 45084.00 c/s virtual
Only one salt:  43903.00 c/s real, 44412.00 c/s virtual

Benchmarking: FreeBSD MD5 [32/64 X2]... DONE
Raw:    8167.00 c/s real, 8182.00 c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/64]... DONE
Raw:    570.12 c/s real, 576.07 c/s virtual

Benchmarking: Kerberos AFS DES [48/64 4K]... DONE
Short:  449816.00 c/s real, 458984.00 c/s virtual
Long:   1251683.00 c/s real, 1258618.00 c/s virtual

Benchmarking: NT LM DES [64/64 BS]... DONE
Raw:    10633686.00 c/s real, 10676429.00 c/s virtual
```