

Network Intrusion Detection with AI

Defensive Context: The tools and techniques in this lesson are intended for authorized network defense and security monitoring. Always operate within your organization's policies and applicable laws. Deploy detection systems only on networks you are authorized to monitor.

IDS vs IPS: Understanding the Difference

An **Intrusion Detection System (IDS)** passively monitors network traffic or host activity and generates alerts when it identifies suspicious behavior. It sits alongside the traffic flow, analyzing copies of packets without interfering with them. Think of it as a security camera: it observes and records, but it does not physically stop an intruder.

An **Intrusion Prevention System (IPS)** goes a step further. It sits inline with the traffic flow — directly in the path between source and destination — and can actively block, drop, or modify packets that match malicious signatures or anomalous patterns. An IPS is the security guard who can physically intervene.

Aspect	IDS	IPS
Placement	Out-of-band (passive tap or mirror port)	Inline (directly in traffic path)
Action	Alerts only	Alerts and blocks/drops
Latency impact	None	Adds slight latency
Risk of disruption	Low — cannot break connectivity	Higher — false positives can block legitimate traffic
Use case	Monitoring, forensics, threat hunting	Active defense, automated response

Many modern tools operate in both modes. Suricata, for example, can function as a passive IDS or be deployed inline as an IPS. The choice depends on your risk tolerance, network architecture, and operational maturity. Organizations often start with IDS to understand their traffic patterns before graduating to IPS mode.

Signature-Based vs Anomaly-Based Detection

Detection engines use two fundamental approaches, and understanding both is essential for effective network defense.

Signature-based detection compares network traffic against a database of known attack patterns. Each signature describes a specific sequence of bytes, protocol behavior, or packet structure associated with a known threat. This approach is fast, precise, and produces few false positives — but it is blind to novel attacks. If a threat does not match an existing signature, it passes through undetected.

Anomaly-based detection builds a statistical model of "normal" network behavior and flags deviations from that baseline. It can catch zero-day attacks and novel threats that signature-based systems miss. The tradeoff is a higher false positive rate, especially during initial deployment when the baseline is still being calibrated. Unusual but legitimate events — a software update, a new service, seasonal traffic spikes — can trigger false alerts.

In practice, the best defense layers both approaches. Signatures catch the known threats with high confidence, while anomaly detection provides coverage against the unknown. AI and machine learning are particularly powerful for anomaly-based detection, as we will explore later in this lesson.

Network IDS Tools: Snort, Suricata, and Zeek

Snort

Snort is the original open-source network IDS, created by Martin Roesch in 1998. It pioneered the rule-based detection model that every subsequent NIDS adopted. Snort operates in three modes: sniffer (packet display), packet logger, and network intrusion detection. Its rule language has become a de facto standard — most commercial IDS/IPS products can import Snort rules.

Suricata

Suricata, developed by the Open Information Security Foundation (OISF), is a multi-threaded IDS/IPS engine that is largely compatible with Snort rules but adds native support for multi-threading, file extraction, TLS/JA3 fingerprinting, and HTTP/2 parsing. Its performance on modern multi-core hardware significantly exceeds single-threaded Snort. Suricata also outputs structured JSON logs (EVE JSON), making it far easier to integrate with SIEMs and analysis pipelines.

Zeek (formerly Bro)

Zeek takes a fundamentally different approach. Rather than matching packets against signatures, Zeek is a network analysis framework that generates rich, structured logs describing every connection, DNS query, HTTP request, SSL certificate, and file transfer it observes. Zeek's scripting language allows you to write custom analyzers that correlate events across multiple connections and protocols. It excels at metadata analysis and threat hunting rather than real-time alerting.

Writing Snort/Suricata Rules

A Snort/Suricata rule consists of two parts: a **header** and **options**. The header defines the action, protocol, source/destination addresses, and ports. The options (enclosed in parentheses) specify the detection logic.

```
# Rule structure:
# action protocol src_ip src_port -> dst_ip dst_port (options)

# Alert on any HTTP request containing "/etc/passwd" in the URI
alert http $HOME_NET any -> $EXTERNAL_NET any (
    msg:"Possible LFI - /etc/passwd in URI";
    flow:to_server,established;
    http.uri;
    content:"/etc/passwd";
    classtype:web-application-attack;
    sid:1000001;
    rev:1;
)

# Detect SSH brute force - multiple failed logins
alert ssh $EXTERNAL_NET any -> $HOME_NET 22 (
```

```

    msg:"Potential SSH brute force attempt";
    flow:to_server,established;
    threshold:type both, track by_src, count 5, seconds 60;
    classtype:attempted-admin;
    sid:1000002;
    rev:1;
)

# Detect DNS exfiltration - unusually long subdomain queries

```

Key rule options include `content` (byte matching), `pcre` (regex matching), `flow` (connection direction and state), `threshold` (rate limiting), and protocol-specific sticky buffers like `http.uri`, `http.header`, and `dns.query` that constrain matching to specific parts of the parsed protocol.

Suricata YAML Configuration

Suricata's behavior is controlled by a YAML configuration file. Here is a representative excerpt showing key detection settings:

```

# suricata.yaml - key detection settings
vars:
  address-groups:
    HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"
    EXTERNAL_NET: "!$HOME_NET"

default-rule-path: /etc/suricata/rules
rule-files:
  - suricata.rules
  - local.rules
  - ai-generated.rules

af-packet:
  - interface: eth0
    cluster-id: 99
    cluster-type: cluster_flow
    defrag: yes

outputs:
  - eve-log:
      enabled: yes
      filetype: regular

```

```
filename: eve.json
types:
  - alert:
      tagged-packets: yes
```

Zeek Scripting for Network Analysis

Zeek scripts let you define custom detection logic that correlates events across protocols and connections. Here is an example that detects potential DNS tunneling by tracking query lengths per source IP:

```
@load base/frameworks/notice

module DNSTunnel;

export {
  redef enum Notice::Type += {
    DNS_Tunnel_Detected
  };

  # Threshold: flag if average query length exceeds this
  const avg_query_len_threshold = 40 &redef;
  const min_queries = 20 &redef;
}

global dns_query_tracker: table[addr] of vector of count;

event dns_request(c: connection, msg: dns_msg, query: string, qtype: count, qdcount: count)
{
  local src = c$id$orig_h;

  if ( src !in dns_query_tracker )
    dns_query_tracker[src] = vector();

  dns_query_tracker[src] += |query|;

  if ( |dns_query_tracker[src]| >= min_queries )
```

Network Traffic Baselines and Deviation Detection

Effective anomaly detection requires a solid understanding of what "normal" looks like. A **traffic baseline** captures typical patterns over time: average bandwidth, common protocols, peak usage hours, standard DNS query volumes, and expected connection durations. Deviations from this baseline — a sudden spike in outbound DNS traffic at 3 AM, an internal host connecting to an unusual country, or a server making thousands of short-lived connections — can indicate compromise.

Building a reliable baseline takes time (typically two to four weeks of data) and must account for regular business cycles. Machine learning models, particularly unsupervised techniques like isolation forests, autoencoders, and clustering algorithms, are well-suited for learning these baselines and flagging outliers.

Packet Inspection: Deep vs Metadata

Deep Packet Inspection (DPI) examines the full payload of every packet, including application-layer data. It can detect malicious content embedded in HTTP responses, command-and-control instructions hidden in DNS TXT records, or data exfiltration within encrypted tunnel payloads. DPI is thorough but computationally expensive and increasingly challenged by widespread TLS encryption.

Metadata analysis focuses on connection-level attributes without inspecting payloads: source/destination IPs, ports, timestamps, byte counts, packet counts, protocol flags, TLS certificate details, and JA3/JA3S fingerprints. Metadata analysis scales better, works even with encrypted traffic, and can detect many threats through behavioral patterns alone. Zeek's entire philosophy is built on this approach.

The trend in modern network defense is to combine both: use DPI where possible (decrypted traffic, internal segments) and rely on metadata analysis for encrypted flows.

AI-Enhanced Detection

Machine learning brings several capabilities to network intrusion detection that traditional rule-based systems cannot match:

- **Unsupervised anomaly detection** — Models like isolation forests and autoencoders learn normal traffic patterns and flag statistical outliers without

labeled training data.

- **Supervised classification** — Given labeled datasets of benign and malicious traffic (such as the CICIDS or UNSW-NB15 datasets), classifiers like random forests or gradient-boosted trees can achieve high accuracy on known attack categories.
- **Clustering** — Grouping similar network flows reveals attack campaigns, lateral movement patterns, and coordinated scanning activity.
- **LLM-assisted rule generation** — LLMs like Claude can generate, refine, and explain Snort/Suricata rules from natural language threat descriptions, dramatically accelerating rule development.

Python for Alert Parsing and Analysis

Suricata's EVE JSON logs are straightforward to parse with Python. The following script reads alerts, groups them by signature, and uses Claude to prioritize the findings:

```
import json
from collections import Counter
import anthropic

def parse_eve_alerts(eve_path: str) -> list[dict]:
    """Parse Suricata EVE JSON log and extract alerts."""
    alerts = []
    with open(eve_path) as f:
        for line in f:
            event = json.loads(line)
            if event.get("event_type") == "alert":
                alerts.append({
                    "timestamp": event["timestamp"],
                    "src_ip": event.get("src_ip"),
                    "dest_ip": event.get("dest_ip"),
                    "src_port": event.get("src_port"),
                    "dest_port": event.get("dest_port"),
                    "proto": event.get("proto"),
                    "signature": event["alert"]["signature"],
                    "severity": event["alert"].get("severity", 3),
                    "category": event["alert"].get("category", "Unknown"),
                })
    return alerts
```

```
def summarize_alerts(alerts: list[dict]) -> dict:
```

Alert Triage and False Positive Management

Raw IDS alerts are noisy. A typical Suricata deployment on a medium-sized network can generate thousands of alerts per day, and the vast majority are false positives or low-priority informational events. Effective triage requires a structured approach:

1. **Severity filtering** — Focus first on severity 1 (high) alerts. Severity 2 and 3 alerts are reviewed in bulk or during scheduled threat hunts.
2. **Contextual enrichment** — Correlate alerts with asset inventories, vulnerability scan results, and threat intelligence feeds. An alert for an Apache exploit against a server running Nginx is a false positive.
3. **Suppression rules** — For known false positives that recur, add Suricata suppression directives rather than deleting the rule entirely. This preserves coverage while reducing noise.
4. **Threshold tuning** — Adjust `threshold` and `rate_filter` settings so that low-severity, high-volume signatures generate summaries rather than individual alerts.
5. **AI-assisted triage** — Use LLMs to explain unfamiliar signatures, correlate alert clusters, and recommend investigation priorities.

Using Claude for Detection Engineering

Claude can accelerate several detection engineering workflows. You can describe a threat scenario in natural language and ask Claude to generate a Suricata rule, explain what an existing rule does, suggest improvements for coverage gaps, or analyze a PCAP summary to identify indicators of compromise. For example:

```
import anthropic

client = anthropic.Anthropic()

# Generate a detection rule from a threat description
message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[{
```

```

        "role": "user",
        "content": (
            "Write a Suricata rule to detect potential C2 beaconing behavior:  

            "an internal host making HTTP GET requests to the same external IP  

            "at regular intervals (every 60 seconds +/- 5 seconds), with small  

            "response sizes under 1KB. Include comments explaining each option  

        )
    }
}

print(message.content[0].text)

```

Key Insight: AI-generated rules should always be tested against representative traffic captures before deployment. Use tools like `suricata -r capture.pcap` to replay PCAPs against new rules and verify both detection accuracy and false positive rates. LLMs can produce syntactically correct rules that are logically flawed — human review remains essential.

Key Takeaways

Network intrusion detection is the frontline of blue team defense. Combining signature-based precision with AI-powered anomaly detection creates a layered defense that catches both known and novel threats.

- IDS monitors and alerts; IPS monitors, alerts, and blocks. Start with IDS to learn your traffic before enabling inline prevention.
- Signature-based detection handles known threats; anomaly-based detection catches the unknown. Layer both approaches.
- Snort, Suricata, and Zeek each serve different roles: Snort/Suricata for rule-based alerting, Zeek for deep metadata analysis and threat hunting.
- Master Snort/Suricata rule syntax — it is the lingua franca of network detection engineering.
- Machine learning models excel at learning traffic baselines and flagging deviations that rules cannot anticipate.

- Alert triage is as important as detection. Without effective false positive management, critical alerts drown in noise.
- Use Claude to generate rules, explain alerts, triage findings, and accelerate detection engineering workflows.

Self-check: Can you explain the difference between IDS and IPS deployment modes? Can you write a basic Snort/Suricata rule from scratch? Can you describe how an isolation forest would detect anomalous network traffic? Can you outline a strategy for managing false positives in a production Suricata deployment?

Download Files

These files contain synthetic network data for the IDS challenges. All files are plain text (.txt) and safe to open.

↓ [snort-alerts.txt](#) ~13 KB

↓ [pcap-summary.txt](#) ~15 KB

↓ [network-baseline.txt](#) ~1 KB

↓ [anomaly-traffic.txt](#) ~1 KB