Pattern Matching Algorithm Comparison for Signature-based Malware Detection

Akbar Maulana Ridho - 13521093[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521093@std.stei.itb.ac.id

***Abstract*—Pattern matching adalah salah satu teknik deteksi malware yang umum digunakan, yakni sebuah teknik yang mencari kecocokan sebagian dari sampel dengan pola tertentu. Proses pencocokan ini merupakan sebuah proses pencocokan string dengan dictionary sebanyak kemungkinan nilai sebuah byte, sehingga kita bisa menggunakan algoritma pencocokan string untuk menjalankan proses ini. Makalah ini akan membandingkan performa tiga buah algoritma pencocokan string, yaitu brute force, Knuth-Morris-Pratt (KMP), dan Boyer Moore (BM). Ketiga algoritma tersebut akan diuji untuk memeriksa pola beberapa sampel malware, yaitu Trickbot, WannaCry, Remcos, dan Nanocore. Hasil percobaan ini menunjukkan bahwa algoritma pencocokan Boyer-Moore (BM) lebih unggul dibandingkan dua algoritma yang lain.**

***Keywords*—Malware Detection, Pattern Matching, Brute Force, Knuth Morris Pratt, Boyer Moore.**

I. PENDAHULUAN

Malware, kependekan dari malicious software, merujuk pada program yang dikembangkan oleh *cybercriminal* untuk mencuri data dan merusak sistem komputer. Beberapa jenis malware yang umum meliputi virus, worms, trojan, spyware, adware, dan ransomware.

Di sisi lain, terdapat software anti-virus yang bisa mengidentifikasi dan menghentikan virus/ malware sebelum mereka bisa menginfeksi komputer. Perangkat lunak ini mampu memberikan perlindungan kepada pengguna dari berbagai ancaman digital.

Terdapat berbagai metode yang digunakan oleh pengembang antivirus untuk mendeteksi malware, seperti *heuristic based detection*, *signature-based detection*, *behavioral detection*, *sandbox detection*, dan yang paling terbaru adalah deteksi menggunakan *machine learning*.

Teknik deteksi yang disinggung pada makalah ini adalah *signature-based detection*. Pendekatan ini mencari bagian kode spesifik untuk mengidentifikasi malware tertentu. Pendekatan ini memiliki berbagai keuntungan karena bisa langsung menargetkan sebuah malware dengan banyak varian berdasarkan kesamaan yang ia miliki.

Proses deteksi berdasarkan *signature* ini sebenarnya pada dasarnya merupakan proses pencocokan pola. Proses ini akan melakukan pemindaian ke sebagian atau seluruh malware lalu mencari apakah terdapat bagian yang cocok dengan pola yang dicari. Meskipun begitu, terdapat perbedaan dengan pencocokan string biasa. Pada pencocokan string, karakter yang diujinya memenuhi encoding ASCII atau UTF-8, sedangkan di sini karakternya adalah sebuah byte yang bernilai 0 sampai 255.

Makalah ini akan membandingkan performa dari tiga buah algoritma pencocokan string, yakni brute force, Knuth-Morris-Pratt (KMP), dan Boyer Moore (BM) untuk mencari algoritma mana yang paling efisien untuk digunakan pada kasus ini.

Setiap algoritma tersebut akan digunakan untuk mencari pola tertentu pada beberapa sampel malware. Sampel yang digunakan adalah sampel malware Trickbot, WannaCry, Remcos, dan Nanocore.

Percobaan yang dilakukan pada makalah ini pada dasarnya membandingkan waktu eksekusi yang dibutuhkan oleh setiap algoritma untuk menemukan pola pada sampel malware. Pola malware sendiri diambil dari YARA Rules yang tersedia pada situs Malpedia.

II. DASAR TEORI

A. Pencocokan String

Teks adalah sebuah string yang panjangnya n karakter. Pola (*pattern*) adalah string dengan panjang m karakter (dengan m <<< n) yang akan dicari dalam teks. Pencocokan string adalah sebuah proses pencarian lokasi pertama pada teks yang bersesuaian dengan *pattern*.

Misalkan S adalah sebuah string dengan ukuran m dengan $S = x_0 x_1 \dots x_{m-1}$, maka

1. Prefiks dari S adalah substring dari S[0...k]
2. Suffiks dari S adalah substring dari S[k...m-1]

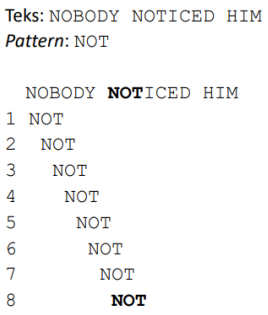
Dengan k adalah sebuah indeks yang memiliki nilai antara 0 dan $m - 1$.

B. Pencocokan String dengan Brute Force

Pada dasarnya, proses pencocokan string dengan brute force merujuk pada proses pengecekan setiap posisi pada teks untuk memeriksa apakah substring pada teks yang dimulai dari posisi tersebut sama dengan pola yang diberikan.

Saat skenario terburuk, algoritma ini memiliki kompleksitas O(mn). Skenario ini terjadi ketika pola berada di akhir teks. Selain itu, skenario terbaiknya memiliki kompleksitas O(n) yang terjadi ketika karakter pertama pola tidak pernah sama dengan teks yang sedang dibandingkan. Saat skenario rata-rata, kompleksitasnya adalah O(m+n).

Berikut adalah ilustrasi pencocokan dengan brute force



Gambar 1 Proses string matching dengan brute force
(Source https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik)

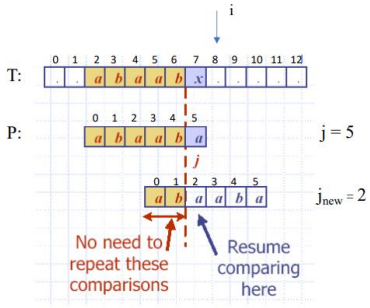
Algoritma ini cepat ketika alfabet teksnya besar, tetapi cukup lambat ketika alfabetnya kecil seperti pada file biner, gambar, dan lain-lain.

C. Pencocokan String dengan Algoritma Knuth-Morris-Pratt (KMP)

Algoritma KMP ini mencari kesamaan pola pada teks dari urutan kiri ke kanan. Meskipun begitu, terdapat sedikit perbedaan dengan algoritma brute force. Algoritma KMP ini melakukan proses penggeseran yang lebih baik dibandingkan dengan algoritma brute force.

Misalkan terjadi ketidakcocokan antara teks dengan pola pada posisi T[i] dan P[j]. Kita bisa melakukan penggeseran yang lebih dari satu untuk menghindari perbandingan yang sia-sia. Penggeseran maksimalnya adalah panjangnya prefiks terbesar P[0..j-1] yang merupakan suffiks dari P[1..j-1].

Berikut adalah ilustrasinya.



Gambar 2 Ilustrasi Proses Penggeseran pada Algoritma KMP
(Source https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik)

Sebelum melakukan pencocokan, algoritma KMP melakukan perhitungan fungsi pinggiran (border function). Border function b(k) didefinisikan sebagai ukuran prefiks terbesar dari P[0..k] yang juga merupakan suffiks dari P[1..k].

Kompleksitas waktu algoritma KMP ini adalah O(m+n) dengan O(m) adalah kompleksitas waktu untuk menghitung fungsi pinggiran dan O(n) untuk proses pencarian string.

Meskipun begitu, algoritma KMP tidak terlalu berjalan dengan baik ketika ukuran alfabet meningkat. Hal ini terjadi karena seiring dengan bertambahnya ukuran alfabet, kemungkinan ketidaksamaannya juga meningkat. Selain itu, ketidaksamaannya juga menjadi lebih sering terjadi di awal pola, sedangkan algoritma KMP bekerja lebih baik ketika ketidaksamaannya muncul tidak di awal pola.

D. Pencocokan String dengan Algoritma Boyer-Moore (BM)

Algoritma pencocokan ini berdasarkan pada dua buah teknik. Teknik pertama adalah teknik *looking glass*, yaitu proses pencarian pola pada teks yang bergerak mundur. Ini berarti proses pencariannya berawal dari akhir pola dan berakhir pada awal pola. Teknik

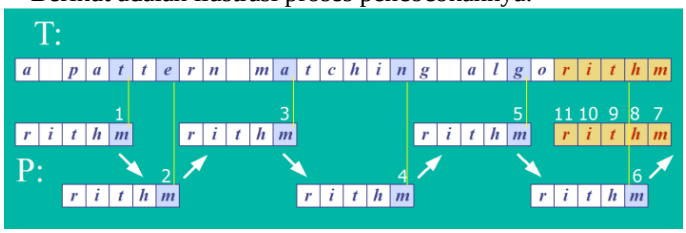
Teknik kedua adalah teknik *character-jump*. Pada dasarnya, ketika terjadi ketidakcocokan pada T[i] == x dan P[j] tidak sama dengan T[i], terdapat tiga buah kemungkinan.

Kemungkinan pertama adalah ketika pola P mengandung x, Pada kasus ini kita bisa menggeser pola ke kanan sehingga x sejajar dengan kemunculan terakhir x pada pola.

Kemungkinan kedua adalah ketika pola p mengandung x, tetapi penggeseran menuju kemunculan terakhir x pada pola tidak mungkin dilakukan. Pada kasus ini kita akan menggeser pola ke kanan sebanyak satu karakter.

Terakhir, ketika kasus satu dan dua tidak memenuhi, seperti ketika x tidak terdapat pada P, kita bisa geser pola P ke kanan sepanjang pola tersebut.

Berikut adalah ilustrasi proses pencocokannya.



Gambar 3 Ilustrasi Proses Pencocokan String pada Algoritma KMP
(Source https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik)

Untuk mengimplementasikan teknik kedua, algoritma ini membuat fungsi *last occurence function*. Fungsi ini akan melakukan preprocessing pada pola P yang menghitung indeks terakhri kemunculan sebuah karakter pada pola.

Fungsi $L(x)$ didefinisikan sebagai indeks terbesar i sedemikian sehingga $P[i] == x$ atau bernilai -1 ketika indeks tersebut tidak ada dengan x adalah sebuah karakter pada alfabet.

Pada kasus terburuk, algoritma Boyer-Moore berjalan pada kompleksitas P(nm+A). Meskipun begitu, algoritma ini sangat cepat ketika ukuran alfabet besar dan akan berjalan lambat ketika ukuran alfabet kecil. Karena hal ini, algoritma BM berjalan cepat untuk pencarian string pada teks berbahasa Inggris dan lambat saat pencarian pola pada file biner.

E. Malware

Malware, singkatan dari *malicious software*, merujuk pada perangkat lunak merusak yang dikembangkan oleh *cybercriminal* untuk mencuri data dan merusak sistem komputer. Secara umum, malware terbagi menjadi virus, worms, trojan, spyware, adware, dan ransomware.

Malware memiliki berbagai tujuan tergantung pada tujuan awal pembuatannya. Tujuan pertama adalah pencurian data, yang meliputi surat elektronik, dokumen, dan informasi sensitif seperti password. Tujuan kedua adalah perusakan dan gangguan sistem atau jaringan sehingga tidak bisa digunakan. Tujuan ketiga adalah untuk perusakan. Tujuan keempat adalah untuk mencuri sumber daya komputasi seperti digunakan untuk menjalankan cryptominer, mengirim email spam, dan lain-lain.

Tujuan terakhir adalah untuk mendapatkan keuntungan finansial dengan mengancam korban atau menjualnya ke *dark web*.

F. Antivirus

Perangkat lunak anti-virus adalalah perangkat lunak yang bisa mengidentifikasi dan menghentikan virus/ malware sebelum mereka bisa menginfeksi komputer. Perangkat lunak ini mampu memberikan perlindungan kepada pengguna dari berbagai ancaman digital.

Terdapat beberapa teknik yang digunakan oleh antivirus untuk mendeteksi malware.

Teknik pertama adalah deteksi berdasarkan pendekatan heuristik. Pendekatan ini melakukan analisis dengan mencari perintah atau instruksi yang tidak secara umum muncul pada sebuah program. Selain itu, pendekatan ini juga bisa mencari sebagian dari program lalu mencocokkannya dengan bagian program yang diketahui secara umum merupakan virus.

Teknik kedua adalah teknik deteksi berdasarkan *signature*. Proses ini mencari potongan spesifik dari sebuah malware untuk proses identifikasi. Meskipun begitu, teknik ini hanya bisa dilakukan pada malware yang sudah dikenali agar diketahui *signature*-nya, sehingga harus ada korban terlebih dahulu.

Teknik ini memiliki berbagai keuntungan dibandingkan dengan pencocokan berdasarkan hash file. Teknik ini bisa mengambil pola umum dari sebuah famili malware dan menargetkan sebuah famili sekaligus, dibandingkan dengan hash file yang hanya bisa menargetkan satu jenis/ varian virus saja.

Teknik lain yang digunakan adalah *behavioral detection*, *cloud antivirus detection*, *sandbox detection*, dan *machine learning* yang tidak terlalu relevan dengan makalah ini.

III. ALAT DAN METODE PERCOBAAN

A. Alat

Berikut adalah alat yang digunakan pada percobaan makalah ini.

1. Golang 1.19
2. Library yeka/zip untuk extract zip file yang terenkripsi
3. Malware bazaar by abuse sebagai sumber sampel malware
4. Malpedia sebagai sumber yara rule untuk pola pencarian yang digunakan untuk mengidentifikasi malware.

B. Proses Percobaan

Percobaan ini akan menguji beberapa sampel malware. Untuk setiap sampel malware, akan dijalankan proses berikut.

1. Unduh sampel malware dari Malware Bazaar
2. Extract berkas malware yang terekripsi ke dalam memori
3. Siapkan beberapa pola yang diketahui sebagai *signature* dari malware tersebut.
4. Untuk setiap jenis algoritma pencocokan string, yaitu brute-force, KMP, dan BM, lakukan proses pencocokan pola dan hitung waktu awal.
5. Untuk setiap pola, konversikan pola dari string menjadi array of byte.
6. Setelah pola dikonversikan, lakukan proses pencocokan string untuk dengan algoritma tertentu.
7. Setelah pencarian pola selesai, ukur waktu akhir dan hitung selisih waktunya.
8. Verifikasi apakah pola tersebut ditemukan pada sampel.

Mengenai alfabet yang digunakan, sebuah perbandingan karakter pada makalah ini menggunakan perbandingan byte, sehingga pada alfabet akan terdapat 256 karakter yang bernilai dari 00000000 hingga 11111111 (dalam bit). Ini dipilih karena unit informasi terkecil yang diproses oleh komputer adalah byte, bukan bit. Selain itu, instruksi biner program juga satuan terkecilnya adalah byte atau beberapa byte, sehingga pemilihan perbandingan berdasarkan byte jauh lebih cocok bila dibandingkan dengan perbandingan berdasarkan bit.

Selain itu, beberapa pola pada sampel ini juga mengandung wildcard (??) yang bisa cocok dengan karakter apa saja, sehingga terdapat fungsionalitas tambahan pada ketiga algoritma yang mengimplementasikan pattern matching dengan wildcard ini.

C. Implementasi Algoritma

Berikut adalah implementasi ketiga algoritma string matching dalam bahasa Go. Implementasi algoritma ini sudah termasuk implementasi pola wildcard yang disebutkan pada bagian sebelumnya.

Brute Force

```go
func BruteForce(pattern []byte, toMatch []byte, wildcard []bool) bool {
    foundMatch := false
    patternLen := len(pattern)

    i := 0
    maxIdx := len(toMatch) - len(pattern) + 1

    for i < maxIdx && !foundMatch {
        equal := true
        j := 0

        for j < patternLen && equal {
            equal = pattern[j] == toMatch[i+j] || wildcard[j]
            j++
        }

        if equal {
            foundMatch = true
        }
        i++
    }

    return foundMatch
}
```

Knuth-Morris-Pratt (KMP)

```go
func KMP(pattern []byte, toMatch []byte, wildcard []bool) bool {
    foundMatch := false

    patternLen := len(pattern)

    // compute border
    borders := make([]int, patternLen)
    j, i := 0, 1

    for i < patternLen {
        if pattern[j] == pattern[i] {
            // j+1 chars match
            borders[i] = j + 1
            i++
            j++
        } else if j > 0 { // j follows matching prefix
            j = borders[j-1]
        } else { // no match
            borders[i] = 0
            i++
        }
    }
```

```
    // begin matching
    i, j = 0, 0

    for i < len(toMatch) && !foundMatch {
        if pattern[j] == toMatch[i] || wildcard[j] {
            if j == patternLen-1 {
                foundMatch = true
                // result = append(result, i-pattern
            }
            i++
            j++
        } else if j > 0 {
            j = borders[j-1]
        } else {
            i++
        }
    }

    return foundMatch
}
```

Boyer Moore (BM)

```
func BadCharHeuristic(pattern []byte, patLen int) []int {
    ret := make([]int, 0)

    for i := 0; i < 256; i++ { //intialize all chars to
        ret = append(ret, -1)
    }
    for i := 0; i < patLen; i++ {
        ret[int(pattern[i])] = i // updates the last inde
    }
    return ret
}
```

```
func BM(patternStr []byte, toMatchStr []byte, wildcard []bool) bool {
    // make both the pattern and to match string into lowercase
    var patLen int = len(patternStr)
    var textLen int = len(toMatchStr)
    // ret := make([]int, 0)
    foundMatch := false

    if patLen <= textLen {
        var badChar []int = BadCharHeuristic(patternStr, patLen) // calculat
        i := 0
        for i <= textLen-patLen && !foundMatch {
            j := patLen - 1
            // check from backwords
            for j >= 0 && (patternStr[j] == toMatchStr[i+j] || wildcard[j])
                j--
            }

            if j < 0 { // if it is a matching string
                // ret = append(ret, i)
                foundMatch = true
                // shift the pattern so the next char is alligned with its l
                if i+patLen < textLen {
                    i += patLen - badChar[toMatchStr[i+patLen]]
                } else {
                    i += 1
                }
            } else {
                // shift the pattern so that the mismatched character aligns
                if diff := j - badChar[toMatchStr[i+j]]; diff > 1 {
                    i += diff
                } else {
                    i += 1
                }
            }
        }
    }
    return foundMatch
}
```

IV. Data Percobaan

Sebelum percobaan dijalankan, berikut adalah informasi dari sampel malware yang akan digunakan.

1. Malware Trickbot

Trickbot adalah trojan yang bertujuan untuk keuntungan finansial. Pertama kali ditemukan pada tahun 2016. Sumber infeksinya adalah berasal dari dokumen yang terinfeksi.

Sampel yang kami gunakan memiliki SHA256 hash 6857d1f39cca9db038166c47cb8742ee8f03601bf4e23394a5b94c35ae3d1726 yang diambil dari Malware Bazaar. Memiliki ukuran sebesar 581632 bytes.

Adapun pola yang digunakan diambil dari yara rule win_trickbot_auto yang diambil dari situs Malpedia pada halaman win.trickbot. Pola yang digunakan adalah sekuens 0 dan 2 dari rule tersebut. Berikut adalah polanya.

No	Pola (sekuens byte dalam hexadecimal)
0	83 e0 70 83 c0 10 eb 25 a9 00 00 00 40 74 11 25 00 00 00 80
2	f7 d8 1b c0 83 e0 20 83 c0 20 eb 36 25 00 00 00 80 f7 d8

Table 1 Sekuens Pola Sampel trickbot

2. WannaCry Ransomware

WannaCry adalah sebuah ransomware yang pertama kali muncul pada tahun 2017. Malware ini bertujuan untuk menenkripsi semua data korban lalu meminta tebusan agar data tersebut bisa dikembalikan.

Sampel yang digunakan memiliki SHA256 hash 561d7f05055800d3eb9d9e150969e2c84a71dc82a362fb3e1a224af420e53b35 yang diambil dari Malware Bazaar. Sampel ini berukuran 245760 bytes.

Adapun pola yang digunakan diambil dari yara rule win_wannacryptor_auto yang diambil dari situs Malpedia pada halaman win.wannacryptor. Pola yang digunakan adalah sekuens 0, 1, dan 2 pada sampel tersebut. Berikut adalah polanya.

No	Pola (sekuens byte dalam hexadecimal)
0	83 ec 1c 56 8b f1 8a 46 5a 84 c0 75 6f 8a 46 58
1	c2 0c 00 8b ce e8 ?? ?? ?? ?? 84 c0
2	74 13 3c 2f 74 04 3c 5c 75 03 8d 6e 01 8a 46 01

Table 2 Sekuens Pola Sampel WannaCry

3. Remcos RAT

Remcos (singkatan dari *remote control & surveillance software*) adalah alat akses jarak jauh komersial yang ditunjukkan untuk mengontrol komputer secara jarak jauh.

Remcos diiklankan sebagai perangkat lunak terpercaya yang bisa digunakan untuk pengawasan dan *penetration testing*, tetapi digunakan dalam berbagai percobaan hacking.

Setelah dipasang, remcos akan membuka *backdoor* sehingga bisa mendapatkan akses penuh pada perangkat pengguna.

Sampel yang digunakan memiliki SHA256 hash 6c0f5a9bf9bfd84be91f3d84335b63ac95ac2b227fedc5de439971577328ac30 yang diambil dari Malware Bazaar. Sampel ini

berukuran 496259 bytes.

Adapun pola yang digunakan diambil dari yara rule win_re,cps_auto yang diambil dari situs Malpedia pada halaman win.remcos. Pola yang digunakan adalah sekuens 0, 1, dan 2 pada yara rule tersebut. Berikut adalah polanya.

No	Pola (sekuens byte dalam hexadecimal)
0	ff 15 ?? ?? ?? ?? 85 c0 74 10 6a 00 ff 35 ?? ?? ?? ??
1	6a 09 ff 35 ?? ?? ?? ?? ff 15 ?? ?? ?? ?? ff 35 ?? ?? ?? ??
2	74 10 6a 00 ff 35 ?? ?? ?? ?? ff 15 ?? ?? ?? ??

Table 3 Sekuens Pola Sampel Remcos

4. NanoCore

Nanocore adalah *remote access tool* yang digunakan untuk mencuri kredensial dan untuk melakukan pengintaian pada kamera pengguna. RAT ini pertama kali ditemukan pada tahun 2017.

Sampel yang digunakan memiliki SHA256 hash 920f8094f26c8540082ba329bbdbdba2ba082d4f6b8427103f6acd2ae66ef89c yang diambil dari Malware Bazaar. Sampel ini berukuran 623616 bytes.

Adapun pola yang digunakan diambil dari yara rule win_re,cps_auto yang diambil dari situs Malpedia pada halaman win.remcos. Pola yang digunakan adalah sekuens 0 pada yara rule tersebut tersebut. Berikut adalah polanya.

No	Pola (sekuens byte dalam hexadecimal)
0	43 6f 24 cb 95 30 38 39

Tabel 4 Sekuens Pola Sampel Nanocore

V. HASIL PENGUJIAN

Berikut adalah data hasil percobaan untuk setiap sampel.

1. Sampel Trickbot

Panjang sampel 581632 dan panjang pola 21 dan 20.

	Waktu Eksekusi (dalam mikrosekon)		
Pola	Brute Force	KMP	BM
0	580.497	482.703	156.663
2	600.935	476.292	112.723
Total	1181.432	958.995	269.384
Relatif terhadap brute force	1	1.232	4.38

Tabel 5 hasil percobaan pada sampel Trickbot

2. Sampel WannaCry

Panjang sampel 245760 dan panjang pola 15, 12, dan 15.

	Waktu Eksekusi (dalam mikrosekon)		
Pola	Brute Force	KMP	BM
0	43.461	22.332	8.376
1	130.624	60.064	35.717
2	181.971	92.894	16.792
Total	356.056	181.29	60.885
Relatif terhadap brute force	1	1.96	5.85

Tabel 6 hasil percobaan pada sampel WannaCry

3. Remcos

Panjang sampel 496259 dan panjang pola 18, 20, dan 16.

	Waktu Eksekusi (dalam mikrosekon)		
Pola	Brute Force	KMP	BM
0	906.558	613.329	225.212
1	838.029	602.779	207.709
2	806.41	617.576	255.288
Total	2550.997	1833.684	688.209
Relatif terhadap brute force	1	1,39	3,71

Tabel 7 hasil percobaan pada sampel Remcos

4. Nanocore

Panjang sampel 623616 dan panjang pola 7

	Waktu Eksekusi (dalam mikrosekon)		
Pola	Brute Force	KMP	BM
0	1102.775	748.031	231.734
Total	1102.775	748.031	231.734
Relatif terhadap brute force	1	1.47	4.76

Tabel 8 hasil percobaan pada sampel Nanocore

Nilai relatif terhadap brute force dihitung dari waktu eksekusi brute force dibagi dengan waktu eksekusi algoritma tersebut. Ini bertujuan untuk mendapatkan nilai kecepatan tersebut berapa kalinya kecepatan brute force.

Berdasarkan percobaan, waktu pencocokan string oleh algoritma KMP berkisar antara 1.2 hingga 1.96 kali waktu eksekusi oleh algoritma brute force. Meskipun begitu, secara modus algoritma KMP hanya sekitar 20-40% lebih cepat dibandingkan dengan algoritma brute force.

Berbeda halnya dengan algoritma BM. Algoritma tersebut mencatatkan waktu eksekusi yang jauh lebih cepat dibandingkan dengan brute force dan KMP, mulai dari 3,7 hingga 5,85 kalinya algoritma brute force. Pada percobaan ini, jelas bahwa algoritma Boyer Boore jauh lebih cocok digunakan pada kasus ini.

Selain itu, kedua algoritma KMP dan BM masih memiliki waktu eksekusi yang lebih cepat bila dibandingkan dengan brute force. Hal ini terjadi karena kedua algoritma tersebut melakukan proses pergeseran yang lebih cerdas dibandingkan dengan algoritma brute force biasa.

Sebagaimana disebutkan sebelumnya, kekurangan algoritma KMP adalah ia semakin tidak efektif ketika ukuran alfabet semakin meningkat karena kemungkinan ketidakcocokannya semakin meningkat. Dalam hal ini, terdapat 256 kemungkinan alfabet, jauh lebih banyak dibandingkan karakter A-Za-z0-9 yang berjumlah 62 buah. Pernyataan ini terbukti dengan hasil eksekusi dengan KMP yang percepatannya tidak sesignifikan algoritma BM.

Lain halnya dengan KMP, ternyata algoritma BM lebih diunggulkan di sini. Sebagaimana disebutkan sebelumnya, algoritma BM akan lebih diuntungkan ketika ukuran alfabetnya

semakin besar. Hal itu terbukti pada hasil percobaan ini. Algoritma BM memiliki waktu eksekusi yang jauh lebih cepat untuk melakukan string matching dibandingkan dengan KMP dan brute force. Ingat kembali bahwa pada kasus ini terdapat 256 alfabet dan hanya sebagian kecil dari alfabet tersebut yang muncul pada pola pencocokan. Oleh karena itu, akan banyak kasus ketika karakter yang diuji pada teks tidak muncul pada pola, sehingga algoritma BM melakukan pergeseran sebanyak panjang polanya. Pergeseran ini yang membuatnya jauh lebih efisien.

VI. KESIMPULAN

Sebagaimana ditunjukkan pada bagian sebelumnya, ketiga algoritma, yakni brute force, KMP, dan BM bisa digunakan sebagai algoritma pencocokan string pada kasus ini. Meskipun begitu, hasil percobaan menunjukkan bahwa algoritma Boyer Moore jauh lebih efisien untuk melakukan pencocokan string yang berupa sekuens byte. Salah satu penyebabnya adalah kemungkinan ketidakcocokan yang sangat tinggi serta terdapat banyak kasus ketika karakter uji pada teks tidak muncul pada pola, sehingga kita bisa menggeser pola sepanjang pola tersebut dan menjadikannya jauh lebih cepat dibandingkan dengan algoritma KMP.

VII. APPENDIX

Source code yang digunakan untuk mengujian pada makalah ini bisa dilihat di https://github.com/akbarmridho/makalah-stima

Berikut adalah bagian video penjelasan makalah ini yang diunggah pada platform youtube. Video tersebut bisa diakses melalui link berikut: https://youtu.be/Jwi-PgJZ_iA

VIII. ACKNOWLEDGMENT

Penulis ucapkan terima kasih sebesar-besarnya kepada berbagai pihak sehingga makalah ini bisa terselesaikan, mulai dari mahasiswa K01, teman-teman penulis, hingga utamanya dosen IF221, Dr. Ir. Rinaldi, M.T. Penulis juga berterima kasih kepada kontributor dan maintainer situs Malpedia dan Malware Bazaar yang memungkinkan penulis untuk memperoleh sampel malware beserta polanya sehingga percobaan pada makalah ini bisa dilakukan.

PUSTAKA

[1] Cisco, *What is Malware?*. Cisco Systems Inc, diakses 22 Mei 2023.
[2] CISA, *Understanding Anti-Virus Software*. America's Cyber Defense Agency (2019), diakses 22 Mei 2023.
[3] FortiNet, *What is Heuristic Analysis?*. Fortinet, Inc, diakses 22 Mei 2023.
[4] Munir. Rinaldi, *Pencocokan String*. Bandung, West Java, 2023.
[5] SentinelOne, *What is a Malware File Signature?*. Fortinet, Inc, diakses 22 Mei 2023.
[6] Malpedia, *Win.Trickbot*. Malpedia, diakses 22 Mei 2023.
[7] Malpedia, *Win.WannaCryptor*. Malpedia, diakses 22 Mei 2023.
[8] Malpedia, *Win.Remcos*. Malpedia, diakses 22 Mei 2023.
[9] Malpedia, *Win.Nanocore*. Malpedia, diakses 22 Mei 2023.
[10] Seifreed, *Trickbot Malware Sample*. Malware Bazaar, diakses 22 Mei 2023.
[11] petikvx, *Wannacry Malware Sample*. Malware Bazaar, diakses 22 Mei 2023.
[12] Adrian__luca, *Remcos RAT Sample*. Malware Bazaar, diakses 22 Mei 2023.
[13] Abuse_ch, *Nanocore RAT Sample*. Malware Bazaar, diakses 22 Mei 2023.

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, 22 Mei 2023

Akbar Maulana Ridho (13521093)