String Matching dalam Pendeteksian Intrusi

Author Jacqueline and 13512074[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512074@std.stei.itb.ac.id

***Abstrak*—Pendeteksian intrusi adalah cara atau alat yang bagus untuk mengidentifikasi serangan pada jaringan. Inti dari hal ini ialah melakukan pencarian pada paket-paket dan mengidentifikasi konten yang sesuai dengan yang diketahui sebagai serangan. Keefisienan penggunaan string matching sangat bergantung pada penggunakan ruang dan waktu.**

***Index Terms*—pendeteksian intrusi, string matching.**

I. PENDAHULUAN

Semakin hari, pengaksesan data pada jaringan menjadi semakin kritis. Sistem apa saja yang dapat diakses kini menjadi sasaran empuk untuk diserang. Bentuk penyerangannya beragam, seperti virus; pengambilalihan server; pengintaian; dan lain-lain. Serangan layanan ini memang tidak langsung menembus sistem, tapi kerusakan yang dihasilkan bisa sebanding dengan penyerangan oleh manusia. Bermunculan ketertarikan untuk menghalau serangan ini di berbagai level, mulai dari host akhir, tap jaringan, hingga core router.

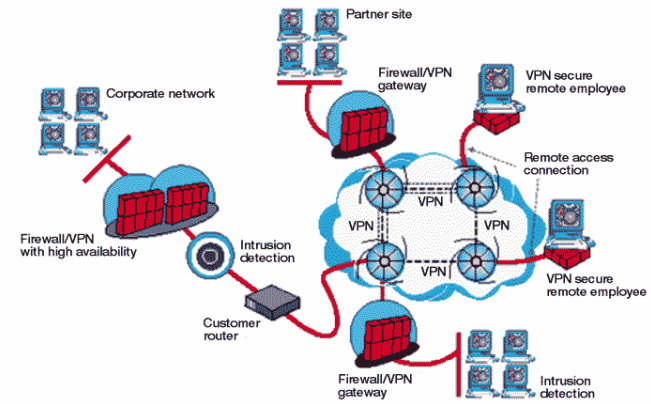
Salah satu cara yang paling menjanjikan adalah pendeteksian intrusi. Ini memonitor lalu lintas jaringan secara berkala. Sehingga, jika ditemukan aktivitas yang mencurigakan, dapat mengirimkan peringatan kepada admin. Hal ini juga berlaku jika terdeteksi worm baru atau penolakan pada serangan terhadap layanan. Selain itu, log aktivitas yang terjadi juga akan tersimpan.

II. STRING MATCHING PADA PENDETEKSIAN INTRUSI

Ada beberapa hal yang menjadi highlight dalam pendeteksian intrusi dengan prinsip string matching.

A. Tolak Ukur Penggunaan String Matching

Bagian terpenting dalam pendeteksian intrusi adalah string matching itu sendiri. String matching merupakan proses pencocokan untai pola yang ada terhadap teks. Prinsip dasarnya, mencocokkan pattern dengan teks pada tempat yang bersesuaian. Jika berbeda, dilakukan penggeseran. Hal ini dilakukan berulang hingga akhir.



Gambar 1. Pendeteksian Intrusi
sumber: www.whitehelm.com

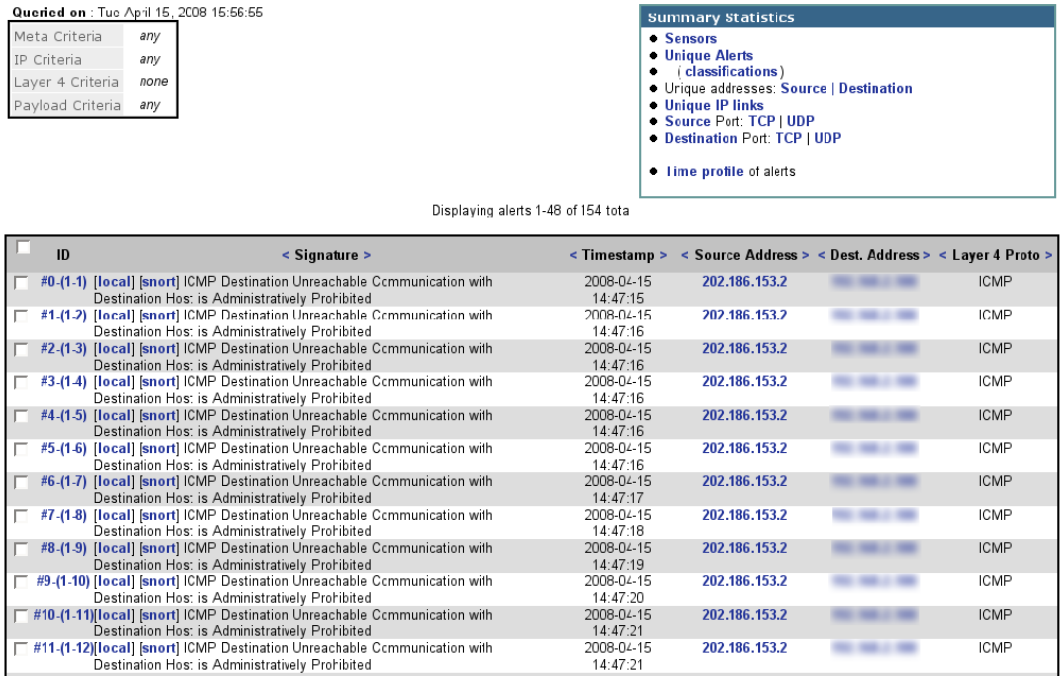
Pendeteksian intrusi memiliki beberapa alat bantu, yang pertama disebut Snort.

Snort menggunakan himpunan yang berisi aturan-aturan yang diturunkan dari serangan-serangan yang ada atau hal lain yang dicurigai. Konten yang relevan pada untai diekstraksi dari header serangan yang diketahui. Akan dicek apakah dalam berbagai kondisi memenuhi aturan yang ada. Untai dicocokkan, menetapkan lokasi pada paket, dan lain-lain. Kemudian, aksi dilakukan bersesuaian dengan aturan tersebut. Pilihannya, yaitu membuat log paket, mengabaikan paket, memberi pesan peringatan kepada admin sistem (lewat email), atau mengaktivasi aturan lain secara dinamis.

Distribusi Snort menggunakan sebuah himpunan aturan-aturan yang mampu meng-cover serangan-serangan yang telah dikenal seperti buffer overflow atau sebaran eksploitasi. Aturan-aturan ditambahkan ke Snort jika kelemahan tersebut ditemukan. Masing-masing aturan menyimpan untai konten, berasosiasi dengan aturan pada lokasinya, dan tipe dari paket yang ia kontrol.

Kemudian, ada Penderajatan dari Basisdata Pendeteksian Intrusi. Himpunan aturan yang telah dipaparkan didukung oleh pendeteksian intrusi dan disimpan dalam basisdata. Distribusi standar dari Snort melebihi 1500 aturan. Aturan-aturan tersebut dapat mencocokkan karakter yang bukan untai huruf, seperti

alamat IP. Banyak aturan yang sangat panjang. Hal ini menyebabkan sangat penting untuk menghindari algoritma string matching yang dalam tekniknya perlu dilakukan secara berkala yang panjangnya hampir sesuai dengan panjang dari aturan itu sendiri.



Gambar2. Contoh Tampilan Snort
sumber: eyereex.eryell.com

Dalam menanggapi serangan baru yang ditemukan, aturan baru ditambahkan pada basisdata Snort. Ukuran dari aturan di basisdata Snort dikarakteristikkan dengan jumlah untai uniknya dan total byte. Jika hanya menggunakan algoritma dengan teknik simpel, yakni pencarian linear, tentu tidak dianggap baik. Basisdata akan tumbuh baik jika menggunakan Hukum Moore (menambahkan ukuran aturan di basidata sedemikian sehingga sangat jauh dari penambahan penghitungan transistor di chip). Dibutuhkan teknik yang berjalan secara berkala, tetapi mandiri terhadap ukuran aturannya di basidata pula. Contoh algoritma yang seperti ini adalah Aho-Corasick. Biaya konstruksinya diabaikan karena pembangunan struktur data terhitung cepat dan hanya dipanggil saat memproses himpunan aturan baru, pendeteksiannya dinyalakan ulang, atau saat beberapa aturan dinamis berjalan.

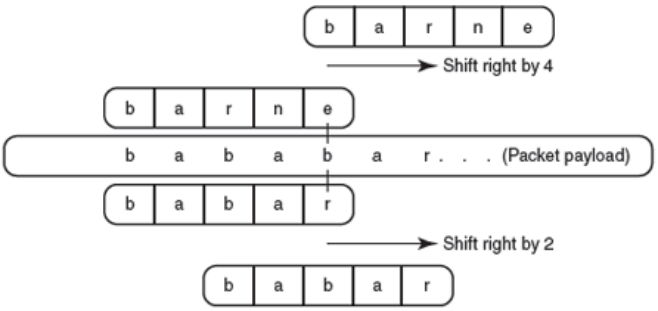
B. Algoritma dalam String Matching

Tidak seperti masalah umum pada string matching, pendeteksian intrusi menggunakan multi-pattern, bukan single-pattern. Pada single-pattern, dikenal algoritma seperti Knuth-Morris-Pratt dan Boyer-Moore. Area kerjanya juga luas karena hanya menggunakan pola tunggal dalam pencarian di teks. Biasanya, yang seperti ini digunakan dalam pengedit kata atau sistem DNA.

Masalah string matching yang ditangani dengan multi-pattern adalah pencarian di teks oleh himpunan untai-untai. Single-pattern dapat dibuat menjadi multi-pattern. Caranya, gunakan pattern tunggal tersebut dalam teks yang ingin dicari pada tiap pattern yang akan digunakan.

Multi-pattern secara umum akan melakukan praproses dari himpunan untai masukan. Kemudian, mencarinya bersamaan di teks.

Pencarian dengan multi-pattern dapat diaplikasikan lewat algoritma Aho-Corasick, Commentz-Walter, Wu-Manber, dan lain-lain. Ada pula penyamaan string menggunakan prinsip hashing dan teknik signature based. Namun, penggunaan algoritma-algoritma ini terkadang perlu diverifikasi ulang dengan algoritma string matching yang persis.



Gambar 3. Contoh Boyer-Moore
sumber: www.searchitchannel.techtarget.com

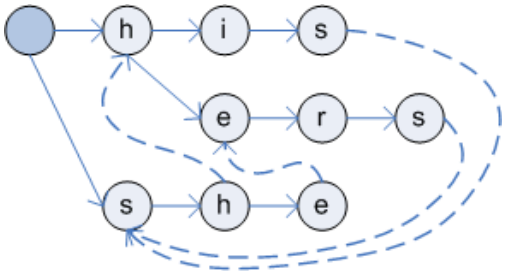
Dengan demikian, penggunaan yang persis dan tidak persis sama-sama penting. String matching yang tidak presisi (imprecise) memberikan kemungkinan data stream sampai pada kesalahan positif yang sekuensial.

Berikut ini adalah beberapa algoritma yang menggunakan multi-pattern dalam pencarian untainya dan ada dalam versi terkini Snort.

1. Heristik Karakter Buruk

Heristik karakter buruk ini sudah dikenal dalam algoritma string matching Boyer-Moore. Diberikan single-pattern sepanjang n untuk dicocokkan, dapat bergerak sejauh n karakter pula. Jika karakter pada posisi tersebut tidak sama dengan karakter pattern yang bersesuaian, pointer untuk pencarian dipindah sejauh n+1 karakter. Jika karakter yang dicari ada di string, tapi bukan karakter terakhir dari string pencarian, dapat dilompati sejauh mungkin namun instansiasinya tidak terlewat.

2. Aho-Corasick



Gambar 4. Contoh Aho-Corasick
sumber: codeproject.com

Algoritma Aho-Corasick ini merupakan salah satu yang pertama yang menggunakan multi-pattern. Ia mampu meng-handle kasus terburuk pada string matching yang berkaitan dengan waktu linear dan ukuran dari masukan. Algoritma ini mengonstruksi mesin status dari string untuk dicocokkan. Mesin status diawali dengan simpul akar kosong yang merupakan status default nonmatching. Tiap pattern yang akan dicocokkan ditambahkan status ke mesin, mulai dari akar hingga ke akhir pattern.

Berikut ini adalah data Aho-Corasick yang dioptimasi.

```
state_aho {
struct state_aho * next_state[256];
struct rule * rule_list;
};
```

Data Aho-Corasick yang tidak dioptimasi.

```
struct aho_state {
struct aho_state * next_state[256];
struct rule * rule_list;
struct aho_state * fail_ptr;
};
```

Mesin status ditaversal dan pointer kesalahan ditambahkan dari tiap simpul ke prefiks terpanjang pada simpul tersebut yang juga mengantarkan ke simpul yang valid.

3. Pencarian SFK

Pencarian SFK digunakan jika Snort dalam kondisi minim memori. Algoritma ini membangun pohon dengan simpul-simpul pada level yang bersesuaian. Setiap level merupkan sekuensial list dari simpul yang bertetangga, mengandung pointer ke aturan yang cocok. Karakter harus dicocokkan secara traversal dengan simpul anak serta pointer ke simpul tetangga yang selanjutnya. Algoritma pencarian SFK ini melakukan penggeseran karakter buruk hingga menemukan tempat mulai yang memungkinkan untuk mencocokkan untai, kemudian dilakukan traversal. Jika terdapat kecocokan antara karakter di simpul yang sedang dilihat dengan karakter di paket, akan menuju anak lewat pointernya dan melakukan inkremen pointer paket karakternya. Jika tidak, pointer tetangga ditelusuri hingga mencapai akhir list. Ketika pencocokan gagal, dilakukan runutbalik.

4. Wu-Manber

Algoritma ini awalnya melakukan prakomputasi dua tabel, tabel penggeseran karakter buruk dan hash. Ketika penggeseran karakter buruk gagal, dua karakter pertama dari untai diindekskan ke tabel hash untuk mencari list dari pointer yang mungkin untuk pencocokan pattern. Pattern ini dibandingkan untuk mencari jika ada kecocokan, kemudian masukan digeser satu karakter ke kanan dan proses tersebut diulang-ulang.

C. Penggunaan Teknik IP Lookup dalam String Matching

Secara sekilas, banyak kesamaan dalam masalah-masalah pencarian IP dengan string matching. Yang paling mirip adalah mengenai kesamaan awalan yang terpanjang.

Pencarian IP dipakai sebagai masukan sebuah himpunan pattern untuk dicocokkan dan mencari kecocokan terpanjang yang memungkinkan. String matching sebagai masukan dari himpunan string untuk mencari seluruh lokasi kecocokan yang terjadi.

Algoritma modern bergantung pada prapemprosesan dari himpunan masukan pola atau untai yang terbatas. Ini juga membangun struktur data mirip pohon yang besar untuk traversal secara berkala dan derajat yang tinggi pada tiap sisi.

Banyak optimasi yang dapat mempercepat pencarian IP dan memiliki kesamaan dengan string matching. Berikut ini merupakan algoritma-algoritma yang berangkat dari pencarian IP.

1. Algoritma Lulea

Algoritma ini menggunakan konsep memasukkan daun dan mengompres bitmap ke ukuran basisdata pencarian IP. Dengan memasukkan informasi routing ke daun, dapat mencegah simpul internal memiliki informasi routing. Pemasukan daun secara klasik membuat simpul daun

terduplikasi. Oleh sebab itu, kemudian direduksi dengan bitmap yang dikompres. Untuk mengetahui routing mana yang informasinya sesuai dengan daun, dilakukan penghitungan himpunan bit utama pada indeks bitmap. Untuk menambah efisiensi jika mengolah bitmap berukuran sangat besar, algoritma ini merangkum hitungan posisi bit pada data bitmapnya.

2. Algoritma Eatherton

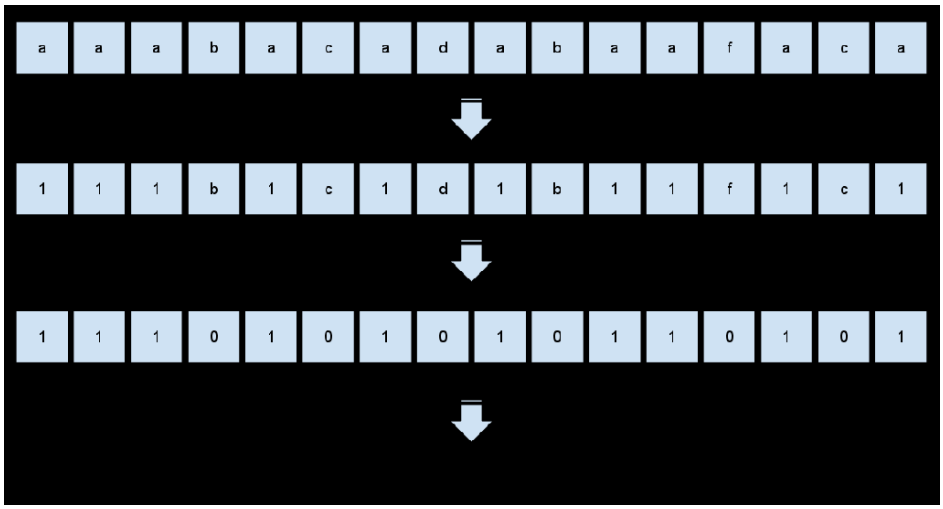
Algoritma pohon bitmap Eatherton merupakan pengembangan dari Lulea. Yang ia lakukan adalah membuat sebuah struktur data yang tidak membutuhkan pemasukan daun dan dapat ditraversal dengan sebuah memori yang tunggal, diakses per simpul. Untuk setiap implementasi pohon bitmap untuk traversal n bit pada satu waktu, simpul-simpulnya mengandung dua bitmap. Internal bitmap berukuran 2n dan eksternalnya 2n-1, sebagai pointer ke array dari anak dan pointer ke array dari aturan pencocokan.

III. PENGOPTIMASIAN UNTUK STRING MATCHING

Dari keadaan dan kebutuhan yang telah dipaparkan, sangat penting mengoptimalisasi penggunaan string matching dalam mendeteksi intrusi. Ada cara penyimpanan data yang baik untuk memenuhi hal tersebut.

A. Kompresi Bitmap

Kompresi bitmap ini digunakan dalam Aho-Corasick untuk mendapatkan fungsi yang sama dengan hasil luar bitmap dari algoritma Eatherton.



Gambar 5. Kompresi Bitmap

Digunakan single pointer yang menunjuk ke status valid pertama selanjutnya dan mengontrol 256 bit bitmap, mengindikasikan apakah traversal dengan karakter yang diberikan itu valid atau terus berputar di jalur pointer yang salah.. Jika status selanjutnya valid, bit utama dijumlahkan dengan nilainya dan menambahkannya pada basis pointer-ke-selanjutnya.

Dalam mesin dengan pointer 32-bit, pengoptimasian dengan Aho-Corasick yang asli adalah membuat struktur data dengan simpul-simpul sebesar 1028 byte. Sementara itu, jika menggunakan bitmap yang dikompresi, hanya butuh 44 byte. Penurunan kebutuhan memori ini menaikkan cache-behavior dari perangkat lunak dan memungkinkan perangkat keras menyimpan seluruh struktur data Aho-Corasick pada SRAM.

Walau demikian, penggunaan kompresi bitmap memiliki kekurangan pula. Pertama, kompresi ini memerlukan struktur data yang tidak dioptimisasi (hal ini membuat jumlah kondisi terburuknya menjadi dua kali lipat lebih banyak daripada dengan melakukan perkarakter masukan). Kedua, traversal dari simpul ke simpul mengharuskan melakukan pengecekan tiap bit dalam tiap bitmap, kemudian menghitungnya hingga 256 bit utama. Hal ini dapat dikontrol dengan penggunaan seperti algoritma Lulea. Penghitungannya dilakukan untuk tiap 32 bit dalam bitmap. Perbedaan cost-nya sangat terlihat pada hampir seluruh proses jaringan.

B. Kompresi Jalur

Prinsip pengoptimasian ini tidak mengurangi ukuran tiap node atau pun meningkatkan peformansi kasus terburuknya. Tapi, yang dilakukan kompresi jalur adalah mengurangi total ruang yang dibutuhkan dalam basisdata Snort. Prinsip ini mirip dengan pengoptimasian pada simpul terakhir yang terjadi pada pohon bitmap Eatherton atau algoritma pencarian IP lainnya.

Kompresi bitmap bekerja dengan baik pada top dari mesin status.

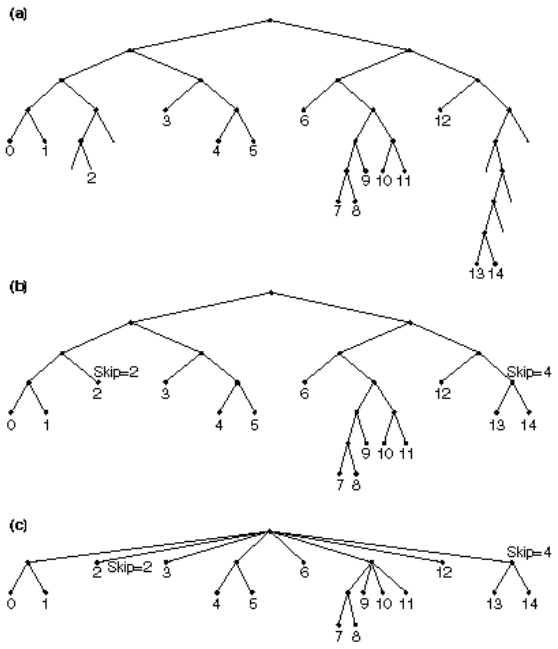
Namun, penggunaannya sangat sia-sia pada simpul-simpul pada bottom-nya.

Agar simpul-simpul sekuensial di bagian bottom ini

terkompres menjadi single node, diterapkanlah kompresi jalur.

Pada kompresi jalur, simpul-simpulnya mengandung sebuah array-dari-karakter, aturan yang bersesuaian, dan pointer ke yang salah (secara sekuensial ditraversal hingga mencapai akhir simpul atau ditemukan ketidakcocokan.

Optimasi ini menurunkan ukuran struktur data yang dibutuhkan Aho-Corasick secara signifikan (struktur data tersebut juga sebelumnya sudah dikompresi bitmapnya). Namun, kompresi jalur juga ada kekurangannya. Penambahan ke single bit untuk membedakan jalur yang dikompres dengan yang tidak membuat ukuran simpul bitmap bertambah sedikit karena kegagalan pointer menjadi lebih kompleks. Pointer tersebut harus menyertakan offset dalam simpul sebagai penyimpan jika terjadi kegagalan transisi di tengah simpul kompresi jalur. Pada pengimplementasian perangkat lunak, data strukturnya bertambah ukuran dua byte. Ini bergantung pada bagaimana compiler melihat strukturnya. Sedangkan pada pengimplentasian perangkat keras, bertambah tiga bit.



Gambar 6. Ilustrasi Kompresi Jalur
sumber: www.drdobbs.com

Kompresi jalur juga menambah tingkat kompleksitasi algortima pencarian. Ini tidak melakukan traversal pada struktur data yang telah diketahui, melainkan menerima banyak data yang tidak terprediksi serta data dari cabang-cabangnya.

Pada mesin dengan 32-bit pointer, sebuah jalur tunggal yang simpulnya terkompres, dapat berisi data yang setara dengan 4 simpul yang bitmapnya terkompres. Hal ini memberi maksimum perbandingan kompresinya 4:1. Sementara itu, perbandingan kompresi pada Snort dengan pada implementasi bitmap adalah 2.5:1.0.

IV. HASIL

Optimasi dengan kompresi pada algoritma Wu-Manber menunjukkan penghematan 20 kali. Sementara itu, lewat algoritma Aho-Corasick, ukuran basidata berkurang hingga 50 kali. Pada router yang bersesuaian, dimungkinkan menyimpan basisdata untuk pencarian SFK, bitmap yang terkompres, dan jalur terkompres, pada chip.

Ketika pendeteksian intrusi berada pada router mereka, sangat penting dapat mengontrol kasus terburuk pula. Jika tidak, router dapat diserang dengan paket kasus terburuk, menyerang data dan membuat overload.

Dari himpunan aturan Snort yang digunakan, pencarian SFK memberikan traversal pada karakter dengan 20 simpul perkarakternya. Sementara untuk Wu-Manber, lebih dari 80 aturan.

Performansi dengan Aho-Corasick yang bitmapnya terkompres meningkat dua kali lipat dibanding yang tidak terkompres.

V. KESIMPULAN

Pendeteksian intrusi kini semakin banyak dan beragam penggunaanya dalam jaringan. Pendeteksian ini sangat bergantung pada optimasi kasus umum, seperti heristik karakter buruk, untuk menambah kecepatan.

Lewat analogi antara pencarian IP dan string matching, dibangun Aho-Corasick untuk penanganannya terhadap kasus terburuk. Aho-Corasick sebagai satu-satunya algoritma string matching yang berjalan pada waktu deterministik untuk kasus terburuknya, memerlukan banyak tempat.

Melalui kompresi bitmap kemudian kompresi jalur pada simpul-simpulnya, didapatkan Aho-Corasick yang lebih baik. Kompresi tersebut mampu mengurangi ukuran yang dibutuhkan jika hanya menggunakan algoritma versi biasa. Dengan demikian, Aho-Corasick yang melakukan kompresi dinilai sebagai pilihan terbaik dalam string matching untuk pendeteksian intrusi.

VI. ACKNOWLEDGMENT

Penulis menyampaikan terima kasih kepada Bapak Rinaldi Munir dan Ibu Masayu Leyla selaku dosen pengajar Strategi Algoritma.

REFERENSI

[1] *www.paloaltonetworks.com/.../what-is-an-**intrusion-detection**-system-ids.html*
[2] *www.snort.org/*
[3] www.blog.ivank.net/aho-corasick-algorithm-in-as3.html
[4] www.sbm.lbl.gov
[5] www.cs.princeton.edu

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, 19 Mei 2014

Jacqueline (13512074)