Jurnal Elektronik Ilmu Komputer Udayana
Volume 14, No 3. February 2026

p-ISSN: 2301-5373
e-ISSN: 2654-5101

Implementasi *Intrusion Prevention System* (IPS) Menggunakan *Signature – Based Detection* Berbasis Suricata

I Wayan Dimas Wirahadi[a1], I Gede Santi Astawa [a2], Made Agung Raharja [a3] I Made Widiartha [a4]

[a]Program Studi Informatika, Universitas Udayana
Bali, Indonesia
[1]iwayandimas20@gmail.com
[2] santi.astawa@unud.ac.id
[3]made.agung@unud.ac.id
[4]madewidiartha@unud.ac.id

Abstract

Network security is a crucial aspect of maintaining the integrity, confidentiality, and availability of data. One solution to strengthen network security is the implementation of an Intrusion Prevention System (IPS). This research implements a hybrid IPS that integrates Suricata as a signature-based detection engine with the Aho-Corasick algorithm for secondary log analysis and pattern matching. The system, implemented on a Windows operating system, is designed to automatically detect and block various types of attacks, such as Port Scanning, DDoS (SYN Flood), and Protocol-Specific Attacks. Testing was conducted through end-to-end attack scenarios to measure the system's effectiveness and response time. The results demonstrate that the proposed architecture is highly effective, achieving an average detection rate (recall) of 86.67% and a precision rate of 81.25%, which yields an F1-Score of 83.87%. Response time analysis revealed that Suricata detects threats within an average of 1-3 seconds, while the Aho-Corasick layer analyzes the resulting logs in a matter of milliseconds. This implementation proves that the combination of Suricata and Aho-Corasick provides a security solution that is not only effective at detecting known threats but also efficient at analyzing the results.

Keyword : *Network Security, Intrusion Prevention System, Suricata, Signature-Based Detection, Attack Detection*

1. Pendahuluan

Keamanan jaringan adalah aspek krusial untuk menjaga integritas, kerahasiaan, dan ketersediaan data. Salah satu solusi untuk memperkuat keamanan adalah penerapan Intrusion Prevention System (IPS). Seiring dengan meningkatnya digitalisasi, keamanan jaringan telah menjadi pilar utama dalam melindungi aset informasi dari berbagai ancaman siber. Kasus pelanggaran data, seperti serangan SQL Injection dan Distributed Denial-of-Service (DDoS), telah terbukti menimbulkan kerugian finansial yang signifikan dan merusak reputasi organisasi [1]. Sistem keamanan konvensional yang bersifat pasif, seperti firewall, seringkali tidak cukup untuk menangkal serangan modern yang semakin canggih dan dinamis. Oleh karena itu, diperlukan pendekatan keamanan proaktif yang tidak hanya mendeteksi, tetapi juga mampu mencegah serangan secara real-time.

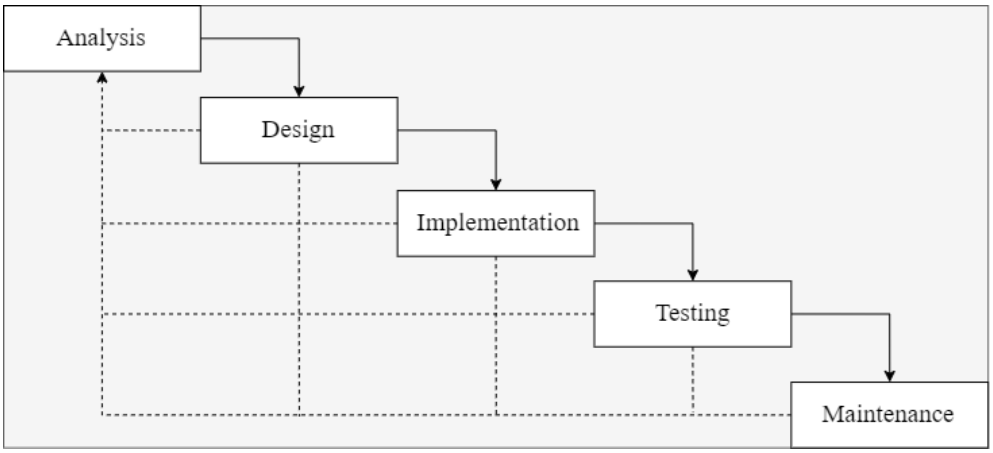
Intrusion Prevention System (IPS) muncul sebagai solusi untuk kebutuhan ini. Berbeda dengan Intrusion Detection System (IDS) yang hanya memberikan peringatan, IPS secara aktif memblokir lalu lintas berbahaya sebelum mencapai target [2]. Penelitian ini berfokus pada implementasi IPS menggunakan Suricata, sebuah engine deteksi ancaman open-source yang dikenal karena performa tingginya berkat arsitektur multi-threading [3]. Suricata mampu melakukan Deep Packet Inspection (DPI) dan menggunakan metode Deteksi Berbasis Signature untuk mengidentifikasi pola-pola serangan yang telah dikenal. Metode ini sangat efektif untuk ancaman yang polanya sudah terdokumentasi, menjadikannya garda terdepan yang andal dalam infrastruktur keamanan. Meskipun deteksi berbasis signature sangat efektif, implementasinya menghadapi tantangan skalabilitas dan performa. Seiring dengan berkembangnya ancaman baru, jumlah signature yang harus diperiksa oleh engine IPS dapat mencapai puluhan hingga ratusan ribu. Proses pencocokan pola secara linear terhadap database signature yang masif ini dapat menimbulkan latensi dan menjadi bottleneck pada

jaringan berkecepatan tinggi. Beban komputasi yang tinggi ini dapat menurunkan throughput jaringan dan bahkan membuka celah bagi penyerang untuk mengeksploitasi keterlambatan respons sistem. Oleh karena itu, diperlukan sebuah mekanisme untuk mengoptimalkan proses pencocokan signature tanpa mengorbankan kedalaman analisis.

Untuk mengatasi tantangan ini, penelitian ini mengusulkan sebuah arsitektur hibrida yang inovatif dengan mengintegrasikan Suricata dengan algoritma Aho-Corasick [4]. Algoritma ini merupakan metode pencocokan string multi-pola yang sangat efisien, mampu mencari ribuan pola secara simultan dalam satu kali proses dengan kompleksitas waktu linear. Dalam arsitektur ini, Suricata bertindak sebagai sensor utama di lapisan jaringan untuk deteksi awal dan pemblokiran, sementara Aho-Corasick berfungsi sebagai mesin analisis sekunder pada lapisan log untuk validasi, korelasi peringatan, dan threat hunting dengan kecepatan tinggi. Kontribusi utama dari penelitian ini adalah merancang dan memvalidasi sebuah model IPS yang tidak hanya efektif dalam deteksi, tetapi juga efisien dalam pemrosesan data peringatan, sehingga menciptakan siklus respons keamanan yang lebih cepat dan cerdas.

2. Metode Penelitian

Metode penelitian ini menggunakan pendekatan metodologi System Development Life Cycle (SDLC) untuk memastikan bahwa setiap tahapan implementasi dilakukan secara sistematis, terstruktur, dan dapat dievaluasi secara menyeluruh dapat dilihat pada Gambar 1. SDLC merupakan sebuah kerangka kerja terorganisir yang umum digunakan dalam pengembangan sistem informasi, yang mencakup seluruh siklus hidup sistem mulai dari tahap inisialisasi hingga pemeliharaan pasca implementasi. Pendekatan ini sangat cocok untuk penelitian yang bersifat teknis dan aplikatif, seperti pengembangan dan penerapan sistem keamanan jaringan berbasis Suricata.



Gambar 1. Alur Metode Penelitian

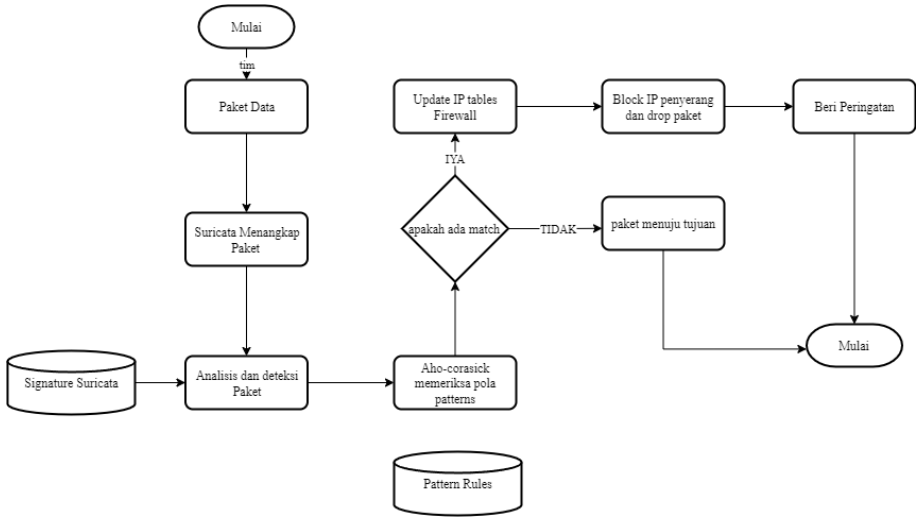
2.1. Landasan Teori

Sistem yang diusulkan dibangun di atas tiga teknologi inti. Pertama, Intrusion Prevention System (IPS), sebuah teknologi keamanan yang secara aktif memonitor dan memblokir lalu lintas berbahaya [5]. Kedua, Suricata, sebuah engine IDS/IPS open-source berkinerja tinggi yang mampu melakukan analisis paket mendalam secara paralel berkat arsitektur multi-threading [6]. Ketiga, Algoritma Aho-Corasick, sebuah metode pencocokan string multi-pola yang sangat efisien. Algoritma ini membangun finite state machine dalam bentuk struktur data Trie dengan failure links, memungkinkan pencarian ribuan pola dalam waktu linear tanpa backtracking [4]. Penelitian sebelumnya telah mengonfirmasi efektivitas Suricata dalam mendeteksi berbagai serangan [1, 7], namun integrasi dengan Aho-Corasick sebagai lapisan analisis log sekunder merupakan pendekatan baru yang ditawarkan dalam penelitian ini untuk meningkatkan efisiensi.

2.2. Arsitektur Sistem Hybrid

p-ISSN: 2301-5373
e-ISSN: 2654-5101

Proses utama sistem divisualisasikan pada Gambar 2, yang menunjukkan alur kerja mulai dari masuknya paket data, deteksi awal oleh Suricata, analisis log oleh Aho-Corasick, hingga eksekusi blokir IP oleh firewall. Diagram ini menegaskan bahwa sistem bekerja berlapis-lapis dengan kombinasi deteksi real-time dan validasi log sekunder.



Gambar 2. Alur Kerja Sistem

Alur dimulai saat paket data masuk ke jaringan. Suricata menangkap dan menganalisis setiap paket berdasarkan ruleset signature. Jika terdeteksi potensi ancaman, log akan dihasilkan. Log ini kemudian dianalisis oleh modul Aho-Corasick untuk validasi pola. Jika pola terkonfirmasi sebagai serangan, sistem akan memblokir IP penyerang melalui firewall dan mengirimkan peringatan. Jika tidak ada kecocokan, paket diteruskan ke tujuan. Arsitektur ini terdiri dari empat lapisan utama:

1. **Lapisan Deteksi Jaringan (Suricata):** Bertugas sebagai sensor utama yang berjalan dalam mode IPS inline pada OS Windows menggunakan driver WinDivert. Lapisan ini melakukan inspeksi paket, deteksi ancaman awal berdasarkan ruleset signature, dan aksi pemblokiran instan.

2. **Lapisan Analisis Log (Python & Aho-Corasick):** Sebuah skrip Python secara real-time memonitor file log eve.json yang dihasilkan Suricata. Skrip ini menggunakan algoritma Aho-Corasick untuk melakukan pencocokan pola sekunder, validasi, dan korelasi alert dengan kecepatan tinggi.

3. **Lapisan Aksi & Manajemen (Firewall & Database):** Berdasarkan hasil analisis, skrip Python mengirimkan perintah pemblokiran IP ke Windows Firewall dan mencatat semua detail insiden ke dalam database MySQL untuk analisis historis.

4. **Lapisan Visualisasi (Flask Web App):** Sebuah dashboard berbasis web yang dibangun menggunakan framework Flask untuk menampilkan statistik dan log insiden secara real-time.

2.3. Konfigurasi Signature dan Implementasi Aho-Corasick

Contoh aturan signature ditunjukkan dalam Tabel 1, misalnya deteksi lebih dari 20 paket SYN dalam 10 detik (port scanning) atau string ${jndi: pada lalu lintas HTTP (exploit Log4j). Tabel ini berfungsi untuk memperlihatkan bagaimana pola serangan diterjemahkan ke dalam rules yang dipahami

Suricata Efektivitas sistem sangat bergantung pada kualitas ruleset signature yang digunakan oleh Suricata dan pola yang dikenali oleh Aho-Corasick. Sebanyak 27 ruleset khusus dikembangkan untuk mendeteksi berbagai ancaman. Beberapa contoh signature kunci disajikan pada Tabel 1.

Tabel 1. Signature rule

ID Aturan	Aksi	Deskripsi Aturan (Ringkas)	Tipe Ancaman
2000001	Drop	Mendeteksi >20 paket SYN dalam 10 detik dari 1 IP.	Port Scan
3000001	Drop	Mendeteksi >100 paket SYN dalam 1 detik.	DDoS
4000002	Drop	Mencari pola select atau '--' di URI HTTP.	SQL Injection
5000002	Drop	Mendeteksi string ${jndi: pada lalu lintas HTTP.	Exploit Log4j

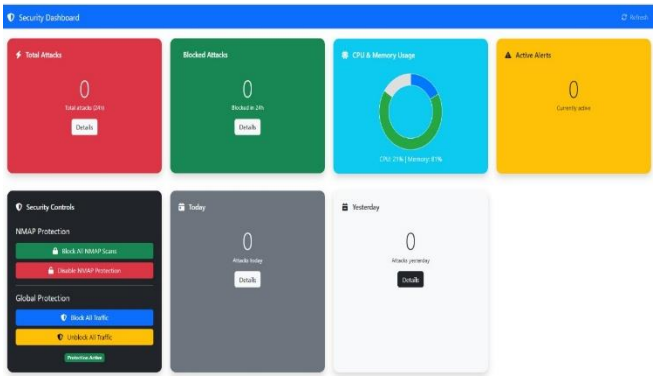
Di lapisan analisis, algoritma Aho-Corasick diimplementasikan menggunakan Python. Prosesnya terdiri dari dua tahap:

1. Tahap Konstruksi: Algoritma membangun struktur data Trie dari semua pola signature yang didefinisikan (misalnya, "Nmap SYN Scan", "SQL Injection Attempt"). Setelah itu, failure links dibuat untuk setiap node dalam Trie, yang memungkinkan transisi cepat saat terjadi ketidakcocokan karakter.
2. Tahap Pencarian: Teks dari log eve.json (khususnya field alert.signature) diumpankan ke automata yang telah dibangun. Algoritma akan menelusuri teks sekali saja (single pass) untuk menemukan semua kemunculan pola yang terdefinisi dengan kompleksitas waktu O(n + m), di mana n adalah panjang teks dan m adalah jumlah total kemunculan pola.

2.4. Desain Dashboard Sistem

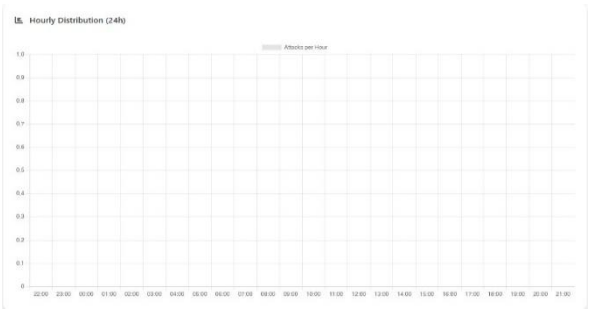
Untuk memfasilitasi pemantauan dan manajemen sistem, dirancang sebuah antarmuka pengguna berbasis web (dashboard) menggunakan kerangka kerja Flask. Dashboard ini berfungsi sebagai pusat kontrol visual yang menyajikan data keamanan secara real-time kepada administrator. tampilan antarmuka utama dashboard ditunjukkan pada Gambar 3 yang memperlihatkan bagaimana informasi serangan dan status jaringan divisualisasikan secara interaktif. Rekapitulasi intensitas serangan per hari ditampilkan pada Gambar 4, sedangkan detail serangan berdasarkan interval jam dapat dilihat pada Gambar 5. Desain sistem Intrusion Prevention System (IPS) berbasis Signature-Based Detection ini mengacu pada kebutuhan mendeteksi dan mencegah serangan secara real-time dengan memanfaatkan engine Suricata. Sistem dirancang dalam bentuk arsitektur modular yang terdiri atas beberapa komponen, yaitu mesin pendeteksi (Suricata), pengelola firewall (IPTables/WinFirewall), penyimpanan data (MySQL), dan antarmuka pengguna (Flask Web Framework). Suricata bertugas memantau lalu lintas jaringan dan menandai aktivitas mencurigakan berdasarkan kumpulan rules (tanda tangan serangan) yang telah dimuat

p-ISSN: 2301-5373
e-ISSN: 2654-5101



Gambar 3. Dashboard Monitoring

Gambar 4. Rekapan Serangan Perhari



Gambar 3. Rekapan Serangan Perjam

2.5. Skenario Pengujian

Pengujian dilakukan pada lingkungan jaringan lokal yang terkontrol untuk mengevaluasi efektivitas dan waktu respons.

a. Pengujian Efektivitas: Simulasi serangan dilakukan menggunakan tools seperti Nmap (Port Scanning), hping3 (DDoS), dan Scapy (Protocol-Specific Attack). Metrik yang diukur adalah True Positive (TP), False Positive (FP), dan False Negative (FN) untuk menghitung Presisi, Recall, dan F1-Score.

b. Pengujian Waktu Respons: Waktu respons diukur sebagai selisih antara timestamp deteksi ancaman oleh sistem dan timestamp eksekusi tindakan pemblokiran oleh firewall.

c. Lingkungan dan Skenario Pengujian: Pengujian dilakukan di lingkungan jaringan lokal yang terkontrol. Spesifikasi perangkat keras dan perangkat lunak yang digunakan dirangkum dalam Tabel 2.

.

Tabel 2. Penggunaan perangkat

Komponen	Spesifikasi	Keterangan
Processor	AMD Ryzen 7 5800H, 16CPU	Dipilih karena kemampuan multi-threading yang baik untuk menangani rule processing paralel
RAM	16 GB	Memory yang cukup untuk menangani ruleset dan traffic

Storage	1 TB SSD	Penyimpanan cepat untuk logging dan packet capture

Simulasi serangan dilakukan menggunakan tools standar industri untuk empat kategori utama:

1. Port Scanning: Menggunakan Nmap dengan berbagai teknik (TCP Connect, SYN Stealth, FIN, NULL, XMAS Scan).
2. DDoS Attack: Menggunakan hping3 untuk melancarkan serangan SYN Flood, UDP Flood, dan ICMP Flood.
3. Protocol-Specific Attack: Menggunakan Scapy untuk membuat paket khusus yang menargetkan kerentanan pada protokol DNS dan SMB.
4. Exploit: Mensimulasikan serangan yang mengeksploitasi kerentanan umum seperti Buffer Overflow dan Remote Code Execution.

3. Hasil dan Pembahasan

3.1. Analisis Mekanisme Deteksi Aho-Corasick

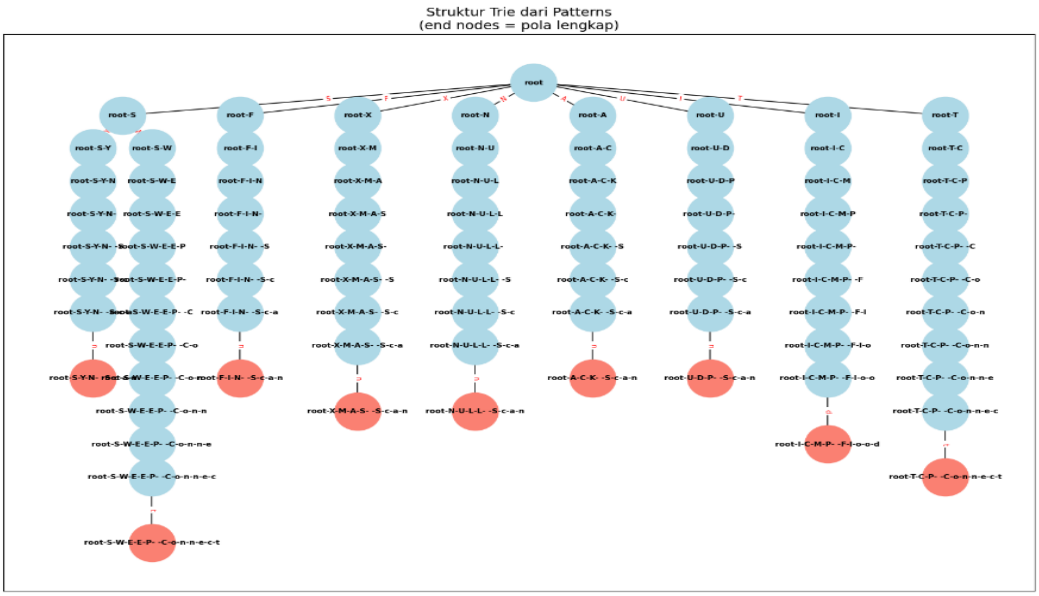
Visualisasi struktur Trie dengan failure links dapat dilihat pada Gambar 6, yang memperlihatkan bagaimana pola signature disusun agar pencocokan lebih efisien. Detail pola serangan yang dimasukkan dalam Trie dirangkum dalam Tabel 3, termasuk SYN Scan, FIN Scan, ICMP Flood, dan lain-lain. Hal ini menegaskan kemampuan Aho-Corasick menangani ribuan pola sekaligus.

1. Pembentukan Automaton: Trie dan Failure Links

Tahap pertama adalah membangun sebuah finite automaton dari kumpulan pola (signature) yang telah didefinisikan. Proses ini menciptakan sebuah struktur data Trie yang diperkaya dengan failure links.

- Pembangunan Trie: Semua pola signature (misalnya, "SYN Scan", "FIN Scan", "ICMP Flood") dimasukkan ke dalam struktur data Trie. Setiap node merepresentasikan sebuah karakter, dan setiap jalur dari root ke node akhir merepresentasikan sebuah pola lengkap. Prefiks yang sama antar pola akan berbagi jalur yang sama, sehingga menghemat ruang penyimpanan secara signifikan.
- Pembangunan Failure Links: Ini adalah langkah krusial yang membedakan Aho-Corasick. Untuk setiap node di dalam Trie, sebuah failure link dibuat. Tautan ini menunjuk ke node lain di dalam Trie yang merepresentasikan sufiks terpanjang dari string saat ini yang juga merupakan prefiks dari pola lain. Fungsi dari failure link adalah untuk menghindari backtracking saat terjadi ketidakcocokan karakter. Alih-alih memulai pencarian dari awal, algoritma cukup mengikuti failure link ke status berikutnya yang paling relevan, sehingga menjaga performa pencarian tetap linear.

p-ISSN: 2301-5373
e-ISSN: 2654-5101



Gambar 4. Pembentukan Failure

Visualisasi pada Gambar 6 menunjukkan struktur Trie yang dihasilkan. Node berwarna biru muda adalah node intermediet, sedangkan node berwarna merah menandakan akhir dari sebuah pola yang valid. Garis putus-putus berwarna hijau merepresentasikan failure links yang menghubungkan berbagai cabang dari Trie, memungkinkan transisi cepat saat terjadi kegagalan pencocokan.

2. Analisis Struktur Trie yang Dihasilkan

Struktur *Trie* yang telah dibangun beserta *failure links*-nya dapat dianalisis lebih lanjut melalui tabel berikut, yang merangkum properti dari setiap node.

Tabel 3. Strukture Node Aho-Corasick

Node Path	Kedalaman	Output
root-S-Y-N- -S-c-a-n	8	SYN Scan
root-S-W-E-E-P- -C-o-n-n-e-c-t	13	SWEEP Connect
root-F-I-N- -S-c-a-n	8	FIN Scan
root-X-M-A-S- -S-c-a-n	9	XMAS Scan
root-N-U-L-L- -S-c-a-n	9	NULL Scan
root-A-C-K- -S-c-a-n	8	ACK Scan
root-U-D-P- -S-c-a-n	8	UDP Scan
root-I-C-M-P- -F-l-o-o-d	10	ICMP Flood
root-T-C-P- -C-o-n-n-e-c-t	11	TCP Connect

Tabel 3 menunjukkan bagaimana setiap pola *signature* direpresentasikan sebagai sebuah jalur di dalam *Trie*. Kolom "Target Failure" menunjukkan ke mana algoritma akan "melompat" jika terjadi ketidakcocokan pada node tersebut. Dalam banyak kasus, tautan ini kembali ke *root*, tetapi pada implementasi yang lebih kompleks, tautan ini dapat menunjuk ke node intermediet lain, yang secara drastis meningkatkan efisiensi pencarian.

3.2. Hasil Pengujian Efektifitas

Untuk mengukur performa sistem secara holistik, hasil dari 45 skenario pengujian dikompilasi ke dalam sebuah confusion matrix. Matriks ini membandingkan hasil prediksi sistem dengan kondisi kenyataan, yang menjadi dasar untuk menghitung metrik performa standar. Hasilnya disajikan pada Tabel 4.

Tabel 4. Confusion Matrix

	Positif	**Negatif**
True	**13**	**2**
False	**3**	**27**

Dari *confusion matrix* tersebut, dapat diuraikan:

- **True Positive (TP) = 13:** Sistem berhasil mengidentifikasi 13 dari 15 serangan dengan benar.
- **False Negative (FN) = 2:** Sistem gagal mendeteksi 2 serangan yang sebenarnya terjadi.
- **False Positive (FP) = 3:** Sistem salah mengklasifikasikan 3 aktivitas normal sebagai serangan.
- **True Negative (TN) = 27:** Sistem dengan benar mengabaikan 27 dari 30 aktivitas normal.

Berdasarkan data di atas, metrik performa sistem dihitung sebagai berikut:

1. **Akurasi (Accuracy)** Akurasi mengukur proporsi prediksi yang benar (TP + TN) dari total keseluruhan kasus.
   - **Formula:** Akurasi = (TP + TN) / (TP + TN + FP + FN) (1)
   - **Perhitungan:** Akurasi = (13 + 27) / (13 + 27 + 3 + 2) = 40 / 45 = **88.89%** (2)
   - **Interpretasi:** Nilai akurasi 88.89% menunjukkan bahwa sistem memiliki keandalan yang sangat tinggi dalam membuat keputusan yang benar, baik dalam mengidentifikasi serangan maupun mengabaikan lalu lintas yang sah.

2. **Presisi (Precision)** Presisi mengukur kualitas dari peringatan yang dihasilkan, yaitu seberapa banyak prediksi serangan yang benar-benar merupakan serangan.
   - **Formula:** Presisi = TP / (TP + FP) (1)
   - **Perhitungan:** Presisi = 13 / (13 + 3) = 13 / 16 = **81.25%** (2)
   - **Interpretasi:** Tingkat presisi 81.25% adalah hasil yang baik, menunjukkan bahwa mayoritas peringatan yang dihasilkan oleh sistem dapat dipercaya. Namun, adanya 3 kasus *False Positive* (FP) menandakan bahwa beberapa *signature* mungkin terlalu umum (kurang spesifik), sehingga memicu alarm pada lalu lintas normal. Ini adalah area krusial untuk perbaikan, karena *false positive* dapat menyebabkan kelelahan peringatan (*alert fatigue*) bagi administrator.

3. **Recall (Sensitivity / Tingkat Deteksi)** *Recall* adalah metrik terpenting untuk sistem keamanan, karena mengukur kemampuan sistem untuk mendeteksi semua serangan yang terjadi.
   - **Formula:** Recall = TP / (TP + FN) (1)
   - **Perhitungan:** Recall = 13 / (13 + 2) = 13 / 15 = **86.67%** (2)
   - **Interpretasi:** Dengan *recall* sebesar 86.67%, sistem ini terbukti sangat efektif dalam menjalankan fungsi utamanya, yaitu mendeteksi ancaman. Dua kasus *False Negative* (FN) yang terlewatkan menjadi perhatian utama. Analisis lebih lanjut menunjukkan bahwa kegagalan ini terjadi pada skenario serangan yang menggunakan teknik *obfuscation* dan fragmentasi paket untuk menghindari deteksi berbasis *signature*. Hal ini menyoroti keterbatasan inheren dari deteksi berbasis *signature* dan perlunya pembaruan *ruleset* secara berkala.

4. **F1-Score** F1-Score adalah rata-rata harmonik dari presisi dan *recall*, memberikan skor tunggal yang menyeimbangkan kedua metrik tersebut.
   - **Formula:** F1-Score = 2 * (Presisi * Recall) / (Presisi + Recall) (1)

p-ISSN: 2301-5373
e-ISSN: 2654-5101

- **Perhitungan:** F1-Score = 2 * (0.8125 * 0.8667) / (0.8125 + 0.8667) = **83.87%** (2)
- **Interpretasi:** F1-Score sebesar 83.87% menegaskan bahwa sistem memiliki keseimbangan yang sangat baik antara kemampuan mendeteksi ancaman secara komprehensif (*recall*) dan menjaga kualitas peringatan agar tetap akurat (*presisi*).

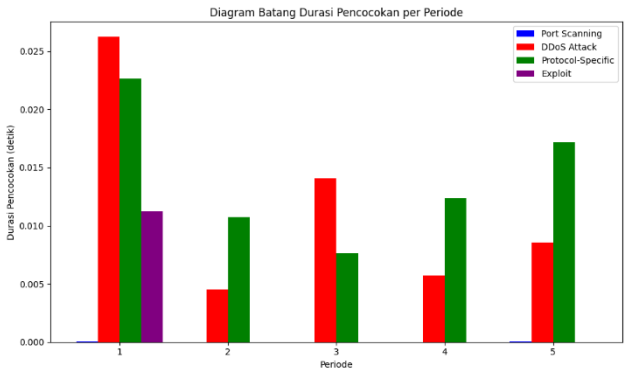
Hasil ini menunjukkan bahwa sistem memiliki **kemampuan deteksi (Recall) yang sangat baik**, dengan berhasil mengidentifikasi 13 dari 15 serangan. Namun, adanya 3 kasus *False Positive* (FP) sedikit menurunkan **tingkat presisi menjadi 81.25%**, yang mengindikasikan bahwa beberapa aturan deteksi perlu disempurnakan untuk meminimalkan alarm palsu. Nilai *F1-Score* sebesar 83.87% mengonfirmasi bahwa sistem menjaga keseimbangan yang solid antara deteksi dan presisi.

3.3. Analisis Waktu Respons

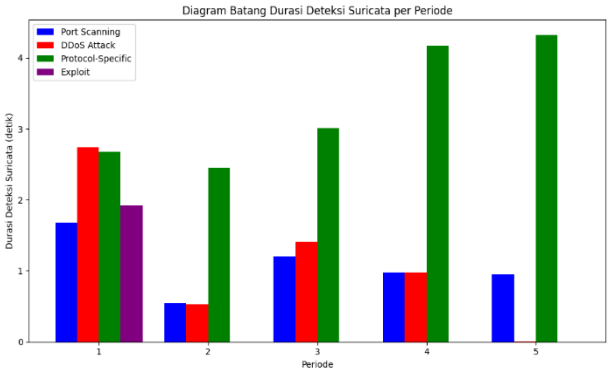
Pada bagian ini, dilakukan analisis terhadap waktu respons atau latensi deteksi dari dua komponen utama sistem: mesin deteksi Suricata dan algoritma pencocokan pola Aho-Corasick. Pengujian ini bertujuan untuk mengetahui kecepatan masing-masing komponen dalam mengidentifikasi berbagai jenis serangan yang disimulasikan.Pengujian waktu respons memisahkan latensi pada dua lapisan utama: deteksi oleh Suricata dan analisis oleh Aho-Corasick, seperti yang ditunjukkan pada Tabel 5.

Tabel 5. Perbandingan Rata-Rata Waktu Respons

Jenis Serangan	Waktu Deteksi Suricata (detik)	Waktu Analisis Aho-Corasick (md)
Port Scanning	1.071	0.012
DDoS Attack	1.133	11.825
Protocol-Specific	3.325	14.114
Exploit	0.959	5.602



Gambar 7. Response Time Ahocorasick



Gambar 8. Response Time Suricata

Perbandingan kinerja Suricata dan Aho-Corasick ditunjukkan pada Tabel 5. Suricata memerlukan 1–3 detik untuk deteksi, sedangkan Aho-Corasick hanya membutuhkan milidetik. Hasil ini divisualisasikan dalam Gambar 7 (rata-rata waktu respons Aho-Corasick) dan Gambar 8 (rata-rata waktu respons Suricata), yang menegaskan efisiensi arsitektur hibrida Hasilnya menunjukkan perbedaan performa yang signifikan. Suricata, yang melakukan analisis paket mendalam (*Deep Packet Inspection*), membutuhkan 1 hingga 3 detik untuk mendeteksi ancaman. Waktu deteksi terlama tercatat pada serangan *Protocol-Specific*, yang memerlukan proses dekode protokol yang kompleks. Di sisi lain, algoritma Aho-Corasick menunjukkan performa yang jauh lebih unggul, dengan memproses dan mencocokkan pola di log dalam hitungan milidetik. Hal ini membuktikan bahwa arsitektur hibrida ini

efektif, di mana Suricata bertindak sebagai sensor garis depan yang andal, dan Aho-Corasick menyediakan mesin analisis sekunder yang sangat efisien untuk investigasi pasca-kejadian.

4. Kesimpulan

Penelitian ini berhasil mengimplementasikan dan memvalidasi arsitektur Intrusion Prevention System (IPS) hibrida yang secara efektif mengombinasikan kekuatan deteksi mendalam dari Suricata dengan efisiensi analisis log dari algoritma Aho-Corasick. Efektivitas sistem ini terbukti secara kuantitatif melalui hasil pengujian yang solid. Dengan kemampuan mendeteksi 13 dari 15 skenario serangan, sistem mencapai tingkat recall (kemampuan deteksi) sebesar 86.67%, yang menunjukkan keandalannya dalam mengidentifikasi mayoritas ancaman. Keseimbangan performa sistem dikonfirmasi oleh F1-Score sebesar 83.87%, yang merefleksikan harmoni yang baik antara kemampuan deteksi yang tinggi dan tingkat presisi (81.25%) yang solid, meskipun masih terdapat ruang untuk mengurangi false positive. Integrasi algoritma Aho-Corasick sebagai lapisan analisis sekunder terbukti secara signifikan mempercepat proses validasi dan korelasi peringatan. Sementara Suricata membutuhkan waktu 1 hingga 3 detik untuk melakukan inspeksi paket secara real-time, modul Aho-Corasick mampu memproses dan mencocokkan pola pada log yang dihasilkan dalam hitungan milidetik. Hal ini menegaskan bahwa arsitektur ini berhasil menerapkan pembagian kerja yang optimal, di mana Suricata berfungsi sebagai sensor garis depan yang tangguh, dan Aho-Corasick sebagai mesin analisis pasca-kejadian yang sangat efisien. Sistem ini juga berhasil menunjukkan kemampuannya dalam memberikan perlindungan aktif secara end-to-end, mulai dari deteksi ancaman di jaringan hingga eksekusi pemblokiran alamat IP penyerang secara otomatis melalui Windows Firewall, melengkapi siklus respons insiden. Untuk pengembangan di masa depan, disarankan untuk mengintegrasikan deteksi berbasis anomali menggunakan machine learning untuk menangani serangan zero-day dan mengotomatiskan manajemen ruleset melalui threat intelligence feeds.

Referensi

[1] F. T. Anugrah, S. Ikhwan, and J. Gusti, "Implementasi Intrusion Prevention System (IPS) Menggunakan Suricata Untuk Serangan SQL Injection," Jurnal Teknik Telekomunikasi, 2022.

[2] D. Stiawan, H. Abdullah, et al., "Intrusion prevention system: A survey," Journal of Theoretical and Applied Information Technology, vol. 40, no. 1, pp. 1-13, 2012.

[3] Open Information Security Foundation, "Suricata User Guide," 2023. [Online]. Available: https://suricata.readthedocs.io/en/latest/. [Accessed: Sep. 19, 2025].

[4] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," Communications of the ACM, vol. 18, no. 6, pp. 333–340, 1975.

[5] S. Ninawe, V. Bariyekar, and R. Asati, "Network Intrusion Prevention System," IJARCCE, vol. 8, no. 2, pp. 196-199, 2019.

[6] D. Kuswanto, "Unjuk Kerja Intrusion Prevention Sistem (IPS) Berbasis Suricata Pada Jaringan Lokal Area Network," Jurnal Ilmiah NERO, vol. 1, no. 2, 2014.

[7] T. Ariyadi, Y. N. Kunang, and R. Santi, "Implementasi dan Analisa Snort dan Suricata Sebagai IDS dan IPS Untuk Mencegah Serangan DOS dan DDOS," in Seminar Nasional Teknologi Informasi & Komunikasi Terapan (SEMANTIK), Semarang, Indonesia, 2012.