

Network Intrusion Detection with AI

From signature matching to AI-powered anomaly detection — building a modern blue team detection pipeline.

⚠ Educational Use Only

Network detection techniques are for defensive security training and CTF competitions. Deploy IDS/IPS only on networks you are authorized to monitor.

IDS/IPS

Short

Suricata

Zeek

AI Detection

CTF Blue Team

Today's Agenda

1. **IDS/IPS Fundamentals** — Signature vs. anomaly-based detection
 2. **Snort & Suricata Rule Writing** — with AI assistance
 3. **Network Baseline & Deviation Detection**
 4. **AI-Powered Traffic Classification** — normal vs. malicious
 5. **Detecting Common Attacks** — port scans, brute force, DDoS, C2, DNS tunneling
 6. **Zeek Log Analysis with AI**
 7. **Detection Playbooks** — building with AI
 8. **False Positive Management & Alert Tuning**
 9. **CTF Challenge** — write rules to catch attack patterns
-

What Is an Intrusion Detection System?

An IDS monitors network traffic (or host activity) and generates alerts when suspicious behavior is detected. An IPS adds the ability to **block** traffic inline.

IDS (Detection)

Passive monitoring. Copies traffic via SPAN port or TAP. Alerts but does not block.

IPS (Prevention)

Inline deployment. Inspects and can **drop** packets in real time. Higher risk of false-positive disruption.

Notes: Most modern tools (Suricata, Snort 3) can operate in either IDS or IPS mode depending on deployment.

Signature-Based Detection

Matches packets against a database of **known attack patterns**. Fast and precise for known threats, but blind to novel attacks.

- **How it works:** Each rule describes specific byte patterns, protocol fields, or behavioral sequences
- **Strengths:** Low false-positive rate, easy to understand, fast at wire speed
- **Weaknesses:** Cannot detect zero-day attacks, requires constant rule updates, evasion via encoding/fragmentation

```
# Example: Snort detecting a known exploit signature
alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (
  msg:"EXPLOIT MS17-010 EternalBlue Attempt";
  content:"|ff|SMB|72 00 00 00 00|";
  offset:4; depth:9;
  sid:2024001; rev:1;
)
```

Anomaly-Based Detection

Builds a model of "**normal**" **behavior** and alerts when traffic deviates from it. Can detect novel threats but generates more false positives.

- **Statistical models:** Track mean/stddev of packet rates, connection durations, payload sizes
- **Machine learning models:** Isolation Forest, autoencoders, clustering on netflow features
- **Behavioral baselines:** Time-of-day patterns, protocol distributions, DNS query frequencies

Where AI Shines

AI excels at anomaly detection because it can learn complex, multi-dimensional baselines that static thresholds miss — a subtle shift in packet timing combined with unusual destination ports that no single rule would catch.

Signature vs. Anomaly: Side by Side

Attribute	Signature-Based	Anomaly-Based
Known threats	Excellent	Good (if modeled)
Zero-day / novel attacks	None	Good
False positive rate	Low	Higher
Setup effort	Rule subscription	Baseline training period
Evasion resistance	Encoding/fragmentation	Mimicry attacks
AI enhancement	Auto-generate rules	Core strength

Best practice: Use both together — signatures for known threats, anomaly detection for the unknown.

The IDS/IPS Tooling Landscape

Tool	Type	Key Strength
Snort 3	Signature IDS/IPS	Industry standard, massive rule ecosystem
Suricata	Signature IDS/IPS	Multi-threaded, protocol-aware, EVE JSON logs
Zeek (Bro)	Network monitor	Rich structured logs, scripting engine for custom analytics
OSSEC / Wazuh	HIDS	Host-based, file integrity, log analysis
Security Onion	Platform	Integrates Suricata + Zeek + ELK + OSSEC

Notes: In CTFs, you will most often encounter Snort/Suricata rules and Zeek logs in PCAP analysis challenges.

Anatomy of a Snort Rule

Every Snort rule has a **header** (action, protocol, IPs, ports, direction) and **options** (detection logic, metadata).

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

alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (
  msg:"WEB-ATTACKS SQL Injection attempt";
  flow:to_server,established;
  content:"UNION"; nocase;
  content:"SELECT"; nocase; distance:0;
  pcre:"/UNION\s+(ALL\s+)?SELECT/i";
  classtype:web-application-attack;
  sid:2024010; rev:1;
)
```

- **alert** — action (alert, log, pass, drop, reject)
- **flow:to_server,established** — only matches established TCP sessions
- **content** — byte/string matching; **nocase** for case-insensitive

- `pcre` — Perl-compatible regex for complex patterns
-

Key Snort Rule Options

Option	Purpose	Example
<code>content</code>	Match byte sequence	<code>content:" ff SMB";</code>
<code>offset / depth</code>	Limit search range	<code>offset:0; depth:4;</code>
<code>distance / within</code>	Relative to last match	<code>distance:0; within:50;</code>
<code>pcre</code>	Regex matching	<code>pcre:"/cmd\.exe/i";</code>
<code>threshold</code>	Rate limiting	<code>threshold:type both, track by_src, count 5, seconds 60;</code>
<code>flowbits</code>	Stateful matching	<code>flowbits:set,login_attempt;</code>
<code>byte_test</code>	Numeric comparison	<code>byte_test:2,>,1024,0;</code>

AI-Assisted Snort Rule Writing

Use an LLM to generate, validate, and optimize detection rules from natural language descriptions or raw packet data.

```
import anthropic

client = anthropic.Anthropic()

prompt = """Write a Snort 3 rule that detects:
- Outbound DNS queries to TXT records larger than 200 bytes
- Only on port 53 UDP traffic
- This pattern is consistent with DNS tunneling/exfiltration
- Include a threshold of 10 queries in 60 seconds from same source"""

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[{"role": "user", "content": prompt}]
)
```

AI Rule-Writing Workflow

1. Describe the attack behavior in plain English. 2. AI generates candidate rule. 3. Test against known-good and known-bad PCAPs. 4. Tune thresholds based on false positive rate.

Suricata Rule Examples

Suricata uses the same rule syntax as Snort with additional keywords for protocol-aware detection.

```
# Detect SSH brute force (>10 failed attempts in 60s)
alert ssh $EXTERNAL_NET any -> $HOME_NET 22 (
  msg:"ET SCAN SSH Brute Force Attempt";
  flow:to_server,established;
  app-layer-event:ssh.banner_missing;
  threshold:type both, track by_src, count 10, seconds 60;
  classtype:attempted-admin;
  sid:2024020; rev:1;
)

# Detect TLS connection to known-bad JA3 hash (Cobalt Strike)
alert tls $HOME_NET any -> $EXTERNAL_NET any (
  msg:"ET MALWARE Cobalt Strike C2 JA3 Hash";
  ja3.hash; content:"72a589da586844d7f0818ce684948eea";
  classtype:trojan-activity;
```

Notes: Suricata's `ja3.hash` keyword lets you fingerprint TLS clients — extremely useful for detecting C2 beacons.

Establishing a Network Baseline

Before detecting anomalies, you need to know what "**normal**" looks like. Capture baseline metrics over 1-4 weeks of typical operations.

Key Baseline Metrics

- **Traffic volume:** Bytes/packets per second by protocol (TCP, UDP, ICMP)
- **Connection patterns:** Average concurrent connections, session durations
- **Protocol distribution:** HTTP 45%, HTTPS 30%, DNS 10%, SSH 5%, Other 10%
- **Time-of-day patterns:** Peak hours, off-hours expected traffic levels
- **DNS behavior:** Query rate, top queried domains, average response sizes
- **Endpoint behavior:** Typical destinations per host, ports used

```
# Example: Netflow baseline statistics
baseline = {
    "avg_packets_per_sec": 12450,
    "stddev_packets_per_sec": 2100,
    "avg_dns_queries_per_min": 340,
    "avg_unique_dst_ips_per_host": 45,
    "avg_session_duration_sec": 28.5,
```

```
"protocol_distribution": {  
  "TCP": 0.72, "UDP": 0.22, "ICMP": 0.03, "Other": 0.03  
}  
}
```

Deviation Detection with AI

Once a baseline exists, use statistical and ML methods to flag deviations.

```
import numpy as np
from sklearn.ensemble import IsolationForest

# Netflow features: [packets/s, bytes/s, unique_dst, avg_pkt_size, dns_queries/min]
training_data = np.array([
    [12400, 8500000, 42, 684, 335],
    [12800, 8700000, 47, 680, 350],
    [11900, 8200000, 40, 690, 320],
    # ... weeks of normal traffic ...
])

model = IsolationForest(contamination=0.01, random_state=42)
model.fit(training_data)

# New observation during potential attack
```

Why Isolation Forest?

It isolates anomalies by randomly splitting features. Anomalous points require fewer splits to isolate, making it efficient and effective for network traffic with many features.

AI-Powered Traffic Classification

Train classifiers to categorize network flows into **normal**, **suspicious**, and **malicious** categories using labeled netflow data.

Feature	Normal	Port Scan	DDoS	C2 Beacon
Packets/flow	50-500	1-3	1000+	5-20
Bytes/packet	200-1400	40-60	40-100	100-300
Flow duration	1-300s	<0.1s	Sustained	Regular interval
Unique dst ports	1-5	100+	1	1-2
Inter-packet gap	Variable	Uniform small	Near zero	Periodic (jitter)

Notes: These features come from standard 5-tuple netflow records (src_ip, dst_ip, src_port, dst_port, protocol) plus derived statistics.

Feature Engineering for Network ML

```
import pandas as pd

def extract_flow_features(netflow_records):
    """Extract ML features from raw netflow data."""
    features = []
    for flow in netflow_records:
        features.append({
            "duration": flow["end_time"] - flow["start_time"],
            "total_packets": flow["packets_fwd"] + flow["packets_bwd"],
            "total_bytes": flow["bytes_fwd"] + flow["bytes_bwd"],
            "avg_pkt_size": flow["total_bytes"] / max(flow["total_packets"], 1),
            "fwd_bwd_ratio": flow["packets_fwd"] / max(flow["packets_bwd"], 1),
            "unique_flags": len(set(flow["tcp_flags"])),
            "dst_port": flow["dst_port"],
            "entropy_payload": calculate_entropy(flow["payload_sample"])
```

Payload entropy is especially useful: encrypted C2 traffic and DNS tunneling payloads have high entropy (6.5-8.0), while normal text is lower (3.5-5.0).

Detecting Port Scans

Port scans probe many ports on a target to discover open services. They produce distinctive traffic patterns.

SYN Scan Signature

SYN packets to many ports, no completed handshakes. Fast, stealthy.

Connect Scan Signature

Full TCP handshake then immediate RST. Slower, more visible.

```
# Snort: Detect SYN scan (>20 ports in 5 seconds)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (
  msg:"SCAN nmap SYN scan detected";
  flags:S,12;
  threshold:type both, track by_src, count 20, seconds 5;
  classtype:attempted-recon;
  sid:2024030; rev:1;
)

# Snort: Detect XMAS scan (FIN+PSH+URG flags)
alert tcp $EXTERNAL_NET any -> $HOME_NET any (
  msg:"SCAN nmap XMAS scan detected";
```

```
flags:FPU,12;  
classtype:attempted-recon;  
sid:2024031 rev:1.
```

Detecting Brute Force Attacks

Brute force attacks send many authentication attempts in rapid succession. Detect by rate-limiting per source IP.

```
# SSH brute force: >5 connection attempts in 30 seconds
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (
  msg:"BRUTE-FORCE SSH login attempt";
  flow:to_server,established;
  content:"SSH-";
  threshold:type both, track by_src, count 5, seconds 30;
  sid:2024040; rev:1;
)

# HTTP login brute force (POST to /login)
alert http $EXTERNAL_NET any -> $HOME_NET any (
  msg:"BRUTE-FORCE HTTP login attempt";
  flow:to_server,established;
  http.method; content:"POST";
  http.uri; content:"/login";
```

Detecting DDoS Attacks

Distributed Denial of Service attacks flood targets with traffic from many sources. Detection relies on traffic volume anomalies and pattern analysis.

```
# SYN flood detection
alert tcp any any -> $HOME_NET any (
  msg:"DDOS SYN Flood detected";
  flags:S,12;
  threshold:type both, track by_dst, count 500, seconds 10;
  classtype:attempted-dos;
  sid:2024050; rev:1;
)

# DNS amplification (large DNS responses from external)
alert udp $EXTERNAL_NET 53 -> $HOME_NET any (
  msg:"DDOS DNS Amplification Attack";
  dsize:>512;
  threshold:type both, track by_dst, count 100, seconds 10;
  classtype:attempted-dos;
```

AI Enhancement: Multi-Source Correlation

```
# Detect distributed attacks by correlating source diversity
def detect_ddos(flows, window_sec=60):
    unique_sources = flows.groupby("dst_ip")["src_ip"].nunique()
    total_packets = flows.groupby("dst_ip")["packets"].sum()
    suspects = unique_sources[
        (unique_sources > 100) & (total_packets > 50000)
    ]
    return suspects # IPs under likely DDoS
```

Detecting Command & Control (C2) Traffic

C2 beacons phone home at regular intervals. The hallmark is **periodic connections** with consistent timing and small payloads.

```
# Detect potential C2 beacon: regular HTTPS to single host
alert tls $HOME_NET any -> $EXTERNAL_NET 443 (
  msg:"MALWARE Potential C2 Beacon - Regular Interval";
  flow:to_server,established;
  threshold:type both, track by_src, count 30, seconds 3600;
  classtype:trojan-activity;
  sid:2024060; rev:1;
)
```

AI-Based Beacon Detection

```
import numpy as np

def detect_beacon(connection_times, jitter_threshold=0.15):
    """Detect C2 beacons by analyzing connection periodicity."""
    intervals = np.diff(connection_times)
```

```
if len(intervals) < 5:
    return False, {}

mean_interval = np.mean(intervals)
std_interval  = np.std(intervals)
jitter_ratio  = std_interval / mean_interval if mean_interval > 0 else 1

is_beacon = (
    jitter_ratio < jitter_threshold # Low jitter = regular timing
```

Detecting DNS Tunneling

DNS tunneling encodes data in DNS queries/responses (often TXT or CNAME records) to exfiltrate data or establish covert channels.

Indicators of DNS Tunneling

- **Long subdomain labels:** `aGVsbG8gd29ybGQ.evil.com` (base64-encoded data)
- **High query rate** to a single domain
- **Unusual record types:** TXT, NULL, CNAME with large responses
- **High entropy** in subdomain strings

```
# Snort: Detect long DNS subdomain (>50 chars before first dot)
alert udp $HOME_NET any -> any 53 (
  msg:"EXFILTRATION DNS Tunneling - Long Subdomain";
  content:"|01 00|"; # Standard query
  byte_test:1,>,50,12; # First label length > 50
  threshold:type both, track by_src, count 5, seconds 60;
  classtype:policy-violation;
  sid:2024070; rev:1;
)
```

```
# Detect high-frequency DNS to single domain
alert udp $HOME_NET any -> any 53 (
  msg:"EXFILTRATION DNS Tunneling - High Query Rate";
  threshold:type both, track by_src, count 100, seconds 60;
```

DNS Tunneling: AI-Powered Detection

```
import math
from collections import Counter

def calculate_entropy(domain_label):
    """Shannon entropy of a string. High entropy suggests encoded data."""
    if not domain_label:
        return 0
    freq = Counter(domain_label)
    length = len(domain_label)
    return -sum(
        (count/length) * math.log2(count/length)
        for count in freq.values()
    )

def analyze_dns_query(query_name):
```

Packet Header Analysis

Understanding packet headers is essential for writing detection rules and analyzing captures.

TCP Packet Header (typical SYN scan packet):

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Source Port: 54321      |      Dest Port: 80      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               | Sequence Number: 0x1a2b3c4d |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               | Acknowledgment Number: 0x00000000 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Offset|Rsvd|C|E|U|A|P|R|S|F|      Window: 1024      |
|   5   | 000 |0|0|0|0|0|0|1|0|      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               | Checksum: 0xab2         |
|                               | Urgent Ptr: 0           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Flags: S = SYN only (no ACK) --> This is a scan probe

Red Flags in Packet Headers

SYN-only with no follow-up ACK = scan. **FIN+PSH+URG** = XMAS scan. **No flags** = NULL scan.

TTL inconsistencies across packets from same source = OS spoofing or proxy chain.

Netflow Data for Detection

Netflow records summarize connection metadata without capturing full payloads — ideal for large-scale traffic analysis.

```
# Sample netflow records (CSV format)
timestamp,src_ip,dst_ip,src_port,dst_port,proto,packets,bytes,duration,flags
2024-01-15 02:14:01,10.0.1.50,203.0.113.10,443,52341,TCP,145,98400,120.5,APSF
2024-01-15 02:14:03,10.0.1.50,203.0.113.10,443,52342,TCP,142,97200,119.8,APSF
2024-01-15 02:16:02,10.0.1.50,203.0.113.10,443,52350,TCP,148,99100,121.2,APSF
2024-01-15 02:18:01,10.0.1.50,203.0.113.10,443,52358,TCP,140,96800,118.9,APSF
                                     ^^^^^^^^^
                                     Regular ~120s intervals = C2 beacon!

# Compare with normal HTTPS browsing:
2024-01-15 02:14:01,10.0.1.22,142.250.80.46,443,61234,TCP,45,32000,5.2,APSF
2024-01-15 02:14:08,10.0.1.22,151.101.1.69,443,61240,TCP,120,185000,12.1,APSF
2024-01-15 02:17:45,10.0.1.22,104.16.132.229,443,61255,TCP,30,22000,3.8,APSF
                                     Variable destinations, variable timing = normal browsing
```

Zeek (Bro) Log Analysis

Zeek generates rich, structured logs from network traffic. Each log type captures different aspects of network activity.

Log File	Contains	Detection Use
<code>conn.log</code>	All connections	Baseline, scan detection, C2
<code>dns.log</code>	DNS queries/responses	Tunneling, DGA domains
<code>http.log</code>	HTTP requests	Web attacks, data exfil
<code>ssl.log</code>	TLS handshakes	C2 certs, expired certs
<code>files.log</code>	File transfers	Malware delivery
<code>notice.log</code>	Zeek-generated alerts	Built-in detections
<code>weird.log</code>	Protocol anomalies	Evasion attempts

Analyzing Zeek conn.log

```
# Zeek conn.log fields (tab-separated)
#fields ts      uid          id.orig_h  id.orig_p  id.resp_h  id.resp_p  proto  service  duration
1705276441.123  CYz1234abc  10.0.1.50  52341      203.0.113.10  443        tcp    ssl      120.50
1705276561.456  CYz1234def  10.0.1.50  52342      203.0.113.10  443        tcp    ssl      119.80
1705276681.789  CYz1234ghi  10.0.1.50  52350      203.0.113.10  443        tcp    ssl      121.20

# conn_state values:
# SF  = Normal established and finished
# S0  = SYN sent, no reply (scan)
# REJ = Connection rejected
# S1  = Connection established, not terminated
# OTH = No SYN (mid-stream pickup)
```

Quick Scan Detection with conn.log

Count states per source IP. Any host with >50 connections in a short window is almost certainly scanning.

AI-Powered Zeek Log Analysis

```
import anthropic

client = anthropic.Anthropic()

zeek_logs = """
# conn.log excerpt (suspicious host 10.0.1.50)
1705276441.123 10.0.1.50 52341 203.0.113.10 443 tcp ssl 120.50 4200 94200 SF
1705276561.456 10.0.1.50 52342 203.0.113.10 443 tcp ssl 119.80 4100 93100 SF
1705276681.789 10.0.1.50 52350 203.0.113.10 443 tcp ssl 121.20 4300 94800 SF
1705276801.012 10.0.1.50 52358 203.0.113.10 443 tcp ssl 118.90 4050 92900 SF
# dns.log excerpt
1705276440.100 10.0.1.50 52000 10.0.0.1 53 udp dns 0.002 update.legit-software.com A
1705276560.200 10.0.1.50 52001 10.0.0.1 53 udp dns 0.001 update.legit-software.com A
"""
```

Zeek dns.log: Tunneling Detection

```
# Normal DNS queries in dns.log
1705276440.100 10.0.1.22 10.0.0.1 53 www.google.com      A      NOERROR 142.250.80.46
1705276440.200 10.0.1.22 10.0.0.1 53 cdn.jsdelivr.net    AAAA   NOERROR 2606:4700::6810:...
1705276441.500 10.0.1.23 10.0.0.1 53 api.github.com     A      NOERROR 140.82.121.5

# DNS tunneling in dns.log (note: long encoded subdomains, TXT records)
1705276440.100 10.0.1.50 10.0.0.1 53 aGVsbG8gd29ybGQ.t.evil.com      TXT   NOERROR
1705276440.150 10.0.1.50 10.0.0.1 53 dGhpcyBpcyBleGZpbA.t.evil.com   TXT   NOERROR
1705276440.200 10.0.1.50 10.0.0.1 53 dHJhdGVkIGRhGEgZnJvbQ.t.evil.com TXT   NOERROR
1705276440.250 10.0.1.50 10.0.0.1 53 dGhlIGNvbXB5b21pc2VkIGhvc3Q.t.evil.com TXT   NOERROR
```

Red flags: All queries go to subdomains of `t.evil.com`; all use `TXT` records; subdomain labels are base64-encoded; query rate is 20/second from single host.

Building Detection Playbooks with AI

A detection playbook documents **what** to detect, **how** to detect it, and **what to do** when an alert fires.

```
playbook:
  name: "DNS Tunneling Detection"
  id: PB-NET-005
  mitre_attack: T1071.004 # Application Layer Protocol: DNS
  severity: HIGH

  detection_sources:
    - zeek_dns_log:
        query: |
          dns.log | where query_length > 50
                  | where record_type in ["TXT", "NULL", "CNAME"]
                  | stats count by src_ip, parent_domain
                  | where count > 50
    - suricata_rule: "sid:2024070"
    - ml_model: "dns_tunneling_classifier_v2"
```

Generating Playbooks with AI

```
import anthropic

client = anthropic.Anthropic()

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=4096,
    messages=[{"role": "user", "content": ""Create a detection playbook for
lateral movement via SMB/PsExec in a Windows network.

Include:
1. MITRE ATT&CK mapping
2. Snort/Suricata rules
3. Zeek log queries
4. Windows Event Log correlation
```

Playbook Library

Build a library of playbooks for each MITRE ATT&CK technique relevant to your environment. AI can generate the initial draft; your team validates and tunes based on real alert data.

False Positive Management

High false positive rates cause **alert fatigue** — the number one enemy of effective detection.
Tuning is an ongoing process.

Common Sources of False Positives

Detection Rule	False Positive Source	Tuning Strategy
Port scan detection	Vulnerability scanners (Nessus, Qualys)	Whitelist scanner IPs
DNS tunneling	CDN subdomains, DKIM records	Allowlist known CDN domains
C2 beacon detection	Software update checks, NTP	Exclude known update servers
Brute force alert	Service accounts, monitoring tools	Separate thresholds per source type
DDoS detection	Legitimate traffic spikes (sales events)	Dynamic baselines by time of day

Alert Tuning with AI

```
import anthropic, json

client = anthropic.Anthropic()

alert_data = {
    "rule": "sid:2024070 - DNS Tunneling Long Subdomain",
    "total_alerts_7d": 2847,
    "unique_sources": 12,
    "top_triggering_domains": [
        {"domain": "*.cdn.cloudflare.com", "count": 1200, "known_legitimate": True},
        {"domain": "*.dkim.amazonses.com", "count": 890, "known_legitimate": True},
        {"domain": "*.t.evil-c2.xyz", "count": 450, "known_legitimate": False},
        {"domain": "*.update.corp-tool.internal", "count": 307, "known_legitimate": True}
    ],
    "true_positive_rate": "15.8%"
}
```

Suricata Suppress & Threshold Tuning

```
# /etc/suricata/threshold.config

# Suppress DNS tunneling alerts for known CDN domains
suppress gen_id 1, sig_id 2024070, track by_src, ip 10.0.1.100
suppress gen_id 1, sig_id 2024070, track by_src, ip 10.0.1.101

# Rate-limit noisy scan detection to 1 alert per source per hour
threshold gen_id 1, sig_id 2024030, type limit, \
  track by_src, count 1, seconds 3600

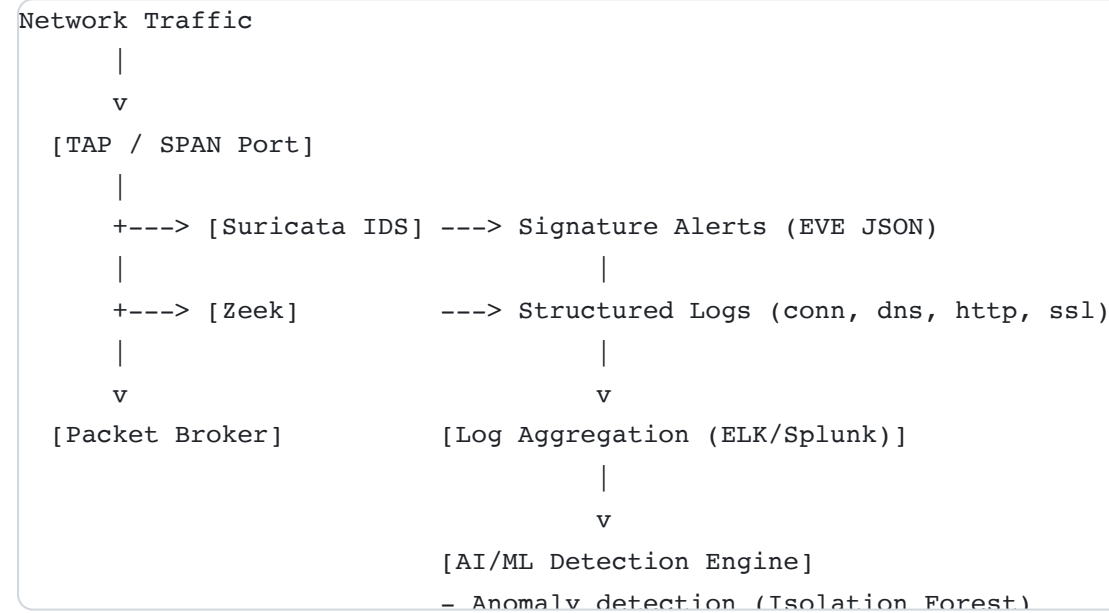
# Raise threshold for brute force on bastion hosts (expected auth volume)
threshold gen_id 1, sig_id 2024040, type both, \
  track by_src, count 50, seconds 60

# Event filter: only alert on SSH brute force if >20 unique users attempted
```

Tuning Best Practices

Never disable a rule entirely. Suppress by known-good IPs or raise thresholds. Review suppression lists monthly. Track true-positive rate per rule over time.

Complete Detection Pipeline



MITRE ATT&CK Coverage Map

Map your detection rules to ATT&CK techniques to identify coverage gaps.

Technique	ID	Our Detection	Coverage
Network Service Scanning	T1046	SYN/XMAS scan rules + conn.log S0 count	Strong
Brute Force	T1110	SSH/FTP/HTTP threshold rules	Strong
Network DoS	T1498	SYN flood + DNS amp rules + volume anomaly	Strong
Application Layer Protocol: DNS	T1071.004	DNS tunneling rules + entropy classifier	Strong
Encrypted Channel (C2)	T1573	JA3 hash + beacon periodicity + ML	Moderate
Proxy: Multi-hop Proxy	T1090.003	TTL analysis + geo-hop detection	Weak
Data Exfiltration over C2	T1041	Byte volume anomaly on C2 channels	Moderate

Notes: Use MITRE ATT&CK Navigator to visualize your detection coverage heat map and prioritize gaps.

CTF Challenge: Network Traffic Detection

Scenario: You are a SOC analyst. A PCAP capture from the last 24 hours contains multiple attack patterns hidden among normal traffic. Write detection rules to catch them all.

Challenge Rules

- You will receive a PCAP file and a set of Zeek logs
- Write Snort/Suricata rules that detect each attack with **zero false positives** on the provided normal traffic
- Each correctly detected attack pattern earns points
- Bonus points for rules that are **general enough** to catch variants

Attack Patterns to Find

1. **SYN scan** from an external host targeting the DMZ
2. **SSH brute force** followed by successful login
3. **DNS tunneling** exfiltrating data to an external domain

4. **C2 beacon** over HTTPS with regular 300-second intervals

5. **DDoS amplification** using DNS reflection

CTF Challenge: Expected Deliverables

```
# Your submission: detection_rules.rules
# Must contain valid Snort/Suricata rules for each attack

# Rule 1: SYN Scan Detection
alert tcp $EXTERNAL_NET any -> $DMZ_NET any (
  msg:"CTF-02 SYN Scan Detected";
  flags:S,12;
  # YOUR LOGIC HERE: threshold, tracking
  sid:9000001; rev:1;
)

# Rule 2: SSH Brute Force
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (
  msg:"CTF-02 SSH Brute Force";
  # YOUR LOGIC HERE
```

Scoring

Criterion	Points
Each attack pattern detected (x5)	20 pts each

Criterion	Points
Zero false positives on normal traffic	25 pts bonus
Generalized rules (catch variants)	15 pts bonus
AI-assisted analysis writeup	10 pts bonus

Using AI to Solve Detection CTFs

```
import anthropic

client = anthropic.Anthropic()

# Step 1: Feed Zeek logs to AI for pattern identification
zeek_excerpt = open("challenge_conn.log").read()[:8000]

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=4096,
    messages=[{"role": "user", "content": f"""You are a SOC analyst in a CTF.
Analyze these Zeek conn.log entries and identify all suspicious patterns.
For each pattern, specify:
1. Type of attack
2. Source and destination IPs involved"""}]
```

Pro Tip

Feed the AI both suspicious and normal traffic samples. Ask it to explain **why** each rule will not fire on the normal traffic. This builds confidence in your detection logic.

Key Takeaways

- **Layer your defenses:** Signature detection (Snort/Suricata) catches known threats; anomaly detection (ML models) catches the unknown
- **Baselines are essential:** You cannot detect deviations without knowing what normal looks like
- **Zeek logs are gold:** Structured network metadata enables powerful AI-driven analysis
- **AI accelerates rule writing:** Generate candidate rules from natural language, then validate against real traffic
- **Tuning never stops:** False positives degrade detection; continuously refine thresholds and suppressions
- **Map to ATT&CK:** Know your coverage gaps and prioritize detection development
- **Playbooks save time:** Document detection logic, triage steps, and response actions for every rule category

Notes: Next lesson we will build on these network detection skills with log analysis and SIEM correlation using AI.

Resources & Further Reading

- **Snort 3 Documentation:** snort.org/documents
- **Suricata Rule Writing:** docs.suricata.io/en/latest/rules/
- **Zeek Documentation:** docs.zeek.org
- **Emerging Threats Rules:** rules.emergingthreats.net
- **MITRE ATT&CK:** attack.mitre.org
- **Security Onion:** securityonionsolutions.com
- **CIC-IDS Dataset:** Labeled network intrusion detection dataset for ML training
- **PCAP repositories:** malware-traffic-analysis.net, netresec.com/pcap

⚠ Educational Use Only

Network detection techniques are for defensive security training and CTF competitions. Deploy IDS/IPS only on networks you are authorized to monitor.
