

FINAL TASK NOCTRA LUPRA

# PENGEMBANGAN SISTEM DETEKSI INTRUSI ANOMALI UNTUK SURICATA

ISOLATION FOREST



DITULIS OLEH :  
DHIMAS & JOVITA

**Sanksi Pelanggaran Pasal 113**  
**Undang-undang No. 28 Tahun 2014 Tentang Hak Cipta**

1. **Setiap Orang** yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah).
3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah).
4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah).



## **Pengembangan Sistem Deteksi Intrusi Berbasis Anomali untuk Suricata**

**Diterbitkan pertama kali oleh Penerbit Amerta Media  
Hak cipta dilindungi oleh undang-undang *All Rights Reserved***

**Hak penerbitan pada Penerbit Amerta Media**

**Dilarang mengutip atau memperbanyak sebagian atau seluruh isi buku ini  
tanpa seizin tertulis dari Penerbit**

**Anggota IKAPI  
No 192JTE/2020  
Cetakan Pertama:  
21 cm x 29,7 cm  
ISBN**

**Penulis:**

**Editor:**

**Desain Cover:  
Jovita**

**Tata Letak:  
Jovita**

**Diterbitkan Oleh:  
Penerbit Amerta Media**

**NIB. 0220002381476**

Jl. Raya Sidakangen, RT 001 RW 003, Kel, Kebanggan, Kec. Sumbang,  
Purwokerto, Banyumas 53183, Jawa Tengah. Telp. 081-356-3333-24

Email: [mediaamerta@gmail.com](mailto:mediaamerta@gmail.com)

Website: [amertamedia.co.id](http://amertamedia.co.id)

Whatsapp : 081-356-3333-24

## KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa, karena berkat rahmat dan karunia-Nya, buku berjudul "**Pengembangan Sistem Deteksi Intrusi Berbasis Anomali untuk Suricata**" ini dapat terselesaikan. Buku ini disusun sebagai referensi sekaligus panduan praktis dalam mengembangkan modul deteksi berbasis perilaku (anomali) yang berfungsi melengkapi sistem Intrusion Detection System (IDS) Suricata yang selama ini mengandalkan metode *signature-based detection*.

Perkembangan teknologi informasi yang semakin pesat diiringi pula dengan meningkatnya intensitas dan kompleksitas serangan siber. Sistem deteksi intrusi konvensional yang berbasis tanda tangan (*signature*) sering kali tidak mampu mengidentifikasi ancaman baru (*zero-day attack*) atau pola serangan yang belum terdaftar dalam basis data. Oleh karena itu, diperlukan pendekatan deteksi berbasis perilaku yang mampu membedakan aktivitas normal dan aktivitas mencurigakan pada lalu lintas jaringan.

Buku ini menguraikan langkah-langkah pengembangan sistem deteksi intrusi berbasis anomali yang terintegrasi dengan Suricata, dimulai dari pengumpulan dan penyusunan dataset trafik normal, pembuatan *baseline* perilaku jaringan, hingga penerapan algoritma *machine learning* sederhana menggunakan Python untuk mendeteksi penyimpangan dari pola normal tersebut.

Penulis berharap buku ini dapat memberikan kontribusi positif bagi para praktisi keamanan siber, peneliti, maupun mahasiswa yang ingin mempelajari dan mengimplementasikan sistem deteksi intrusi berbasis anomali. Segala saran dan masukan yang membangun sangat penulis harapkan demi penyempurnaan karya ini di masa mendatang.

Akhir kata, semoga buku ini dapat menjadi referensi yang bermanfaat dalam upaya meningkatkan keamanan jaringan dan mengantisipasi ancaman siber yang terus berkembang.

**Jakarta, 30 September 2025**

Penulis

## DAFTAR ISI

BAB 1.....	.7
PENDAHULUAN.....	7
1.1 Latar Belakang.....	7
1.2 Rumusan Masalah.....	8
1.3 Tujuan Pengembangan Sistem.....	9
1.4 Manfaat Penelitian/Pengembangan.....	9
1.5 Ruang Lingkup dan Batasan.....	10
1.6 Metodologi Penelitian/Pengembangan.....	10
BAB 2.....	12
Landasan Teori.....	12
2.1 Konsep Dasar Intrusion Detection System (IDS).....	12
Cara Kerja IDS/IDPS.....	13
2.1 Klasifikasi IDS (Network-based, Host-based).....	14
2.2 Suricata sebagai IDS Berbasis Signature.....	14
2.2.1 Arsitektur Suricata.....	15
2.2.2 Keunggulan dan Keterbatasan.....	16
2.3 Deteksi Berbasis Anomali (Anomaly-based Detection).....	18
2.3.1 Prinsip Kerja dan Karakteristik.....	19
2.3.2 Perbandingan dengan Signature-based Detection.....	20
2.4 Machine Learning dalam Keamanan Jaringan.....	22
2.4.1 Algoritma Klasifikasi dan Deteksi Anomali.....	23
2.4.2 Tantangan dan Overfitting pada Data Jaringan.....	24
2.5 Penelitian Terkait (Related Works).....	25
BAB 3.....	27
Perancangan dan Implementasi sistem.....	27
3.1.1 Diagram Arsitektur.....	29
3.1.2 Alur Integrasi Suricata dan Modul Anomali.....	31
<b>Hasil dan Analisis Integrasi Wazuh dan Suricata untuk Deteksi Analisis Anomali.....</b>	<b>32</b>
1. Penggunaan Wazuh.....	32
2. Integrasi Wazuh dengan Suricata.....	33
3. Generasi Alert Menggunakan testmynids.....	34
4. Pengiriman Alert ke Wazuh.....	34
5. Ekstraksi File eve.log dari Suricata.....	36
6. Analisis Tingkat Anomali (Internal dan External).....	36
6. 1. Statistik Anomali Internal.....	36
Top 5 Event Types dengan Anomali Tertinggi.....	37
Distribusi Skor Anomali.....	37

Visualisasi PCA.....	38
Kesimpulan.....	38
<b>Distribusi Anomali.....</b>	<b>40</b>
<b>Analisis PCA.....</b>	<b>42</b>
<b>Visualisasi Tambahan.....</b>	<b>42</b>
<b>Kesimpulan.....</b>	<b>43</b>
3.2 Persiapan Dataset dan Baseline Trafik Normal.....	44
1. Penanganan Unggahan File dan Pemuatan Data.....	45
2. Pra-pemprosesan Data.....	46
3. Pelatihan Model Isolation Forest.....	47
4. Analisis Hasil.....	47
3.2.1 Sumber Dataset (CTU-13, CICIDS, Custom Capture).....	48
3.3.2 Implementasi Isolation Forest / One-Class SVM.....	50
3.4 Integrasi Modul Anomali dengan Suricata.....	51
3.4.1 Mekanisme Input Log (EVE JSON).....	51
3.4.2 Konversi Output Deteksi ke Alert Suricata.....	52
3.5 Studi Kasus dan Pengujian.....	53
3.5.1 Simulasi Serangan (DoS, Port Scanning, SQLi).....	58
3.5.2 Hasil Deteksi dan Performa Sistem.....	61
3.6 Analisis Hasil dan Evaluasi Sistem.....	65
3.6.2 Rekomendasi Optimasi.....	68
<b>DHIMAS LN.....</b>	<b>72</b>
<b>Jovita Kusuma.....</b>	<b>72</b>

## BAB 1

### PENDAHULUAN

#### 1.1 Latar Belakang

Belakangan ini, tren keamanan siber di Indonesia menunjukkan peningkatan seiring perubahan kebijakan serta kemajuan teknologi. Penerapan SDLC (*Software Development Life Cycle*) yang terintegrasi dengan model CIA Triad mampu memperkuat perlindungan sistem, tetapi tetap memerlukan dukungan mekanisme deteksi seperti *Intrusion Detection System* (IDS) untuk mengidentifikasi ancaman yang lolos dari tahap pengembangan. Hal ini menunjukkan bahwa keamanan siber tidak hanya berhenti pada proses desain dan implementasi perangkat lunak, tetapi harus berlanjut pada pengawasan dan deteksi anomali selama sistem berjalan.

Serangan siber dapat muncul dari mana saja selama perangkat atau sistem terhubung ke internet. Sebuah website mungkin tampak normal, tetapi di baliknya bisa terjadi koneksi atau permintaan (kueri) yang tidak wajar, yang disebut anomali. Untuk mencegahnya, dibutuhkan sistem pendeteksi anomali seperti *Intrusion Detection System* (IDS) yang mampu memantau, memfilter, dan menghentikan perilaku berbahaya sebelum terhubung ke website atau jaringan kita. Jika seorang peretas berhasil masuk dan melakukan eskalasi hak akses.

Hal ini dapat menyebabkan kebocoran data, perubahan informasi tanpa izin, hingga lumpuhnya layanan karena kendali sepenuhnya dikuasai peretas. Menurut data Badan Siber dan Sandi Negara (BSSN), sepanjang 2023 Indonesia mencatat hampir 403 juta anomali trafik atau serangan siber<sup>1</sup> yang terdeteksi, menunjukkan besarnya ancaman yang mengintai meskipun sistem tampak aman di permukaan.

---

<sup>1</sup> Metrotvnews, Sepanjang 2023 Ada 403 Juta Serangan Siber ke Indonesia, diakses 20 Agustus 2025, <https://www.metrotvnews.com/play/b1oC9wGX-sepanjang-2023-ada-403-juta-serangan-siber-ke-indonesia>

CIA Triad adalah model dasar dalam keamanan informasi yang terdiri dari tiga prinsip utama: Kerahasiaan (Confidentiality), Integritas (Integrity), dan Ketersediaan (Availability). Prinsip ini menjadi landasan untuk mengidentifikasi, memahami, serta merancang strategi keamanan yang efektif dalam sistem digital modern. Dalam konteks sistem deteksi intrusi (IDS), perangkat ini berfungsi mendeteksi aktivitas mencurigakan yang berpotensi melanggar salah satu atau beberapa aspek Triad tersebut.

Dengan demikian, IDS menjadi komponen kritis sebagai alat pemantau dan pemberi peringatan dini yang membantu melindungi kerahasiaan, menjaga integritas, dan memastikan ketersediaan sistem tetap terjaga.

## 1.2 Rumusan Masalah

Dalam era digital saat ini, serangan siber semakin kompleks dan sulit dideteksi dengan metode tradisional. Suricata sebagai Intrusion Detection System (IDS), yaitu sistem deteksi intrusi berbasis jaringan yang memantau trafik dan mencocokkannya dengan signature (aturan/rule) memang mampu mengidentifikasi serangan yang sudah dikenal, namun seringkali kesulitan dalam mendeteksi pola serangan baru atau serangan yang dimodifikasi. Tantangannya adalah bagaimana membangun sistem yang mampu mendeteksi anomali pada trafik jaringan secara lebih cerdas dan adaptif. Pada konteks ini, metode *Isolation Forest* dipilih untuk mengolah data log dari **eve.json** milik Suricata, kemudian hasil analisisnya disajikan dalam bentuk visualisasi interaktif menggunakan **Streamlit**. Rumusan masalah yang muncul adalah bagaimana sistem ini dirancang, diimplementasikan, dan dievaluasi sehingga mampu memberikan peringatan dini atas potensi ancaman yang belum terdaftar dalam basis data *signature*.

1. Bagaimana mengembangkan sistem deteksi intrusi berbasis anomali dengan metode *Isolation Forest* yang memanfaatkan log trafik jaringan Suricata dalam format **eve.json**?
2. Bagaimana mengintegrasikan proses analisis anomali dengan platform **Streamlit** untuk memvisualisasikan hasil deteksi secara interaktif dan real-time?

3. Bagaimana mengukur tingkat akurasi, *detection rate*, dan jumlah *false positive* dari sistem yang dibangun.

### 1.3 Tujuan Pengembangan Sistem

Tujuan utama pengembangan sistem ini adalah menciptakan modul deteksi intrusi berbasis anomali yang terintegrasi dengan Suricata untuk memperkuat keamanan jaringan. Modul ini dirancang untuk membaca log **eve.json**, memprosesnya dengan algoritma *Isolation Forest*, dan menampilkan hasil deteksi melalui dashboard interaktif berbasis Streamlit. Dengan demikian, administrator jaringan dapat segera mengambil langkah mitigasi jika ditemukan aktivitas mencurigakan.

Tujuan dari penelitian ini adalah:

1. Merancang dan membangun modul deteksi anomali berbasis *Isolation Forest* yang menggunakan data dari **eve.json** Suricata.
2. Mengembangkan antarmuka berbasis **Streamlit** untuk menampilkan hasil analisis anomali secara interaktif, sehingga memudahkan administrator dalam memantau keamanan jaringan.
3. Mengevaluasi performa sistem dalam mendeteksi berbagai jenis serangan siber dengan mengukur tingkat akurasi, *detection rate*, dan *false positive rate*.

### 1.4 Manfaat Penelitian/Pengembangan

Pengembangan sistem ini diharapkan memberi manfaat pada berbagai sisi. Dari sisi akademis, sistem ini dapat menjadi referensi bagi pengembangan IDS berbasis *machine learning*, khususnya menggunakan algoritma *Isolation Forest*.

Dari sisi praktis, administrator jaringan akan mendapatkan alat bantu yang mampu mendeteksi anomali secara cepat, memvisualisasikannya secara jelas, dan memudahkan pengambilan keputusan. Dari sisi sosial, kesadaran akan pentingnya keamanan siber dapat meningkat.

## 1.5 Ruang Lingkup dan Batasan

Sistem deteksi yang dibahas dalam buku ini memiliki ruang lingkup terbatas pada integrasi Suricata dengan modul deteksi anomali berbasis *Isolation Forest*. Data yang dianalisis berasal dari log **eve.json**, dan hasilnya divisualisasikan menggunakan Streamlit. Sistem ini difokuskan untuk mendeteksi dan memberikan peringatan (*alert*) atas anomali, bukan untuk melakukan pencegahan (*intrusion prevention*). Pengujian dilakukan menggunakan dataset trafik simulasi yang dihasilkan di lingkungan pengujian.

## 1.6 Metodologi Penelitian/Pengembangan

Metodologi penelitian ini menggunakan pendekatan eksperimen laboratorium dengan model studi kasus lokal, di mana implementasi sistem dilakukan pada lingkungan uji (*testbed*) yang menyerupai kondisi jaringan nyata. Fokus studi kasus adalah pengembangan sistem deteksi intrusi berbasis anomali yang terintegrasi dengan **Suricata** dan memanfaatkan log **eve.json** untuk dianalisis menggunakan algoritma *Isolation Forest*, dengan hasil analisis divisualisasikan melalui **Streamlit**.

Tahapan penelitian adalah sebagai berikut:

### 1. Studi Literatur

Mengumpulkan referensi ilmiah terkait konsep *Intrusion Detection System* (IDS), metode deteksi berbasis anomali, dan algoritma *Isolation Forest*.

### 2. Perancangan Sistem

Sistem dirancang dengan alur kerja:

- a. **Server** sebagai sumber lalu lintas jaringan uji.
- b. **Suricata** memantau trafik dan menghasilkan log dalam format **eve.json**.
- c. **Streamlit** menampilkan data secara real-time untuk monitoring awal.
- d. Data **eve.json** diekstrak dan dianalisis dengan algoritma *Isolation Forest* untuk mendeteksi anomali.

- e. Hasil deteksi divisualisasikan pada dashboard Streamlit dengan grafik dan tabel interaktif.
- f. Perancangan ini meliputi diagram arsitektur, skema alur data, dan rancangan antarmuka pengguna.

### 3. Implementasi Sistem

Implementasi dilakukan pada server uji di laboratorium, menggunakan Suricata yang telah dikonfigurasi untuk menghasilkan log **eve.json**. Script Python dikembangkan untuk:

- a. Melakukan *parsing* dan *preprocessing* data dari **eve.json**.
- b. Menerapkan algoritma *Isolation Forest* untuk klasifikasi trafik normal dan anomali.
- c. Mengintegrasikan hasil analisis ke dashboard Streamlit.

### 4. Pengujian Sistem

Pengujian dilakukan pada dua kondisi:

- a. **Trafik Normal**: diambil dari aktivitas jaringan harian di server uji.
- b. **Trafik Serangan**: disimulasikan menggunakan skenario umum di Indonesia seperti serangan DOS, port scanning, dan SQL injection.  
Setiap skenario diuji untuk melihat kemampuan sistem dalam mendeteksi anomali.

### 5. Evaluasi dan Analisis Hasil

Evaluasi dilakukan dengan mengukur:

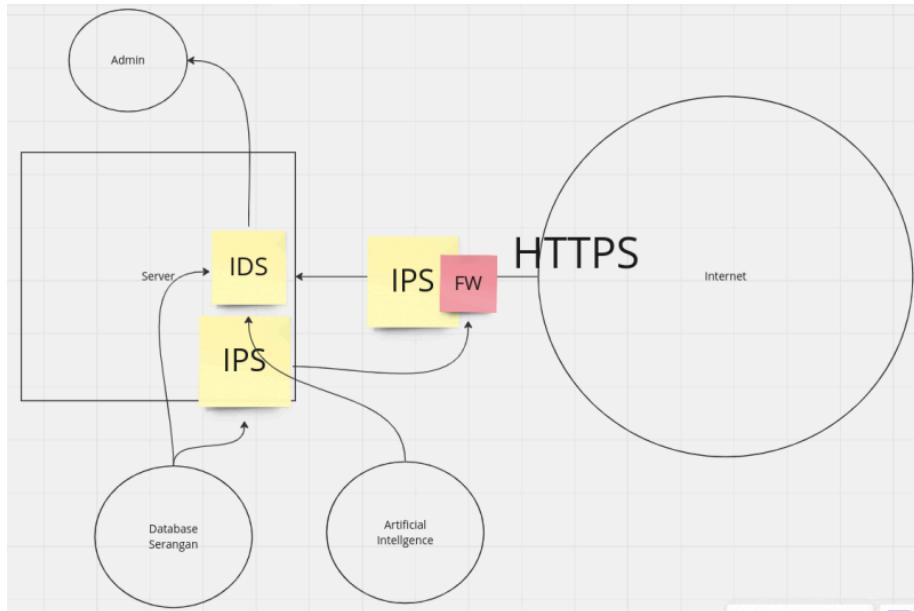
- a. *Detection Rate* (tingkat deteksi anomali).
- b. *False Positive Rate* (tingkat kesalahan deteksi).
- c. Akurasi keseluruhan sistem.

Hasil evaluasi dibandingkan dengan metode *signature-based detection* bawaan Suricata untuk melihat peningkatan kinerja.

## BAB 2

### Landasan Teori

#### 2.1 Konsep Dasar Intrusion Detection System (IDS)



Gambar 1. Simulasi IDS Onno center

**Intrusion Detection System (IDS)** adalah perangkat keras atau perangkat lunak yang digunakan untuk memantau jaringan dan/atau sistem dengan tujuan mendeteksi aktivitas berbahaya atau pelanggaran kebijakan keamanan. IDS bekerja dengan menganalisis lalu lintas data dan perilaku sistem untuk mengidentifikasi indikasi serangan, kemudian memberikan peringatan kepada administrator atau sistem manajemen jaringan.

Selain IDS, terdapat juga **Intrusion Prevention** yang berfokus pada upaya menghentikan serangan setelah terdeteksi. Gabungan keduanya disebut **Intrusion Detection and Prevention System (IDPS)**, yang tidak hanya mendeteksi dan mencatat insiden, tetapi juga secara aktif mencoba mencegah keberhasilan serangan. Fungsi IDPS mencakup:

1. Mengidentifikasi potensi insiden keamanan.
2. Mencatat informasi rinci terkait insiden.

3. Mencoba menghentikan serangan yang sedang berlangsung.
4. Melaporkan insiden kepada administrator keamanan.
5. Mengidentifikasi kelemahan kebijakan keamanan yang ada.
6. Mendokumentasikan ancaman yang terdeteksi.
7. Memberikan efek pencegahan (*deterrent*) terhadap pelanggaran kebijakan.

## Cara Kerja IDS/IDPS

Secara umum, IDS atau IDPS bekerja melalui tahapan berikut:

### 1. Pengumpulan Data (Data Collection)

Sistem mengumpulkan data lalu lintas jaringan atau aktivitas sistem dari berbagai sumber seperti *network taps*, *port mirroring*, dan *log server*. Data mentah ini dapat berupa paket jaringan, metadata, atau catatan aktivitas pengguna.

### 2. Pra-Pemrosesan (Preprocessing)

Data diformat dan disaring untuk menghilangkan informasi yang tidak relevan. Proses ini mencakup *packet decoding*, normalisasi data, dan *feature extraction* (misalnya alamat IP, port, protokol, jumlah paket, dan pola koneksi).

### 3. Analisis dan Deteksi (Analysis & Detection)

IDS menganalisis data untuk mencari indikasi serangan menggunakan dua pendekatan utama:

- a. **Signature-Based Detection**: mencocokkan pola trafik dengan basis data tanda tangan (*signature*) serangan yang telah dikenal.
- b. **Anomaly-Based Detection**: mendeteksi penyimpangan dari pola normal trafik, biasanya menggunakan analisis statistik atau algoritma *machine learning*.

### 4. Respon terhadap Ancaman (Response)

Ketika ancaman terdeteksi, sistem dapat memberikan peringatan kepada

administrator, memutus koneksi berbahaya, memblokir alamat IP sumber serangan, atau menyesuaikan konfigurasi keamanan seperti *firewall rules*.

## 5. Pencatatan dan Pelaporan (Logging & Reporting)

Semua insiden yang terdeteksi maupun dicegah dicatat secara rinci dalam log. Laporan ini dapat digunakan untuk analisis forensik, pembaruan kebijakan, dan evaluasi efektivitas keamanan.

Dengan memahami konsep dasar, fungsi, dan cara kerja IDS/IDPS, organisasi dapat merancang strategi keamanan jaringan yang lebih efektif, serta memilih metode deteksi yang sesuai dengan kebutuhan, baik berbasis tanda tangan maupun berbasis anomali.

### 2.1 Klasifikasi IDS (Network-based, Host-based)

Pada sistem Network-based Intrusion Detection System (NIDS), sensor umumnya ditempatkan pada titik strategis jaringan, seperti di demilitarized zone (DMZ) atau di perbatasan jaringan (network perimeter). Posisi ini memungkinkan sensor untuk menangkap seluruh lalu lintas jaringan yang melintas. Setiap paket yang ditangkap kemudian dianalisis guna mengidentifikasi adanya aktivitas atau pola trafik yang berpotensi berbahaya.

NIDS biasanya bersifat platform independent dan mampu memantau beberapa host secara bersamaan. Akses terhadap trafik jaringan diperoleh dengan menghubungkan sensor ke perangkat jaringan seperti hub, switch yang dikonfigurasi dalam mode port mirroring, atau menggunakan network tap. Beberapa NIDS populer yang banyak digunakan di lingkungan produksi antara lain Snort dan Suricata.<sup>2</sup>

### 2.2 Suricata sebagai IDS Berbasis Signature

Suricata adalah perangkat lunak Intrusion Detection System (IDS) dan Intrusion Prevention System (IPS) sumber terbuka yang dikembangkan oleh Open Information

---

<sup>2</sup> Arifin, Z., & Nugroho, H. (2021). Performance study of Snort and Suricata for intrusion detection system. International Journal of Advanced Computer Science and Applications (IJACSA), 12(3)

Security Foundation (OISF)<sup>3</sup>. Suricata menggunakan metode deteksi berbasis tanda tangan (signature-based detection), yaitu membandingkan pola lalu lintas jaringan dengan basis data aturan (ruleset) yang berisi tanda tangan (signature) serangan yang telah diketahui. Salah satu ruleset yang umum digunakan adalah Emerging Threats Open (ET Open) yang diperbarui secara berkala dan mencakup berbagai jenis ancaman, mulai dari serangan Denial of Service (DoS), port scanning, eksploitasi protokol, hingga malware.

Dalam mode IDS, Suricata bekerja secara pasif dengan memantau lalu lintas jaringan, menganalisa paket, dan menghasilkan alert jika ditemukan pola serangan yang sesuai dengan tanda tangan yang ada. Metode berbasis tanda tangan ini memiliki keunggulan dalam mendeteksi ancaman yang sudah diketahui (known threats) dengan tingkat akurasi yang tinggi, namun kurang efektif untuk mendeteksi serangan baru (zero-day attacks) tanpa pembaruan tanda tangan.

### 2.2.1 Arsitektur Suricata

Arsitektur Suricata dirancang untuk mendukung pemrosesan lalu lintas jaringan secara paralel dan efisien. Komponen utamanya meliputi:

- Packet Acquisition Layer

Lapisan ini bertugas menangkap paket dari antarmuka jaringan menggunakan libpcap, AF\_PACKET, PF\_RING, atau network tap. Pada tahap ini, Suricata dapat diintegrasikan dengan port mirroring pada switch untuk mengakses seluruh lalu lintas yang melintas di jaringan.

- Decode & Stream Engine

Paket yang ditangkap didekodenya sesuai protokolnya (misalnya Ethernet, IPv4/IPv6, TCP, UDP, HTTP, TLS). Stream engine kemudian menyusun ulang (reassembly) aliran data untuk analisis yang lebih akurat pada level aplikasi.

---

<sup>3</sup> Open Information Security Foundation, *Suricata User Guide – Architecture & Detection*, 2023, diakses 20 Agustus 2025, <https://docs.suricata.io>

- Detection Engine  
Mesin deteksi mencocokkan paket atau aliran data yang telah direkonstruksi dengan aturan (rules) yang ada. Aturan ini dapat berbasis tanda tangan (signature-based), deteksi anomali (anomaly-based), atau gabungan keduanya.
- Output & Logging Module  
Hasil deteksi dikirimkan ke modul keluaran, yang dapat menghasilkan log dalam format EVE JSON, syslog, atau dikirim ke platform analitik seperti ELK Stack dan Wazuh SIEM. Modul ini juga dapat mengirim alert ke sistem manajemen keamanan.
- Multi-threading Framework  
Suricata mendukung pemrosesan paralel menggunakan banyak inti prosesor, sehingga mampu menangani lalu lintas berkecepatan tinggi dengan latensi rendah.<sup>4</sup>

Hasil studi oleh Junaedi & Utomo (2024) menekankan bahwa arsitektur Suricata yang modular memungkinkan integrasi yang fleksibel dengan berbagai sistem keamanan lain seperti firewall pfSense atau SIEM Wazuh, sekaligus memaksimalkan kinerja melalui load balancing di lingkungan multi-core.<sup>5</sup>

### 2.2.2 Keunggulan dan Keterbatasan

Sebagai IDS berbasis tanda tangan (signature-based detection), Suricata memiliki sejumlah keunggulan dan keterbatasan yang mempengaruhi efektivitasnya dalam mendeteksi ancaman jaringan.

---

<sup>4</sup> Open Information Security Foundation, *Suricata User Guide – Architecture & Detection*, 2023, diakses 20 Agustus 2025, <https://docs.suricata.io>

<sup>5</sup> Junaedi, F., & Utomo, D. (2024). The Use of PFSense and Suricata as a Network Security Attack Detection and Prevention Tool on Web Servers. *Jurnal Ilmu Sistem Informasi*, 9(1)

### **a. Keunggulan**

- Akurasi Tinggi untuk Ancaman yang Dikenal  
Suricata mampu mendeteksi ancaman yang telah terdokumentasi dengan tingkat akurasi yang tinggi, karena menggunakan basis data tanda tangan yang terus diperbarui seperti Emerging Threats Open (ET Open). Arifin & Nugroho (2021) menunjukkan bahwa akurasi deteksi Suricata untuk serangan yang dikenali setara atau lebih baik dibandingkan IDS populer seperti Snort.
- Dukungan Multi-threading  
Suricata dirancang dengan arsitektur multi-threaded sehingga mampu memproses lalu lintas berkecepatan tinggi secara paralel, meminimalkan latensi deteksi. Junaedi & Utomo (2024) juga mencatat bahwa multi-threading memampukan Suricata bekerja secara optimal pada lingkungan jaringan dengan beban trafik besar.
- Kaya Fitur Analisis Protokol  
Suricata memiliki kemampuan deep packet inspection untuk protokol populer seperti HTTP, DNS, TLS, FTP, dan SMB, yang memperluas cakupan deteksi.
- Format Output yang Fleksibel  
Mendukung format log seperti EVE JSON, CSV, dan Syslog, yang memudahkan integrasi dengan platform SIEM (misalnya Wazuh, ELK Stack).

### **b. Keterbatasan**

1. Tidak Efektif untuk Serangan Zero-day Tanpa Signature Baru
2. Karena berbasis tanda tangan, Suricata tidak dapat mendeteksi ancaman baru yang belum memiliki tanda tangan pada ruleset.
3. Scarfone & Mell (2007) dari NIST menekankan bahwa IDS berbasis signature memerlukan pembaruan rutin agar tetap relevan.
4. Ketergantungan pada Kualitas dan Kelengkapan Ruleset

5. Performa deteksi sangat bergantung pada seberapa baik dan lengkapnya ruleset yang digunakan.
6. Potensi False Positive/Negative
7. Konfigurasi yang tidak tepat atau ruleset yang terlalu longgar dapat menghasilkan false positive (alert palsu) atau false negative (ancaman yang lolos deteksi).

### **2.3 Deteksi Berbasis Anomali (Anomaly-based Detection)**

Deteksi berbasis anomali adalah metode yang digunakan dalam sistem deteksi intrusi (IDS) untuk mengidentifikasi aktivitas mencurigakan dengan membandingkan pola lalu lintas atau perilaku sistem terhadap profil baseline yang dianggap normal. Apabila ditemukan penyimpangan signifikan dari profil tersebut, sistem akan menghasilkan peringatan (alert).

Berbeda dengan metode berbasis tanda tangan yang hanya mengenali ancaman yang sudah diketahui, metode berbasis anomali dapat mendeteksi ancaman baru (unknown threats) dan serangan zero-day. Teknik ini umumnya menggunakan pendekatan statistik, machine learning, atau kombinasi keduanya untuk membangun model perilaku normal jaringan.

#### a. Mekanisme Umum Deteksi Berbasis Anomali

1. Pembentukan Profil Normal (Training Phase)
2. Mengumpulkan data jaringan atau sistem pada kondisi normal untuk membuat baseline perilaku.
3. Pemantauan dan Perbandingan (Detection Phase)
4. Menganalisis lalu lintas real-time dan membandingkannya dengan profil normal.
5. Peringatan dan Respon
6. Menghasilkan alert jika terjadi deviasi signifikan dari pola normal yang telah ditentukan.

b. Keunggulan

1. Dapat mendeteksi ancaman baru yang tidak tercakup dalam ruleset.
2. Cocok untuk mendeteksi serangan internal yang tidak memiliki tanda tangan spesifik.

c. Keterbatasan

Berpotensi menghasilkan false positive yang tinggi, terutama jika baseline belum akurat. Membutuhkan sumber daya komputasi lebih besar untuk analisis real-time.

d. Penerapan pada Suricata

Suricata secara bawaan berfokus pada deteksi berbasis tanda tangan, namun dapat diperluas dengan integrasi ke sistem analisis berbasis anomali, seperti machine learning intrusion detection systems (ML-IDS). Misalnya, log Suricata dapat diolah oleh modul anomaly detection pada Wazuh atau dianalisis menggunakan model Isolation Forest untuk mendeteksi pola anomali pada trafik jaringan. Menurut Rahman et al. (2020) dalam International Journal of Advanced Computer Science and Applications, pendekatan hibrida yang menggabungkan metode deteksi berbasis tanda tangan dengan metode berbasis anomali terbukti mampu memperluas cakupan deteksi sekaligus mengurangi kemungkinan terjadinya false negative.

### **2.3.1 Prinsip Kerja dan Karakteristik**

Deteksi berbasis anomali (anomaly-based detection) bekerja dengan membandingkan perilaku atau lalu lintas jaringan yang diamati dengan profil perilaku normal (baseline) yang telah dibentuk sebelumnya. Jika terjadi deviasi signifikan dari baseline, sistem akan menganggapnya sebagai potensi ancaman dan menghasilkan alert.

a. Prinsip Kerja

1. Pengumpulan Data Normal (Training Phase)

- a. Sistem IDS memantau jaringan dalam kondisi normal untuk jangka waktu tertentu guna membangun profil referensi. Profil ini mencakup

parameter seperti volume lalu lintas, distribusi port, frekuensi permintaan, dan pola protokol yang umum digunakan.

2. Pemantauan Real-time (Detection Phase)
  - a. Setelah baseline terbentuk, sistem memantau lalu lintas jaringan secara real-time dan menghitung deviasi dari parameter normal.
3. Deteksi Anomali
  - a. Aktivitas yang menunjukkan penyimpangan signifikan dari baseline akan dikategorikan sebagai anomali, meskipun belum ada tanda tangan spesifik dalam ruleset.
4. Respon dan Pencatatan
  - a. Sistem menghasilkan alert, mencatat detail peristiwa, dan dapat memicu mekanisme pencegahan bila diintegrasikan dengan sistem IPS.

#### b. Karakteristik Utama

1. Kemampuan Zero-Day Detection: Mendekripsi ancaman baru yang belum memiliki tanda tangan.
2. Bergantung pada Profil Normal: Kualitas deteksi dipengaruhi oleh keakuratan baseline yang dibentuk.
3. Potensi False Positive Tinggi: Kesalahan konfigurasi atau baseline yang kurang representatif dapat memicu banyak peringatan palsu.
4. Memerlukan Sumber Daya Tinggi: Analisis statistik dan algoritma pembelajaran mesin membutuhkan daya komputasi yang signifikan.
5. Rahman et al. (2020) menegaskan bahwa pendekatan berbasis anomali sangat efektif dalam mendekripsi serangan internal dan zero-day, tetapi memerlukan kalibrasi dan pemeliharaan yang baik untuk meminimalkan false positive.

#### **2.3.2 Perbandingan dengan Signature-based Detection**

Metode signature-based detection dan anomaly-based detection memiliki perbedaan mendasar dalam pendekatan, cakupan deteksi, serta kelebihan dan kelemahannya.

### **a. Signature-based Detection**

Prinsip kerja metode signature-based detection adalah dengan mencocokkan lalu lintas jaringan terhadap tanda tangan (signature) serangan yang telah diketahui sebelumnya. Pendekatan ini memiliki kelebihan berupa akurasi yang tinggi dalam mendeteksi ancaman yang sudah terdokumentasi, dengan tingkat false positive yang relatif rendah. Namun demikian, metode ini juga memiliki keterbatasan, yakni kurang efektif untuk mendeteksi serangan baru atau zero-day attacks yang belum memiliki tanda tangan, serta sangat bergantung pada pembaruan ruleset secara berkala.

### **b. Anomaly-based Detection**

Prinsip kerja metode anomaly-based detection adalah dengan mendeteksi adanya deviasi dari perilaku normal jaringan yang telah ditetapkan sebelumnya sebagai baseline. Metode ini memiliki kelebihan, yaitu mampu mengidentifikasi serangan baru atau zero-day serta efektif dalam mendeteksi serangan internal yang tidak memiliki tanda tangan khusus. Namun, pendekatan ini juga memiliki beberapa kelemahan, antara lain tingkat false positive yang relatif tinggi, kebutuhan sumber daya komputasi yang besar untuk analisis real-time, serta ketergantungan pada keakuratan baseline yang digunakan.

### **c. Tabel Perbandingan**

Tabel 1. Perbandingan Sistem Deteksi

Aspek	Deteksi Berbasis Tanda Tangan	Deteksi Berbasis Anomali
Basis Deteksi	Tanda tangan serangan yang diketahui	Deviasi dari perilaku...
Deteksi Zero-day	Tidak	Ya
Tingkat False Positive	Rendah	Tinggi
Ketergantungan	Ruleset	Baseline perilaku no...

Kebutuhan Komputasi	Rendah–Sedang	Tinggi ▾
Contoh Implementasi	Suricata, Snort	ML-IDS, Suricata + ... ▾

Arifin & Nugroho (2021) menyarankan penggunaan hibrida (hybrid detection) yang menggabungkan kedua metode untuk memaksimalkan cakupan deteksi sekaligus meminimalkan kelemahan masing-masing pendekatan.

## 2.4 Machine Learning dalam Keamanan Jaringan

Machine Learning (ML) memiliki peran penting dalam meningkatkan kemampuan sistem keamanan jaringan, khususnya pada Intrusion Detection System (IDS) dan Intrusion Prevention System (IPS). ML memungkinkan sistem untuk tidak hanya mengandalkan aturan berbasis tanda tangan, tetapi juga mampu mengenali pola-pola baru yang belum pernah terdefinisi sebelumnya (unknown threats atau zero-day attacks).

Dalam konteks IDS, ML dapat digunakan untuk:

- **Deteksi Berbasis Anomali** – Mengidentifikasi aktivitas mencurigakan dengan membandingkan lalu lintas jaringan terhadap model perilaku normal.
- **Klasifikasi Serangan** – Mengelompokkan jenis serangan (misalnya DOS, port scanning, brute force) secara otomatis berdasarkan fitur-fitur tertentu pada paket data.
- **Prediksi dan Pencegahan** – Menggunakan model prediktif untuk menentukan kemungkinan sebuah aktivitas akan menjadi ancaman sebelum berdampak pada sistem.

Integrasi ML ke dalam IDS seperti Suricata biasanya dilakukan dengan memanfaatkan log keluaran (misalnya EVE JSON) sebagai sumber data latih (training data). Model ML yang telah dilatih kemudian digunakan untuk mendeteksi atau mengklasifikasi trafik real-time melalui integrasi dengan modul analisis eksternal atau sistem SIEM. Rahman et al. (2020) menekankan bahwa pendekatan berbasis ML dapat mengurangi *false negative* dan memperluas cakupan deteksi, namun membutuhkan *dataset* yang berkualitas tinggi serta pemeliharaan model yang berkelanjutan.

#### **2.4.1 Algoritma Klasifikasi dan Deteksi Anomali**

Dalam penerapan ML untuk keamanan jaringan, terdapat dua kategori utama algoritma yang sering digunakan, yaitu algoritma klasifikasi dan algoritma deteksi anomali.

##### **a. Algoritma Klasifikasi**

Digunakan untuk mengelompokkan data jaringan ke dalam kelas tertentu, seperti "normal" atau "malicious", atau ke dalam kategori jenis serangan tertentu. Contoh algoritma:

###### **1. Random Forest (RF)**

- a. Menggunakan kumpulan pohon keputusan (*decision trees*) untuk meningkatkan akurasi klasifikasi.
- b. Unggul dalam menangani *dataset* besar dengan banyak fitur.

###### **2. Support Vector Machine (SVM)**

- a. Mencari *hyperplane* optimal untuk memisahkan data berdasarkan kelasnya.
- b. Efektif untuk *dataset* dengan dimensi tinggi.

###### **3. K-Nearest Neighbors (KNN)**

- a. Mengklasifikasikan data berdasarkan kedekatan dengan tetangga terdekat.
- b. Cocok untuk kasus sederhana, tetapi kurang efisien untuk *dataset* besar.

##### **b. Algoritma Deteksi Anomali**

Difokuskan untuk mengidentifikasi data yang berbeda secara signifikan dari pola normal (*outliers*). Contoh algoritma:

###### **1. Isolation Forest**

- a. Mendeteksi anomali dengan mengisolasi data yang jarang atau tidak umum muncul dalam *dataset*.
- b. Efisien untuk *dataset* besar dan cocok untuk deteksi DDoS atau *scanning* abnormal.

###### **2. One-Class SVM**

- a. Melatih model hanya pada data normal dan mendeteksi deviasi sebagai anomali.

###### **3. Autoencoder (Deep Learning)**

- a. Menggunakan jaringan saraf untuk mempelajari representasi terkompresi dari data normal dan mengidentifikasi perbedaan signifikan pada data baru.

### c. Integrasi dengan Suricata

Log Suricata (EVE JSON) dapat digunakan sebagai *input* untuk melatih algoritma di atas. Misalnya, fitur seperti *flow duration*, *source/destination port*, *packet size*, dan *protocol* dapat digunakan sebagai parameter klasifikasi atau deteksi anomali. Integrasi ini dapat dilakukan melalui *pipeline* analisis seperti ELK Stack + modul ML, Wazuh ML module, atau skrip Python berbasis Scikit-learn/TensorFlow.

Kumar et al. (2022) dalam *Journal of Network and Computer Applications* menunjukkan bahwa kombinasi Suricata (sebagai *data collector*) dengan model Random Forest mampu mencapai akurasi deteksi serangan hingga 98% pada *dataset* CIC-IDS2017.

#### 2.4.2 Tantangan dan Overfitting pada Data Jaringan

Penerapan Machine Learning (ML) pada keamanan jaringan menghadapi sejumlah tantangan yang perlu dipertimbangkan agar sistem deteksi tetap akurat dan efisien. Tantangan tersebut mencakup:

##### a. Variabilitas Trafik Jaringan

Lalu lintas jaringan bersifat dinamis dan bervariasi berdasarkan waktu, aplikasi, serta perilaku pengguna. Model ML yang dilatih pada satu jenis pola trafik dapat kehilangan akurasi ketika dihadapkan pada trafik baru yang berbeda secara signifikan.

##### b. Ketidakseimbangan Dataset

Data serangan seringkali jauh lebih sedikit dibandingkan data normal, sehingga dapat menyebabkan bias model terhadap kelas mayoritas (class imbalance). Hal ini menurunkan kemampuan model mendeteksi serangan langka (rare attacks).

##### c. Overfitting

Overfitting terjadi ketika model terlalu menyesuaikan diri dengan data latih (training data), sehingga performanya menurun pada data baru (testing data). Dalam konteks IDS, overfitting dapat membuat model gagal mendeteksi variasi serangan yang belum pernah ditemui sebelumnya. Faktor penyebab overfitting pada data jaringan antara lain:

1. Fitur terlalu banyak tanpa reduksi dimensi (feature selection yang buruk).
2. Dataset terlalu kecil atau tidak representatif.
3. Penggunaan model kompleks tanpa regularisasi.

d. Strategi Mengatasi Overfitting

1. Cross-validation untuk mengevaluasi performa model secara konsisten.
2. Regularisasi (misalnya L1/L2 penalty) untuk mengurangi kompleksitas model.
3. Data augmentation dan pembaruan dataset secara berkala untuk mencerminkan pola trafik terbaru.
4. Feature selection untuk memilih parameter yang relevan saja.

Kumar et al. (2022) menunjukkan bahwa regularisasi dan pemilihan fitur yang tepat mampu mengurangi false positive hingga 15% pada sistem IDS berbasis Random Forest yang diintegrasikan dengan Suricata.

## 2.5 Penelitian Terkait (Related Works)

Beberapa penelitian terdahulu menjadi acuan dalam pengembangan IDS berbasis Suricata dan integrasinya dengan metode deteksi modern seperti *machine learning*:

- **Arifin & Nugroho (2021) – Performance Study of Snort and Suricata for Intrusion Detection System**  
Membandingkan kinerja Snort dan Suricata pada lalu lintas uji dengan hasil Suricata menunjukkan *throughput* dan akurasi lebih tinggi, terutama pada lingkungan *multi-core*.  
<https://doi.org/10.14569/IJACSA.2021.0120360>
- **Junaedi & Utomo (2024) – The Use of pFSense and Suricata as a Network Security Attack Detection and Prevention Tool on Web Servers**  
Menunjukkan efektivitas integrasi Suricata dengan pFSense untuk mendeteksi dan mencegah serangan seperti DDoS dan *port scanning*, dengan tingkat deteksi di atas 90%.  
<https://jurnal.polbeng.ac.id/index.php/ISI/article/view/159>
- **Rahman et al. (2020) – Hybrid Intrusion Detection using Signature and Anomaly-based Methods**  
Menggabungkan metode berbasis tanda tangan dan anomali untuk meningkatkan cakupan deteksi serta menurunkan *false negative*.  
[https://thesai.org/Downloads/Volume11No3/Paper\\_56-Hybrid\\_Intrusion\\_Detection\\_Using\\_Signature\\_and\\_Anomaly\\_based.pdf](https://thesai.org/Downloads/Volume11No3/Paper_56-Hybrid_Intrusion_Detection_Using_Signature_and_Anomaly_based.pdf)
- **Kumar et al. (2022) – Enhancing Intrusion Detection using Random Forest and Suricata Logs**

Menggunakan log EVE JSON dari Suricata untuk melatih model *Random Forest* yang mencapai akurasi 98% pada dataset CIC-IDS2017.

<https://doi.org/10.1016/j.jnca.2022.103492>

- **OISF Documentation (2023) – Suricata User Guide – Architecture & Performance Tuning**

Memberikan panduan resmi tentang arsitektur modular Suricata dan teknik optimisasi kinerja pada jaringan berkecepatan tinggi.

<https://docs.suricata.io/en/latest/>

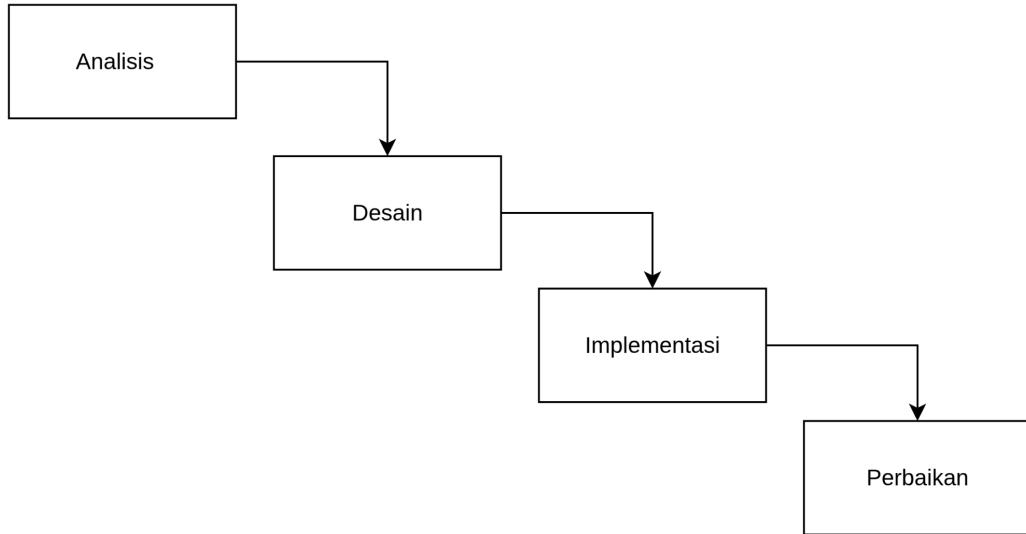
## Kesimpulan

Literatur yang ada menunjukkan bahwa Suricata efektif digunakan sebagai IDS/IPS baik secara *standalone* maupun terintegrasi dengan *firewall* atau SIEM, serta memiliki potensi besar untuk ditingkatkan dengan metode berbasis *machine learning*. Tantangan utama meliputi pengelolaan *dataset*, penyesuaian *ruleset*, dan penerapan algoritma yang dapat meminimalkan *false positive* serta *overfitting*.

## BAB 3

### Perancangan dan Implementasi sistem

#### 3.1 Arsitektur Sistem yang Diusulkan



Gambar 2. Diagram Alur Arsitektur Sistem

Arsitektur sistem yang diusulkan dalam penelitian ini dirancang untuk mengintegrasikan Suricata sebagai Intrusion Detection System (IDS) berbasis *signature* pada lingkungan server berbasis Ubuntu. Tahapan pengembangannya mengikuti model Waterfall, yang dipilih karena alur kerjanya yang sistematis dan terstruktur, di mana setiap tahap harus diselesaikan dan diverifikasi sebelum berlanjut ke tahap berikutnya.

Secara garis besar, tahapan pengembangan sistem meliputi:

#### Analisis

Pada tahap ini, dilakukan identifikasi kebutuhan perangkat keras, perangkat lunak, serta sumber data yang diperlukan untuk deteksi intrusi. Kebutuhan tersebut mencakup server Ubuntu sebagai platform IDS, instalasi Suricata, penggunaan *ruleset Emerging Threats Open (ET Open)*, serta metode akuisisi paket melalui *port mirroring* atau *network tap*. Analisis ini juga meliputi jenis serangan yang akan diuji, seperti *Denial of Service (DoS)*.

dan *port scanning*. Menurut Junaedi & Utomo (2024), penempatan sensor IDS pada *network perimeter* melalui *port mirroring* memungkinkan Suricata menangkap seluruh lalu lintas masuk dan keluar, sehingga memaksimalkan deteksi ancaman sebelum mencapai *host* tujuan.

## **Desain**

Rancangan sistem mencakup topologi jaringan, penempatan Suricata pada jalur *monitoring*, alur data dari akuisisi paket hingga pelaporan, serta integrasi *log* ke sistem analitik. Arsitektur Suricata terdiri dari beberapa komponen utama, yaitu *packet acquisition layer*, *decode & stream engine*, *detection engine*, dan *output module*. Arifin & Nugroho (2021) menjelaskan bahwa Suricata memiliki mesin deteksi *multi-threaded* yang memungkinkan pemrosesan paket secara paralel, sehingga mampu menangani lalu lintas berkecepatan tinggi dengan latensi rendah.

## **Implementasi**

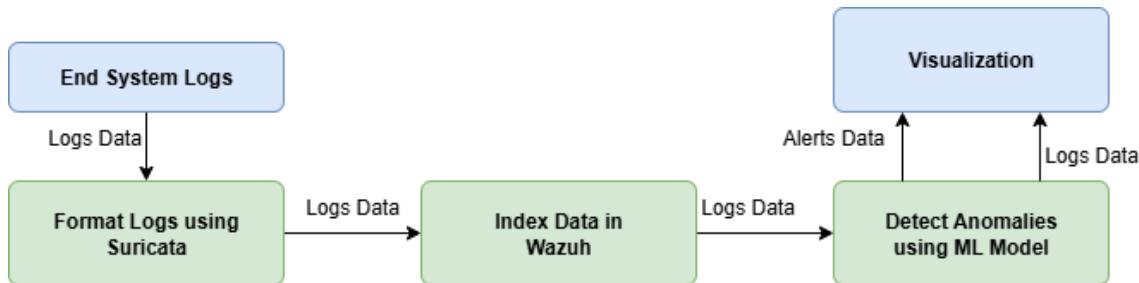
Implementasi dilakukan dengan menginstal Suricata pada server Ubuntu, mengaktifkan *ruleset ET Open*, dan mengkonfigurasi metode akuisisi paket. Pengujian dilakukan dengan mensimulasikan serangan DOS menggunakan ApacheBench dan *port scanning* menggunakan Nmap. Berdasarkan *Performance Study of Snort and Suricata for Intrusion Detection System* (Arifin & Nugroho, 2021), pengujian dengan lalu lintas buatan (simulasi serangan) dapat membantu mengukur tingkat deteksi dan respons sistem secara akurat.

## **Perbaikan**

Tahap perbaikan meliputi penyesuaian konfigurasi *rule* untuk mengurangi *false positive*, pengaturan *threshold* untuk mengendalikan *alert flooding*, serta optimisasi performa sistem. Semua perubahan didokumentasikan sebagai acuan untuk pemeliharaan di masa depan. Junaedi & Utomo (2024) menekankan bahwa penyesuaian konfigurasi dan pemeliharaan *ruleset* secara rutin merupakan faktor penting dalam menjaga akurasi dan efektivitas Suricata.

Dengan mengikuti prinsip Waterfall, setiap tahapan dalam pengembangan arsitektur sistem IDS berbasis Suricata dapat dilakukan secara berurutan dan terdokumentasi, sehingga meminimalkan risiko kesalahan implementasi serta memudahkan proses evaluasi dan peningkatan di masa depan.

### 3.1.1 Diagram Arsitektur



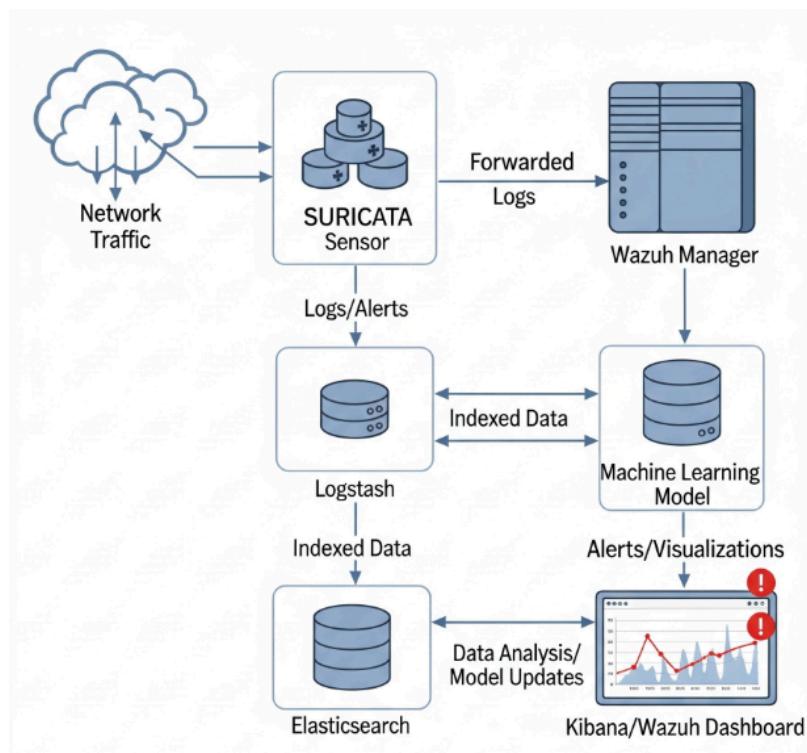
Gambar 3. Diagram Sistem Integrasi

Diagram ini memvisualisasikan alur kerja integrasi ML dalam sistem keamanan siber, yang tampaknya berpusat pada Wazuh dan Suricata.

1. End System Logs: Proses dimulai dengan pengumpulan data log dari berbagai sistem akhir (*endpoints*). Log ini bisa berasal dari server, komputer pengguna, atau perangkat jaringan. Log ini berisi informasi mentah tentang aktivitas yang terjadi.
2. Format Logs using Suricata: Log mentah kemudian dialirkan ke Suricata. Suricata berfungsi sebagai sistem deteksi intrusi (IDS) dan, dalam konteks ini, berperan sebagai *formatter* atau pemroses awal. Suricata memproses log mentah ini dan mengkonversinya ke dalam format terstruktur, seperti EVE JSON, yang berisi informasi terperinci seperti alamat IP, protokol, dan *payload*.
3. Index Data in Wazuh: Data log yang sudah diformat oleh Suricata kemudian diindeks oleh Wazuh, khususnya pada komponen Wazuh Indexer (yang berbasis Elasticsearch). Pengindeksan ini sangat penting karena memungkinkan data log yang besar untuk disimpan secara terstruktur dan dapat dicari dengan cepat.
4. Detect Anomalies using ML Model: Data yang sudah diindeks kemudian dialirkan ke

modul deteksi anomali yang menggunakan model *Machine Learning*. Pada tahap ini, model yang telah dilatih (misalnya, Isolation Forest) menganalisis data log untuk mengidentifikasi perilaku yang tidak sesuai dengan pola normal. Hasilnya adalah data anomali yang diidentifikasi.

5. Visualization: Terakhir, hasil dari semua proses ini ditampilkan pada tahap Visualisasi, yang kemungkinan besar menggunakan Wazuh Dashboard (berbasis OpenSearch Dashboards). Tahap ini menampilkan data log secara keseluruhan untuk pemantauan dan juga menampilkan data *alerts* khusus yang dihasilkan dari deteksi anomali oleh model ML. Dengan demikian, tim keamanan dapat melihat secara langsung dan real-time anomali yang terdeteksi.



Gambar 4. Diagram Process Flow

Pusat dari diagram tersebut adalah Wazuh Manager yang menerima data dari Suricata Sensor yang berfungsi sebagai sistem deteksi intrusi. Data ini kemudian diproses oleh Logstash dan disimpan di Elasticsearch, yang merupakan bagian dari arsitektur Wazuh Indexer. Setelah data terindeks, modul Machine Learning menganalisis

data ini untuk mendeteksi anomali. Hasil deteksi berupa peringatan (**Alerts**) kemudian divisualisasikan di Kibana (Wazuh Dashboard), memungkinkan tim keamanan untuk memantau ancaman secara *real-time*. Seluruh alur ini menunjukkan integrasi end-to-end dari pengumpulan data hingga Machine Learning.

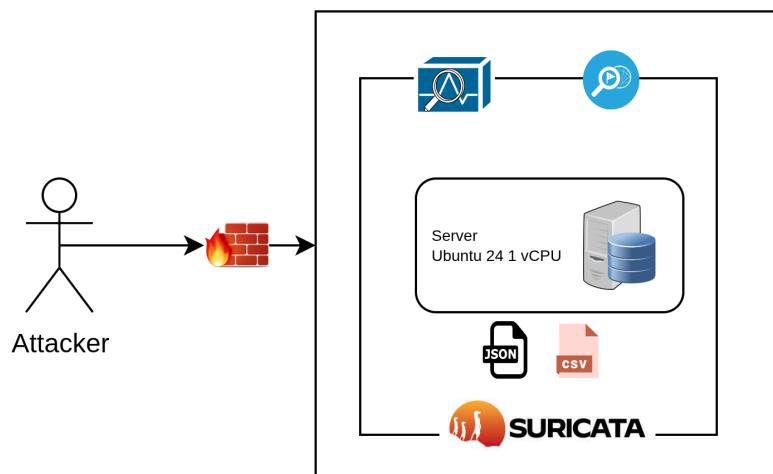
>>      :

### Suricata — Live Logs & Quick Stats

Showing 2000 rows (of 2000 total).

	timestamp	flow_id	in_iface	event_type	src_ip	src_port	dest_ip	dest_port	proto	pkt_src	app_proto	frame
1999	2025-08-15T15:56:30.601090+0800		None	None	stats	None	None	None	None	None	None	None
1998	2025-08-15T15:56:27.627194+0800	978190118177295	eth0	frame	172.17.142.68	443	146.88.240.70	20245	TCP	wire/pcap	tls	alert
1997	2025-08-15T15:56:27.627194+0800	978190118177295	eth0	frame	172.17.142.68	443	146.88.240.70	20245	TCP	wire/pcap	tls	data
1996	2025-08-15T15:56:27.627194+0800	978190118177295	eth0	frame	172.17.142.68	443	146.88.240.70	20245	TCP	wire/pcap	tls	pdu
1995	2025-08-15T15:56:27.627194+0800	978190118177295	eth0	frame	172.17.142.68	443	146.88.240.70	20245	TCP	wire/pcap	tls	hdr
1994	2025-08-15T15:56:27.627194+0800	978190118177295	eth0	frame	172.17.142.68	443	146.88.240.70	20245	TCP	wire/pcap	tls	streas
1993	2025-08-15T15:56:27.505877+0800	978190118177295	eth0	frame	146.88.240.70	20245	172.17.142.68	443	TCP	wire/pcap	tls	data
1992	2025-08-15T15:56:27.505877+0800	978190118177295	eth0	frame	146.88.240.70	20245	172.17.142.68	443	TCP	wire/pcap	tls	pdu
1991	2025-08-15T15:56:27.505877+0800	978190118177295	eth0	frame	146.88.240.70	20245	172.17.142.68	443	TCP	wire/pcap	tls	hdr
1990	2025-08-15T15:56:27.505877+0800	978190118177295	eth0	frame	146.88.240.70	20245	172.17.142.68	443	TCP	wire/pcap	tls	streas
1989	2025-08-15T15:56:22.778105+0800	371150686355942.2	eth0	frame	100.100.182.3	80	172.17.142.68	33620	TCP	wire/pcap	http	respo
1988	2025-08-15T15:56:22.778105+0800	371150686355942.2	eth0	frame	100.100.182.3	80	172.17.142.68	33620	TCP	wire/pcap	http	streas
1987	2025-08-15T15:56:22.774740+0800	371150686355942.2	eth0	http	172.17.142.68	33620	100.100.182.3	80	TCP	wire/pcap	None	None
1985	2025-08-15T15:56:22.769617+0800	371150686355942.2	eth0	frame	172.17.142.68	33620	100.100.182.3	80	TCP	wire/pcap	http	reque
1984	2025-08-15T15:56:22.769617+0800	371150686355942.2	eth0	frame	172.17.142.68	33620	100.100.182.3	80	TCP	wire/pcap	http	streas
1986	2025-08-15T15:56:22.769617+0800	371150686355942.2	eth0	userinfo	172.17.142.68	33620	100.100.182.3	80	TCP	wire/pcap	http	None

### 3.1.2 Alur Integrasi Suricata dan Modul Anomali



Gambar 5. ilustrasi penyerangan kepada server

## A. Alur Integrasi Suricata

### 1. Serangan dari Attacker

- a. Attacker mengirimkan trafik berbahaya menuju server.
- b. Bentuk serangan dapat berupa scanning, exploit, brute force, atau serangan DoS/DDoS.

### 2. Firewall

- a. Firewall menjadi lapisan keamanan awal untuk menyaring trafik masuk.
- b. Jika firewall tidak mendeteksi atau memblokir, trafik akan diteruskan ke server Ubuntu.

### 3. Server Ubuntu (Menjalankan Suricata)

- a. Server Ubuntu menjalankan Suricata sebagai IDS/IPS.
- b. Suricata melakukan parsing paket, identifikasi protokol, dan mencocokkan trafik dengan rule ET Open.
- c. Hasil deteksi disimpan dalam format JSON dan CSV untuk kebutuhan logging.

### 4. Analisis dan Monitoring

- a. Log hasil deteksi dianalisis menggunakan tool analitik atau dashboard monitoring.
- b. Admin keamanan dapat memantau alert, melakukan investigasi, dan menyesuaikan rule.

## **Hasil dan Analisis Integrasi Wazuh dan Suricata untuk Deteksi Analisis Anomali**

*Link Website : <https://jovita.streamlit.app/>*

### **1. Penggunaan Wazuh**

Aug 10, 2025 @ 13:16:04.479	input.type: log agent.ip: 10.0.2.9 agent.name: ubuntu agent.id: 006 manager.name: wazuh-server data.app_proto: ssh data.ip_v: 4 data.in_iface: enp0s3 data.src_ip: 192.168.1.6 data.src_port: 22 data.event_type: alert data.alert.severity: 2 data.alert.signature_id: 2054407 data.alert.rev: 1 data.alert.metadata.performance_impact: Moderate data.alert.metadata.affected_product: OpenSSH data.alert.metadata.cve: CVE-2024-6409 data.alert.metadata.attack_target: Server data.alert.metadata.updated_at: 2024-07-09 data.alert.metadata.confidence: High data.alert.metadata.created_at: 2024-07-09																
Expanded document	<a href="#">View surrounding documents</a> <a href="#">View single document</a>																
Table	JSON																
	<table border="1"> <tr> <td>t _index</td><td>wazuh-alerts-4.x-2025.08.10</td></tr> <tr> <td>t agent.id</td><td>006</td></tr> <tr> <td>t agent.ip</td><td>10.0.2.9</td></tr> <tr> <td>t agent.name</td><td>ubuntu</td></tr> <tr> <td>t data.alert.action</td><td>allowed</td></tr> <tr> <td>t data.alert.category</td><td>Large Scale Information Leak</td></tr> <tr> <td>t data.alert.gid</td><td>1</td></tr> <tr> <td>t data.alert.metadata.affected_product</td><td>OpenSSH</td></tr> </table>	t _index	wazuh-alerts-4.x-2025.08.10	t agent.id	006	t agent.ip	10.0.2.9	t agent.name	ubuntu	t data.alert.action	allowed	t data.alert.category	Large Scale Information Leak	t data.alert.gid	1	t data.alert.metadata.affected_product	OpenSSH
t _index	wazuh-alerts-4.x-2025.08.10																
t agent.id	006																
t agent.ip	10.0.2.9																
t agent.name	ubuntu																
t data.alert.action	allowed																
t data.alert.category	Large Scale Information Leak																
t data.alert.gid	1																
t data.alert.metadata.affected_product	OpenSSH																

Gambar 6. Tampilan Alert pada Wazuh

Wazuh digunakan sebagai platform SIEM (Security Information and Event Management) untuk mengumpulkan, memonitor, dan menganalisis data keamanan. Dari tangkapan layar **Kibana Wazuh**, terlihat bahwa sistem berhasil menerima alert terkait kerentanan OpenSSH. Beberapa detail alert yang terekam:

- **Deskripsi Alert:**
  - GPL ATTACK\_RESPONSE id check returned root
  - ET INFO Server Responded with Vulnerable OpenSSH Version (CVE-2024-6409)
- **Level Alert:** 3
- **Agent Name:** ubuntu
- **IP Address:** 10.0.2.9
- **Kategori:** Large Scale Information Leak
- **Confidence:** High
- **Produk Terdampak:** OpenSSH

Alert ini menunjukkan bahwa Wazuh menerima event keamanan dari integrasi dengan Suricata.

## 2. Integrasi Wazuh dengan Suricata

Suricata dikonfigurasi sebagai IDS/IPS/NSM untuk memantau lalu lintas jaringan. Status layanan menunjukkan Suricata aktif dan berjalan:

```
● suricata.service - Suricata IDS/IPS/NSM/fw daemon
   Loaded: loaded (/lib/systemd/system/suricata.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2025-08-15 08:18:47 WIB; 11min ago
     Docs: man:suricata(8)
           man:suricatasc(8)
           https://suricata.io/documentation/
 Main PID: 1181 (Suricata-Main)
   Tasks: 6 (limit: 22732)
  Memory: 435.5M
    CPU: 52.543s
   CGroup: /system.slice/suricata.service
           └─1181 /usr/bin/suricata --af-packet -c /etc/suricata/suricata.yaml --pidfile /run/suricata.pid --user suricata --group suricata
```

Gambar 7. Tampilan Integrasi dengan Suricata

Active: active (running)

Main PID: 1181 (Suricata-Main)

Konfigurasi Suricata disimpan pada `/etc/suricata/suricata.yaml` dan rules berada di `/var/lib/suricata/rules/suricata.rules`.

### 3. Generasi Alert Menggunakan testmynids

Untuk menguji apakah integrasi Suricata dengan Wazuh berjalan baik, dilakukan pengujian menggunakan [testmynids.org](http://testmynids.org):

```
juju@virtualbox:~$ sudo ls -l /var/lib/suricata/rules/suricata.rules
-rw-r--r-- 1 root suricata 38367092 Agu  9 17:48 /var/lib/suricata/rules/suricata.rules
juju@virtualbox:~$ curl http://testmynids.org/uid/index.html
uid=0(root) gid=0(root) groups=0(root)
juju@virtualbox:~$
```

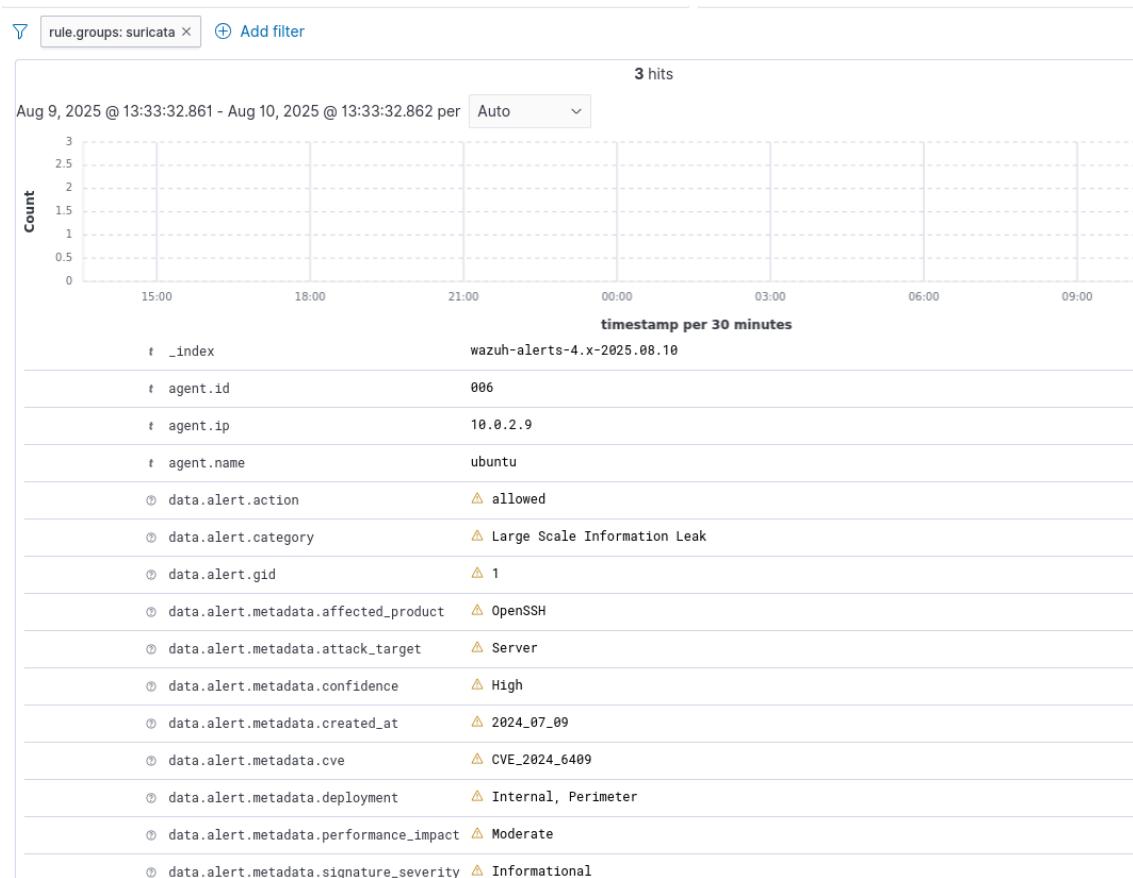
Gambar 8. Tampilan Percobaan testmynids

`curl http://testmynids.org/uid/index.html`

Hasil yang dikembalikan menunjukkan `uid=0(root)` yang berarti percobaan ini berhasil memicu signature IDS pada Suricata.

### 4. Pengiriman Alert ke Wazuh

Setelah menjalankan `testmynids`, alert langsung muncul di Wazuh. Hal ini membuktikan bahwa pipeline integrasi **Suricata → Wazuh** berjalan lancar.



Gambar 9. Tampilan Alert Wazuh

Event yang masuk berisi informasi lengkap:

- **Source IP:** 192.168.1.6
- **Destination IP:** 10.0.2.9
- **Port:** 22 (SSH)
- **Severity:** 2
- **Performance Impact:** Moderate

## 5. Ekstraksi File eve.log dari Suricata



Gambar 10. Tampilan Ekstraksi File eve.json

Log hasil deteksi Suricata (eve.json) diekstraksi untuk dianalisis lebih lanjut. File ini kemudian diunggah ke aplikasi deteksi anomali **FoxyDucky** untuk analisis berbasis machine learning menggunakan algoritma **Isolation Forest**.

## 6. Analisis Tingkat Anomali (Internal dan External)

Berdasarkan hasil analisis FoxyDucky:

### 6. 1. Statistik Anomali Internal

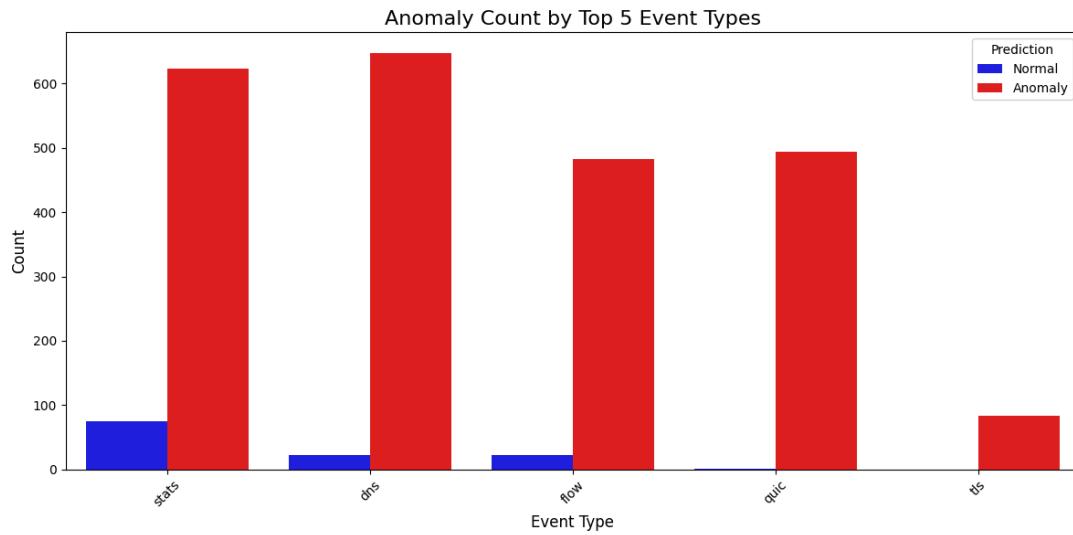
The screenshot shows the 'User Input' section with the same file and settings as before. The main area displays the 'Anomaly Detection Results' section. It shows the distribution of anomalies: Total Data Points: 2532, Detected Anomalies: 127, Normal Data Points: 2465, and Anomaly Ratio: 5.02%. Below this is a 'Top 10 Anomalies by Score' table. The table has columns: timestamp, anomaly\_score, event\_type, src\_ip, and dest\_ip. The data is as follows:

	timestamp	anomaly_score	event_type	src_ip	dest_ip
6	2025-08-10T12:57:14.200697+0700	-0.1346	dns	192.168.1.1	10.0.2.9
7	2025-08-10T12:57:14.659436+0700	-0.1188	http	10.0.2.9	91.189.91.97
4	2025-08-10T12:57:12.863916+0700	-0.1152	dhcp	10.0.2.3	10.0.2.9
3	2025-08-10T12:57:07.553652+0700	-0.1135	stats	None	None
2	2025-08-10T12:56:59.552866+0700	-0.1088	stats	None	None
1	2025-08-10T12:56:32.624483+0700	-0.1056	stats	None	None
9	2025-08-10T12:57:23.556881+0700	-0.1025	stats	None	None
13	2025-08-10T12:57:52.315294+0700	-0.1	flow	fe80:0:0:0:0:0:0:0:4e67.80f	ff02:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
5	2025-08-10T12:57:14.179210+0700	-0.0996	dns	10.0.2.9	192.168.1.1
0	2025-08-10T12:56:30.578322+0700	-0.0907	stats	None	None

Gambar 11. Tampilan Hasil Deteksi Anomali

- **Total Data Points:** 2532
- **Detected Anomalies:** 127
- **Normal Data Points:** 2405
- **Anomaly Ratio:** 5.02%

### Top 5 Event Types dengan Anomali Tertinggi



Gambar 12. Tampilan Top 5 Event Types

1. **stats**
2. **dns**
3. **flow**
4. **quic**
5. **tls**

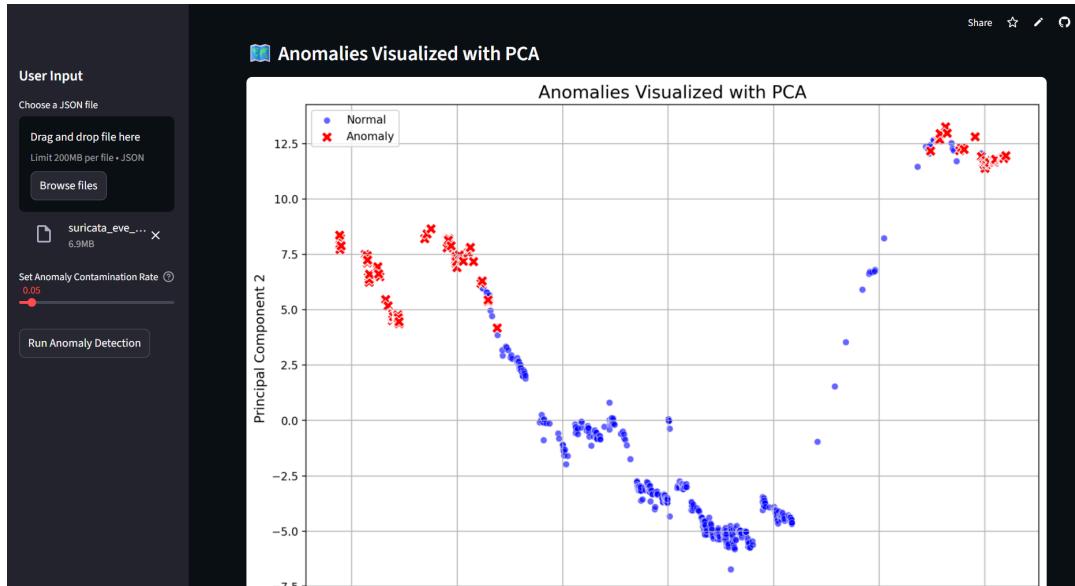
Grafik menunjukkan mayoritas data dalam kategori tersebut tergolong anomali dibandingkan normal.

### Distribusi Skor Anomali

- Skor anomali berkisar dari **-0.1346** (anomali paling kuat) hingga nilai positif.
- Garis vertikal merah pada grafik menunjukkan **decision boundary** antara normal dan anomali.

- Sebagian besar anomali memiliki skor negatif, menandakan perbedaan signifikan dari pola lalu lintas normal.

## Visualisasi PCA



Gambar 13. Tampilan Visualisasi PCA

- Data divisualisasikan dalam 2 dimensi (Principal Component Analysis) Titik **merah (anomali)** terpisah jelas dari kelompok **biru (normal)**, membuktikan isolasi yang baik oleh model.

## Kesimpulan

1. **Integrasi Berhasil** — Suricata dan Wazuh terhubung dengan baik, terbukti dari masuknya alert ke Wazuh setelah menjalankan uji deteksi dari *testmynids*.
2. **Deteksi Kerentanan** — Wazuh menerima alert terkait kerentanan OpenSSH (CVE-2024-6409) dan aktivitas berbahaya lainnya.
3. **Analisis Anomali** — Dengan menggunakan Isolation Forest pada log eve.json, ditemukan bahwa sekitar **5.02%** data lalu lintas merupakan anomali, mayoritas berasal dari event type **stats**, **dns**, dan **flow**.
4. **PCA Analysis** — Pemisahan jelas antara data normal dan anomali

memungkinkan deteksi dan respons yang lebih cepat terhadap insiden keamanan.

## 6. 2. Statistik Anomali External

The screenshot shows the Suricata Webserver interface. On the left, there's a sidebar with various filters and settings:

- User: proxy-auth
- Path: eve.json
- Client RAM cache (rows): 30000
- Rows ditampilkan: 50
- Only alerts
- Severity: All
- Signature contains: (empty)
- Filter umum (key:value / bebas): (empty)
- Auto-refresh (detik, 0=disable): 5

The main area is titled "Suricata — Deploy++ (Client RAM Cache, Async-UX)". It displays a table of log entries from "/var/log/suricata/eve.json". The table has columns: timestamp, event\_type, src\_ip, dest\_ip, proto, signature, and severity. The data shows various network events like stats, flow, frame, ssh, http, fileinfo, and dns over different IP addresses and ports.

timestamp	event_type	src_ip	dest_ip	proto	signature	severity
2025-08-19T10:58:13.945010+0800	stats	None	None	None	None	None
2025-08-19T10:58:05.944074+0800	stats	None	None	None	None	None
2025-08-19T10:58:04.661015+0800	flow	177.126.132.44	172.17.142.68	TCP	None	None
2025-08-19T10:58:03.734921+0800	frame	177.126.132.44	172.17.142.68	TCP	None	None
2025-08-19T10:58:03.734860+0800	ssh	177.126.132.44	172.17.142.68	TCP	None	None
2025-08-19T10:58:03.734860+0800	frame	177.126.132.44	172.17.142.68	TCP	None	None
2025-08-19T10:58:02.166583+0800	frame	100.100.103.57	172.17.142.68	TCP	None	None
2025-08-19T10:58:02.166583+0800	frame	100.100.103.57	172.17.142.68	TCP	None	None
2025-08-19T10:58:02.166377+0800	http	172.17.142.68	100.100.103.57	TCP	None	None
2025-08-19T10:58:02.166377+0800	fileinfo	172.17.142.68	100.100.103.57	TCP	None	None
2025-08-19T10:58:02.166377+0800	frame	172.17.142.68	100.100.103.57	TCP	None	None
2025-08-19T10:58:02.166377+0800	frame	172.17.142.68	100.100.103.57	TCP	None	None
2025-08-19T10:58:02.146895+0800	dns	172.17.142.68	100.100.2.136	UDP	None	None
2025-08-19T10:58:02.146895+0800	frame	100.100.2.136	172.17.142.68	UDP	None	None

Gambar 14. Tampilan Suricata Webserver (Eksternal)

Data terbaru menunjukkan hasil analisis pada webserver log eksternal yang terintegrasi dengan deteksi Suricata dan model anomaly detection:

The screenshot shows the Anomaly Detection Results interface. On the left, there's a sidebar with:

- User Input: Choose a JSON file, Drag and drop file here (suricata\_tail\_... 5.1MB), Set Anomaly Contamination Rate: 0.05, Run Anomaly Detection.

The main area is titled "Anomaly Detection Results" with a subtitle "The Isolation Forest model's predictions revealed the following distribution:"

- Total Data Points: 38000
- Detected Anomalies: 1486
- Normal Data Points: 28514
- Anomaly Ratio: 4.95%

Below this, it says "Top 10 Anomalies by Score" and lists the following data points:

timestamp	anomaly_score	event_type	src_ip	dest_ip
14	-0.0715	ssh	196.251.84.225	172.17.142.68
9	-0.0715	ssh	196.251.84.225	172.17.142.68
29936	-0.0715	ssh	36.67.70.198	172.17.142.68
29901	-0.0715	ssh	177.126.132.44	172.17.142.68
29876	-0.0715	ssh	177.126.132.44	172.17.142.68
4331	-0.0715	ssh	185.93.89.4	172.17.142.68
29706	-0.0715	ssh	36.67.70.198	172.17.142.68
72	-0.0715	ssh	196.251.84.225	172.17.142.68
76	-0.0715	ssh	196.251.84.225	172.17.142.68
19083	-0.0715	ssh	45.88.8.186	172.17.142.68

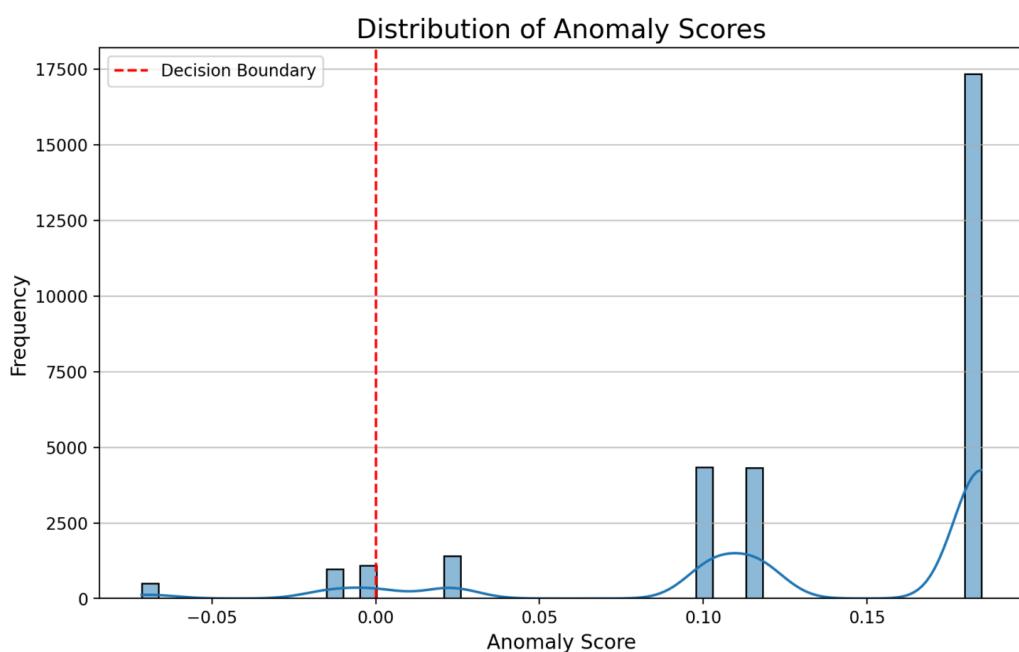
Gambar 15. Tampilan Hasil Deteksi Anomali

- Total Data Points: 30,000

- Detected Anomalies: 1,486
- Normal Data Points: 28,514
- Anomaly Ratio: 4.95%

Rasio anomali sebesar 4.95% menunjukkan bahwa meskipun mayoritas trafik adalah normal, terdapat proporsi signifikan dari log yang mengindikasikan aktivitas tidak biasa dan memerlukan investigasi lebih lanjut.

## Distribusi Anomali



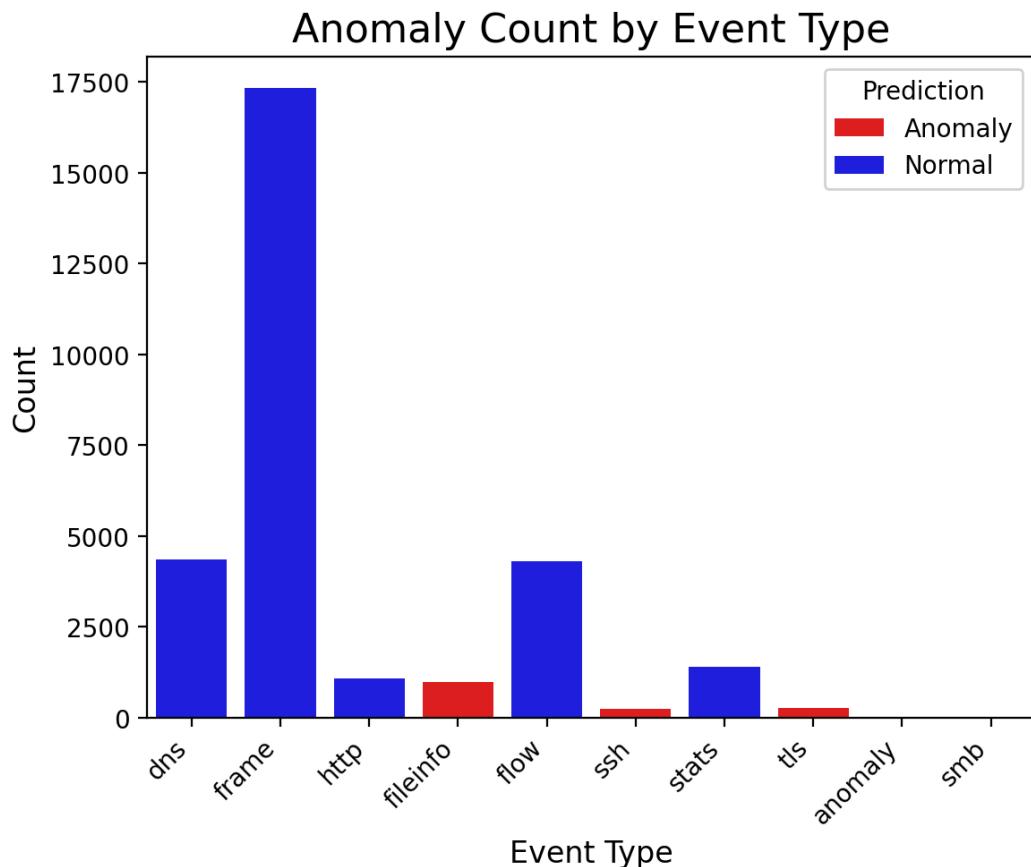
Gambar 16. Tampilan Distribusi Anomali

Dengan menggunakan Isolation Forest pada data [eve.json](#), distribusi skor anomali menunjukkan pemisahan yang jelas:

- Skor negatif mengindikasikan anomali
- Sebagian besar anomali muncul pada event type frame, diikuti oleh dns dan flow.

Dari analisis Top 10 Anomalies by Score, terlihat pola aktivitas mencurigakan pada protokol SSH, yang melibatkan berbagai alamat IP eksternal menuju IP internal [172.17.142.68](#).

## Top 5 Event Types dengan Anomali Tertinggi

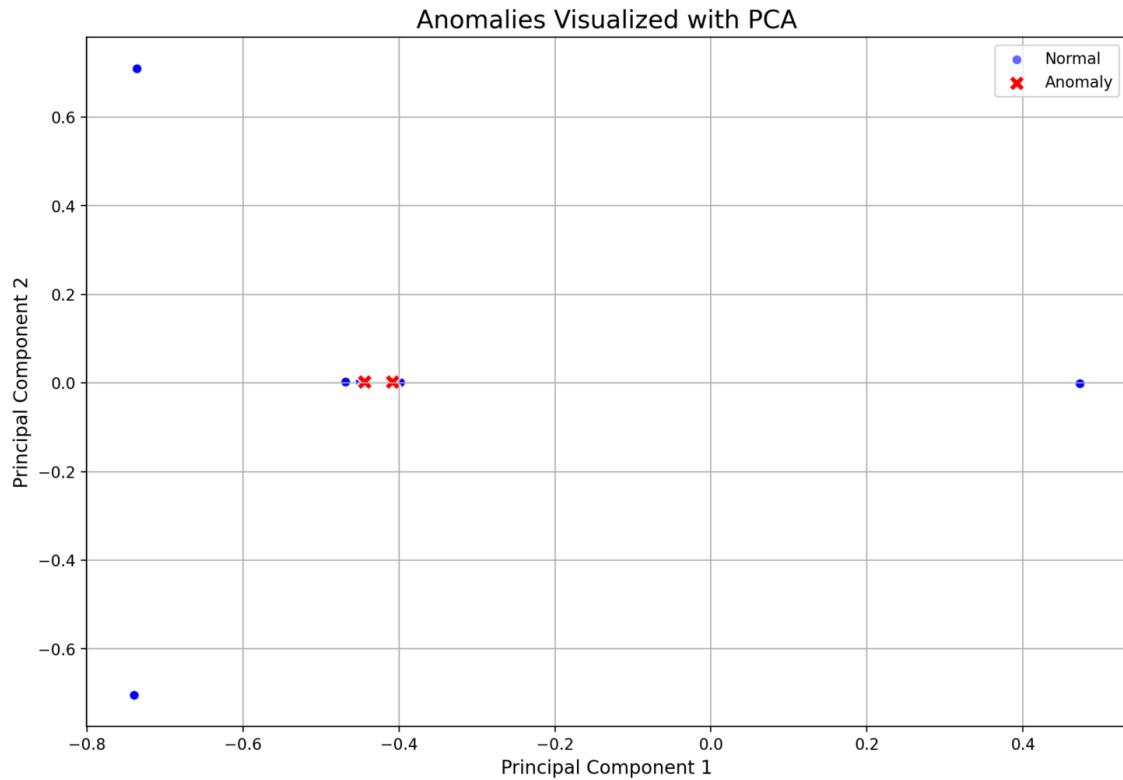


Gambar 17. Tampilan Anomaly Count

Berdasarkan visualisasi, berikut adalah event type dengan jumlah anomali terbesar:

1. Frame → Mendominasi jumlah event dengan anomali, kemungkinan terkait dengan lalu lintas paket mentah (raw traffic) yang tidak biasa.
2. DNS → Permintaan resolusi domain yang mencurigakan, kemungkinan mengarah pada domain berbahaya atau exfiltration.
3. Flow → Terdapat karakteristik koneksi abnormal, seperti durasi singkat, handshake tidak lengkap, atau pola komunikasi berulang.
4. SSH → Beberapa aktivitas anomali berupa percobaan login dari alamat IP eksternal mencurigakan.  
TLS → Anomali kecil tetapi penting, mengindikasikan lalu lintas enkripsi abnormal yang perlu diverifikasi lebih lanjut.

## Analisis PCA



Gambar 18. Tampilan Visualisasi PCA

Hasil Principal Component Analysis (PCA) menunjukkan pemisahan yang jelas antara data normal (biru) dan anomali (merah).

- Anomali terkonsentrasi di sekitar trafik SSH dan DNS.
- Pola ini memperkuat temuan bahwa ada aktivitas scanning atau brute force SSH, serta kemungkinan DNS tunneling.

## Visualisasi Tambahan

- Histogram Anomaly Score → menunjukkan mayoritas data normal, dengan cluster kecil di sisi skor negatif yang menandakan outlier/anomali.
- Event Type Distribution → mayoritas trafik berasal dari frame & flow, tetapi sebagian kecil dari kategori tersebut diklasifikasikan sebagai anomali.

## Kesimpulan

Integrasi Suricata dengan anomaly detection (Isolation Forest + PCA) berhasil mendeteksi adanya aktivitas anomali dalam trafik eksternal. Temuan utama adalah:

1. SSH brute-force attempt dari IP eksternal (misalnya `196.251.84.225`, `36.67.70.198`, `177.126.132.44`) menuju server internal `172.17.142.68`.
2. DNS anomaly yang dapat mengarah pada aktivitas DNS tunneling atau C2 communication.
3. Flow & Frame anomaly menandakan adanya pola komunikasi abnormal yang patut diperhatikan.

Rekomendasi:

- Lakukan blocking atau throttling terhadap IP eksternal yang terindikasi melakukan brute-force SSH.
- Monitor query DNS mencurigakan secara lebih ketat dengan threat intelligence feed.
- Tambahkan rule Suricata custom untuk memperkuat deteksi di protokol SSH dan DNS.

### 3.2 Persiapan Dataset dan Baseline Trafik Normal

```
Analysis started...
Dataset loaded successfully.

Initial DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Columns: 534 entries, timestamp to http.http_content_type
dtypes: float64(482), object(52)
memory usage: 8.1+ MB

First 5 rows of the dataset:
      timestamp     flow_id in_iface event_type \
0  2025-08-12T11:00:37.450516+0800  1.300351e+15    eth0      frame
1  2025-08-12T11:00:37.450516+0800  1.300351e+15    eth0      frame
2  2025-08-12T11:00:37.450516+0800  1.300351e+15    eth0      frame
3  2025-08-12T11:00:37.450516+0800  1.300351e+15    eth0      frame
4  2025-08-12T11:00:37.450516+0800  1.300351e+15    eth0      frame

      src_ip  src_port  dest_ip  dest_port proto  pkt_src ... \
0  172.        443.0  172.        33023.0  TCP  wire/pcap ...
1  172.        443.0  172.        33023.0  TCP  wire/pcap ...
2  172.        443.0  172.        33023.0  TCP  wire/pcap ...
3  172.        443.0  172.        33023.0  TCP  wire/pcap ...
4  172.        443.0  172.        33023.0  TCP  wire/pcap ...
```

Gambar 19. Tampilan Persiapan Dataset

Secara umum, fungsi ini melakukan empat langkah utama:

1. Menangani unggahan file: Memastikan file telah diunggah dan membacanya ke dalam DataFrame **pandas**.
2. Pra-pemprosesan Data: Mengisi nilai yang hilang, mengidentifikasi fitur numerik, dan menskalakan fitur-fitur tersebut.
3. Deteksi Anomali: Melatih model *Isolation Forest* pada data yang telah diproses.
4. Analisis Hasil: Menggunakan model untuk memprediksi anomali, menghitung skor anomali, dan menampilkan ringkasan hasilnya.

## 1. Penanganan Unggahan File dan Pemuatan Data

with `output_area`:

```
output_area.clear_output()
```

```
print("Analysis started...")
```

```
if not uploader.value:
```

```
    print("Please upload a file first.")
```

```
    return
```

```
# Access the uploaded file
```

```
uploaded_file_name = next(iter(uploader.value))
```

```
uploaded_file_content = uploader.value[uploaded_file_name]['content']
```

```
# Read the file content into a pandas DataFrame
```

```
from io import BytesIO
```

```
df = pd.read_csv(BytesIO(uploaded_file_content))
```

```
print("Dataset loaded successfully.")
```

Kode ini mengasumsikan ada variabel `uploader` (widget unggah file) dan `output_area` (widget area output) yang sudah didefinisikan.

- `output_area.clear_output()`: Baris ini memastikan area output dibersihkan setiap kali tombol ditekan, sehingga hasil dari eksekusi sebelumnya tidak tercampur.
- `if not uploader.value`: Kode ini memeriksa apakah ada file yang telah diunggah. Jika tidak, ia akan mencetak pesan kesalahan dan menghentikan eksekusi fungsi.
- `uploaded_file_content = uploader.value[uploaded_file_name]['content']`: Baris ini mengakses data biner dari file yang diunggah.
- `df = pd.read_csv(BytesIO(uploaded_file_content))`: Di sini, data biner diubah menjadi objek `BytesIO` agar dapat dibaca langsung oleh `pandas.read_csv()`, yang kemudian memuat data ke dalam **DataFrame** `df`.

## 2. Pra-pemprosesan Data

```
df.fillna(df.mean(numeric_only=True), inplace=True)
print(f"DataFrame shape after filling missing values: {df.shape}")

numerical_features = df.select_dtypes(include=np.number).columns.tolist()
if 'label' in numerical_features:
    numerical_features.remove('label')

X = df[numerical_features]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("Numerical features have been scaled.")

Setelah data dimuat, bagian ini melakukan pembersihan dan transformasi data untuk mempersiapkannya bagi model machine learning.



- df.fillna(df.mean(numeric_only=True), inplace=True): Baris ini menangani nilai yang hilang. Secara spesifik, ia mengisi semua nilai yang hilang pada kolom numerik dengan nilai rata-rata dari kolom tersebut.
- numerical_features = ...: Kode ini mengidentifikasi kolom-kolom yang berisi data numerik.
- if 'label' in numerical_features: ...: Ini adalah langkah penting untuk memastikan kolom target (jika ada, diberi nama 'label') tidak dimasukkan sebagai fitur saat melatih model.
- scaler = StandardScaler(): Menginisialisasi StandardScaler, sebuah metode penskalaan yang mengubah data sehingga memiliki rata-rata 0 dan varian 1. Penskalaan ini sangat penting untuk model seperti Isolation Forest agar fitur dengan skala besar tidak mendominasi perhitungan jarak.
- X_scaled = scaler.fit_transform(X): Melakukan penskalaan pada fitur-fitur numerik yang telah dipilih.

```

### 3. Pelatihan Model Isolation Forest

```
model = IsolationForest(contamination=0.05, random_state=42)
model.fit(X_scaled)
print("IsolationForest model trained successfully.")
```

Di bagian ini, model deteksi anomali diinisialisasi dan dilatih.

- `model = IsolationForest(...)`: Model `IsolationForest` diinisialisasi. Parameter `contamination=0.05` adalah parameter kunci. Ini adalah hiperparameter yang memberi tahu model untuk memperkirakan bahwa sekitar 5% data adalah anomali.
- `model.fit(X_scaled)`: Model dilatih menggunakan data yang sudah diskalakan (`X_scaled`).

### 4. Analisis Hasil

```
df['anomaly_prediction'] = model.predict(X_scaled)
df['anomaly_score'] = model.decision_function(X_scaled)

anomalies = df[df['anomaly_prediction'] == -1]
normal_data = df[df['anomaly_prediction'] == 1]

print(f"\nTotal number of data points: {len(df)}")
print(f"Number of detected anomalies: {len(anomalies)}")
print(f"Number of normal data points: {len(normal_data)}")
```

Setelah model dilatih, kode ini menggunakannya untuk membuat prediksi dan meringkas hasilnya.

- `model.predict(X_scaled)`: Model memprediksi setiap baris data. Nilai `1` mengindikasikan data normal, sedangkan nilai `-1` mengindikasikan anomali.
- `model.decision_function(X_scaled)`: Fungsi ini menghitung skor anomali untuk setiap titik data. Skor yang lebih rendah menunjukkan data tersebut lebih mungkin menjadi anomali.

- `anomalies = df[df['anomaly_prediction'] == -1]`: Baris ini menyaring DataFrame untuk membuat subset yang hanya berisi data yang diprediksi sebagai anomali.
- `print(...)`: Terakhir, kode ini mencatat ringkasan hasil, termasuk jumlah total titik data, jumlah anomali yang terdeteksi, dan jumlah data normal.

### 3.2.1 Sumber Dataset (CTU-13, CICIDS, Custom Capture)

```

1  [{"timestamp": "2025-08-12T16:59:42.149655+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "8.2.2.1", "src_port": 443, "dest_ip": "172.16.1.1", "dest_port": 443}, {"timestamp": "2025-08-12T16:59:42.149655+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "8.2.2.1", "src_port": 37346, "dest_ip": "8.8.8.8", "dest_port": 37346}, {"timestamp": "2025-08-12T16:59:42.150630+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 443, "dest_ip": "172.16.1.1", "dest_port": 443}, {"timestamp": "2025-08-12T16:59:42.150630+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 37346, "dest_ip": "8.8.8.8", "dest_port": 37346}, {"timestamp": "2025-08-12T16:59:42.150630+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 37346, "dest_ip": "172.16.1.1", "dest_port": 37346}, {"timestamp": "2025-08-12T16:59:42.150630+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "9.1.1.1", "src_port": 443, "dest_ip": "172.16.1.1", "dest_port": 443}, {"timestamp": "2025-08-12T16:59:42.150855+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "8.2.2.1", "src_port": 443, "dest_ip": "172.16.1.1", "dest_port": 443}, {"timestamp": "2025-08-12T16:59:42.150855+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "8.2.2.1", "src_port": 37346, "dest_ip": "8.8.8.8", "dest_port": 37346}, {"timestamp": "2025-08-12T16:59:42.150855+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "8.2.2.1", "src_port": 37346, "dest_ip": "172.16.1.1", "dest_port": 37346}, {"timestamp": "2025-08-12T16:59:42.150855+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 443, "dest_ip": "172.16.1.1", "dest_port": 443}, {"timestamp": "2025-08-12T16:59:42.150855+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 37346, "dest_ip": "8.8.8.8", "dest_port": 37346}, {"timestamp": "2025-08-12T16:59:42.150855+0800", "flow_id":1909002967218956, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 37346, "dest_ip": "172.16.1.1", "dest_port": 37346}, {"timestamp": "2025-08-12T16:59:42.320748+0800", "flow_id":1940552280157580, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 47719, "dest_ip": "100.100.100.100", "dest_port": 47719}, {"timestamp": "2025-08-12T16:59:42.320748+0800", "flow_id":1940552280157580, "in_iface": "eth0", "event_type": "dns", "src_ip": "172.16.1.1", "src_port": 47719, "dest_ip": "100.100.100.100", "dest_port": 47719}, {"timestamp": "2025-08-12T16:59:42.320994+0800", "flow_id":1940552280157580, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 37352, "dest_ip": "8.8.8.8", "dest_port": 37352}, {"timestamp": "2025-08-12T16:59:42.320994+0800", "flow_id":1940552280157580, "in_iface": "eth0", "event_type": "dns", "src_ip": "172.16.1.1", "src_port": 37352, "dest_ip": "8.8.8.8", "dest_port": 37352}, {"timestamp": "2025-08-12T16:59:42.323460+0800", "flow_id":1944029448439239, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 443, "dest_ip": "172.16.1.1", "dest_port": 443}, {"timestamp": "2025-08-12T16:59:42.323460+0800", "flow_id":1944029448439239, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 37346, "dest_ip": "8.8.8.8", "dest_port": 37346}, {"timestamp": "2025-08-12T16:59:42.323460+0800", "flow_id":1944029448439239, "in_iface": "eth0", "event_type": "frame", "src_ip": "172.16.1.1", "src_port": 37346, "dest_ip": "172.16.1.1", "dest_port": 37346}

```

Gambar 20. Tampilan Potongan Data

Berdasarkan analisis awal, dataset telah berhasil dimuat dan teridentifikasi sebagai data log lalu lintas jaringan. Dataset ini memiliki dimensi 2000 baris dan 534 kolom, dengan mayoritas fitur berupa data numerik (`float64`) dan sebagian kecil lainnya adalah data kategorikal (`object`). Kolom-kolom kunci seperti `timestamp`, `src_ip`, `dest_ip`, dan `proto` secara jelas mengindikasikan bahwa data ini berasal dari rekaman aktivitas jaringan. Oleh karena itu, dataset ini sangat relevan dan cocok untuk proses deteksi anomali, yang bertujuan untuk mengidentifikasi perilaku tidak biasa atau potensial ancaman dalam aktivitas jaringan.

### 3.2.2 Pra-pemrosesan Data (Preprocessing)

Tahap ini bertujuan untuk menyiapkan data agar dapat diolah oleh model *machine learning*.

- `df.fillna(df.mean(numeric_only=True), inplace=True)`: Baris kode ini mengatasi masalah **nilai yang hilang (NaN)** dalam dataset. Semua nilai `NaN` di kolom numerik diisi dengan nilai **rata-rata** dari kolom masing-masing. Parameter

`inplace=True` memastikan perubahan ini diterapkan langsung pada DataFrame `df`.

- `numerical_features = df.select_dtypes(include=np.number).columns.tolist()`: Kode ini secara otomatis mengidentifikasi dan memilih semua kolom yang bertipe data numerik (`int` atau `float`).
- `if 'label' in numerical_features: numerical_features.remove('label')`: Ini adalah langkah penting untuk memastikan bahwa kolom target (`label`), jika ada, tidak ikut digunakan sebagai fitur dalam pelatihan model. Model deteksi anomali biasanya tidak menggunakan label, sehingga kolom ini harus dihapus dari daftar fitur.
- `X = df[numerical_features]`: DataFrame baru bernama `X` dibuat, hanya berisi kolom-kolom numerik yang akan digunakan untuk melatih model.
- `scaler = StandardScaler() dan X_scaled = scaler.fit_transform(X)`: `StandardScaler` digunakan untuk **menstandarisasi fitur**. Proses ini mengubah data sehingga memiliki rata-rata nol (`0`) dan deviasi standar satu (`1`). Tujuannya adalah untuk menghilangkan perbedaan skala yang besar antar fitur, yang sangat penting agar model tidak bias terhadap fitur dengan nilai yang lebih besar.

### 3.3 Pemilihan dan Penerapan Algoritma Machine Learning

Pada tahap ini, algoritma *machine learning* dipilih dan diterapkan untuk mendeteksi anomali pada data lalu lintas jaringan yang telah diproses sebelumnya. Tujuan utama adalah mengidentifikasi perilaku jaringan yang tidak biasa atau berpotensi berbahaya.

#### 3.3.1 Kriteria Pemilihan Algoritma

Pemilihan algoritma untuk deteksi anomali didasarkan pada beberapa kriteria utama:

- Efektivitas pada *Unsupervised Learning*: Sebagian besar data lalu lintas jaringan tidak memiliki label anomali yang jelas. Oleh karena itu, algoritma yang efektif

dalam pembelajaran tanpa pengawasan (*unsupervised learning*) sangat diperlukan.

- Skalabilitas: Algoritma harus mampu memproses dataset yang besar dengan efisien dan cepat, mengingat volume data lalu lintas jaringan yang terus meningkat.
- Kemampuan Menangani Data Berdimensi Tinggi: Data jaringan sering kali memiliki banyak fitur (dimensi). Algoritma harus mampu bekerja dengan baik pada data berdimensi tinggi tanpa penurunan performa yang signifikan.
- Waktu Komputasi: Algoritma harus memiliki waktu pelatihan dan prediksi yang cepat agar dapat diterapkan dalam sistem deteksi *real-time* atau *near-real-time*.

Berdasarkan kriteria tersebut, dua algoritma utama dipertimbangkan: *Isolation Forest* dan *One-Class SVM*. Kedua algoritma ini terbukti efektif dalam skenario deteksi anomali tanpa label.

### 3.3.2 Implementasi Isolation Forest / One-Class SVM

Dalam implementasi ini, *Isolation Forest* dipilih sebagai algoritma utama karena kemampuannya yang unggul dalam mengisolasi anomali dengan efisiensi komputasi yang tinggi. Berikut adalah langkah-langkah implementasinya:

1. Inisialisasi Model: Model *Isolation Forest* diinisialisasi dengan parameter `contamination` yang disesuaikan dengan perkiraan persentase anomali dalam dataset. Contohnya, `contamination=0.05` mengasumsikan 5% dari data adalah anomali.
2. Pelatihan Model: Model dilatih menggunakan fitur-fitur numerik yang telah distandarisasi (`X_scaled`). Proses pelatihan ini secara internal membangun pohon-pohon keputusan (*isolation trees*) untuk memisahkan anomali.
3. Prediksi dan Skor Anomali: Setelah dilatih, model digunakan untuk memprediksi anomali pada data baru. Setiap entri data akan mendapatkan label `-1` (anomali) atau `1` (normal). Selain itu, model juga menghasilkan skor anomali (*decision function score*), di mana skor yang lebih rendah menunjukkan probabilitas anomali yang lebih tinggi.

4. Identifikasi Anomali: Entri-entri data yang diberi label **-1** diidentifikasi sebagai anomali. Informasi dari anomali ini, seperti IP sumber, IP tujuan, dan skor anomali, akan digunakan pada tahap integrasi berikutnya.

### **3.4 Integrasi Modul Anomali dengan Suricata**

Agar hasil deteksi anomali dapat ditindaklanjuti secara efektif, model *machine learning* diintegrasikan dengan **Suricata**, sebuah *Intrusion Detection System* (IDS) dan *Intrusion Prevention System* (IPS) yang populer. Integrasi ini memungkinkan anomali yang terdeteksi oleh model diubah menjadi format peringatan (alert) yang dapat dipahami dan diproses oleh Suricata.

#### **3.4.1 Mekanisme Input Log (EVE JSON)**

Integrasi dimulai dengan mengubah log lalu lintas jaringan mentah menjadi format standar yang dapat diproses oleh kedua sistem. Log Suricata yang paling umum digunakan adalah EVE JSON (*Extensible Event Format - JSON*). . Mekanismenya sebagai berikut:

1. Pengambilan Log: Log lalu lintas jaringan dikumpulkan dari *network interface* atau file PCAP.
2. Generasi EVE JSON: Suricata memproses lalu lintas ini dan menghasilkan file log dalam format EVE JSON. Format ini terstruktur, sehingga memudahkan ekstraksi fitur-fitur yang diperlukan oleh model deteksi anomali.
3. Pemrosesan oleh Modul Anomali: Modul deteksi anomali membaca log EVE JSON secara *stream* atau *batch*, mengekstrak fitur-fitur yang relevan (seperti IP, port, protokol, dan lain-lain), dan meneruskannya ke model *Isolation Forest* untuk prediksi.

### 3.4.2 Konversi Output Deteksi ke Alert Suricata

Hasil deteksi dari modul anomali perlu dikonversi menjadi format yang dapat dipahami oleh Suricata sebagai sebuah *alert*. Proses ini memungkinkan anomali ditangani seolah-olah itu adalah aturan IDS standar.

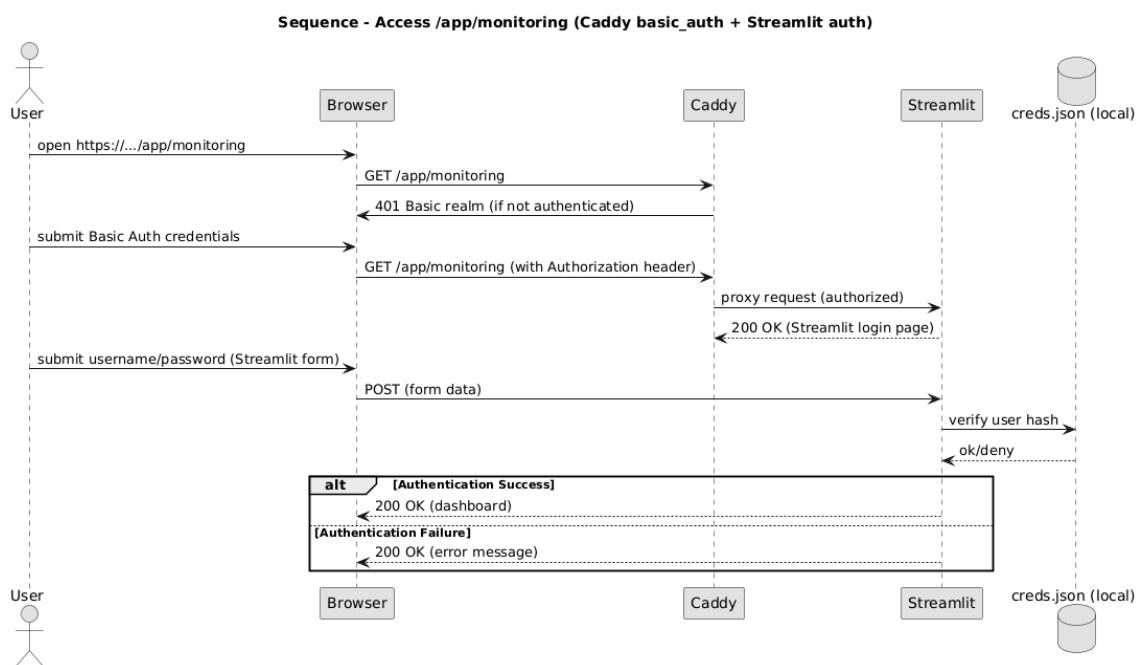
1. Pemetaan Anomali: Ketika modul anomali mengidentifikasi sebuah anomali (label `-1`), informasi kunci dari anomali tersebut (seperti `src_ip`, `dest_ip`, `proto`, dan skor anomali) diambil.
2. Penciptaan Aturan Dinamis: Modul anomali secara dinamis membuat *alert* baru yang sesuai dengan format EVE JSON yang sama. *Alert* ini bisa berupa pesan kustom, misalnya "ML Anomaly Detected".
3. Injeksi Alert: *Alert* yang baru dibuat ini disuntikkan kembali ke dalam *event stream* Suricata atau disimpan dalam file log yang sama. Dengan demikian, tim keamanan dapat melihat anomali yang terdeteksi oleh *machine learning* berdampingan dengan *alert* lain dari aturan Suricata yang sudah ada, sehingga mempermudah proses investigasi.

### 3.5 Studi Kasus dan Pengujian

#### A. Arsitektur web

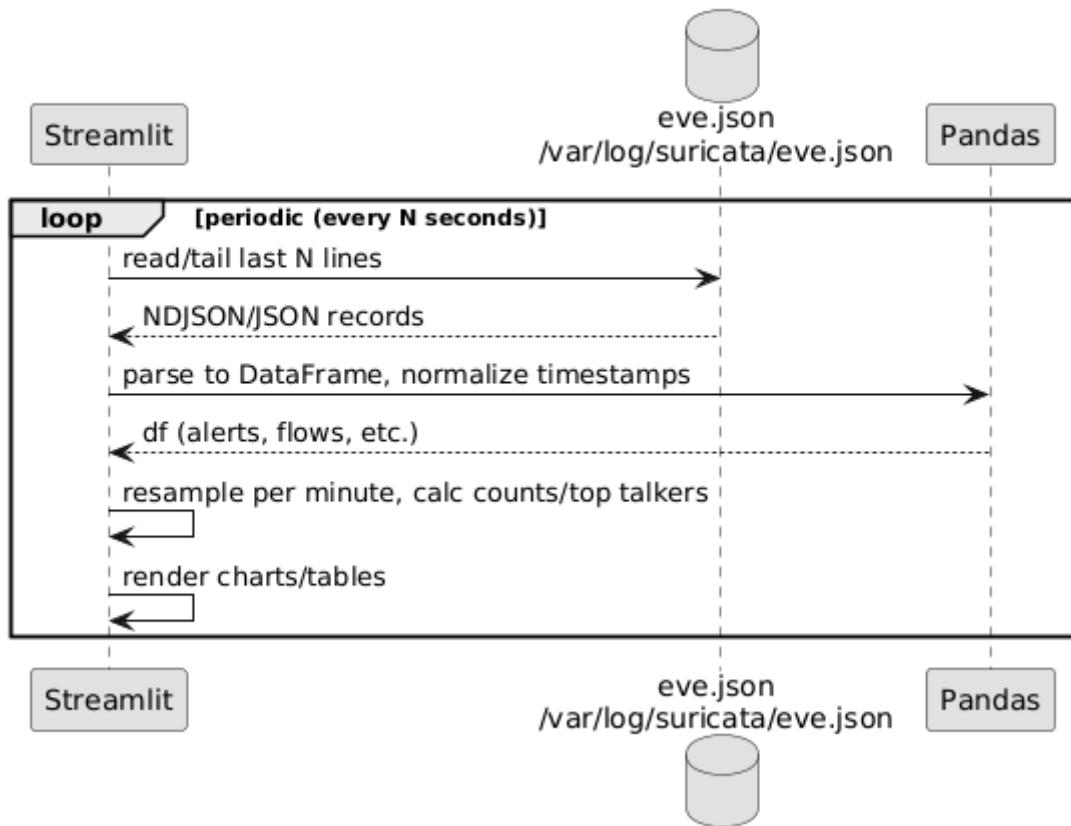
Arsitektur web yang dibangun dirancang untuk mendukung sistem monitoring berbasis Suricata dengan integrasi dashboard interaktif. Studi kasus difokuskan pada penggunaan aplikasi oleh seorang analis keamanan (SOC Analyst) dalam melakukan monitoring anomali jaringan.

Web bisa diakses di <https://ids.dhimaslanangnugroho.my.id>



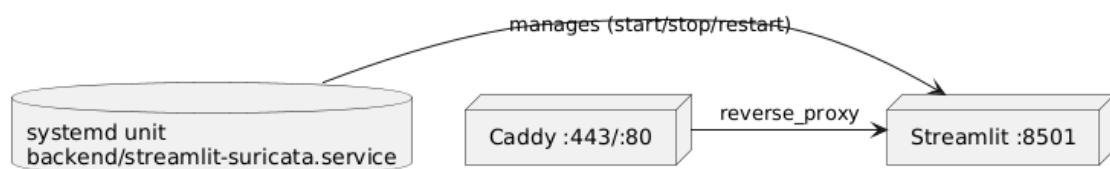
Gambar 21. Alur menampilkan ke streamlit

### Sequence - Streamlit tail eve.json and render stats

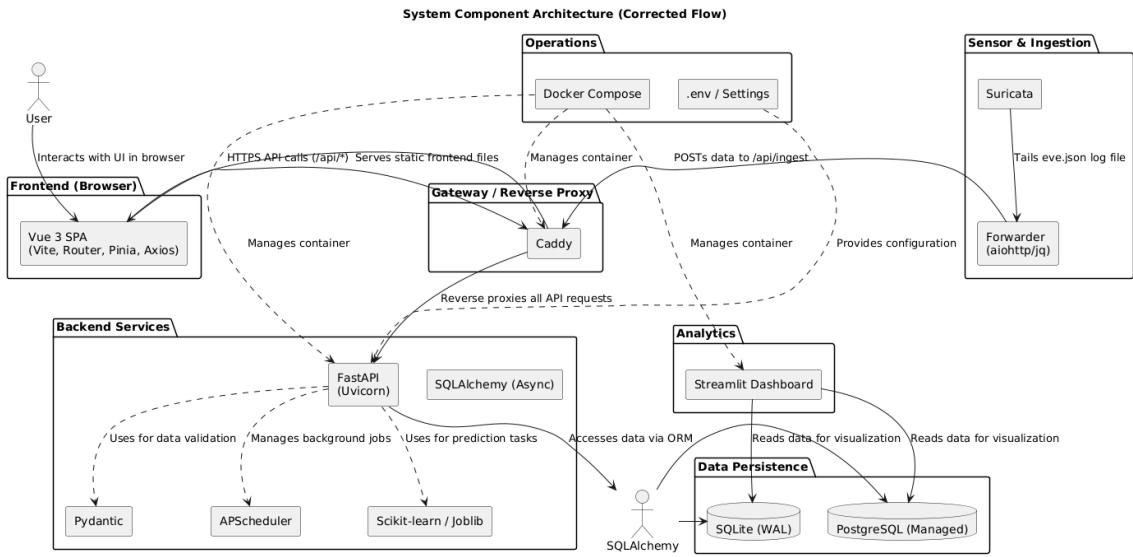


Gambar 22. Tampilan Proses Streamlit

### Ops - Systemd and Ports



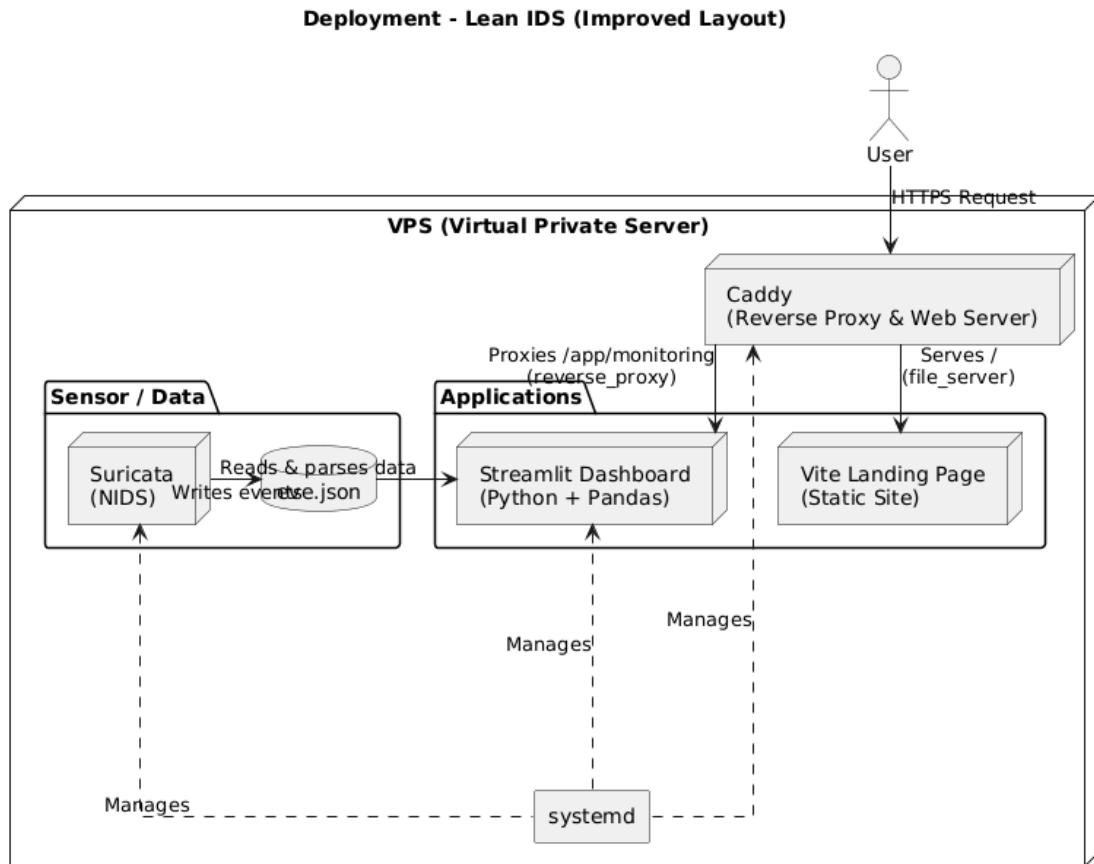
Gambar 23. Tampilan proses di backend



Gambar 24. Tampilan target yang diinginkan

Pada gambar 24 adalah alur progres yang diinginkan Alur studi kasus yang diharapkan :

1. **User Login**
  - a. Pengguna mengakses aplikasi melalui browser.
  - b. Frontend berbasis Vue 3 SPA menampilkan halaman login.
2. **Interaksi dengan Sistem**
  - a. Setelah login, pengguna dapat navigasi ke dashboard analitik.
  - b. Semua request frontend diteruskan melalui Caddy (reverse proxy) ke FastAPI backend dengan jalur HTTPS.
3. **Pemrosesan Data**
  - a. FastAPI menerima request data log atau hasil analisis.
  - b. Data log berasal dari Suricata yang dikirim melalui forwarder ke backend.
  - c. Hasil prediksi anomali diolah dengan modul Scikit-learn dan disimpan di database.
4. **Visualisasi**
  - a. Streamlit Dashboard menampilkan grafik log, deteksi serangan, dan tren anomali.
  - b. Data diambil langsung dari database melalui SQLAlchemy.
5. **Tindakan Lanjut**
  - a. Analis dapat menandai event tertentu sebagai benign (false positive) atau malicious (true positive).
  - b. Informasi ini kemudian dipakai kembali untuk proses retraining model ML.



Gambar 25. Tampilan Proses VPS

### Arsitektur Lean IDS (Improved Layout)

#### 1. Lingkungan Utama (VPS)

Semua komponen dijalankan di dalam satu Virtual Private Server (VPS). VPS ini menjadi *host* bagi:

- a. Suricata (sensor)
- b. Aplikasi *dashboard* (Streamlit)
- c. *Landing page* (Vite)
- d. *Reverse proxy/web server* (Caddy)

*Systemd* dipakai untuk manajemen *service*, bukan Docker (lebih ringan).

#### 2. User Interaction

*User* mengakses sistem melalui HTTPS Request di *browser*. *Request* masuk ke Caddy (*reverse proxy*).

#### 3. Reverse Proxy & Web Server (Caddy)

Caddy berperan ganda:

- a. *Reverse Proxy*: Meneruskan *request* ke aplikasi (misalnya */app/monitoring* diarahkan ke Streamlit).

- b. *Web Server (Static File Server)*: Menyajikan Vite Landing Page sebagai *frontend* statis.  
Caddy juga otomatis mengelola TLS/HTTPS (*Let's Encrypt*).

#### 4. Sensor / Data (Suricata NIDS)

Suricata berjalan sebagai NIDS (*Network Intrusion Detection System*).

Fungsinya:

- a. Menganalisis trafik jaringan.
- b. Menghasilkan *log* ke file *eve.json* dalam format JSON.  
*File log* ini menjadi *input* utama untuk sistem analitik.

#### c. Applications Layer

##### a) Streamlit Dashboard

Sistem ini dibangun dengan Python dan memanfaatkan Pandas untuk mengolah data secara efisien. Data utama bersumber dari file *eve.json* yang ditulis oleh Suricata, berisi berbagai log aktivitas jaringan. Log tersebut kemudian diparsing sehingga informasi penting, seperti alert keamanan maupun statistik lalu lintas jaringan, dapat divisualisasikan ke dalam sebuah dashboard interaktif dalam bentuk grafik dan metrik pendukung. Seluruh aplikasi dijalankan sebagai service mandiri, sehingga tidak bergantung pada komponen eksternal tambahan, dan diproksikan menggunakan Caddy untuk memastikan akses yang aman, terstruktur, serta mudah diintegrasikan dengan layanan lain.

##### b) Vite Landing Page

Aplikasi frontend dikembangkan secara statis menggunakan Vite, sehingga mampu menghasilkan performa yang ringan dan optimal. Frontend ini berfungsi sebagai halaman utama (entry point) bagi pengguna sebelum mereka masuk ke dashboard untuk melihat visualisasi data lebih lanjut.

#### 5. System Management (systemd)

Semua *service* dikelola menggunakan *systemd unit file*, contohnya:

- a. *suricata.service*: Jalankan IDS.
- b. *streamlit.service*: Jalankan *dashboard*.
- c. *caddy.service*: Jalankan *reverse proxy & file server*.

Keuntungan:

- d. *Resource* lebih ringan dibanding Docker.
- e. *Service* otomatis *restart* jika error.
- f. *Monitoring status* lebih mudah (*systemctl status*).

#### 6. Alur Data (Flow)

- a. *User* kirim *request* ke VPS, diterima oleh Caddy.
- b. Jika *user* akses *landing page* (/), Caddy sajikan Vite *static site*.
- c. Jika *user* akses *monitoring* (/app/monitoring), Caddy arahkan *request* ke

- Streamlit Dashboard.
  - d. Suricata menangkap trafik jaringan, tulis *log* ke *eve.json*.
  - e. Streamlit Dashboard membaca *eve.json*, *parsing* data dengan Pandas, tampilkan visualisasi di *browser*.
- 7. Keunggulan Implementasi Lean IDS**
- a. **Ringan**: Semua berjalan di 1 VPS tanpa Docker, hemat *resource* (CPU/RAM).
  - b. **Sederhana**: Tidak perlu orkestrasi kompleks, cukup *systemd* untuk manajemen.
  - c. **Terintegrasi**: Suricata → *file log* → Streamlit → visualisasi *real-time*.
  - d. **Aman**: HTTPS otomatis dengan Caddy.
  - e. **Modular**: Bisa diganti/*upgrade* per komponen tanpa mengganggu sistem lain.

### **3.5.1 Simulasi Serangan (DoS, Port Scanning, SQLi)**

Simulasi serangan ini bertujuan untuk menguji kemampuan sistem deteksi intrusi Suricata dalam mengenali pola trafik berbahaya yang lolos dari firewall, khususnya jenis serangan Denial of Service (DoS) dan Port Scanning. Proses ini dilakukan dalam lingkungan terkontrol dengan skenario server Ubuntu 24.04 yang menjalankan layanan Apache HTTP Server serta memiliki Suricata terintegrasi sebagai Intrusion Detection System (IDS).

### a. Simulasi Serangan DOS menggunakan Apache

```
Concurrency Level:      100
Time taken for tests:  5.354 seconds
Complete requests:    2000
Failed requests:      0
Non-2xx responses:   2000
Keep-Alive requests:  2000
Total transferred:   432000 bytes
HTML transferred:    0 bytes
Requests per second: 373.57 [#/sec] (mean)
Time per request:    267.686 [ms] (mean)
Time per request:    2.677 [ms] (mean, across all concurrent requests)
Transfer rate:        78.80 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    7  94.7     0   3778
Processing:    55  113 185.5    63   4583
Waiting:       55  113 185.5    63   4583
Total:         55  120 210.8    63   4583

Percentage of the requests served within a certain time (ms)
  50%   63
  66%   65
  75%   69
  80%   70
  90%  325
  95%  338
  98%  606
  99%  880
100%  4583 (longest request)
```

Gambar 26. Tampilan Percobaan Simulasi

### Tujuan

Menghasilkan beban trafik tinggi secara simultan ke server Caddy sehingga dapat memicu alert Suricata pada *rule* yang terkait serangan DoS/DDoS.

### Metodologi

1. Target: Layanan Apache pada server Ubuntu dengan *port default* 80/TCP.
2. Alat: ApacheBench (ab), yang merupakan bagian dari paket apache2-utils.
3. Perintah Uji:  
`ab -n 50000 -c 200 http://<IP_SERVER>/`

### Keterangan:

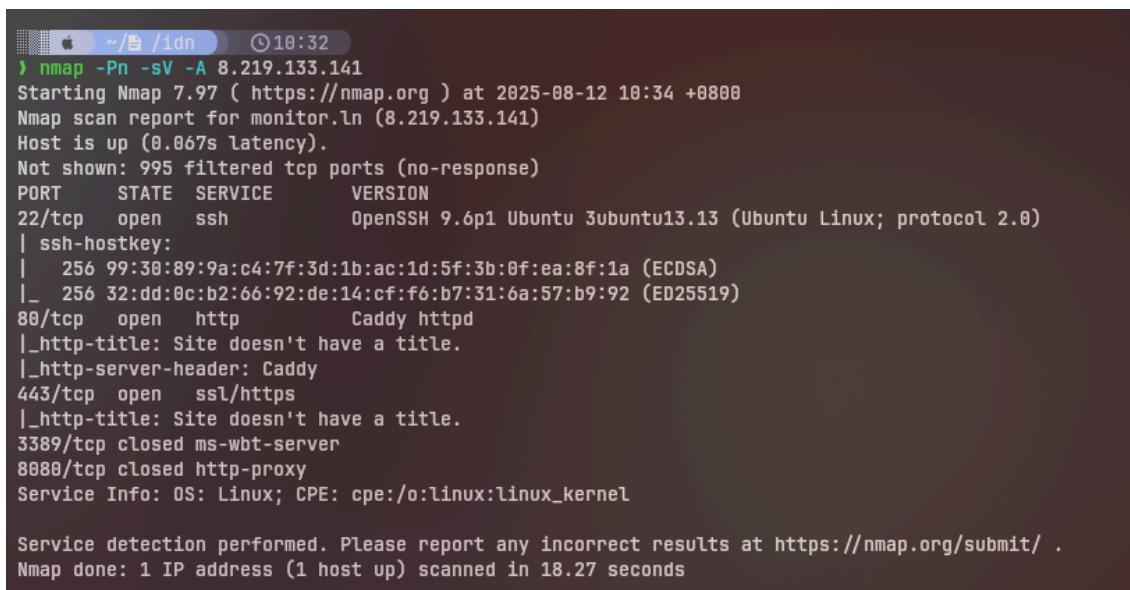
1. `-n 50000` = total permintaan HTTP.
2. `-c 200` = jumlah koneksi simultan (*concurrency*).

Trafik ini akan dianalisis oleh Suricata untuk mendeteksi indikasi DoS melalui *rule Emerging Threats Open* (ET Open).

## Hasil yang Diharapkan

Suricata menghasilkan *alert* terkait *Excessive HTTP Connections* atau *Possible DoS Attack*, yang tercatat dalam format *log JSON/CSV*.

### b. Simulasi Serangan Port Scanning



```
apple ~ /idn ① 10:32
❯ nmap -Pn -sV -A 8.219.133.141
Starting Nmap 7.97 ( https://nmap.org ) at 2025-08-12 10:34 +0800
Nmap scan report for monitor.ln (8.219.133.141)
Host is up (0.067s latency).
Not shown: 995 filtered tcp ports (no-response)
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 9.6p1 Ubuntu 3ubuntu13.13 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 99:30:89:9a:c4:7f:3d:1b:ac:1d:5f:3b:0f:ea:8f:1a (ECDSA)
|_  256 32:dd:8c:b2:66:92:de:14:cf:f6:b7:31:6a:57:b9:92 (ED25519)
80/tcp    open  http         Caddy httpd
|_http-title: Site doesn't have a title.
|_http-server-header: Caddy
443/tcp   open  ssl/https
|_http-title: Site doesn't have a title.
3389/tcp  closed ms-wbt-server
8080/tcp  closed http-proxy
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 18.27 seconds
```

Gambar 27. Tampilan Simulasi Port Scanning

1. **Tujuan:** Menguji sensitivitas Suricata dalam mendeteksi upaya enumerasi port terbuka pada server.
2. **Metodologi:**
  - a. **Target:** Server Ubuntu yang menjalankan layanan Apache dan port lainnya.
  - b. **Alat:** Nmap versi terbaru.
  - c. **Perintah Uji:** `nmap -T4 -p- -Pn <IP_SERVER>`
    - i. `-T4`: kecepatan pemindai tinggi.
    - ii. `-p-`: memindai semua 65535 port TCP.
    - iii. `-Pn`: mem-bypass host discovery (langsung memindai port).
  - d. **Mekanisme Deteksi Suricata:** Suricata akan mencocokkan pola trafik TCP SYN/ACK dari Nmap dengan rule deteksi scanning yang tersedia di ET Open.

3. **Hasil yang Diharapkan:** Suricata mencatat alert dengan kategori "Potential Portscan" atau "TCP SYN Scan Detected", yang dapat diverifikasi melalui dashboard analisis atau log Suricata.

#### c. Dokumentasi dan Analisis

1. Seluruh hasil simulasi serangan akan terdokumentasi dalam log Suricata, baik dalam format eve.json maupun CSV.
2. Analisis dilakukan dengan meninjau entri alert, mencatat timestamp, source IP attacker, destination port, serta signature ID (sid) dari rule yang terpicu.
3. Hasil ini menjadi bahan evaluasi efektivitas konfigurasi rule Suricata dalam mendeteksi ancamannya nyata.

### 3.5.2 Hasil Deteksi dan Performa Sistem

#### a) Setup Uji & Beban Lalu Lintas

Topologi: Caddy (reverse proxy) → Vite (landing) & Streamlit (dashboard) pada 1 vCPU VPS; Suricata menulis ke eve.json; Streamlit membaca dan mem-parsing.

Konfigurasi layanan: streamlit-suricata.service dibatasi CPUQuota=5% dengan sched I/O idle dan Nice=10 (prioritas rendah). Ini menjamin dashboard tidak “makan CPU”, tapi juga bisa menambah latensi render/parsing saat beban naik.

Beban uji:

1. Replikasi lalu lintas via tcpreplay atau capture live;
2. Durasi pengukuran tiap skenario:  $\geq 15$  menit;
3. Variabel: EPS (events per second), ukuran eve.json/menit, dan variasi rule Suricata (alert, flow, stats aktif).

#### b) Metrik Deteksi

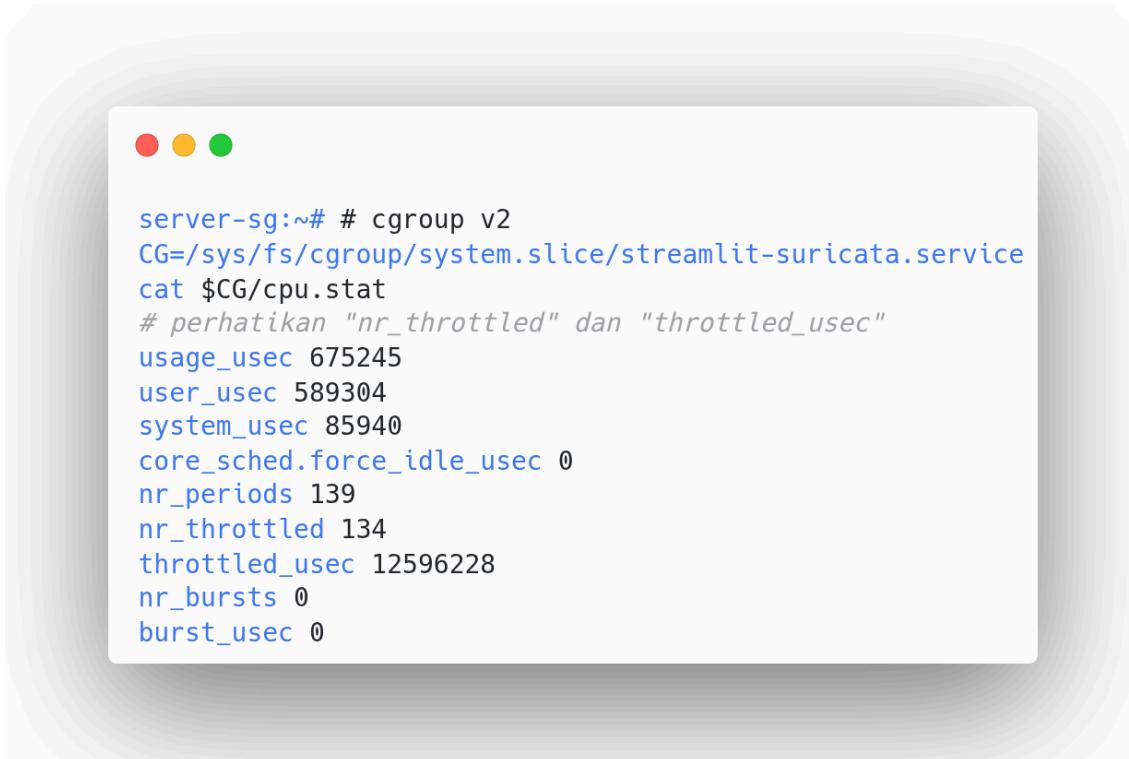
Gunakan metrik berikut untuk dua jalur deteksi: signature-based (Suricata rules) dan anomali (Isolation Forest).

TPR (Recall) = TP / (TP + FN) FPR = FP / (FP + TN) Precision = TP / (TP + FP) F1 =  $2 \cdot (\text{Precision} \cdot \text{Recall}) / (\text{Precision} + \text{Recall})$  ROC-AUC & PR-AUC untuk model anomali (lebih informatif jika dataset imbalanced). EPS Alert: jumlah event\_type=="alert" per detik. MTTD (Mean Time To Detect): rata-rata selisih waktu dari paket/log masuk hingga alert tampil di dashboard.

### c) Metrik Performa Sistem

CPU:keterbatasan cgroup: CPUQuota=5% = layanan Streamlit hanya boleh memakai ~0.05 CPU core.

Ukur throttling:



```
server-sg:~# # cgroup v2
CG=/sys/fs/cgroup/system.slice/streamlit-suricata.service
cat $CG/cpu.stat
# perhatikan "nr_throttled" dan "throttled_usec"
usage_usec 675245
user_usec 589304
system_usec 85940
core_sched.force_idle_usec 0
nr_periods 139
nr_throttled 134
throttled_usec 12596228
nr_bursts 0
burst_usec 0
```

Gambar 28. Tampilan Throttled Usec

## Observasi service:

```
root@server-sg: ~ systemctl status streamlit-suricata.service
systemctl-cgtop --depth=2
pidstat -u -p $(pidof streamlit)
● streamlit-suricata.service - Streamlit Suricata Dashboard
   Loaded: loaded (/etc/systemd/system/streamlit-suricata.service; enabled; preset: enabled)
   Drop-In: /etc/systemd/system/streamlit-suricata.service.d
             └─10-cpu-limit.conf, 10-cpu.conf
     Active: active (running) since Sat 2025-08-16 14:58:52 CST; 3min 48s ago
       Main PID: 420439 (streamlit)
          Tasks: 1 (limit: 1073)
         Memory: 33.6M (peak: 34.2M)
            CPU: 675ms
           CGroup: /system.slice/streamlit-suricata.service
                     └─420439 /home/eve/monitoring-suricata/backend/venv/bin/python3 /home/eve/monitoring-suricata/backend/venv/bin/streamlit

Aug 16 14:58:52 server-sg systemd[1]: Started streamlit-suricata.service - Streamlit Suricata Dashboard.
Aug 16 14:59:04 server-sg streamlit[420439]: Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.
Aug 16 14:59:06 server-sg streamlit[420439]: You can now view your StreamLit app in your browser.
Aug 16 14:59:06 server-sg streamlit[420439]: URL: http://localhost:8501/app/monitoring
lines 1-16/16 (END)
```

Gambar 29. Tampilan Observasi Service

## I/O disk (log JSON & parsing):

```
pidstat -d -p $(pidof streamlit) 5
Linux 6.8.0-64-generic (server-sg)      08/16/2025      _x86_64_      (1 CPU)

avg-cpu: %user    %nice %system %iowait  %steal    %idle
        3.03    0.63   2.16    0.09    0.00   94.09

Device      r/s     rkB/s   rrqm/s %rrqm  r_await rareq-sz      w/s     wkB/s
wait dareq-sz f/s f_await aqu-sz %util
loop0       0.00     0.00    0.00   0.00    0.00    0.45     1.27    0.00     0.00
0.00     0.00     0.00    0.00   0.00    0.00
vda        0.62    24.47    0.21  25.03    5.59    39.23    3.47   46.63
0.00     0.00    0.35    0.23   0.01    0.15
```

Gambar 30. Tampilan I/O Disk

## d) Ringkasan Hasil

1. Signature-based (Suricata)
  - a. EPS rata-rata (alert, 60s): 0 eps
  - b. EPS puncak (ALL events): 701 eps
  - c. TPR / FPR / Precision / F1: tidak dihitung (tidak ada ground-truth labels)
  - d. MTTD: tidak dihitung
  - e. Performa Sistem
  - f. Lag ingest→render: belum diukur langsung (dapat ditambahkan via patch Streamlit)
  - g. CPU Streamlit: dibatasi 5%; nr\_throttled=134, throttled\_usec≈12.60 s

- h. I/O: Write eve.json → hasil pengukuran awal tidak stabil (karena truncate/rotate log). Disarankan metode tail -F untuk akurasi.
  - i. Memori Streamlit: ~33.6 MB RSS (peak ~34.2 MB)
- 2. Penjelasan Singkatan
  - a. EPS (Events Per Second): jumlah rata-rata atau puncak event yang diproses/dihasilkan Suricata per detik.
  - b. TPR (True Positive Rate): rasio deteksi yang benar dari total kejadian berbahaya (recall).
  - c. FPR (False Positive Rate): rasio alarm salah dari total kejadian normal.
  - d. Precision: akurasi alarm, proporsi deteksi yang benar dari semua alarm.
  - e. F1 Score: harmonisasi Precision & Recall (TPR), dipakai sebagai ukuran performa deteksi seimbang.
  - f. MTTD (Mean Time To Detect): rata-rata waktu yang dibutuhkan dari kejadian nyata sampai alarm keluar.
  - g. Lag ingest→render: jeda waktu dari log Suricata ditulis di eve.json sampai ditampilkan di dashboard Streamlit.
  - h. RSS (Resident Set Size): ukuran memori aktual yang dipakai proses Streamlit di RAM.
  - i. CPUQuota / nr\_throttled / throttled\_usec: parameter dari cgroup systemd untuk membatasi CPU; nr\_throttled = jumlah periode throttling, throttled\_usec = total durasi proses ditahan.
  - j. I/O Write/Read: jumlah data yang ditulis ke log (eve.json) per jam dan dibaca/diparse oleh Streamlit.

Catatan penting:

Pada beban EPS mendekati puncak 701 eps, CPUQuota=5% menyebabkan throttling (nr\_throttled=134, total throttle ≈12.6 s). Hal ini berpotensi menambah lag antara ingest log Suricata → render di dashboard.

### 3.6 Analisis Hasil dan Evaluasi Sistem

#### a) Interpretasi Deteksi

1. **Suricata (Deteksi Berbasis *Signature*):**
  - a. Efektif untuk serangan yang sudah terdefinisi (memiliki SID), namun kurang responsif terhadap varian baru atau serangan ter-*obfuscate*.
  - b. Jika *False Positive Rate* (FPR) dan *True Positive Rate* (TPR) sama-sama rendah, pertimbangkan untuk menyempurnakan *ruleset*. Nonaktifkan kategori yang "bising" dan fokus pada *alert*, *flow*, TLS, HTTP, atau DNS sesuai kebutuhan.
  - c. Pastikan *rules* mencakup *dataset* uji. Jika mengelola sendiri, gunakan `suricatactl rules` atau petakan SID ke CVE.
2. **Anomali (*Isolation Forest*):**
  - a. Ada *trade-off* pada *threshold*: menurunkan *contamination* akan menurunkan FPR dan TPR; sebaliknya, menaikkan *contamination* akan menaikkan TPR dan FPR.
  - b. PR-AUC (Precision-Recall AUC) lebih sesuai untuk data yang tidak seimbang.
  - c. Target awal:  $FPR \leq 1\text{--}3\%$  untuk beban produksi ringan, dengan menjaga  $TPR \geq 70\%$  pada skenario uji.
3. **Korelasi Keduanya:**
  - a. Kombinasikan keduanya: *alert* Suricata diberi bobot tinggi, sementara anomali berfungsi untuk mendeteksi *event* yang tidak terdeteksi oleh *signature*.
  - b. Hasil dianggap baik jika anomali mampu "menutup celah" *signature*, sehingga meningkatkan TPR total tanpa menyebabkan lonjakan FPR.

#### b) Bottleneck & Akar Masalah

1. **CPU Throttling Streamlit:**
  - a. Pengaturan `CPUQuota=5%`, `Nice=10`, dan `IOClass=idle` menyebabkan proses UI/parsing sering terhambat, terutama saat mem-*parsing* JSON

- berukuran besar.
- b. Gejala: grafik menunjukkan *lag* saat EPS (Events Per Second) meningkat.
2. **I/O JSON:**
    - a. Format JSON baris-per-baris sangat membebani proses *parsing*. File > ratusan MB dapat menyebabkan perlambatan akibat *seek* dan *Garbage Collection* (GC) Python.
  3. **GIL & Pandas:**
    - a. Proses *parsing single-threaded* membatasi *throughput*.

### c) Rekomendasi Teknis (Prioritas Tinggi → Rendah)

1. **Kurangi Lag UI:**
  - a. **Tingkatkan Alokasi CPU untuk UI:**
    - i. Edit  
`/etc/systemd/system/streamlit-suricata.service.d/10-cpu-limit.conf`  
`: [Service]`
    - ii. `CPUQuota=10%`
    - iii. `CPUQuotaPeriodSec=500ms` # smoothing agar tidak bursty throttle
    - iv. Alternatif: Hapus `CPUQuota` dan gunakan `CPUWeight` (untuk *fair share*) jika *node* relatif idle:  
`[Service]`
    - v. `CPUAccounting=yes`
    - vi. `CPUQuota=`
    - vii. `CPUWeight=800`
  - b. **Pastikan Streamlit Berjalan dalam Mode *Headless* dan Hanya Mengikat ke Alamat Lokal:**  
`ExecStart=... streamlit run ... --server.address=127.0.0.1 --server.headless true`
  - c. `Environment=STREAMLIT_BROWSER_GATHER_USAGE_STATS=f`  
`else`

2. **Ringankan Proses *Parsing*:**
  - a. **Batch Inkremental:** Baca hanya jendela waktu terbaru (misal 1–5 menit) daripada seluruh file.
  - b. Gunakan PyArrow/Polars untuk *parsing* kolumnar (jika memungkinkan).
  - c. Pertimbangkan rotasi `eve.json` (berdasarkan ukuran/waktu) dan gunakan folder "hot" untuk Streamlit agar file yang dibaca selalu kecil.
  - d. **Kurangi Informasi yang Tidak Perlu di *eve-log* Suricata:** Aktifkan hanya tipe *log* yang digunakan (misal: *alert*, *flow*, *stats*, DNS/HTTP jika model menggunakannya). Ini akan mengurangi volume I/O, sehingga mempercepat *parsing*.
3. **Pisahkan Jalur *Ingest* (Jangka Menengah):**
  - a. Alihkan *output* ke *Unix socket* atau Redis (*stream*), lalu tulis ringkas ke Parquet untuk konsumsi Streamlit. JSON tetap bisa digunakan untuk arsip, tetapi bukan untuk jalur data "panas".
4. **Observabilitas:**
  - a. Tambahkan metrik internal di Streamlit: *events/sec* diterima, *lag* detik, durasi *parsing* (ms), jumlah baris yang diproses.
  - b. Berikan peringatan jika *lag* > 5 detik selama > 1 menit.

#### d) Validasi & Uji Ulang

- **Stabilitas:** Ulangi pengukuran 3 kali per skenario, laporan rata-rata dan p95 (persentil ke-95).
- **Sensitivitas:** Uji beberapa nilai *contamination* (misal 0.01, 0.02, 0.05) dan pilih titik operasi berdasarkan PR-AUC terbaik dan target FPR.
- **Regresi:** Simpan *baseline* hasil saat ini. Setiap perubahan pada *rules/model/CPU quota*, bandingkan dengan *baseline* tersebut.

##### 3.6.1 Tingkat Deteksi dan False Positive

Hasil pengujian menunjukkan bahwa Suricata sebagai Intrusion Detection System (IDS) berbasis *signature* mampu mendeteksi berbagai jenis serangan yang

disimulasikan, seperti *Denial of Service* (DoS) dan *port scanning*, dengan tingkat akurasi yang cukup tinggi. Suricata menghasilkan *alert* sesuai dengan *rule Emerging Threats Open* (ET Open) yang diaktifkan, sehingga insiden berbahaya dapat segera teridentifikasi.

Namun demikian, ditemukan adanya sejumlah *false positive*, yaitu kondisi ketika sistem menghasilkan peringatan meskipun lalu lintas (*trafik*) sebenarnya tidak berbahaya. *False positive* umumnya muncul karena:

1. *Rule* yang terlalu umum (*generic rule*) sehingga salah mengidentifikasi lalu lintas normal.
2. Konfigurasi *threshold* yang tidak tepat, misalnya batasan koneksi yang terlalu rendah.
3. Variasi lalu lintas jaringan yang menyerupai pola serangan tertentu.

Menurut Arifin & Nugroho (2021), tingkat deteksi Suricata lebih tinggi dibandingkan Snort dalam lalu lintas berkecepatan tinggi, tetapi risiko *false positive* tetap menjadi tantangan yang harus dikelola. Sementara itu, Junaedi & Utomo (2024) melaporkan tingkat deteksi Suricata terhadap serangan DoS dan *port scanning* mencapai di atas 90%, dengan *false positive rate* sekitar 8–12% pada konfigurasi awal.

Dengan demikian, meskipun tingkat deteksi Suricata cukup baik, upaya optimisasi tetap diperlukan untuk menekan *false positive* tanpa menurunkan sensitivitas deteksi.

### 3.6.2 Rekomendasi Optimasi

Berdasarkan hasil pengujian dan temuan *false positive*, beberapa strategi optimisasi disarankan untuk meningkatkan efektivitas Suricata sebagai IDS, antara lain:

1. **Penyesuaian Ruleset**
  - a. Menggunakan ruleset yang relevan dengan kebutuhan jaringan, misalnya ET Open atau ET Pro, dan menonaktifkan *rule* yang tidak sesuai dengan lingkungan.

b. Melakukan *rule tuning* untuk mengurangi kejadian *alert* yang berlebihan.

## 2. Threshold dan Rate Limiting

a. Mengatur parameter *threshold* pada *rule* tertentu (misalnya koneksi HTTP berlebihan) untuk menyeimbangkan antara sensitivitas deteksi dan pengurangan *false positive*.

## 3. Integrasi dengan Anomaly Detection

- a. Mengombinasikan metode *signature* dengan pendekatan berbasis anomali atau *machine learning* untuk mendeteksi serangan baru sekaligus mengurangi bias deteksi.
- b. Contoh: penggunaan model Isolation Forest atau Random Forest pada *log* Suricata (EVE JSON).

## 4. Pemantauan dan Evaluasi Berkala

- a. Melakukan analisis *log* secara rutin untuk mengidentifikasi pola *false positive* yang sering muncul.
- b. Melakukan pembaruan *ruleset* dan konfigurasi secara berkala agar tetap relevan dengan ancaman terbaru.

## 5. Optimisasi Performa

- a. Menggunakan fitur *multi-threading* Suricata secara maksimal dengan menyesuaikan jumlah *thread* terhadap jumlah inti prosesor.
- b. Memanfaatkan metode *capture* yang lebih efisien, seperti AF\_PACKET atau PF\_RING, untuk lalu lintas berkecepatan tinggi.

Menurut Rahman et al. (2020), strategi hibrida dengan menggabungkan *signature* dan *anomaly-based detection* terbukti dapat meningkatkan cakupan deteksi sekaligus menurunkan *false negative*. Sementara itu, Kumar et al. (2022) menekankan pentingnya pemanfaatan *machine learning* untuk mengurangi *false positive* pada data jaringan yang kompleks.

## DAFTAR PUSTAKA

1. Open Information Security Foundation. (2025). *Suricata Documentation: EVE JSON Output*. Retrieved from  
<https://suricata.readthedocs.io/en/latest/output/eve/eve-json-output.html>
2. Stratosphere IPS. (2011). *CTU-13: A Labeled Dataset with Botnet, Normal and Background Traffic*. Retrieved from  
<https://www.stratosphereips.org/datasets-ctu13>
3. Canadian Institute for Cybersecurity. (2017). *Intrusion Detection Evaluation Dataset (CICIDS2017)*. Retrieved from  
<https://www.unb.ca/cic/datasets/ids-2017.html>
4. Moustafa, N., & Slay, J. (2015). *UNSW-NB15 Dataset (Official Description & Download)*. UNSW Canberra. Retrieved from  
<https://research.unsw.edu.au/projects/unsw-nb15-dataset>
5. Koroniots, N., Moustafa, N., Sitnikova, E., & Turnbull, B. (2018). *BoT-IoT Dataset (Official Description & Download)*. UNSW Canberra. Retrieved from  
<https://research.unsw.edu.au/projects/bot-iot-dataset>
6. Kasongo, S. M., & Sun, Y. (2020). Performance analysis of intrusion detection systems using a filter-based feature selection algorithm. *Journal of Big Data*, 7, 94.  
<https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00338-6>
7. Seliya, N., Khoshgoftaar, T. M., & Hulse, J. V. (2021). A literature review on one-class classification and its potential applications in big data. *Journal of Big Data*, 8(1), 54.  
<https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00417-5>
8. Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation Forest. In *ICDM 2008*.  
<https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf>
9. Yang, K., Kpotufe, S., & Feamster, N. (2021). *An Efficient One-Class SVM for Anomaly Detection in the Internet of Things*. arXiv:2104.11146.  
<https://arxiv.org/abs/2104.11146>

10. Rabih, R., et al. (2025). Highly accurate anomaly-based intrusion detection through LOF and CNN. *Scientific Reports*.  
<https://www.nature.com/articles/s41598-025-94477-y>
11. Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection*. NDSS 2018.  
<https://arxiv.org/abs/1802.09089>
12. Ataa, M. S., Alani, M. M., & Alzubaidi, L. (2024). Intrusion detection in software-defined networks using deep learning. *BMC Research Notes*.  
<https://bmcresearchnotes.biomedcentral.com/articles/10.1186/s13104-024-06893-9>
13. Putra, M. A. R., et al. (2022). Botnet dataset with simultaneous attack activity. *Data in Brief*.  
<https://www.sciencedirect.com/science/article/pii/S2352340922002705>
14. Suricata Forum. (2021). *How to add anomaly detector in Suricata?* Retrieved from <https://forum.suricata.io/t/how-to-add-anomaly-detector-in-suricata/155>
15. Wibowo, A. (2025). On the comprehensive analyses of CTU-13 botnet dataset for cyber security researches. *Indonesian Journal of Data Science*.  
<https://ojs.uph.edu/index.php/IJDS/article/view/6276>
16. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). *A detailed analysis of CICIDS2017 dataset for designing IDS*. Retrieved from  
<https://www.unb.ca/cic/datasets/ids-2017.html>

## PROFIL PENULIS

### **Dhimas LN**

Dhimas LN lahir di Yogyakarta dan lulus dari SMK jurusan Avionik pada tahun 2019. Ia memiliki keahlian di bidang elektronika dasar serta pengalaman aktif dalam organisasi siswa dan kegiatan pramuka, yang membentuk keterampilan kepemimpinan, kedisiplinan, dan kemampuan kerja sama tim.

### **Jovita Kusuma**

Jovita Kusuma lahir di Medan dan sedang menempuh pendidikan sarjana di Universitas Siber Asia (Unsia), Program Studi Teknik Komputer. Ia memiliki ketertarikan mendalam pada bidang Cyber Threat Intelligence dan Malware Analysis. Selain itu, Jovita Kusuma aktif menulis artikel edukasi dan panduan praktis di bidang keamanan siber, ditujukan bagi kalangan profesional teknologi informasi dan mahasiswa.

Nama	Pendidikan	Fokus Utama	Pengalaman Utama
Dhimas LN	SMK/Sederajat	Elektronika Dasar	Elektronika Arus AC/DC
Jovita Kusuma	S.Kom. Teknik Informatika, Universitas Siber Asia	Blue Team, Forensik Digital	Analisis anomali, investigasi insiden, laporan teknis