APLIKASI ALGORITMA PENCOCOKAN STRING UNTUK PENGAMANAN JARINGAN

Andrew Pratomo Budianto

Institut Teknologi Bandung
Bandung, Indonesia
e-mail: andrewpratomo@yahoo.com

ABSTRAK

Jaringan yang bebas dari penyusup merupakan salah satu prasyarat sebuah jaringan dikatakan aman dan layak digunakan sebagai media pengiriman data. Seiring dengan meningkatnya jumlah pengguna teknologi jaringan, jumlah para penyusup yang ingin mengintrusi jaringan-jaringan tersebut juga turut meningkat. Pada akhirnya, performa dari sistem untuk mendeteksi penyusup pada jaringan tersebut menjadi sebuah hal yang sangat menarik dan penting untuk dikaji lebih dalam. Performa sistem pendeteksi yang terdahulu menyebabkan kecepatan pengiriman data pada jaringan menjadi *bottleneck* pada sistem deteksi intrusi yang digunakan. Salah satu solusi brilian dalam menyelesaikan permasalahan ini adalah dengan menggunakan algoritma pencocokan string yang mangkus untuk mendeteksi terjadinya anomali pada pengiriman data pada jaringan tersebut. Contoh konkret dari penggunaan algoritma string matching pada pengamanan jaringan adalah sebuah aplikasi yang dikenal dengan nama Snort. Snort bekerja dengan menggunakan teknik inspeksi data yang mengalir pada jaringan. Pada tulisan ini, akan dibahas bagaimana cara kerja Snort dengan algoritma pencocokan string Boyer-Moore, dan efektifitasnya dalam pengamanan jaringan.

Kata kunci: Boyer-Moore, deteksi instrusi, keamanan jaringan, pencocokan *string*, Snort.

1. PENDAHULUAN

Seperti yang telah kita ketahui, semakin meningkatnya jumlah pemakaian teknologi jaringan, makin dibutuhkan algoritma yang mangkus dalam berbagai aspek yang berhubungan dengan jaringan tersebut.

Pencocokan *string* pada tiap paket yang mengalir dalam jaringan diperlukan pada aplikasi-aplikasi intrusi deteksi, seperti *firewall*, *firewall* akan mengecek dengan pencocokan string apakah terdapat virus yang lewat, pada layer tujuh, dilakukan *web load balancing* berbasis URL, *cookie*, maupun pembatasan content yang diakses. Berlatar belakang masalah tersebut, dapat diasumsikan bahwa pencocokan string pada isi paket merupakan masalah yang menarik untuk didiskusikan.

Sebuah paket jaringan pada umumnya harus periksa kecocokannya dengan ratusan *string*. Disinilah kelebihan dari algoritma pencocokan *string* diperlukan. Bagaimanapun aturan deteksi intrusi berbeda dari kamus bahasa pada umumnya. Paket yang harus diperiksa muncul dalam ukuran yang sangat berbeda, kedua tiap aturan dapat menspesifikasikan *string* yang berbeda, dan tiap *string* juga tidak memiliki aturan *case-sensivity* yang berbeda-beda.

Organisasi makalah ini kurang lebih dapat dijelaskan sebagai berikut. Pada bab pendahuluan akan dijelaskan mengapa penggunaan algoritma yang mangkus dalam pencocokan *string* amat diperlukan dalam dunia jaringan komputer, pada bab dua dijelaskan algoritma yang umum digunakan sebagai algoritma pencocokan *string* terhadap paket data yang melewati jaringan, dilanjutkan pada bab 3 yang membahas tentang contoh konkret aplikasi yang terdapat untuk menangani masalah ini. Pada bagian akhir terdapat kesimpulan dari seluruh isi makalah ini, beserta daftar pustaka yang digunakan.

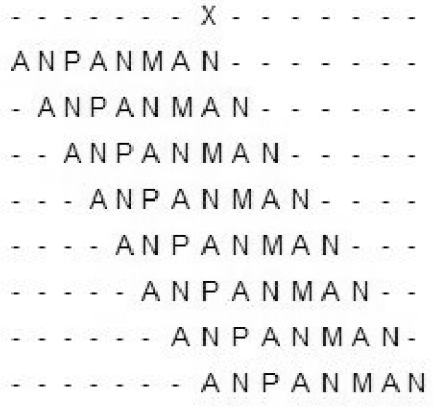
2. ALGORITMA BOYER-MOORE

Algoritma pencarian *string* Boyer-Moore merupakan algoritma pencocokan *string* yang dapat dikatapakan paling efektif. Ditemukan oleh Bob Boyer dan J. Strother Moore pada tahun 1977. Algoritma dimulai dengan memproses *string* yang tempat pencarian (bukan *string* yang ingin dicari, hal ini berbeda dengan algoritma pencocokan *string* pada umumnya, dimana proses dimulai dari *string* yang ingin dicari, baru kemudian memproses tempat pencarian *string*). Waktu eksekusi dari algoritma Boyer-Moore dapat dikatakan mendekati *sub-linear*, karena algoritma ini tidak perlu memeriksa semua karakter dalam *string* untuk dicari, namun akan melewati beberapa diantaranya. Pada umumnya, algoritma ini akan semakin cepat apabila *string* kunci yang ingin dicari semakin panjang [4]. Kemangkusan dari algoritma ini diturunkan dari fakta bahwa, tiap usaha pencarian yang gagal antara *string* kunci dan tempat pencarian, akan

menggunakan informasi yang didapat dari percobaan pada posisi-posisi yang lain.

2.1 Cara kerja

Hal yang sering mengejutkan bagi orang yang pertama kali melihat algoritma ini adalah dari segi verifikasinya. Algoritma ini mencoba memeriksa, apakah terdapat kesamaan pada sebuah posisi, mulai dari karakter paling belakang [1]. Sebagai contoh, pada string "ANPANMAN" apabila kita memulai pencocokan dari depan maka, kita akan memeriksa apakah posisi ke delapan mengandung "N", apabila "N" ditemukan, maka pencocokan akan dilanjutkan ke posisi ke tujuh untuk memeriksa apakah terdapat huruf "A", dan begitu pula untuk seterusnya dilakukan pengecekan "A" sampai pada posisi awal.



Gambar 1. Verifikasi pada algoritma Boyer-Moore

Alasan mengapa Boyer-Moore memulai pencocokan dari belakang terlihat lebih jelas apabila kita membayangkan apabila sebuah verifikasi gagal, sebagai contoh, apabila kita menemukan "X", dan bukan "N" pada posisi ke delapan. "X" tidak muncul pada posisi manapun pada *string* "ANPANMAN", dan hal itu berarti tidak terdapat *string* "ANPANMAN" pada posisi paling awal dari teks maupun tujuh posisi kedepan berikutnya, karena seluruh posisi tersebut tidak dapat memberikan "X". Setelah melakukan pengecekan hanya pada sebuah karakter, kita dapat melewati karakter-karakter berikutnya dan mulai mencari kecocokan mulai pada posisi ke sembilan dari teks, tepat setelah "X".

Hal ini menjelaskan mengapa performa terbaik dari algoritma ini untuk sebuah teks sepanjang N karakter dan sebuah pola dengan panjang M adalah N/M, pada kasus terbaik, hanya sebuah M karakter yang perlu diperiksa kecocokannya. Hal ini juga menjelaskan mengapa bila pola yang dicari makin panjang, maka algoritma ini dapat menghitung makin cepat [4].

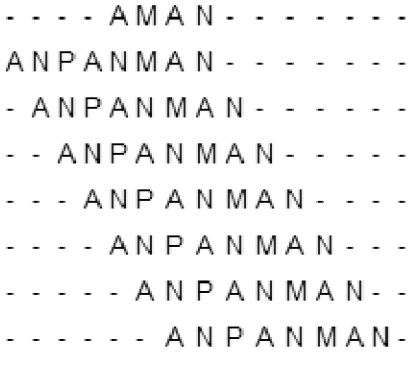
Algoritma ini menghitung terlebih dahulu dua buah tabel untuk memproses informasi tiap kesalahan verifikasi, tabel pertama digunakan untuk menghitung berapa banyak posisi kedepan yang dapat diloncati tiap ditemukannya sebuah kesalahan pencocokan, tabel berikutnya menghitung berapa banyak karakter telah sukses dicocokan sebelum terjadi kegagalan pencocokan berikutnya (Karena dua buah tabel ini berguna untuk menghitung seberapa jauh loncatan sebuah pencocokan, maka tabel ini biasa disebut tabel loncatan) [4].

Tabel pertama lebih mudah dihitung, mulai dari karakter terakhir *string* kunci, bergerak kedepan sampai pada karakter pertama. Tiap kali bergerak kedepan, apabila karakter yang ditemukan belum terdapat pada tabel, tambahkan karakter tersebut dan beri nilai yang berisi jaraknya dari karakter paling kanan. Karakter yang lain dihitung sama dengan panjang dari *string* kunci tersebut [4].

Tabel 1. Prekomputasi dari algoritma Boyer-Moore

Karakter	Shift
N	0
A	1
M	2
P	5
Semua karakter yang lain	0

Tabel kedua sedikit lebih sulit untuk dihitung, untuk tiap nilai *i* yang kurang dari panjang string pencarian, kita harus menghitung terlebih dahulu pola yang berakhiran *i* buah karakter dari *string* pencarian, diikuti oleh kesalahan karakter sebelumnya, lalu kita menginisialisasi dengan pola pencarian dan menentukan jumlah karakter paling sedikit yang harus digeser kekiri sebelum dua buah pola bersesuaian [4].



Gambar 2. Kesalahan "A" pada posisi 5 menyisihkan 6 kemungkinan yang lain,

Sebagai contoh, untuk *string* pencarian ANPANMAN, tabel akan menjadi seperti ini:

Tabel 2. Prekomputasi dari algoritma Boyer-Moore

i	Pola	Shift
0	~~N~~	1

1	AN	8
2	MAN	3
3	NMAN	6
4	ANMAN	6
5	PANMAN	6
6	NPANMAN	6
7	ANPANMAN	6

Jumlah shift yang dihitung oleh tabel kedua biasa disebut "*good suffix shift*". Tabel yang sebenarnya ditemukan oleh Boyer-Moore masih menggunakan versi aturan *good suffix* yang lebih sederhana dan lemah dimana tiap entri dari tabel di atas tidak menggunakan *mis-match* untuk karakter paling kiri. Aturan ini biasa disebut "*weak good suffix rule*" dan tidak cukup untuk membuktikan bahwa Boyer-Moore berjalan dengan performa *linear* pada kasus terburuk.

Berikut di bawah ini adalah contoh kode lengkap penggunaan algoritma Boyer-Moore dalam bahasa C:

```
void preBmBc(char *x, int m, int bmBc[]) {
   int i;

   for (i = 0; i < ASIZE; ++i)
      bmBc[i] = m;
   for (i = 0; i < m - 1; ++i)
      bmBc[x[i]] = m - i - 1;
}


void suffixes(char *x, int m, int *suff) {
   int f, g, i;

   suff[m - 1] = m;
   g = m - 1;
   for (i = m - 2; i >= 0; --i) {
      if (i > g && suff[i + m - 1 - f] < i - g)
         suff[i] = suff[i + m - 1 - f];
      else {
         if (i < g)
            g = i;
         f = i;
         while (g >= 0 && x[g] == x[g + m - 1 -
f])
            --g;
         suff[i] = f - g;
      }
   }
}

void preBmGs(char *x, int m, int bmGs[]) {
   int i, j, suff[XSIZE];

   suffixes(x, m, suff);

   for (i = 0; i < m; ++i)
      bmGs[i] = m;
   j = 0;
   for (i = m - 1; i >= 0; --i)
      if (suff[i] == i + 1)
         for (; j < m - 1 - i; ++j)
            if (bmGs[j] == m)
               bmGs[j] = m - 1 - i;
   for (i = 0; i <= m - 2; ++i)
      bmGs[m - 1 - suff[i]] = m - 1 - i;
}


void BM(char *x, int m, char *y, int n) {
   int i, j, bmGs[XSIZE], bmBc[ASIZE];

   /* Preprocessing */
   preBmGs(x, m, bmGs);
   preBmBc(x, m, bmBc);

   /* Searching */
   j = 0;
   while (j <= n - m) {
      for (i = m - 1; i >= 0 && x[i] == y[i + j];
--i);
      if (i < 0) {
         OUTPUT(j);
         j += bmGs[0];
      }
      else
         j += MAX(bmGs[i], bmBc[y[i + j]] - m + 1
+ i);
   }
}
```

2.2 Performa algoritma pencocokan string Boyer-Moore

Performa untuk kasus terburuk untuk menghitung seluruh kecocokan kurang lebih N*M. Kasus terburuk ini terjadi ketika *string* yang ingin dicari mengandung perulangan sebuah karakter. Pada skenario ini, algoritma harus memeriksa N-M+1 *offset* yang berbeda pada teks untuk kesamaan, dan tiap pengecekan membutuhkan M buah perbandingan.

Kasus terbaik untuk menemukan seluruh kemunculan pada teks membutuhkan kurang lebih 3*N perbandingan, dengan kompleksitas O(n), baik teks yang diberikan mengandung *string* yang ingin dicari atau tidak. Pembuktiannya memerlukan beberapa tahun sebelum ditemukan. Pada tahun algoritma ini ditemukan, 1977, jumlah perbandingan maksimum yang dilakukan tidak lebih dari 6*N, pada tahun 1980, jumlah perbandingan maksimum tersebut ternyata masih dapat diperbaiki menjadi 4*N [6].

2.3 Varian algoritma pencocokan string Boyer-Moore

Algoritma Boyer-Moore Turbo membutuhkan lebih banyak waitu konstan untuk menelesaikan pencarian dalam 2n perbandingan (3n dengan Boyer-Moore standar).

Algoritma Boyer-Moore-Horspool merupakan simplifikasi dari algoritma Boyer-Moore dengan menghapus tabel kedua. Algoritma Boyer-Moore Horspool membutuhkan M*N perbandingan, sementara algoritma Boyer-Moore membutuhkan hanya 3*N perbandingan.

3. SNORT DAN PENCOCOKAN STRING

3.1 Penjelasan singkat tentang Snort

Snort adalah sistem deteksi intrusi jaringan yang tersedia secara gratis (dibawah lisensi GNU) dan terkonfigurasi dengan aturan-aturan yang mendefinisikan string kunci dan deskripsinya masing-masing. Snort juga memiliki arsitektur *plug-in* yang memungkinkan proses stateful analysis dilakukan. Karena tersedia dengan gratis, Snort cukup umum digunakan dan sangat banyak database *string* kunci yang diatur dalam dunia maya [5].

```
alert tcp !$HOME_NET any -> $HOME_NET 80
(msg:"IDS219 - WEB-CGI-Perl access attempt";
flags:PA; content:"perl.exe"; nocase;)
```

Gambar 3. Contoh aturan Snort

Pada gambar 2, diberikan contoh sederhana dari Snort. Tiap aturan mengandung aksi (*alert* pada contoh di atas), protokol (*tcp*), netmask sumber (didefinisikan sebagai seluruh sumber kecuali jaringan yang sedang dimonitor), port sumber (semua *port*), dan *netmask* dan *port* tujuan (80).

3.2 Performa Snort

Seperti telah disebutkan sebelumnya, pencocokan string merupakan komputasi paling banyak yang dilakukan oleh Snort. Performa aktual Snort sangat bergantung pada aturan yang diberikan dan karakteristik dari jaringan yang sedang dimonitor. Percobaan dilakukan dengan menggunakan Snort 1.6.3 yang dijalankan dengan dataset 8.7 juta paket, dengan ukuran keseluruhan sebesar 1 gigabyte selama 25 menit dari koneksi Internet perusahaan besar dengan pengguna berasal dari pebisnis dan peneliti. Menggunakan seluruh aturan yang diberikan dari Snort.org, terlihat bahwa pencocokan *string* merupakan bagian yang paling mahal dari eksekusi dan memakan sebesar 31% dari total waktu eksekusi [2,7].

Purpose	Routine	Portion
String match	mSearch	31%
Packet classification	EvalHeader	8.5%
Packet classification	CheckSrcIPNotEq	6.7%
Other matching	EvalOpts	5.8%

Gambar 4. Performa Snort

Salah satu optimisasi yang umum adalah menggunakan *kernel* atau *hardware prefilter* untuk menyisihkan paket-paket yang tidak diinginkan oleh sistem deteksi. Beberapa *prefilter* hanya akan meningkatkan jumlah pecahan paket untuk dilakukan deteksi [3].

3.3 Kelemahan algoritma

Seperti yang kita ketahui, tiap algoritma memiliki batasan performa untuk kasus terbaik dan kasus terburuknya masing-masing, tak terkecuali algoritma pencocokan *string*. Umumnya, penyusup akan memasukkan paket yang menyebabkan terjadinya kasus terburuk terus menerus, sehingga sistem deteksi akan kelebihan muatan. Pada saat ini, penyusup dapat mengirim paket yang bebas dari sistem deteksi. Untuk mengatasi masalah ini, bagaimanapun performa dari algoritma deteksi intrusi harus dikurangi, dan lebih fokus terhadap penanganan *algorithmic performance vulnerability*[2].

IV. KESIMPULAN

Perkembangan teknologi dilihat dari satu sisi dapat memberikan kemudahan dalam pembuatan suatu program, dengan algoritma dan permasalahan yang sama, waktu eksekusi yang diperlukan pada teknologi masa kini sudah jauh lebih cepat daripada masa-masa awal ditemukannya teknologi komputer. Namun, dilihat dari sudut pandang yang berbeda, peningkatan teknologi, menyebabkan bertambahnya para pengguna teknologi, yang terkadang terlampau cepat, melebihi perkembangan teknologi itu sendiri. Terbatasnya teknologi perangkat keras memaksa kita untuk menggunakan perangkat keras yang sama dan menekankan pada kemangkusan algoritma yang digunakan guna menyelesaikan sebuah masalah.

Contoh konkrit dari masalah diatas adalah penggunaan algoritma pencocokan *string* ini, pada adalnya, permasalahan deteksi intrusi pada jaringan sudah dapat diselesaikan dengan algoritma pencocokan string biasa, namun seiring waktu, akhirnya algoritma Boyer-Moore digunakan. Pada saat ini, algoritma tersebut masih memiliki kelemahan pada kasus terburuknya. Dengan demikian pada dasarnya, permasalahan deteksi intrusi pada jaringan masih harus dikaji lebih jauh lagi.

REFERENSI

[1] Rinaldi Munir, "Strategi Algoritmik", Institut Teknologi Bandung, 2007.

[2] Mike Fisk dan George Varghese, "Applying Fast String Matching to Intrusion Detection", http://woozle.org/.

[3] Yaron Weinsberg, Shimrit Tzur-David, Danny Dolev, dan Tal Anker, "High Performance String Matching Algorithm for a Network Intrusion Prevention System (NIPS)", http://www.cs.huji.ac.id/.

[4] "Wikipedia.org", http://en.wikipedia.org/wiki/Boyer-Moore_string_search_algorithm

[5] "Snort.org", http://snort.org/.

[6] G. Stephen, String Searching Algorithms, World Scientific, 1994.

[7] D. Knuth, J. Morris, and V. Pratt, "Fast pattern matching in strings," SIAM Journal on Computing, vol. 6, no. 2, pp. 323-50, June 1977.