Application of Boyer-Moore and Regex for Advanced Log Pattern Analysis in SIEM for Threat Detection

Azfa Radhiyya Hakim - 13523115[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*aradhihakim@gmail.com*, *13523115@std.stei.itb.ac.id*

***Abstract*— In the modern cybersecurity landscape, SIEM is crucial for threat detection through log analysis. However, the volume and diversity of log data demand efficient strategies. This paper explores applying the Boyer-Moore Algorithm and Regular Expressions (Regex) for advanced log pattern analysis in Security Information and Event Management (SIEM). Boyer-Moore is effective for precise string searches while Regex offers flexibility for complex patterns. The combination of these algorithmic strategies enhances incident detection precision and speed, strengthening defenses against cyber threats.**

***Keywords*—Boyer-Moore, Regex, SIEM, Log Analysis, Pattern Matching**

I. INTRODUCTION

Algorithm Strategy is a general approach or a framework that used to design an efficient and effective algorithm that solved a computation problem. It is not really the algorithm, but more like the “blueprint” or “recipe” to use to solve a various specific problem.

One of the most learned algorithms, especially relevant in areas like cybersecurity and data analysis, is pattern matching. Pattern matching algorithms refers to a process on finding occurrences of a specific pattern within a larger data (usually called text). The efficiency and accuracy of pattern matching algorithms are paramount, especially when dealing with vast amounts of information.

In a world that has been filled with technology, there will be many advantages that can make life easier for humans. But on the other hand, there will also be losses caused by irresponsible people. When it comes to technology, it is impossible not to mention one of the most popular implementations, that is software. Software is a set of instructions, data or programs that tell a computer what to do and how to do it. As software becomes deeply integrated into daily life, powering everything from personal devices to critical infrastructure, it unfortunately attracts bunch of individuals intent on exploiting vulnerabilities. These malicious actors often aim to steal data, disrupt services, or gain unauthorized access, leading to significant financial and reputational damage.

A critical aspect of detecting and mitigating these threats lies in monitoring the vast streams of operational data generated by software and systems. This data, often in the form of log files, provides a detailed record of events, activities, and interactions within a technological environment. This is where Security Information and Event Management become the most critical thing on developing and maintaining the software. The effectiveness of a SIEM system heavily relies on its ability to accurately and efficiently identify malicious patterns hidden within this massive dataset. Advanced pattern matching such as Boyer-Moore and Regex can be used to identify those malicious patterns within a short complexity time.

II. THEORETICAL BASIS

A. Pattern Matching

Pattern matching is the process of finding one or more matching substrings of a specific pattern within a text to be identified. The goal is to locate the index position of the first occurrence of the desired pattern. In the context of pattern matching, the text string (T) is a sequence of symbols or characters from a specific alphabet, which is generally very large. The pattern (P) is a shorter sequence of characters sought within string T (usually P is much smaller than T). The pattern matching problem, given a text T of length n and a pattern P of length m, aims to find all positions (or the first position) in T where P appears as a substring.

There are some pattern-matching algorithms that are widely used, such as:

1. Brute Force Algorithm
   This is one of the most classical algorithms, where the user simply iterates through possibilities one by one until a solution is found (or not). However, this algorithm is inefficient due to its high time complexity, which is $O(nm)$[1].
2. Knuth-Morris-Pratt (KMP) Algorithm
   The KMP algorithm improves pattern matching efficiency by utilizing information from previous comparisons to avoid unnecessary comparisons. KMP uses a border function (also known as a failure function or partial match table) to speed up the search and has a time complexity of $O(n + m)$[1].
3. Boyer-Moore Algorithm
   The Boyer-Moore algorithm is one of the most efficient patterns matching algorithms for searching within long texts. This algorithm uses two main rules: the "bad character rule" (implicitly described in Case 1, 2, and 3 of the character-jump technique) and the "good suffix rule" (implied by the "looking-glass technique" and detailed shift rules) to skip multiple characters at once in the text, thereby reducing the number of necessary comparisons. The worst-case time complexity of this algorithm is $O(nm + A)$, where A is the alphabet size ,

but in practice, it is often much faster.

B. Boyer-Moore

a. Definition

Boyer-Moore is a pattern matching algorithm that is faster and more efficient than the brute force and KMP algorithms, which only requires $O(nm + A)$ of time complexity. This algorithm was invented by two main figures, namely Robert S. Boyer and J Srother Moore.

b. Basic Concept

Unlike other algorithms, Boyer-Moore performs pattern matching backwards, i.e. from the last character to the first. Instead of comparing the pattern with each position in the text sequentially, it uses two main rules to ignore multiple characters at once, thus reducing the number of comparisons required. In the matching process, 3 cases will arise when the characters being compared are different, as follows [1].

1. If P contains x somewhere, then try to shift P right to align the last occurrence of x in P with $T[i]$.

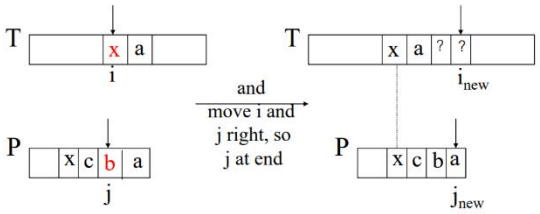

Figure 1: First case Boyer-Moore
Source: [1]

2. If P contains x somewhere, but a shift right to the last character occurrence is not possible, then shift P right by 1 character to $T[i + 1]$.

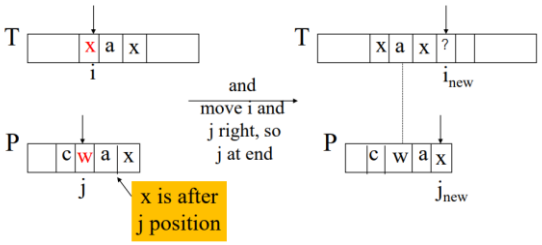

Figure 2: Second case Boyer-Moore
Source: [1]

3. If case 1 and 2 do not apply, then shift P to align $P[0]$.

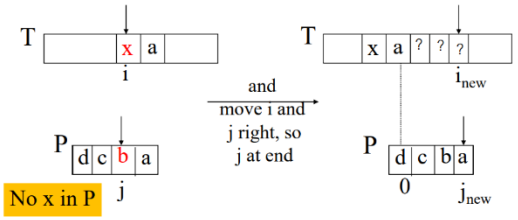

Figure 3: Third case Boyer-Moore
Source: [1]

To continue this algorithm, a table called the Last Occurrence Function (often referred to as LSP) is also required. This table preprocesses the pattern by finding the last occurrence of each character within the pattern. Here is an example of a Last Occurrence Function table of pattern “abacab”.

x	a	b	c	d
L(x)	4	5	3	-1

Figure 4: LSP Table
Source: writer’s archive

c. Implementation

Below is the pseudocode logic of Boyer-Moore algorithm.

```
FUNCTION boyer_moore_algorithm(text, pattern):
    found_positions <- empty list
    text_length <- length of text
    pattern_length <- length of pattern
    bad_char_table <- preprocess_bad_char(pattern)

    IF pattern_length > text_length THEN
        RETURN found_positions
    END IF

    text_position <- 0
    WHILE   text_position   ≤   (text_length   -
pattern_length) DO
        pattern_position <- pattern_length - 1

      WHILE pattern_position ≥ 0 AND
              pattern[pattern_position]         =
text[text_position + pattern_position] DO
            pattern_position <- pattern_position -
1
        END WHILE

        IF pattern_position < 0 THEN
            ADD text_position to found_positions
            text_position <- text_position + 1
        ELSE
             bad_char_shift_amt                <-
bad_char_shift(bad_char_table,

text[text_position + pattern_position],

pattern_position)
            good_suffix_shift_amt              <-
good_suffix_shift(pattern, pattern_position)
            text_position   <-   text_position   +
MAX(bad_char_shift_amt, good_suffix_shift_amt)
        END IF
    END WHILE

    RETURN found_positions
END FUNCTION
```

Here is the example of Boyer-Moore, with pattern “abacab” and text “abacaabadcabacabaabb”.

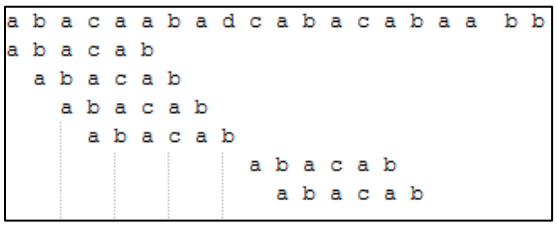

Figure 5: LSP Table
Source: writer's archive

C. Regular Expression (Regex)

Regular Expression (Regex) is a powerful, compact, and flexible notation or standard format used to describe patterns, which are sequences of characters or strings. Regex is employed for efficient string matching. It has become a widespread standard adopted across numerous tools and programming languages. Regex provides various notations for identifying patterns, which are shown below [3].

Table 1: LSP Table
Source: [2]

Regex Pattern	Description
.	Any character except newline.
\.	A period (and so on for *, \, (, \\, etc.)
^	The start of the string.
$	The end of the string.
\d, \w, \s	A digit, word character [A-Za-z0-9_], or whitespace.
\D, \W, \S	Anything except a digit, word character, or whitespace.
[abc]	Character a, b, or c.
[a-z]	Characters from a through z.
[^abc]	Any character except a, b, or c.
aa | bb	Either aa or bb
?	Zero or one of the preceding element.
*	Zero or more of the preceding element.
+	One or more of the preceding element.
{n}	Exactly *n* of the preceding element.
{n,}	*n* or more of the preceding element.
{m,n}	Between *m* and *n* of the preceding element.
??, *?, +?, {n,}?, etc.	Same as above, but as few as possible (lazy match).
(expr)	Capture *expr* for use with \1, etc.
(?:expr)	Non-capturing group.
(?=expr)	Followed by *expr*.
(?!expr)	Not followed by *expr*.

D. Security Information and Event Management (SIEM)

Security Information and Event Management (SIEM) is a software system that is widely used as a powerful tool to prevent, detect, and react to cyberattacks. SIEM solutions have evolved into comprehensive systems that provide broad visibility to identify high-risk areas and proactively focus on mitigation strategies aimed at reducing costs and incident response times. Today, SIEM systems and related solutions are slowly merging with big data analytics tools.

Basically, all SIEMs have the capacity to collect, store, and correlate events generated by the managed infrastructure. They are the central platform of the modern security operations center as they collect events from various sensors (intrusion detection systems, anti-virus, firewalls, etc.), correlate these events, and provide a synthetic view of alerts for threat handling and security reporting [4].

SIEMs consist of several components, which is shown in the figure below.

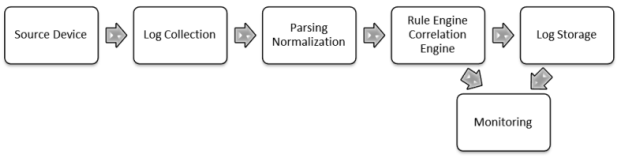

Figure 6: SIEM components
Source: [4]

a. Source device: device that generated the log.
b. Log collection: the process of collecting logs from various source device
c. Parsing normalization: normalize diverse log data to a consistent standard format
d. Rule engine: analyzes correlates event to identify threats pattern
e. Log storage: for forensic purposes
f. Monitoring: A real time monitoring of security events

E. Cyber Security Threat Patterns

In the cybersecurity landscape, threats often appear in the form of specific identifiable patterns. These patterns, known as cybersecurity threat patterns, are sequences of activities or signatures that indicate an attempted attack or system compromise. Understanding these patterns is critical to designing effective detection strategies, especially in systems like SIEM that rely on log analysis to identify suspicious behavior. There are several common attack problems that was used by the attacker to break the systems. This type of attack was commonly written at the system log or application.

1. SQL Injection
   This is the type of attack where the attacker input some of dangerous query into the application. This query later can be executed by the database, and allowed the attacker to read, edit, delete, or even execute administrative

command on database. For example, if the attacker input `'; DROP TABLE--`. This is a valid SQL query since it uses ‘, and will delete table in the database. Another example is using `' or 1=1 --`. This query is commonly used for authentication, where the equation is always true because 1=1.

2. Cross-Site Scripting (XSS).
   This is the type of vulnerability in a website that allow the attackers to inject a dangerous client script (usually javascript) into the site that was shown to another user. When the victim user accesses the site, this dangerous script will be executed that will gain their cookie session, manipulate web content, phising, or even direct the user into a dangerous site. For example, `<img src=x onerror=alert('XSS')>`. This pattern will try to generate <img> with the unknown path. When the browser failed to load the image (onerror), it will run the script that was injected.
3. Denial-of-Service (DoS)
   This is the type of attack that used to make the resource or service down and can’t be used by the verified user by making the service crash.
4. Brute Force Attack
   Brute force attacks are attempts to gain access to user accounts, systems, or other resources by systematically trying every possible combination of passwords or usernames until the correct one is found. These attacks are typically "noisy," leaving behind numerous failed authentication log entries.
   The patterns sought in logs include a high number of failed login attempts from the same IP address to the same account, or failed login attempts from multiple IP addresses to a single account within a short period.

III. IMPLEMENTATION

The basic implementation flow of the application is as follows. First, the user (the system in an automated SIEM) provides the raw log data. This data is then structured into log object that contains information or metadata. Program continues by checking for threat using Boyer-Moore and regex algorithm. For simplicity, we create 3 main classes that becomes the main core of the program: LogPattern, LogEntry, and ThreadDetection.

```python
@dataclass
class ThreatPattern:
    name: str
    boyer_moore_signature: str
    regex_pattern: str
    severity: str
    description: str

@dataclass
class LogEntry:
    timestamp: str
    source_ip: str
    log_content: str
    log_type: str

@dataclass
class ThreatDetection:
    pattern_name: str
    severity: str
    matched_content: str
    source_ip: str
    timestamp: str
    confidence: float
```

Figure 7: Core class
Source: writer’s archive

To identify cyber threats, this advanced log pattern analysis system leverages several instances of ThreatPattern specifically designed to adapt to common vulnerabilities and attack techniques employed by adversaries. Each ThreatPattern instance within the system serves as a formal definition of a potential threat signature. By using pattern matching, some of the threat pattern that can be used is shown as follows.

1. ThreatPattern 1
   - name = “SQL_Injection”
   - boyer_moore_signature = “union_select”
   - regex_pattern = r"(?i)(union\s+select|insert\s+into|delete\s+from|drop\s+table|exec\s*\(|';\s*--)"
   - severity = “HIGH”
   - description = "SQL Injection attempt detected"

2. ThreatPattern 2
   - name = “XSS_Attack”
   - boyer_moore_signature = “<script”
   - regex_pattern = r"(?i)(<script[^>]*>|javascript\s*:|on\w+\s*=|<iframe[^>]*>|eval\s*\(**)**"
   - severity = “HIGH”
   - description = " Cross-Site Scripting attempt detected"
3. ThreatPattern 3
   - name = “Directory_Traversal”
   - boyer_moore_signature = “../”
   - regex_pattern = r"(?i)(\.\.[\\/]|%2e%2e[\\/]|%252e%252e[\\/]|\.\.[\\/].*etc[\\/]passwd)"
   - severity = “MEDIUM”
   - description = “Directory traversal attempt detected”

4. ThreatPattern 4
   - name = “Brute_Force”
   - boyer_moore_signature = “failed login”
   - regex_pattern = r"(?i)(failed\s+login|authentication\s+failed|invalid\s+password|login\s+attempt)"
   - severity = “MEDIUM”
   - description = “Potential brute force attack detected”

5. ThreatPattern 5
   - name = “Command_Injection”
   - boyer_moore_signature = “system(”
   - regex_pattern = r"(?i)(system\s*\(|exec\s*\(|cmd\s*\(|passthru\s*\(|shell_exec)"
   - severity = “HIGH”
   - description = “Command injection attempt detected”

6. ThreatPattern 6
   - name = “Header_Injection”
   - boyer_moore_signature = “\r\n”
   - regex_pattern = r"(?i)(%0a|%0d|%0d%0a|\r\n|\n|\r).*?(Set-Cookie|Location|Content-Type)"
   - severity = “MEDIUM”
   - description = “HTTP Header Injection detected”

7. ThreatPattern 7
   - name = “Privilege_Gained”
   - boyer_moore_signature = “sudo”
   - regex_pattern =
   - r"(?i)(sudo\s+-u|su\s+-|whoami|id\s*;|uname\s*-a|cat\s+\/etc\/passwd)"
   - severity = “HIGH”
   - description = “Privilege escalation attempt detected”

The process continues by analyzing each log entry for threat detection. For every log entry, the system first applies the Boyer-Moore algorithm to quickly scan for predefined threat signatures. If a signature match is found, the system then performs detailed regex pattern matching on the corresponding ThreatPattern to validate and extract the threat. The implementation of this step is shown in the figure below.

```python
def process_log_entry(self, log_entry: LogEntry) -> List[ThreatDetection]:
    detections = []
    self.performance_stats['total_logs_processed'] += 1

    for pattern in self.threat_patterns:
        start_time = time.time()
        bm_matches = self.boyer_moore_engines[pattern.name].search(log_entry.log_content)
        self.performance_stats['boyer_moore_time'] += time.time() - start_time

        if bm_matches:
            start_time = time.time()
            regex_matches = self.regex_patterns[pattern.name].finditer(log_entry.log_content)
            self.performance_stats['regex_time'] += time.time() - start_time

            for match in regex_matches:
                confidence = self._calculate_confidence(pattern, match.group(), log_entry)

                detection = ThreatDetection(
                    pattern_name=pattern.name,
                    severity=pattern.severity,
                    matched_content=match.group(),
                    source_ip=log_entry.source_ip,
                    timestamp=log_entry.timestamp,
                    confidence=confidence
                )
                detections.append(detection)
                self.performance_stats['threats_detected'] += 1

    return detections
```

Figure 8: process_log_entry method
Source: writer’s archive

IV. TEST CASE

To test the validity of the threat detection system, comprehensive test cases have been designed to evaluate both the accuracy and performance of the two-stage detection mechanism. The test cases are categorized into several types to ensure thorough validation of the system's capabilities. Some of the log that will be tested is shown as follows.

1. timestamp="2024-06-09 14:30:15",
   source_ip="192.168.1.100",
   log_content="GET /admin.php?id=1' UNION SELECT username,password FROM users-- HTTP/1.1",
   log_type="web_access"
2. timestamp="2024-06-09 14:31:22",
   source_ip="10.0.0.50",
   log_content="POST /login.php - Failed login attempt for user 'admin'",
   log_type="authentication"

3. timestamp="2024-06-09 14:32:08",
   source_ip="172.16.0.25",
   log_content="GET /uploads/shell.php?cmd=system('cat /etc/passwd') HTTP/1.1",
   log_type="web_access"

4. timestamp="2024-06-09 14:33:45",
   source_ip="192.168.1.200",
   log_content="<script>alert('XSS Test')</script> in user input field",
   log_type="application"

Upon execution of the program, the results are presented as depicted in the figure below.

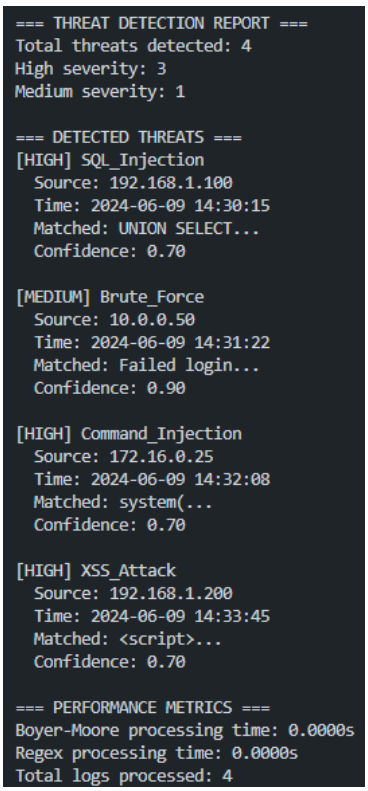

Figure 9: Threat detection report
Source: writer's archive

The analysis of the Threat Detection Report and Performance Metrics indicates that the Cybersecurity SIEM simulation successfully processed a batch of log entries, identifying a total of four distinct threats. The detection results highlight a proactive capability in identifying various attack patterns, categorized by their severity. Overall, the system reported four threats detected, with the majority falling into the high-severity category. Specifically, three threats were classified as High severity, while one threat was classified as Medium severity. This underscores the system's ability to prioritize potentially critical incidents for immediate attention.

V. CONCLUSION

Based on the implementation before, the use of pattern matching (Boyer-Moore and Regex) in the process of detecting threats in SIEM proved capable of improving the monitoring of software on a fairly large scale. Log processing techniques are the main key in this algorithm, where we can detect the types of threats that are detected. These results indicate that the modular pattern matching approach has great potential to be applied in modern security systems. In this paper, the author has not fully included all types of threats that attackers might use. Further research can focus more on increasing the number of such threat types.

VI. ACKNOWLEDGMENT

I express my gratitude to Allah SWT, by whose grace I was given the ease to complete this paper well and on time. Furthermore, I extend my thanks to Dr. Ir. Rinaldi, M.T., who, as the lecturer for the IF 2211 Algorithm Strategy course, taught this 4th-semester material exceptionally well and enjoyably. I also express my sincere thanks to my parents who have greatly assisted and supported me throughout the process of completing this paper. Lastly, I would like to thank my friends, whom I cannot mention one by one, for their invaluable help and support in the creation of this paper.

REFERENCES

[1] Munir, Rinaldi, 2025. "Pencocokan String". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf
[2] Munir, Rinaldi, 2025. "24-String-Matching-dengan-Regex". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf
[3] H. Hosoya, J. Vouillon, and B. C. Pierce, "Regular Expression Types for XML," in Proceedings of the fifth ACM SIGPLAN international conference on Functional programming, Montreal, Canada, 2000
[4] G. González-Granadillo, S. González-Zarzosa, and R. Diaz, "Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures," Sensors, vol. 21, no. 14, p. 4759, 2021.

STATEMENT

I hereby declare that this paper is my original work and has not been copied, adapted, translated from, or plagiarized from any other source.

Bandung, 24 June 2025

Azfa Radhiyya Hakim, 13523115