IDS Pipeline Architecture Guide

📚 Table of Contents

Overview

This IDS (Intrusion Detection System) uses a multi-stage pipeline combining **Suricata** for signature-based detection and **Machine Learning** for anomaly detection. The system can operate in two modes:

1. **AF_PACKET Mode** - Standard Linux packet capture (compatible with all interfaces)
2. **DPDK Mode** - High-performance kernel bypass (requires DPDK-compatible NIC)

Both pipelines follow the same architectural pattern but differ in packet capture mechanisms.

AF_PACKET Pipeline

🔄 Architecture Flow

```
┌─────────────────────────────────────────────────────────────────────┐
│                      AF_PACKET MODE PIPELINE                        │
└─────────────────────────────────────────────────────────────────────┘

  Network Traffic
        │
        ▼
┌───────────────────┐
│  Network Interface│  ← Any interface (USB, PCIe, WiFi, etc.)
│   (AF_PACKET)     │
└─────────┬─────────┘
          │
          │ Raw Packets
          ▼
┌───────────────────┐
│    Suricata       │  ← Signature-based detection
│   (AF_PACKET)     │  ← Rule-based alerts
└─────────┬─────────┘
          │
          │ EVE JSON logs (flows + alerts)
          ▼
┌───────────────────┐
```

PROF

```
│  Kafka Bridge      │  ← Reads EVE.json file
│  (Python script)   │  ← Streams to Kafka
└─────────┬──────────┘
          │
          │ JSON messages
          ▼
┌────────────────────┐
│   Apache Kafka     │  ← Message broker
│  (suricata-alerts) │  ← Decouples producers/consumers
└─────────┬──────────┘
          │
          │ JSON events
          ▼
┌────────────────────┐
│  ML Consumer       │  ← Feature extraction (65 features)
│  (Python)          │  ← ML inference (Random Forest/LightGBM)
│                    │  ← Threat scoring
└─────────┬──────────┘
          │
          │ Enhanced alerts
          ▼
┌────────────────────┐
│   Kafka Topic      │  ← Enhanced predictions
│ (ml-predictions)   │  ← Combined threat scores
└────────────────────┘
```

🔧 How It Works

Step 1: Packet Capture (AF_PACKET)

```
# Interface setup
ip link set enx00e04c36074c up
ip link set enx00e04c36074c promisc on
```

PROF

AF_PACKET is a Linux socket type that captures packets at Layer 2:

- **Pros**: Works with ANY network interface (USB adapters, WiFi, PCIe NICs)
- **Cons**: Packets traverse kernel network stack (some overhead)
- **Performance**: Good for moderate traffic (< 1 Gbps)

Step 2: Suricata Detection

```
suricata --af-packet=enx00e04c36074c \
         -c /etc/suricata/suricata.yaml \
         --set outputs.1.eve-log.enabled=yes
```

Suricata processes packets using:

- **Signature matching**: Rules from Emerging Threats, custom rules
- **Protocol analysis**: HTTP, DNS, TLS, SSH, etc.
- **Flow tracking**: Monitors bi-directional connections
- **Outputs**: EVE JSON format (flows, alerts, HTTP logs, etc.)

Step 3: Kafka Bridge

```
# Watches eve.json file and streams to Kafka
with open('/var/log/suricata/eve.json', 'r') as f:
    for line in follow_file(f):
        event = json.loads(line)
        producer.send('suricata-alerts', event)
```

The bridge:

- Tails the EVE JSON log file
- Parses each JSON event
- Publishes to Kafka topic
- Handles reconnections and errors

Step 4: ML Consumer Processing

```
# Extract features from flow events
features = extract_cicids2017_features(flow_event)
# 65 features: packet counts, byte counts, IAT stats, flags, etc.

# ML inference
prediction = model.predict(features)
# Classes: BENIGN, DDoS, PortScan, BotNet, etc.

# Combine with Suricata alerts
threat_score = calculate_combined_score(ml_prediction, suricata_alerts)
```

PROF

The ML consumer:

- **Extracts 65 CICIDS2017 features** from network flows
- **Maps to 34 model features** (feature engineering)
- **Performs ML inference** using trained Random Forest/LightGBM
- **Combines scores** with Suricata signature-based alerts
- **Publishes enhanced alerts** back to Kafka

📊 Feature Extraction

The pipeline extracts **65 network flow features** including:

Category	Features	Examples
Basic	8 features	Source/Dest IP, Port, Protocol, Timestamp
Flow Duration	1 feature	Total flow duration
Packet Stats	14 features	Total fwd/bwd packets, lengths, rates
Byte Stats	8 features	Total bytes, mean packet sizes
Inter-Arrival Time	12 features	IAT mean, std, min, max (fwd/bwd)
Flags	6 features	FIN, SYN, RST, PSH, ACK, URG counts
Header Lengths	4 features	Fwd/Bwd header lengths
Bulk Stats	6 features	Bulk rates, averages
Active/Idle	6 features	Active mean/std, Idle mean/std

🎯 Attack Detection

The ML models detect:

- **DDoS** (SYN flood, UDP flood, HTTP flood)
- **Port Scanning** (Nmap, Masscan patterns)
- **Brute Force** (SSH, FTP, Web login attacks)
- **Botnet** (C&C communication patterns)
- **Web Attacks** (SQL injection, XSS patterns)
- **Infiltration** (Internal reconnaissance)

DPDK Pipeline

🔄 Architecture Flow

PROF

```
┌──────────────────────────────────────────────────────────────────────┐
│                          DPDK MODE PIPELINE                          │
└──────────────────────────────────────────────────────────────────────┘

  Network Traffic
        │
        ▼
┌───────────────────┐
│  Network Interface│  ← DPDK-compatible NIC only
│   (DPDK PMD)      │  ← Intel, Mellanox, Broadcom
│                   │  ← Kernel bypass!
└─────────┬─────────┘
          │
          │ Raw Packets (Zero-copy)
          ▼
┌───────────────────┐
```

```
│   Hugepages        │  ← 2MB/1GB pages
│  Memory Pool       │  ← Pre-allocated buffers
└─────────┬──────────┘
          │
          │ Packet buffers
          ▼
┌────────────────────┐
│    Suricata        │  ← DPDK Poll Mode Driver
│     (DPDK)         │  ← Multi-core processing
└─────────┬──────────┘
          │
          │ EVE JSON logs
          ▼
┌────────────────────┐
│  Kafka Bridge      │  ← Same as AF_PACKET
│  (Python script)   │
└─────────┬──────────┘
          │
          ▼
    (Same as AF_PACKET pipeline)
```

🚀 How It Works

Step 1: DPDK Setup

Hugepages Allocation:

```
# Allocate 2GB of 2MB hugepages
echo 1024 > /proc/sys/vm/nr_hugepages
mount -t hugetlbfs nodev /mnt/huge
```

PROF

Hugepages provide:

- **Reduced TLB misses**: Fewer page table lookups
- **Contiguous memory**: Better for DMA operations
- **Performance**: 10-20% improvement for packet processing

Interface Binding:

```
# Unbind from kernel driver
dpdk-devbind.py --unbind 0000:01:00.0

# Bind to DPDK driver
dpdk-devbind.py --bind=vfio-pci 0000:01:00.0
```

This removes the NIC from kernel control and gives DPDK direct access.

Step 2: DPDK Packet Capture

Poll Mode Driver (PMD):

```
Traditional (Interrupt-driven):
  Packet arrives → Hardware interrupt → Context switch → Kernel → Copy →
Userspace

DPDK (Polling):
  Packet arrives → Direct DMA → Hugepage buffer → Userspace reads
  (No interrupts, no context switches, no kernel involvement)
```

Benefits:

- **Ultra-low latency**: < 100 ns packet processing
- **Zero-copy**: Direct memory access to packet buffers
- **Multi-core**: Dedicated CPU cores for packet processing
- **Throughput**: 10+ Gbps on commodity hardware

Step 3: Suricata DPDK Mode

```
suricata --dpdk \
         -c /etc/suricata/suricata-dpdk.yaml \
         --set dpdk.interfaces.0.interface=0000:01:00.0
```

Suricata with DPDK:

- Uses **DPDK libraries** for packet I/O
- Bypasses kernel network stack completely
- Allocates **packet buffers from hugepages**
- Uses **lock-free queues** for multi-core processing
- **Same detection logic** as AF_PACKET mode (rules, signatures)

PROF

Step 4-6: Kafka → ML Consumer

Same as AF_PACKET pipeline - Once packets are processed by Suricata, the rest of the pipeline is identical.

⚡ Performance Characteristics

Metric	AF_PACKET	DPDK
Throughput	1-2 Gbps	10+ Gbps
Latency	10-50 μs	< 1 μs
CPU Usage	Moderate	High (dedicated cores)

Metric	AF_PACKET	DPDK
Packet Loss	5-10% @ high load	< 1% @ high load
Setup Complexity	Easy	Complex

Comparison

When to Use AF_PACKET Mode

✅ **Use AF_PACKET if:**

- You have a USB network adapter
- You're testing/development
- Traffic < 1 Gbps
- You want simple setup
- You don't have DPDK-compatible hardware
- You need WiFi or virtual interfaces

When to Use DPDK Mode

✅ **Use DPDK if:**

- You have a DPDK-compatible NIC (Intel X710, Mellanox ConnectX, etc.)
- Traffic > 1 Gbps
- You need minimal packet loss
- Low latency is critical
- You can dedicate CPU cores
- Production deployment

Hardware Requirements

PROF

Component	AF_PACKET	DPDK
NIC	Any interface	DPDK-compatible only
RAM	4GB+	8GB+ (hugepages)
CPU	2+ cores	4+ cores (dedicated)
OS	Any Linux	Linux with hugepages

Component Details

1. Suricata

Role: Signature-based intrusion detection

Key Features:

- 30,000+ community rules (Emerging Threats)
- Protocol parsers: HTTP, DNS, TLS, SSH, SMB, etc.
- Flow tracking: Monitors connection state
- File extraction: Can extract files from traffic
- Performance: Multi-threaded, hardware accelerated

Configuration:

```
# suricata.yaml
af-packet:
  - interface: enx00e04c36074c
    threads: 4
    cluster-type: cluster_flow

outputs:
  - eve-log:
      enabled: yes
      filetype: regular
      types:
        - alert
        - http
        - dns
        - flow
```

2. Apache Kafka

Role: Message broker and event streaming platform

Why Kafka?

- **Scalability**: Can handle millions of events/sec
- **Durability**: Persists messages to disk
- **Decoupling**: Producers and consumers are independent
- **Replay**: Can replay historical events
- **Multi-consumer**: Multiple consumers can read same stream

PROF

Topics:

- `suricata-alerts`: All Suricata events (flows + alerts)
- `ml-predictions`: ML-enhanced threat predictions

3. ML Consumer

Role: Machine learning inference and threat scoring

Components:

1. **Feature Extractor** (`feature_extractor.py`)
    - Parses Suricata flow events

    - Calculates 65 CICIDS2017 features
    - Handles missing data

2. **Feature Mapper** (`feature_mapper.py`)

    - Maps 65 features → 34 model features
    - Handles feature engineering
    - Normalizes values

3. **Model Loader** (`model_loader.py`)

    - Loads trained ML models (Random Forest, LightGBM)
    - Manages model versions
    - Performs inference

4. **Alert Processor** (`alert_processor.py`)

    - Combines ML predictions with Suricata alerts
    - Calculates composite threat scores
    - Generates enhanced alerts

ML Models:

- **Random Forest** (2017 model): 99.2% accuracy on CICIDS2017
- **LightGBM** (2018 model): 99.5% accuracy on CICIDS2018

4. Kafka Bridge

Role: Stream Suricata logs to Kafka

Implementation:

```python
def tail_file(filename):
    """Follow a file like 'tail -f'"""
    with open(filename, 'r') as f:
        f.seek(0, 2)  # Go to end
        while True:
            line = f.readline()
            if not line:
                time.sleep(0.1)
                continue
            yield line

for line in tail_file('/var/log/suricata/eve.json'):
    event = json.loads(line)
    producer.send('suricata-alerts', event)
```

PROF

Data Flow Example

Example: HTTP Request Detection

```
1. Client sends HTTP GET request to suspicious domain
   └─ Source: 192.168.1.100:54321
   └─ Dest: 192.0.2.50:80
   └─ Payload: "GET /malware.exe HTTP/1.1"

2. Suricata captures packet via AF_PACKET/DPDK
   └─ Signature match: "ET MALWARE Suspicious .exe Download"
   └─ HTTP parser extracts: hostname, URI, user-agent
   └─ Flow tracker: This is packet 5 in the flow

3. Suricata writes EVE JSON:
{
  "timestamp": "2025-10-09T10:15:30.123456+0000",
  "event_type": "alert",
  "alert": {
    "signature": "ET MALWARE Suspicious .exe Download",
    "category": "Malware Command and Control Activity Detected",
    "severity": 1
  },
  "flow": {
    "pkts_toserver": 5,
    "pkts_toclient": 4,
    "bytes_toserver": 480,
    "bytes_toclient": 8420
  },
  "http": {
    "hostname": "bad-domain.com",
    "url": "/malware.exe",
    "http_method": "GET"
  }
}

4. Kafka Bridge reads and publishes to Kafka

5. ML Consumer receives event:
   a. Extracts 65 features from flow data:
      - Total packets: 9
      - Total bytes: 8900
      - Flow duration: 2.5 seconds
      - Average packet size: 988 bytes
      - Forward IAT mean: 0.5s
      - SYN flags: 1, ACK flags: 8
      ... (59 more features)

   b. Maps to 34 model features

   c. ML model predicts:
      - Class: "Web Attack" (0.85 confidence)
      - Anomaly score: 0.92
```

PROF

```
   d. Combines with Suricata alert:
      - Suricata severity: 1 (High)
      - ML confidence: 0.85
      - Combined threat score: 0.95 (CRITICAL)

   e. Publishes enhanced alert:
{
  "timestamp": "2025-10-09T10:15:30.500000+0000",
  "source_event": "suricata",
  "ml_prediction": {
    "class": "Web Attack",
    "confidence": 0.85,
    "model": "random_forest_2017"
  },
  "combined_threat_score": 0.95,
  "threat_level": "CRITICAL",
  "suricata_alert": { ... },
  "recommended_action": "BLOCK"
}
```

Performance Tuning

AF_PACKET Optimizations

```
# Increase ring buffer size
ethtool -G enx00e04c36074c rx 4096

# Disable offloading for accurate capture
ethtool -K enx00e04c36074c gro off lro off gso off tso off

# Enable promiscuous mode
ip link set enx00e04c36074c promisc on
```

PROF

DPDK Optimizations

```
# CPU isolation (dedicate cores to DPDK)
# Edit /etc/default/grub:
GRUB_CMDLINE_LINUX="isolcpus=2,3,4,5"

# Increase hugepages
echo 2048 > /proc/sys/vm/nr_hugepages

# Use 1GB hugepages for better performance
echo 4 > /sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages
```

Troubleshooting

AF_PACKET Issues

Problem: No packets captured

```
# Check interface is up and in promiscuous mode
ip link show enx00e04c36074c

# Check Suricata is running
ps aux | grep suricata

# Check Suricata logs
tail -f /var/log/suricata/suricata.log
```

Problem: High packet loss

```
# Increase Suricata threads
suricata --af-packet=enx00e04c36074c --set af-packet.threads=4

# Check CPU usage
top -H -p $(pgrep suricata)
```

DPDK Issues

Problem: Interface binding fails

```
# Check IOMMU is enabled
dmesg | grep -i iommu

# Try different driver
dpdk-devbind.py --bind=uio_pci_generic 0000:01:00.0
```

PROF

Problem: Suricata crashes with DPDK

```
# Check hugepages
cat /proc/meminfo | grep Huge

# Check DPDK EAL parameters
suricata --dpdk --dump-config | grep dpdk
```

Summary

Both pipelines provide comprehensive threat detection combining:

- ✅ **Signature-based detection** (Suricata rules)
- ✅ **Anomaly detection** (Machine Learning)
- ✅ **Flow-based analysis** (Network behavior)
- ✅ **Real-time processing** (Kafka streaming)
- ✅ **Scalable architecture** (Microservices)

Choose AF_PACKET for ease of use and compatibility.
Choose DPDK for maximum performance and minimal packet loss.

Both modes share the same ML detection capabilities and achieve the same detection accuracy!