

Penerapan Algoritma *Pattern Matching* pada Log SSH dalam Peningkatan Keamanan Jaringan

Yanuar Sano Nur Rasyid - 13521110
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521110@std.stei.itb.ac.id

Abstract— Keamanan Jaringan merupakan perlindungan terhadap jaringan yang meliputi perangkat di dalamnya. Salah satu tipe dari Keamanan jaringan adalah *intrusion detection and prevention* yaitu perlindungan jaringan dengan mendeteksi ancaman. Algoritma *Pattern Matching* dapat digunakan untuk menerapkan *intrusion detection and prevention* dengan menggunakan BM, KMP, dan Regex. Secure Shell atau SSH merupakan salah satu protokol jaringan untuk komunikasi antara komputer. Algoritma *Pattern Matching* tersebut digunakan untuk mendapatkan data dari file log Secure Shell. Data yang didapatkan dapat dianalisis lebih lanjut sebagai bahan untuk meningkatkan keamanan jaringan yang lebih baik.

Keywords— *BM; KMP; Network Security; Pattern Matching; SSH;*

I. PENDAHULUAN

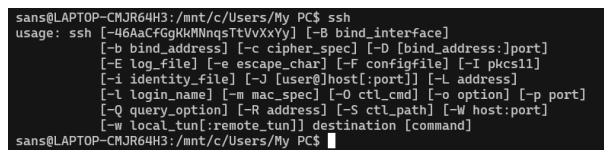
Dalam era digital saat ini, keamanan jaringan menjadi salah satu aspek yang penting yang perlu diperhatikan, terutama saat menggunakan internet. Keamanan diperlukan untuk memastikan kerahasiaan, integritas, dan ketersediaan data yang ditransmisikan melalui jaringan. Salah satu protokol yang umum digunakan untuk mengamankan koneksi jaringan adalah Secure Shell (SSH). SSH merupakan protokol yang memungkinkan pengguna untuk melakukan komunikasi yang aman antara dua perangkat melalui jaringan yang tidak aman.

Aktivitas jaringan dari Secure Shell dapat dianalisis dari file log yang dihasilkan. Dalam konteks keamanan jaringan, aktivitas tersebut perlu dimonitori dengan baik untuk mengidentifikasi hal-hal yang mencurigakan yang berpotensi sebagai sebuah ancaman atau bekas dari ancaman tersebut. Contohnya adalah terdapat percobaan login dengan password salah yang dilakukan berulang-ulang.

Untuk menganalisis lebih lanjut data, file log dari SSH memiliki pola-pola yang dapat dideteksi dan dikumpulkan sebagai data yang dapat dianalisa, seperti user yang mengakses serta ip dan port dari pengaksesnya, tanpa perlu melihat data-data yang tidak relevan. Pola tersebut dapat dengan mudah dideteksi menggunakan Algoritma *Pattern Matching*, seperti Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Regular Expression (Regex).

Penggunaan Algoritma *Pattern Matching* pada Log SSH dapat memberikan manfaat yang khususnya untuk

meningkatkan keamanan jaringan. Dari data aktivitas SSH dan identifikasi pola perilaku yang mencurigakan, administrator sistem dapat segera mengambil tindakan lebih lanjut untuk mengatasi ancaman keamanan yang mungkin muncul.



Gambar 1.1 *Command Line* SSH pada Terminal
Sumber: Dokumentasi Penulis

II. TEORI DASAR

A. *Pattern Matching*

Pattern Matching atau Pencocokan String adalah sebuah proses pencarian P atau *pattern* yaitu sebuah string dengan panjang tertentu, misal m di dalam T atau teks yaitu sebuah string dengan panjang, misal n dengan $n \gg m$ [1].

```
T: the rain in spain stays mainly on the plain  
P: main
```

Gambar 2.1 Contoh *Pattern Matching*
Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>

Hasil dari sebuah *pattern matching* adalah letak pertama dari *pattern* yang ditemukan di dalam teks. Aplikasi Pencocokan String sangatlah banyak mulai dari teks editor hingga bioinformatika.

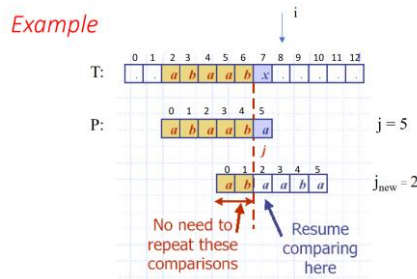
Pattern matching dapat diselesaikan dengan menggunakan *brute force*, tetapi ada algoritma lain yang lebih efisien untuk melakukan pencocokan string, seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM).

B. *Knuth-Morris-Pratt (KMP)*

Knuth-Morris-Pratt atau KMP adalah sebuah algoritma *pattern matching* yang dikembangkan oleh James H. Morris, Donald Knuth, dan Vaughan Pratt. Algoritma ini mencari *pattern* dari kiri ke kanan kemudian saat terjadi ketidaksamaan

akan bergeser hingga *pattern* selanjutnya dapat ditemukan sehingga tidak ada perbandingan *redundant* yang dilakukan.

Pergeseran tersebut ditentukan oleh tabel longest prefix suffix, sesuai dengan namanya, berisi informasi mengenai panjang prefix dari indeks 0..n-1 yang juga merupakan suffix dari indeks 0..n.



Gambar 2.2 Ilustrasi Algoritma Knuth-Morris-Pratt
Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>

Kompleksitas waktu dari KMP adalah $O(n + m)$, dengan n adalah panjang dari text dan m adalah panjang dari pola. Kompleksitas ini sangatlah lebih cepat jika dibandingkan dengan algoritma *brute force*.

C. Boyer-Moore (BM)

Boyer-Moore atau BM adalah algoritma yang dikembangkan oleh Robert S. Boyer dan J Strother Moore. Algoritma ini digunakan untuk mencari *pattern* yang berukuran lebih kecil dibandingkan dengan teks.

Algoritma BM terdiri dari dua teknik [1] yaitu

1. Looking Glass

Melakukan pencarian dilakukan dengan mundur yaitu dari akhir *pattern* hingga ke awal.

2. Character Jump

Ketika ada *mismatch* dapat terjadi tiga kemungkinan yaitu, jika karakter yang tidak sama ada di *pattern* maka bergeser hingga karakter itu. Jika tidak bisa bergeser satu. Jika tidak ada bergeser sebesar *pattern*.



Gambar 2.3 Ilustrasi Algoritma Boyer-Moore
Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>

Pergeseran karakter yang ada di dalam *pattern* dapat dilakukan *preprocessing* dengan membuat tabel last occurrence yang berisi index terakhir dari suatu karakter di dalam *pattern*.

Kompleksitas waktu BM adalah $O(nm + A)$ dengan A adalah alfabet sehingga Boyer-Moore buruk untuk teks yang memiliki alfabet sedikit seperti binary.

D. Regular Expression (Regex)

Regular Expression atau Regex adalah sebuah notasi standar yang terdiri karakter atau string dan merepresentasi sebuah *pattern* di dalam teks [3]. Regex dapat digunakan untuk pencocokan pola dalam teks, mencari kemunculan pertama pola, menggantikan suatu pola, dan membagi teks berdasarkan pola.

Regex terdiri dari beberapa jenis kelas yang memiliki arti-arti tertentu dalam membuat sebuah pola. Berikut adalah contohnya.

| Cheat Sheet | |
|--------------------------------------|--------------------------------|
| Character classes | |
| . | any character except newline |
| \w \d \s | word, digit, whitespace |
| \W \D \S | not word, digit, whitespace |
| [abc] | any of a, b, or c |
| [^abc] | not a, b, or c |
| [a-g] | character between a & g |
| Anchors | |
| ^abc\$ | start / end of the string |
| \b | word boundary |
| Escaped characters | |
| \. * \\ | escaped special characters |
| \t \n \r | tab, linefeed, carriage return |
| \u00A9 | unicode escaped © |
| Groups & Lookaround | |
| (abc) | capture group |
| \1 | backreference to group #1 |
| (?abc) | non-capturing group |
| (?=abc) | positive lookahead |
| (?!abc) | negative lookahead |
| Quantifiers & Alternation | |
| a* a+ a? | 0 or more, 1 or more, 0 or 1 |
| a(5) a(2,) | exactly five, two or more |
| a(1,3) | between one & three |
| a+? a(2,)? | match as few as possible |
| ab cd | match ab or cd |

Gambar 2.4 Notasi Regex
Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Modul-Praktikum-NLP-Regex.pdf>

E. Network Security

Network Security atau Keamanan Jaringan adalah sebuah *sub-studi* dari *cybersecurity* yang pada dasarnya berfungsi untuk melindungi data, aplikasi, perangkat dan sistem yang terhubung dengan sebuah jaringan [4].

Berikut adalah beberapa tipe dari keamanan jaringan.

1. Firewall Protection

Firewall adalah sebuah *software* atau *hardware* yang mengatur pengaksesan sebuah jaringan sehingga hanya pengguna yang sudah di-*approve* yang dapat mengakses.

2. Intrusion detection and prevention (IDPS)

Sebuah proses yang berada di belakang *firewall* untuk memberikan proteksi tambahan melawan ancaman. IDPS biasanya bersifat *passive*, tetapi jika mendeteksi dapat ada aksi otomatis tambahan yang dapat dilakukan, seperti mematikan koneksi. Perlindungan tipe juga yang mengimplementasikan algoritma *pattern matching* untuk mendeteksi aktor yang berbahaya.

3. Network access control (NAC)

Perlindungan yang berada di *frontline* sebagai pengatur akses ke jaringan.

4. Cloud security

Perlindungan terhadap *ressources* online seperti, data sensitif, aplikasi, IP virtual, dan lainnya dari *leak*, kehilangan, dan pencurian.

5. Virtual Private Networks (VPNs)

Sebuah *software* yang melindungi identitas penggunaannya dengan mengenkripsi data dan menyembunyikan IP dan lokasi.

6. Data loss prevention (DLP)

Alat dan strategi yang diimplementasikan kepada user agar tidak, dengan sengaja atau tidak, menyebarkan informasi sensitif.

7. Endpoint protection

Melindungi *endpoint* dari jaringan yaitu laptop, tablet, smartphone, dan lainnya.

F. Secure Shell (SSH)

Secure Shell atau SSH adalah sebuah protokol jaringan yang menghubungkan akses komunikasi ke sebuah komputer dengan aman melalui jaringan yang tidak aman.

SSH memiliki beberapa fitur yang penting untuk memastikan bahwa koneksi aman, seperti enkripsi *end-to-end*, autentikasi, dan integritas data. Selain sebagai penghubung dua buah komputer, Secure Shell juga digunakan untuk manajemen *routers*, server, platform virtualisasi, dan *operating system* (OSes).

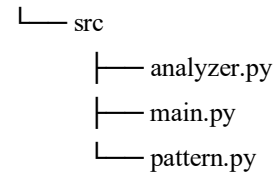
Setiap ada aktivitas dalam SSH, Secure Shell akan menulis semua aktivitas tersebut pada file log. File ini meliputi waktu dan tanggal akses, pengguna, host sumber, host tujuan, tindakan atau perintah, dan status. Aktivitas tersebut perlu diawasi untuk memastikan bahwa keamanan jaringan tetap terjaga.

```
Mar 29 19:39:32 ip-10-77-29-248 system-logind[1118]: New session 134 of user elastic_user_0.
Mar 29 19:39:32 ip-10-77-29-248 sshd[5820]: Received disconnect from 85.245.187.41 port 55700:11: disconnected by user
Mar 29 19:39:32 ip-10-77-29-248 sshd[5820]: Disconnected from 85.245.187.41 port 55700
Mar 29 19:39:32 ip-10-77-29-248 sshd[5750]: pam_unix(sshd:session): session closed for user elastic_user_0
Mar 29 19:39:32 ip-10-77-29-248 system-logind[1118]: Removed session 134.
Mar 29 19:47:54 ip-10-77-29-248 sshd[5830]: Accepted password for elastic_user_1 from 24.151.187.17 port 52323: ssh2
Mar 29 19:47:54 ip-10-77-29-248 sshd[5830]: pam_unix(sshd:session): session opened for user elastic_user_1 by (uid=0)
Mar 29 19:47:54 ip-10-77-29-248 system: pam_unix(system-user:session): session opened for user elastic_user_1 by (uid=0)
Mar 29 19:47:54 ip-10-77-29-248 system-logind[1118]: New session 135 of user elastic_user_1.
Mar 29 19:47:54 ip-10-77-29-248 sshd[5829]: Received disconnect from 24.151.187.17 port 52323:11: disconnected by user
Mar 29 19:47:54 ip-10-77-29-248 sshd[5829]: Disconnected from 24.151.187.17 port 52323
Mar 29 19:47:54 ip-10-77-29-248 sshd[5830]: pam_unix(sshd:session): session closed for user elastic_user_1
Mar 29 19:47:54 ip-10-77-29-248 system-logind[1118]: Removed session 135.
```

Gambar 2.6 Contoh File Log SSH
Sumber: Dokumentasi Penulis

III. IMPLEMENTASI

Penerapan algoritma *pattern matching* untuk menganalisis file log dari aktivitas SSH diimplementasikan dalam bahasa pemrograman Python dengan struktur kode seperti berikut.



File *analyzer.py* berisi kelas log analyzer yang akan digunakan pada *main.py* sementara *pattern.py* berisi implementasi algoritma *pattern matching* dalam Python.

A. pattern.py

Algoritma *pattern matching* yang diimplementasikan adalah Knuth-Morris-Pratt (KMP) dengan fungsi *KMP(pattern,text)* dan *border_function(pattern)*, Boyer-Moore (BM) dengan fungsi *BM(pattern, text)* dan *last_occurance(text)* dan regex dengan fungsi *regex(pattern, text)*.

```
def border_function(pattern: str) -> list:
    """
    Preprocesses pattern for KMP algorithm
    """
    lps = [0] * len(pattern) # init all length of prefix suffix to 0
    i = 1
    j = 0
    while i < len(pattern):
        if pattern[i] == pattern[j]: # if match
            lps[i] = j + 1 # set length of prefix suffix
            # continue next character
            i += 1
            j += 1
        else: # if not match
            if j != 0: # if not at the beginning of pattern
                j = lps[j - 1] # jump to the last prefix
            else:
                lps[i] = 0 # set length of prefix suffix to 0
                i += 1 # continue next character
    return lps
```

Gambar 3.1.1 Implementasi Border Function
Sumber: Dokumentasi Penulis

```
def KMP(pattern: str, text: str) -> int:
    """
    KMP algorithm for pattern searching in a text
    return index of pattern in text if found else -1
    """
    # Preprocessing
    lps = border_function(pattern) # longest prefix suffix

    # Searching
    i = 0
    j = 0
    while i < len(text) and j < len(pattern):
        if text[i] == pattern[j]: # if match
            # continue next character
            i += 1
            j += 1
        else: # if not match
            if j != 0: # if not at the beginning of pattern
                j = lps[j - 1] # jump to the last prefix
            else:
                i += 1 # continue next character

    if j == len(pattern): # if found
        return i - j
    else: # if not found
        return -1
```

Gambar 3.1.2 Implementasi KMP
Sumber: Dokumentasi Penulis

```
def last_occurrence(pattern: str) -> list:
    """
    Preprocesses pattern for BM algorithm
    """
    last = [-1] * 256 # init all (ASCII) last occurrence to -1
    for i in range(len(pattern)):
        last[ord(pattern[i])] = i # set last occurrence of character in pattern
    return last
```

Gambar 3.1.3 Implementasi fungsi *last occurrence*
 Sumber: Dokumentasi Penulis

Tabel last dibuat dengan ukuran 256 menyesuaikan dengan jumlah karakter ASCII.

```
def BM(pattern: str, text: str) -> int:
    """
    BM algorithm for pattern searching in a text
    return index of pattern in text if found else -1
    """
    # Preprocessing
    last = last_occurrence(pattern) # last occurrence of each character in pattern

    # Searching
    i = len(pattern) - 1
    j = len(pattern) - 1
    while i < len(text):
        if text[i] == pattern[j]: # if match
            if j == 0: # if found
                return i
            else: # if not found
                # continue before character
                i -= 1
                j -= 1
        else:
            # jump to the last occurrence of character in pattern or move to the right of pattern once
            i += len(pattern) - min(j, 1 + last[ord(text[i])])
            j = len(pattern) - 1 # reset j to the end of pattern

    return -1
```

Gambar 3.1.4 Implementasi BM
 Sumber: Dokumentasi Penulis

```
def regex(pattern: str, text: str) -> int:
    """
    Regex algorithm for pattern searching in a text
    return index of pattern in text if found else -1
    """
    # search for pattern in text with case insensitive
    match = re.search(pattern, text, re.IGNORECASE)
    if match:
        return match.start()
    else:
        return -1
```

Gambar 3.1.5 Implementasi regex
 Sumber: Dokumentasi Penulis

B. analyzer.py

Implementasi program dibuat dari dua buah kelas yaitu LogAnalyzer sebagai kelas abstract yang berisi method abstrak analyze(). Implementasi konkrit dibuat di dalam kelas SSHLogAnalyzer yang diturunkan dari kelas LogAnalyzer.

```
class LogAnalyzer(ABC):
    @abstractmethod
    def analyze(self):
        """
        Analyze the log file
        output file: result.txt
        """
        pass

class SSHLogAnalyzer(LogAnalyzer):
    def __init__(self, file_path: str, search_func: callable, *pattern: list):...
    def analyze(self):...
    def _get_match_in_file(self, search_func: callable):...
    def _extract_network(self, line: str):...
    def _extract_user(self, line: str):...
    def _output_analysis(self):...
    def _output_summary(self):...
```

Gambar 3.2.1 Kelas analyzer.py
 Sumber: Dokumentasi Penulis

```
def __init__(self, file_path: str, search_func: callable, *pattern: list):
    self.log_file_path: str = file_path
    self.search_func: callable = search_func

    self.search: str = "sshd"
    default_pattern = [
        "Illegal user",
        "Invalid user",
        "Did not receive identification string",
        "Failed password",
        "error: Could not get shadow information",
        "User dcid not allowed",
    ]

    self.pattern: list = pattern if pattern else default_pattern
    self.result: list = [[] for _ in range(len(self.pattern))]
    self.user_result: dict = {}
    self.ip_result: dict = {}
    self.port_result: dict = {}
```

Gambar 3.2.2 SSHLogAnalyzer init
 Sumber: Dokumentasi Penulis

Kelas SSHLogAnalyzer menerima string file_path sebagai lokasi log serta sebuah fungsi pattern matching yang akan digunakan untuk menganalisis log.

Pattern yang digunakan juga dapat dimasukkan oleh user atau memakai pattern default. Pattern default ini mendeteksi beberapa kejanggalan yaitu ada akses login yang mencurigakan untuk pattern indeks 0, 1, dan 3. Sementara itu, pattern dengan indeks sisanya adalah aktivitas yang mungkin terjadi saat serangan DDoS atau Denial Of Service [6].

```
def analyze(self):
    self._get_match_in_file(self.search_func)

    self._output_summary()
    self._output_analysis()
```

Gambar 3.2.3 Implementasi analyze
 Sumber: Dokumentasi Penulis

Implementasi abstraksi dari analyze dilakukan dengan memanggil fungsi _get_match_in_file yang akan mengisi semua variabel self.result dengan hasil pencocokan sesuai dengan indeks pattern.

```
def _get_match_in_file(self, search_func: callable):
    """
    Get the match line number in the log file using search_method from Pattern.py
    """
    with open(self.log_file_path, 'r') as file:
        for line_number, line in enumerate(file, start=1):
            for idx, pattern in enumerate(self.pattern, start=0):
                i = search_func(pattern.lower(), line.lower())
                if i != -1:
                    self.result[idx].append(line[i-len(pattern):].strip() + f" at line {line_number}")
                    self._extract_user(line)
                    self._extract_network(line)
                    break
```

Gambar 3.2.4 Implementasi pattern matching pada log
Sumber: Dokumentasi Penulis

Pencocokan string itu sendiri dilakukan dengan mengiterasi semua line yang ada pada log dengan salah satu pattern. Jika ada yang sama maka line tersebut akan ditambahkan ke self.result dan juga dilakukan pencarian user dan network pada line itu.

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Analyze network activity from ssh log file. Output result.txt and summary.txt")
    parser.add_argument("file_path", type=str, help="Path to ssh log file")
    parser.add_argument("-algo", type=str.lower, help="Pattern Algorithm to use. Default: KMP", default="kmp", choices=["kmp", "bm", "regex"])
    args = parser.parse_args()
    pattern_algo = {
        "kmp": KMP,
        "bm": BM,
        "regex": regex
    }
    print(f"Analyzing {args.file_path}")
    print(f"Using {args.algo} algorithm")
    analyzer = SSHLogAnalyzer(args.file_path, pattern_algo[args.algo])
    try:
        analyzer.analyze()
    except Exception as e:
        print("Error occured", end=" ")
        print(e)
        exit(1)
    print("Done")
```

Gambar 3.3.1 Implementasi *command line interface*
Sumber: Dokumentasi Penulis

Kode program lebih lanjut dapat dilihat pada link repository github.

IV. PENGUJIAN

Pengujian program dilakukan dengan memasukkan file log SSH yang memiliki data-data yang dapat dianalisis.

A. Percobaan 1

Menggunakan data dari [6] dan menjalankan command

```
python .\src\main.py .\test\test.log
```

Hasil output program
summary.txt

```
Summary of .\test\test.log

Pattern found
Illegal user: 32 attempt
Invalid user: 0 attempt
Did not receive identification string: 0 attempt
Failed password: 20 attempt
error: Could not get shadow information: 16 attempt
User dcid not allowed: 0 attempt

Five most frequent users:
NOUSER with 16 count
root with 12 count
test with 8 count
admin with 8 count
guest with 4 count

Five most frequent IP addresses:
218.49.183.17 with 52 count

Five most frequent ports:
48849 with 1 count
49090 with 1 count
49266 with 1 count
49468 with 1 count
49680 with 1 count
```

Gambar 3.4.1 *Summary* Percobaan 1
Sumber: Dokumentasi Penulis

```
def _output_summary(self):
    """
    Summary from the result _get_match_in_file
    """
    with open("summary.txt", "w") as file:
        file.write(f"Summary of {self.log_file_path}\n\n")

        file.write(f"Pattern found\n")
        for id, pattern in enumerate(self.result, start=0):
            file.write(f"{self.pattern[id]}: {len(pattern)} attempt\n")

        sorted_user = sorted(self.user_result.items(), key=lambda x: x[1], reverse=True)
        file.write("\nFive most frequent users:\n")
        for user, count in sorted_user[:5]:
            file.write(f"{user} with {count} count\n")
        if not(sorted_user):
            file.write("No match found\n")

        sorted_ip = sorted(self.ip_result.items(), key=lambda x: x[1], reverse=True)
        file.write("\nFive most frequent IP addresses:\n")
        for ip, count in sorted_ip[:5]:
            file.write(f"{ip} with {count} count\n")
        if not(sorted_ip):
            file.write("No match found\n")

        sorted_port = sorted(self.port_result.items(), key=lambda x: x[1], reverse=True)
        file.write("\nFive most frequent ports:\n")
        for port, count in sorted_port[:5]:
            file.write(f"{port} with {count} count\n")
        if not(sorted_port):
            file.write("No match found\n")
```

Gambar 3.2.5 Summary dari log
Sumber: Dokumentasi Penulis

Method summary akan menghitung semua pattern yang ditemukan pada log kemudian akan mengurutkan dan meng-output lima pattern user dan jaringan yang paling banyak ditemukan pada log.

C. main.py

Implementasi program dengan *interface command line* menggunakan library *argparse* dengan membuat deskripsi dan argumen diterima oleh program. Setelah itu, argumen yang didapatkan akan dimasukkan ke dalam objek *SSHLogAnalyzer*.

B. Percobaan 2

Menggunakan data dari [7] dan menjalankan command

```
python .\src\main.py .\test\auth.log
```

Hasil output program
summary.txt

```
Summary of .\test\auth.log

Pattern found
Illegal user: 0 attempt
Invalid user: 760 attempt
Did not receive identification string: 14 attempt
Failed password: 382 attempt
error: Could not get shadow information: 0 attempt
User dcid not allowed: 0 attempt

Five most frequent users:
root with 211 count
admin with 175 count
elastic_user_0 with 148 count
from with 43 count
ubnt with 38 count

Five most frequent IP addresses:
24.151.103.17 with 157 count
34.204.227.175 with 87 count
91.197.232.109 with 58 count
49.4.143.105 with 40 count
122.163.61.218 with 36 count

Five most frequent ports:
52408 with 10 count
50378 with 7 count
50538 with 7 count
49596 with 7 count
43749 with 7 count
```

Gambar 3.4.2 Summary Percobaan 2
Sumber: Dokumentasi Penulis

Percobaan pertama ditemukan pattern illegal user, failed password, dan error shadow information. Kemudian, user yang diakses pada saat ditemukan pattern adalah NOUSER kemudian root. Terakhir, hanya ada satu IP address 218.49.183.17 yang mengakses SSH dengan port yang berbeda-beda.

Pada percobaan kedua, pattern yang ditemukan adalah invalid user, no identification string, dan failed password. Sedangkan untuk user yang paling banyak adalah root, admin, dan elastic_user_0 dan network dengan IP 24.151.103.17 dengan jumlah kemunculan 157.

Dari kedua percobaan tersebut, didapatkan aktivitas-aktivitas dari Secure Shell yang mencurigakan khususnya pengaksesan tanpa password yang benar pada alamat IP yang sama berkali-kali. Data tersesbut dapat dimanfaatkan untuk meningkatkan keamanan jaringan khususnya penggunaan Secure Shell dengan melakukan tindakan lanjut, seperti mengganti user atau memblok IP tertentu. Hal ini termasuk dalam Network Security tipe Intrusion detection and prevention.

V. KESIMPULAN

Network Security adalah aspek penting dalam era digital ini yang sudah tidak bisa dipisahkan dengan internet. Salah satu

jenis keamanan jaringan adalah Intrusion detection and prevention yang bertugas untuk mendeteksi aktivitas-aktivitas yang dapat mengancam keamanan. Salah satu implementasi deteksi tersebut adalah dengan menggunakan Algoritma Pattern Matching seperti KMP, BM, atau menggunakan Regex. Dari hasil deteksi, didapatkan data mengenai aktivitas jaringan seperti Secure Shell (SSH) yang dapat digunakan untuk meningkatkan keamanan jaringan lebih lanjut.

REPOSITORY LINK

https://github.com/yansans/log_analyzer

UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena berkat izin-Nya pengerjaan makalah Strategi Algoritma dengan judul “Penerapan Algoritma Pattern Matching pada Log SSH dalam Peningkatan Keamanan Jaringan” telah berhasil diselesaikan. Penulis juga mengucapkan terima kasih dosen pengampu mata kuliah IF2211 Strategi Algoritma Semester II Tahun 2022/2023 beserta asisten yang turut membantu pelaksanaan mata kuliah.

REFERENSI

- [1] R. Munir, "Pencocokan String," <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf> (diakses pada 22 Mei 2023).
- [2] M.L. Khodra, "String Matching dengan Regular Expression," <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf> (diakses pada 22 Mei 2023).
- [3] Y.Wibisono dan M.L. Khodra, "Modul Praktikum Kuliah Pengantar Regular Expression," <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Modul-Praktikum-NLP-Regex.pdf> (diakses pada 22 Mei 2023).
- [4] IBM, "What is network security?" <https://www.ibm.com/id-en/topics/network-security> (diakses pada 22 Mei 2023).
- [5] P. Loshin, "Secure Shell (SSH)" <https://www.techtarget.com/searchsecurity/definition/Secure-Shell> (diakses pada 22 Mei 2023).
- [6] OSSEC, "Log Samples from sshd" https://www.ossec.net/docs/log_samples/auth/sshd.html (diakses pada 22 Mei 2023).
- [7] elastic, "examples/MachineLearning/SecurityAnalytics/Recipes/suspicious_login_activity/data/auth.log" https://github.com/elastic/examples/blob/master/Machine%20Learning/Security%20Analytics%20Recipes/suspicious_login_activity/data/auth.log (diakses pada 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

Yanuar Sano Nur Rasyid / 13521110