Real-Time IDS Pipeline with ML Inference

🎯 Architecture Overview

```
┌─────────────────────────────────────────────────────────────────┐
│                Real-Time Flow-Based Detection                    │
└─────────────────────────────────────────────────────────────────┘

External Device (192.168.100.2)
         │
         │ Ethernet Cable
         ▼
   USB Adapter (enx00e04c36074c)
         │
         │ AF_PACKET (Promiscuous)
         ▼
  ┌─────────────────┐
  │    Suricata     │  ◄──  Captures packets in real-time
  │  (AF_PACKET)    │       Extracts flow features
  └────────┬────────┘       Generates alerts
           │
           │ EVE JSON (flow + alert data)
           ▼
  ┌─────────────────┐
  │     Kafka       │  ◄──  Message queue
  │  Topic: alerts  │       Decouples components
  └────────┬────────┘
           │
           │ Consumes messages
           ▼
  ┌─────────────────┐
  │ Feature         │  ◄──  Extracts ML features
  │ Extractor       │       Per flow/alert
  └────────┬────────┘
           │
           │ Feature vectors
           ▼
  ┌─────────────────┐
  │ ML Inference    │  ◄──  Random Forest/LightGBM
  │ (Real-time)     │       Predicts: Benign/Attack
  └────────┬────────┘       Confidence scores
           │
           │ Predictions
           ▼
  ┌─────────────────┐
  │  Alert Action   │  ◄──  High confidence attacks
  │  (Log/Block)    │       Real-time response
  └─────────────────┘
```

PROF

🚀 Quick Start - Complete Pipeline

Step 1: Setup IDS System (One-Time)

```
cd ~/Programming/IDS/dpdk_suricata_ml_pipeline/scripts

# Configure USB adapter for external traffic
sudo ./00_setup_external_capture.sh
```

Step 2: Start Complete Pipeline

Option A: Interactive Menu (Recommended)

```
sudo ./quick_start.sh
# Select option 1: Start Complete Pipeline
```

Option B: Manual Start (More Control)

Open 4 terminals:

Terminal 1: Start Kafka

```
cd ~/Programming/IDS/dpdk_suricata_ml_pipeline/scripts
sudo ./02_setup_kafka.sh
```

Terminal 2: Start Suricata (AF_PACKET)

PROF

```
cd ~/Programming/IDS/dpdk_suricata_ml_pipeline/scripts
sudo ./03_start_suricata_afpacket.sh
```

Terminal 3: Start ML Consumer (Real-time Inference)

```
cd ~/Programming/IDS/dpdk_suricata_ml_pipeline/scripts
./04_start_ml_consumer.sh
```

Terminal 4: Monitor (Optional)

```
cd ~/Programming/IDS/dpdk_suricata_ml_pipeline
tail -f logs/suricata/eve.json | jq 'select(.event_type=="alert" or
.event_type=="flow")'
```

Step 3: Setup External Device (Traffic Generator)

On Windows:

```
# Run as Administrator
New-NetIPAddress -InterfaceAlias "Ethernet" -IPAddress 192.168.100.2 -
PrefixLength 24 -DefaultGateway 192.168.100.1
ping 192.168.100.1
```

On Linux:

```
sudo ip addr add 192.168.100.2/24 dev eth0
sudo ip link set eth0 up
ping 192.168.100.1
```

Step 4: Generate Traffic

From External Device:

```
# Simple HTTP traffic
curl http://192.168.100.1

# Port scan (triggers alerts)
nmap 192.168.100.1

# Replay PCAP with attacks
sudo tcpreplay -i eth0 --mbps=10 attack.pcap
```

PROF

Step 5: Watch Real-Time Detection

Open multiple terminals on IDS system:

Terminal 1: Raw Packets

```
sudo tcpdump -i enx00e04c36074c -n -l
```

Terminal 2: Suricata Alerts

```
tail -f logs/suricata/eve.json | jq 'select(.event_type=="alert") |
{timestamp, src_ip, dest_ip, alert: .alert.signature}'
```

Terminal 3: ML Predictions

```
tail -f logs/ml/consumer.log | grep -E "(ATTACK|BENIGN)"
```

Terminal 4: Stats

```
watch -n 2 'suricatasc -c dump-counters | grep -E "
(capture|decoder|detect|flow)"'
```

📊 Understanding Real-Time Flow-Based Detection

What is a "Flow"?

A **network flow** is a sequence of packets sharing:

- Same source IP + port
- Same destination IP + port
- Same protocol (TCP/UDP/ICMP)
- Same time window

Example:

```
Flow: 192.168.100.2:45678 → 192.168.100.1:80 (TCP)
Packets: SYN, SYN-ACK, ACK, HTTP GET, HTTP Response, ...
Duration: 2.5 seconds
Bytes: 4,523 total
```

PROF

How Real-Time Inference Works

```
1. Suricata captures packets in AF_PACKET mode
   └─> Groups packets into flows
   └─> Extracts features per flow:
         • Duration
         • Packet count
         • Byte count
         • Flags (SYN, FIN, RST, PSH, ACK, URG)
         • Inter-arrival times
         • Packet size statistics

2. For each flow, Suricata outputs:
   └─> EVE JSON with flow metadata
```

```
    └─> Alerts if Suricata rules triggered

3. Kafka receives EVE JSON events
    └─> Buffers messages
    └─> ML consumer polls in real-time

4. Feature Extractor processes each message
    └─> Extracts 80+ ML features
    └─> Normalizes values

5. ML Model predicts in real-time
    └─> Random Forest / LightGBM inference
    └─> Output: BENIGN or ATTACK class
    └─> Confidence score

6. High-confidence attacks logged/blocked
    └─> Threshold: 0.7+ confidence
    └─> Action: Log, alert, block (configurable)
```

🔧 Configuration for Real-Time Performance

1. Suricata Configuration

Check your Suricata config for AF_PACKET settings:

```
# View current config
cat /etc/suricata/suricata.yaml | grep -A 20 "af-packet:"
```

Recommended settings for real-time:

PROF

```
af-packet:
  - interface: enx00e04c36074c
    threads: 2              # Number of capture threads
    cluster-id: 99
    cluster-type: cluster_flow  # Flow-based load balancing
    defrag: yes
    use-mmap: yes
    mmap-locked: yes
    ring-size: 2048         # Larger ring = less drops
    block-size: 32768
    buffer-size: 32768
```

2. Flow Timeout Settings

```
flow:
  managers: 1
  hash-size: 65536
  prealloc: 10000

  # Timeouts (seconds)
  emergency-recovery: 30
  timeout:
    new: 30
    established: 300      # Active connections
    closed: 0
    bypassed: 100
    emergency-new: 10
    emergency-established: 100
    emergency-closed: 0
```

3. ML Consumer Configuration

Edit `config/ids_config.yaml`:

```
ml:
  model_path: "../ML Models/random_forest_model_2017.joblib"
  batch_size: 100         # Process in batches for efficiency
  confidence_threshold: 0.7

kafka:
  bootstrap_servers: "localhost:9092"
  topic: "suricata-alerts"
  group_id: "ids-ml-consumer"
  auto_offset_reset: "latest"   # Only process new messages

performance:
  enable_monitoring: true
  stats_interval: 10      # Log stats every 10 seconds
  max_lag: 1000          # Alert if consumer lags behind
```

PROF

📈 Monitoring Real-Time Performance

Key Metrics to Watch

1. Packet Capture Rate

```
# Live packet rate
sudo tcpdump -i enx00e04c36074c -n | pv -l -i 1 > /dev/null

# Expected: 100-10,000 packets/sec depending on traffic
```

2. Suricata Processing Rate

```
suricatasc -c dump-counters | grep -E "
(capture|decoder.pkts|flow.mgr.flows_checked)"

# Key counters:
# - capture.kernel_packets: Total captured
# - capture.kernel_drops: Packet drops (should be 0)
# - decoder.pkts: Successfully decoded
# - flow.mgr.flows_checked: Flows processed
```

3. Kafka Lag

```
# Check consumer lag
kafka-consumer-groups.sh --bootstrap-server localhost:9092 \
  --describe --group ids-ml-consumer

# Lag should be < 100 for real-time
```

4. ML Inference Rate

```
# Check ML consumer logs
tail -f logs/ml/consumer.log | grep "Processed"

# Example output:
# 2025-10-07 16:30:15 - Processed 245 messages in 0.8s (306 msg/sec)
```

PROF

5. System Resources

```
# CPU and memory usage
htop -p $(pgrep -d, suricata),$(pgrep -d, kafka),$(pgrep -d, python)

# Network stats
watch -n 1 'ip -s link show enx00e04c36074c'
```

🎯 Example Real-Time Detection Scenarios

Scenario 1: Port Scan Detection

From External Device:

```
# Generate port scan
nmap -sS 192.168.100.1 -p 1-1000
```

Watch on IDS:

Terminal 1: Packets

```
sudo tcpdump -i enx00e04c36074c 'tcp[tcpflags] & tcp-syn != 0' -n -c 20
# Shows SYN packets
```

Terminal 2: Suricata Alerts

```
tail -f logs/suricata/eve.json | jq 'select(.event_type=="alert" and
(.alert.signature | contains("scan"))) | {time: .timestamp, alert:
.alert.signature}'

# Example output:
# {
#   "time": "2025-10-07T16:30:45.123456+0000",
#   "alert": "GPL SCAN nmap TCP"
# }
```

Terminal 3: ML Prediction

```
tail -f logs/ml/consumer.log | grep -A 5 "Port Scan"

# Example output:
# [2025-10-07 16:30:45] Flow: 192.168.100.2:45678 -> 192.168.100.1:80
# [2025-10-07 16:30:45] Features: {fwd_pkts: 1, duration: 0.001, ...}
# [2025-10-07 16:30:45] ML Prediction: ATTACK (Port Scan)
# [2025-10-07 16:30:45] Confidence: 0.94
# [2025-10-07 16:30:45] Action: LOGGED
```

PROF

Scenario 2: DDoS Detection

From External Device:

```
# Generate SYN flood
sudo hping3 -S 192.168.100.1 -p 80 --flood -c 1000
```

Watch on IDS:

```
# Suricata will detect:
# - High SYN rate
# - Many flows from same source
# - Incomplete connections

# ML will detect:
# - Abnormal packet rate
# - Short flow durations
# - High SYN/ACK ratio
```

Scenario 3: SQL Injection Detection

From External Device:

```
# Python script
import requests
import time

target = "http://192.168.100.1/login"
payloads = [
    "' OR '1'='1",
    "admin'--",
    "1' OR 1=1--",
    "'; DROP TABLE users--"
]

for payload in payloads:
    try:
        requests.get(f"{target}?user={payload}")
    except:
        pass
    time.sleep(1)
```

PROF

Watch on IDS:

```
# Suricata HTTP inspection
tail -f logs/suricata/eve.json | jq 'select(.event_type=="alert" and
(.alert.category=="Web Application Attack"))'

# ML will detect:
# - HTTP payload patterns
# - URI anomalies
# - Query string characteristics
```

🔬 Deep Dive: ML Feature Extraction

Features Extracted Per Flow

The ML model uses **80+ features** extracted from each flow:

1. Basic Flow Features

- `duration` - Flow duration in seconds
- `fwd_pkts_tot` - Forward packets count
- `bwd_pkts_tot` - Backward packets count
- `fwd_data_pkts_tot` - Forward data packets
- `bwd_data_pkts_tot` - Backward data packets

2. Packet Size Statistics

- `fwd_pkts_per_sec` - Forward packet rate
- `bwd_pkts_per_sec` - Backward packet rate
- `flow_pkts_per_sec` - Total packet rate
- `down_up_ratio` - Download/Upload ratio
- `fwd_header_size_tot` - Total forward header size
- `fwd_header_size_min` - Minimum forward header size
- `fwd_header_size_max` - Maximum forward header size

3. Timing Features

- `flow_iat_mean` - Mean inter-arrival time
- `flow_iat_max` - Maximum inter-arrival time
- `flow_iat_min` - Minimum inter-arrival time
- `flow_iat_std` - Standard deviation of IAT
- `fwd_iat_tot` - Total forward IAT
- `fwd_iat_mean` - Mean forward IAT
- `fwd_iat_max` - Maximum forward IAT
- `fwd_iat_min` - Minimum forward IAT
- `fwd_iat_std` - Standard deviation forward IAT

PROF

4. TCP Flags

- `fwd_psh_flag` - PSH flag count (forward)
- `bwd_psh_flag` - PSH flag count (backward)
- `fwd_urg_flag` - URG flag count (forward)
- `bwd_urg_flag` - URG flag count (backward)
- `fin_flag_cnt` - FIN flag count
- `syn_flag_cnt` - SYN flag count
- `rst_flag_cnt` - RST flag count
- `psh_flag_cnt` - PSH flag count
- `ack_flag_cnt` - ACK flag count
- `urg_flag_cnt` - URG flag count
- `ece_flag_cnt` - ECE flag count

5. Payload Features

- `payload_bytes_per_second` - Payload rate
- `avg_fwd_segment_size` - Average forward segment
- `avg_bwd_segment_size` - Average backward segment
- `fwd_header_size_tot` - Total header size
- `fwd_avg_bytes_per_bulk` - Bulk transfer stats
- `fwd_avg_pkts_per_bulk`
- `fwd_avg_bulk_rate`

Example Feature Vector:

```
{
    'duration': 2.45,
    'fwd_pkts_tot': 156,
    'bwd_pkts_tot': 142,
    'flow_pkts_per_sec': 121.6,
    'down_up_ratio': 0.91,
    'fwd_iat_mean': 0.015,
    'syn_flag_cnt': 1,
    'ack_flag_cnt': 298,
    'avg_fwd_segment_size': 512,
    ...  # 80+ total features
}
```

How ML Inference Works

```
# Simplified ML inference flow

# 1. Receive flow data from Kafka
flow_data = kafka_consumer.poll()

# 2. Extract features
features = feature_extractor.extract(flow_data)
# Result: numpy array [80 features]

# 3. Normalize features
normalized = scaler.transform(features)

# 4. ML prediction
prediction = model.predict(normalized)
# Result: 0 (BENIGN) or 1 (ATTACK)

confidence = model.predict_proba(normalized)
# Result: [0.05, 0.95] = 95% confidence ATTACK

# 5. Classification
if prediction == 1 and confidence[1] > 0.7:
    label = "ATTACK"
    action = "LOG_AND_ALERT"
```

PROF

```
else:
    label = "BENIGN"
    action = "PASS"

# 6. Output
print(f"{timestamp} | {src_ip}:{src_port} -> {dst_ip}:{dst_port} |
{label} ({confidence[1]:.2f})")
```

📊 Performance Tuning

For Low-Traffic Scenarios (<100 packets/sec)

```
# Suricata
af-packet:
  - threads: 1
    ring-size: 1024

# ML Consumer
ml:
  batch_size: 10          # Smaller batches
  inference_timeout: 0.1  # Fast inference
```

For Medium-Traffic Scenarios (100-1000 packets/sec)

```
# Suricata
af-packet:
  - threads: 2
    ring-size: 2048

# ML Consumer
ml:
  batch_size: 100
  inference_timeout: 1.0
```

PROF

For High-Traffic Scenarios (>1000 packets/sec)

```
# Suricata
af-packet:
  - threads: 4
    ring-size: 4096
    cluster-type: cluster_flow

# ML Consumer
ml:
  batch_size: 500
  parallel_workers: 2     # Multiple inference threads
  inference_timeout: 5.0
```

🐛 Troubleshooting Real-Time Pipeline

Issue: High Kafka Lag

Symptom: ML consumer can't keep up with Suricata output

Check:

```
kafka-consumer-groups.sh --bootstrap-server localhost:9092 \
  --describe --group ids-ml-consumer
```

Solution:

```
# Increase batch size
# Edit config/ids_config.yaml
ml:
  batch_size: 500  # Increase from 100

# Add more consumer workers
# Or reduce Suricata output rate
```

Issue: Packet Drops

Symptom: `capture.kernel_drops` counter increasing

Check:

```
suricatasc -c dump-counters | grep drops
```

Solution:

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

# Increase socket buffers
sudo sysctl -w net.core.rmem_max=134217728
sudo sysctl -w net.core.netdev_max_backlog=5000
```

```
# Disable NIC offloading
sudo ethtool -K enx00e04c36074c gro off gso off tso off
```

Issue: Slow ML Inference

Symptom: ML consumer log shows high processing time

Check:

```
tail -f logs/ml/consumer.log | grep "Processed.*in"
```

Solution:

```
# Use lighter model (if available)
# Or increase batch size
# Or add GPU acceleration (if available)

# Check CPU usage
htop -p $(pgrep -f ml_kafka_consumer)
```

Issue: No Alerts Generated

Symptom: Traffic flows but no alerts/predictions

Check:

```
# Is Suricata detecting?
suricatasc -c dump-counters | grep detect.alert

# Is Kafka receiving?
kafka-console-consumer.sh --bootstrap-server localhost:9092 \
  --topic suricata-alerts --from-beginning | head -10

# Is ML consumer running?
ps aux | grep ml_kafka_consumer
```

PROF

Solution:

```
# Check Suricata rules are loaded
suricatasc -c ruleset-stats

# Check Kafka connection
# Check ML model is loaded
tail -100 logs/ml/consumer.log | grep -i error
```

🎓 Complete Workflow Checklist

Pre-Flight

- □ USB adapter connected and configured (192.168.100.1)
- □ External device configured (192.168.100.2)
- □ Physical Ethernet cable connected
- □ Can ping between devices

Start Pipeline

- □ Kafka started (`ps aux | grep kafka`)
- □ Suricata started (`ps aux | grep suricata`)
- □ ML consumer started (`ps aux | grep ml_kafka_consumer`)
- □ All services healthy (check logs)

Verify Real-Time Detection

- □ Generate test traffic
- □ tcpdump shows packets on IDS
- □ Suricata EVE JSON updating
- □ Kafka messages flowing
- □ ML predictions appearing
- □ No packet drops
- □ Kafka lag < 100

Monitor Performance

- □ Packet capture rate acceptable
- □ CPU usage < 80%
- □ Memory usage < 80%
- □ No errors in logs
- □ ML inference < 100ms per flow

PROF

📚 Summary

Pipeline Flow

```
External Device → USB Adapter → Suricata (AF_PACKET) → Kafka → ML
Consumer → Predictions
```

Key Commands

Start:

```
sudo ./quick_start.sh  # Interactive
```

Monitor:

```
# Packets
sudo tcpdump -i enx00e04c36074c -n

# Alerts
tail -f logs/suricata/eve.json | jq 'select(.event_type=="alert")'

# ML
tail -f logs/ml/consumer.log

# Stats
suricatasc -c dump-counters
```

Stop:

```
sudo ./stop_all.sh
```

Expected Performance

- **Latency:** < 10ms packet-to-prediction
- **Throughput:** 100-1000 flows/sec
- **Accuracy:** 95%+ (depending on model)
- **Packet Loss:** < 0.1%

PROF

🚀 You're Ready!

Your real-time IDS pipeline with ML inference is now operational. Generate traffic from your external device and watch the system detect attacks in real-time! 🎉