Algoritma *Brute force* untuk Membongkar *Password* serta Estimasi Waktu Pemecahannya

Muhamad Rizky Yanuar - 13508015
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18015@students.if.itb.ac.id

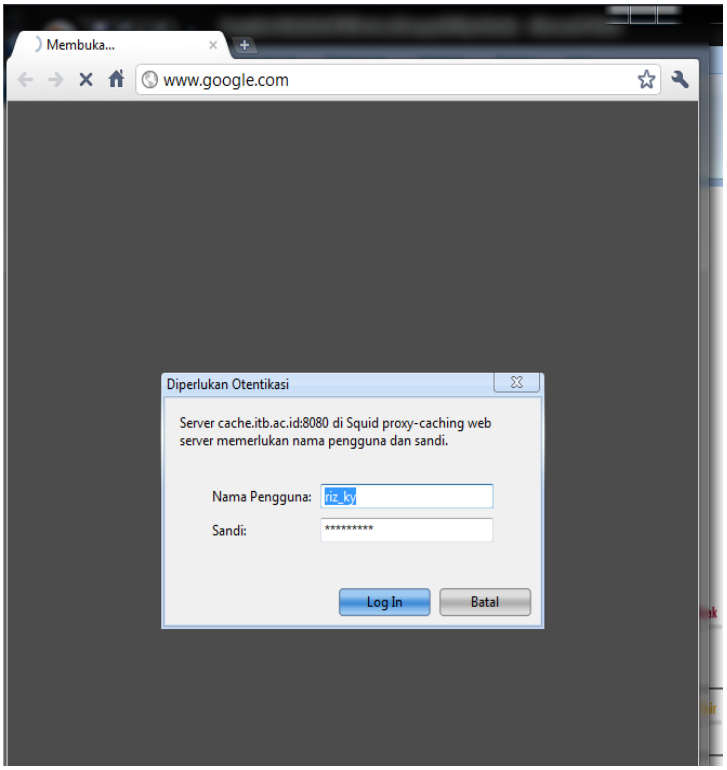
ABSTRAK
Keamanan suatu sistem merupakan hal yang sangat penting. Melihat berkembangnya kecerdasan *hacker*, bahkan *cracker*, pada zaman sekarang membuat keamanan suatu sistem makin rentan untuk ditembus. Hal tersebut bisa mengakibatkan penyalahgunaan dan mengakibatkan kerugian yang sangat besar. Salah satu sistem keamanan yang perlu diperhatikan lebih jauh yaitu pemilihan suatu kata sebagai *password.*

Kata kunci—password, brute force, password strength,

I. PENDAHULUAN

Password merupakan kombinasi karakter atau string yang berfungsi sebagai autentikasi pengguna dalam suatu sistem. *Password* ini biasanya memungkinkan seseorang untuk mendapatkan kekuasaan untuk mengakses suatu resource tertentu. *Password* sebaiknya tidak boleh diketahui oleh pihak luar karena bisa mengakibatkan penyalahgunaan. *Password* telah digunakan sejak dahulu sebagai kata kunci untuk OS, mesin ATM, *decoder TV* kabel, dan sebagainya. *Password* bisa pula tidak berbentuk kata yang memiliki makna, tapi berupa kumpulan karakter atau angka-angka. *Password* seperti itu biasa disebut juga dengan *passphrase*. Contoh *passphrase* adalah PIN dalam ATM.

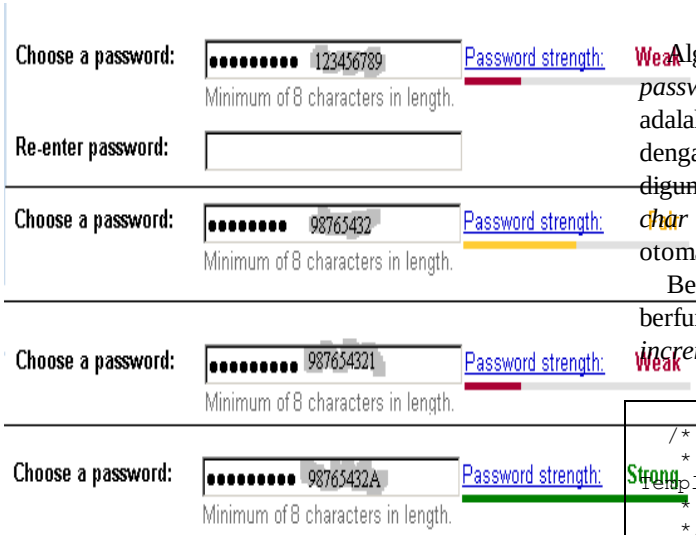
Penggunaan *password* sudah dilakukan sejak zaman dahulu. Dalam peperangan misalnya, digunakan *password* dan counter*password*. Contohnya saat terjadi peperangan *Normandy*, *paratroopers U.S.* menggunakan *password* "*thunder*", lalu dijawab dengan kata "*flash*" untuk mengidentifikasi anggota. *Counterpassword* juga selalu berubah secara berkala saat itu agar keamanan lebih terjamin. Selain itu *password* juga digunakan pada zaman-zaman awal komputer. CTSS buatan MIT, salah satu *sharing system* generasi awal, memiliki fitur *LOGIN* yang mengharuskan *user* untuk memasukan *password*. Dalam proses pengetikan *password*, sistem mematikan mekanisme printing sehingga *password* yang diketik tidak akan tertulis ke layar, sehingga memberikan privasi lebih pada *user* saat memasukan *password*. Berikutnya Robert Morris menemukan gagasan untuk menyimpan *login password* dalam bentuk hashed form sebagai bagian dari Unix *operating system*. Saat ini *password* sudah umum digunakan dalam berbagai aplikasi di sekitar kita.



Gambar 1 Contoh aplikasi yang memerlukan *password* sebagai autentikasi

II. KEKUATAN *PASSWORD*

Kekuatan *password* merupapakan kombinasi dari panjang *password*, kompleksitas *password*, dan tingkat keacakannya. Kekuatan dari suatu *password* biasanya berupa ketahan *password* tersebut dari serangan *brute force* untuk menebak *password* tersebut. Walaupun sebenarnya serangan terhadap suatu *password* tidak harus selalu melalui metode *brute force*. Salah satu metode lain adalah dengan mengetahui lebih jauh mengenai *user* bisa memberikan petunjuk untuk menebak-nebak *password user* yang bersangkutan. Hal tersebut sebaiknya dijadikan pertimbangan juga dalam pemilihan *password*.



Gambar 2 Contoh pengukur kekuatan *password* pada GMail

Password yang kuat bisa meminimalkan resiko terbobolnya keamanan sistem. Keefektifan dari suatu *password* tergantung dari desain dan implementasi dari sistem autentikasi. Berdasarkan sumbernya, *password* bisa tercipta dari dua hal, yakni secara otomatis oleh mesin atau secara manual oleh manusia. *Password* yang dibuat oleh manusia ada kecenderungan memiliki peraturan tertentu yang terkait dengan sang pengguna. Hal tersebut biasanya terjadi saat pembuatan akun di dalam sistem komputer ataupun saat pembuatan akun di dalam suatu *website*. Hal tersebut membuat *password* tersebut bisa lebih mudah ditebak. Bahkan program khusus yang berisi list kata-kata yang paling sering dipakai sebagai *password* telah dibuat, dan hasilnya diperkirakan 40% dari semua *password* yang mungkin ada bisa ditebak oleh program tersebut. Di sisi lain *password* yang dibuat secara otomatis bisa menghindari penebakan *password* melalui sisi personal *user*. *Password* tersebut hanya bisa ditebak secara bruteforce dan bisa dipastikan akan memakan waktu lebih lama untuk bisa ditebak. Namun *password* yang dibuat secara otomatis cukup rumit untuk dibuat generatornya dan cenderung lebih sulit untuk diingat.

III. *BRUTE FORCE* UNTUK MENGUJI KEKUATAN *PASSWORD*

Sebelumnya perlu diperhatikan jika dalam makalah ini, penulis hanya menjelaskan mengenai kekuatan *password* dengan algoritma *brute force* saja. Sebenarnya masih ada metode-metode lain untuk mengkalkulasikan kekuatan dari sebuah *password* namun akan dijelaskan pada lain kesempatan.

Kekuatan *password* ditentukan hanya dengan dari panjang *password* dan karakter yang dipakai dalam *password* tersebut. Karakter yang dipakai mencakup karakter kapital, karakter kecil dan angka 0 sampai dengan 9.

A. Algoritma Brute force untuk Memecahkan Password

Algoritma berikut akan mencoba untuk menebak *password* secara *brute force*. Bahasa yang digunakan adalah bahasa *java*. Algoritma *brute force* tersebut bekerja dengan cara menyimpan terlebih dahulu set karakter yang digunakan sebagai *password*, lalu menyediakan *array of char* untuk diisi oleh kata yang di-*generate* secara otomatis untuk dicocokan dengan *password* yang dicari.

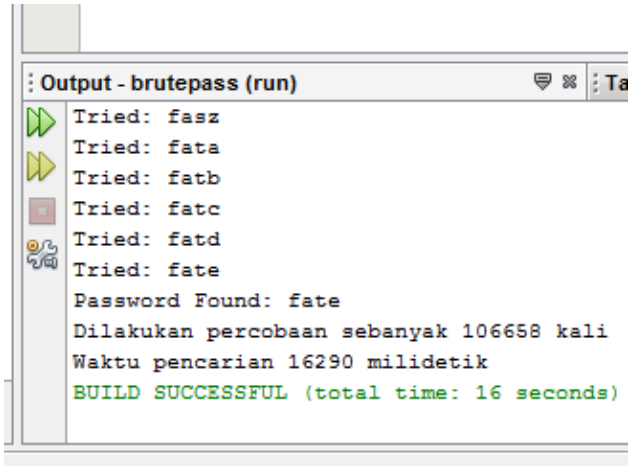
Berikut merupakan kelas *BruteForce.java* yang berfungsi untuk melakukan generate kata dengan *increment* per karakter anggota dari *character set*.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package brutepass;

import java.util.Arrays;

/**
 *
 * @author Gear Arcleife
 */
public class BruteForce {

    private char[] cs; // Character set
    private char[] cg; // Current Guess

public BruteForce(char[] characterSet, int guessLength)
{
    cs = characterSet;
    cg = new char[guessLength];
    Arrays.fill(cg, cs[0]);
}

public void increment()
{
    int index = cg.length - 1;
    while(index >= 0)
  {
      if (cg[index] == cs[cs.length-1])
    {
        if (index == 0)
      {
        cg = new char[cg.length+1];
        Arrays.fill(cg, cs[0]);
        break;
        }
      else
      {
        cg[index] = cs[0];
        index--;
        }
      }
    else
    {
        cg[index] = cs[Arrays.binarySearch(cs, cg[index]) + 1];
        break;
      }
    }
  }

public String PasstoString()
{
    return String.valueOf(cg);
  }
}
```

Berikut adalah kelas *main* dari program *java* tersebut. Proses iterasi terjadi pada kelas ini. Pada kelas ini juga dituliskan *character set* dari *password* serta *password* yang akan dicari.

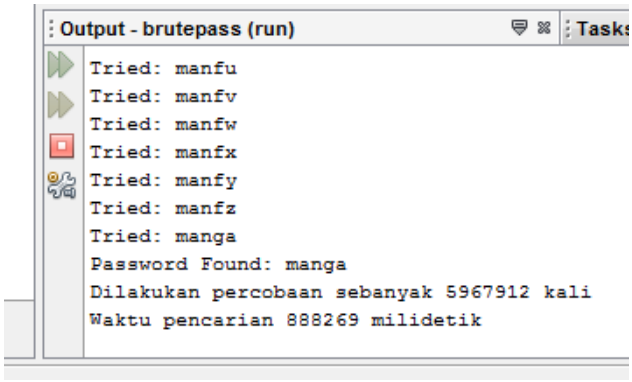
```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package brutepass;

/**
 *
 * @author Gear Arcleife
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        String password = "fate";
   char[] charset =   "abcdefghijklmnopqrstuvwxyz".toCharArray();
   BruteForce bf = new BruteForce(charset, 1);

    String katacoba = bf.PasstoString();
        long i=0;
        long timebefore=System.currentTimeMillis();
   while (true)
  {
    i++;
    if (katacoba.equals(password))
   {
    System.out.println("Password Found: " +   katacoba);
    System.out.println("Dilakukan percobaan sebanyak "+i+" kali");
    long timeafter=System.currentTimeMillis()-timebefore;
    System.out.println("Waktu pencarian "+timeafter+" milidetik");
    break;
    }
    katacoba = bf.PasstoString();
    System.out.println("Tried: " + katacoba);
    bf.increment();
   }
    }
}
```

Setelah proses selesai, jumlah percobaan, waktu pencarian dan *password* yang dicari akan ditampilkan di layar. Pada kelas main di atas, dicari kata "*fate*" sebagai *password* dan semua huruf alfabet kecil sebagai *character set*. Maka didapatkan hasil sebagai berikut.



Gambar 3 Hasil program untuk mencari kata "*fate*"

Berikut adalah contoh lain untuk mencari kata yang lebih panjang. *Password* yang dimasukkan adalah "*manga*", ditambah satu huruf menjadi 5 huruf.



Gambar 4 Hasil program untuk mencari kata "manga"

Dari kedua contoh diatas bisa dilihat bahwa menambah 1 huruf saja, perbedaan waktu pencarian bertambah cukup jauh menjadi 888 detik.

B. Mengkalkulasikan Waktu Pemecahan dengan Brute force

Dikarenakan tidak mungkin menghitung waktu pencarian dengan *brute force* melalui kode diatas secara langsung, maka estimasi waktu terburuk bisa dicari dengan rumus berikut.

Estimasi waktu = (*character set*$^{\text{Jumlah karakter}}$)/2800000 detik

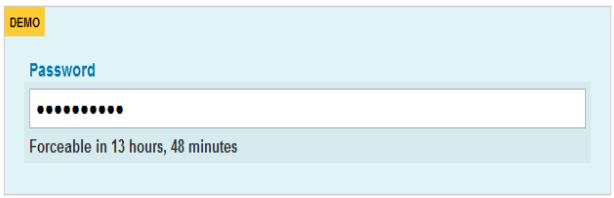
Pada rumus diatas diasumsikan pencarian *password* bisa dilakukan sebanyak 2800000 kali per detik dengan PC yang memakai software pendukung dan menggunakan *high-end graphic card*.

Character set bertambah sesuai jenis karakter yang masuk. Berikut adalah kode *jquery* untuk menghitung estimasi waktu pencarian *password* yang didapat dari suatu laman di internet. Dari kode tersebut dapat dilihat

bahwa dengan menambahkan karakter yang berbeda jenis seperti alfabet kapital atau angka akan menambah waktu pencarian jauh lebih lama.

```
(function($){
$.fn.extend({
pwdstr: function(el) {
return this.each(function() {



$(this).keyup(function(){
$(el).html(getTime($(this).val()));
});

function getTime(str){

var chars = 0;
var rate = 2800000000;

if((/[a-z]/).test(str)) chars +=  26;
if((/[A-Z]/).test(str)) chars +=  26;
if((/[0-9]/).test(str)) chars +=  10;
if((/[^a-zA-Z0-9]/).test(str)) chars +=  32;

var pos = Math.pow(chars,str.length);
var s = pos/rate;

var decimalYears = s/(3600*24*365);
var years = Math.floor(decimalYears);

var decimalMonths =(decimalYears-years)*12;
var months = Math.floor(decimalMonths);

var decimalDays = (decimalMonths-months)*30;
var days = Math.floor(decimalDays);

var decimalHours = (decimalDays-days)*24;
var hours = Math.floor(decimalHours);

var decimalMinutes = (decimalHours-hours)*60;
var minutes = Math.floor(decimalMinutes);

var decimalSeconds = (decimalMinutes-minutes)*60;
var seconds = Math.floor(decimalSeconds);

var time = [];

if(years > 0){
if(years == 1)
time.push("1 year, ");
else
time.push(years + " years, ");
}
if(months > 0){
if(months == 1)
time.push("1 month, ");
else
time.push(months + " months, ");
}
if(days > 0){
if(days == 1)
time.push("1 day, ");
else
time.push(days + " days, ");
}
if(hours > 0){
if(hours == 1)
time.push("1 hour, ");
else
time.push(hours + " hours, ");
}
if(minutes > 0){
if(minutes == 1)
time.push("1 minute, ");
else
time.push(minutes + " minutes, ");
}
if(seconds > 0){
if(seconds == 1)
time.push("1 second, ");
else
time.push(seconds + " seconds, ");
}

if(time.length <= 0)
time = "less than one second, ";
else if(time.length == 1)
time = time[0];
else
time = time[0] + time[1];

return time.substring(0,time.length-2);
}

});
}
});
})(jQuery);
```

Berikutnya akan dilihat hasil estimasi pencarian dengan beberapa masukan *string* yang berbeda.



Gambar 5 Hasil estimasi pencarian *password* dengan *brute force* untuk kata "thepainter"

Ternyata untuk mencari kata "*thepainter*" saja dibutuhkan waktu hingga 13 jam 48 menit. Berikutnya akan diuji kembali estimasi waktu untuk mencari kata dengan 1 karakter lebih banyak. Kata yang dicari adalah "thepainters".

Gambar 6 Hasil estimasi pencarian *password* dengan *brute force* untuk kata "thepainters"

Pencarian kata "*thepainters*" dengan program di atas memperkirakan waktu 14 hari 23 jam untuk memecahkan kata tersebut. Sekarang akan dicoba untuk mengkalkulasikan waktu pencarian untuk kata yang

mengandung huruf kapital, yaitu "*thepainterS*".

Gambar 7 Hasil estimasi pencarian *password* dengan *brute force* untuk kata "*thepainterS*"

Terbukti estimasi waktu pencarian untuk kata "*thepainterS*" jauh melebihi estimasi waktu pencarian "*thepainters*" sampai 85 tahun 1 bulan.

IV. KESIMPULAN

Berikut adalah kesimpulan yang bisa diambil dari analisis di atas.

1. Algoritma *brute force* sangat tidak efektif untuk mencari *password*.
2. Walaupun waktu yang dibutuhkan untuk mencari *password* dengan menggunakan *brute force* sangat mahal, namun *brute force* memastikan solusi dari pencarian pasti ditemukan.
3. *Password* yang aman adalah *password* yang didalamnya merupakan kombinasi dari berbagai jenis karakter berupa angka, alfabet kecil, alfabet kapital, maupun simbol karena akan lebih susah untuk diprediksi.

REFERENCES

[1] http://en.wikipedia.org/wiki/*Password*_strength
[2] http://en.wikipedia.org/wiki/*Password*
[3] http://www.unwrongest.com/projects/*password*-strength/
[4] http://www.ajaxupdates.com/*password*-strength-estimates-brute-force-time-jquery-plugin/
[5] http://www.hellboundhackers.org/forum/brute_force_algorithm_*java*_brute_forcer-22-11469_20.html

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2010

ttd

Muhamad Rizky Yanuar - 13508015