

AI-Powered Log Monitoring & Alerting

Blue Team Lesson 01 — The foundation of defensive security operations: seeing everything, understanding what matters, and responding before damage is done.

What You'll Learn

Log sources, SIEM concepts, AI-enhanced detection, alert pipeline design, and prompt engineering for threat hunting.

CTF Challenge

Identify all malicious activity hidden within noisy production log streams using AI-powered analysis.

Blue Team

Log Analysis

SIEM

AI/LLM

Detection Engineering

Before We Begin

⚠ Educational Use Only

Log monitoring techniques and detection rules are demonstrated for defensive security training and CTF competitions. Always ensure proper authorization before accessing system logs.

All examples in this lesson use synthetic or sanitized log data. In production environments, log access is governed by strict access controls, data retention policies, and privacy regulations (GDPR, HIPAA, SOC 2).

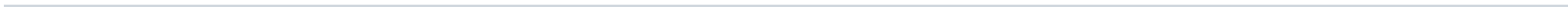
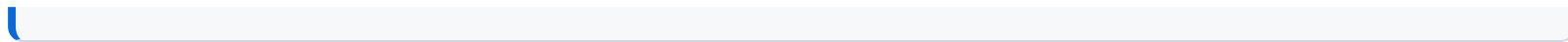
Why Log Monitoring Is the Foundation

Every action on a system leaves a trace. Logs are the single source of truth for what actually happened — not what was supposed to happen.

- **Visibility** — You cannot defend what you cannot see. Logs illuminate the dark corners of your infrastructure.
- **Forensics** — After a breach, logs are the primary evidence. Without them, incident response is guesswork.
- **Compliance** — Regulations (PCI-DSS, HIPAA, SOX) mandate log collection, retention, and review.
- **Threat Detection** — Patterns in logs reveal brute force, lateral movement, data exfiltration, and privilege escalation in real time.
- **Accountability** — Logs tie actions to identities, timestamps, and source addresses.

The Blue Team Mantra

"If it's not logged, it didn't happen. If it's logged but nobody reads it, it might as well not have happened."



The Log Monitoring Challenge

Modern enterprises generate **millions of log events per day**. The challenge is not collection — it is making sense of the flood.

Challenge	Scale
Average enterprise log volume	1–10 TB/day
Median time to detect a breach (without monitoring)	197 days
Median time to detect (with active monitoring)	~70 days
Typical false positive rate (rule-based SIEM)	40–60%
SOC analyst alert fatigue threshold	~500 alerts/day

This is exactly where AI changes the game — processing volume that no human team can handle, while learning to distinguish signal from noise.

Log Sources: Linux — Syslog

Syslog is the backbone of Linux/Unix logging. Most security-relevant events flow through

`/var/log/`.

Log File	Contents
<code>/var/log/auth.log</code>	Authentication events — logins, sudo, SSH
<code>/var/log/syslog</code>	General system messages
<code>/var/log/kern.log</code>	Kernel messages — hardware, drivers, firewall
<code>/var/log/apache2/access.log</code>	HTTP requests to web server
<code>/var/log/audit/audit.log</code>	auditd — fine-grained syscall logging

Syslog Format

```
<priority>timestamp hostname process[pid]: message
```

Priority encodes both **facility** (auth, kern, daemon) and **severity** (emerg, alert, crit, err, warning, notice, info, debug).

Real Log: SSH Brute Force in auth.log

This is what an SSH brute force attack actually looks like in the logs:

```
Mar 15 03:14:07 webservr01 sshd[28341]: Failed password for root from 198.51.100.23 port 44821 ssh2
Mar 15 03:14:08 webservr01 sshd[28343]: Failed password for root from 198.51.100.23 port 44822 ssh2
Mar 15 03:14:08 webservr01 sshd[28345]: Failed password for root from 198.51.100.23 port 44823 ssh2
Mar 15 03:14:09 webservr01 sshd[28347]: Failed password for invalid user admin from 198.51.100.23 port 44824 ssh2
Mar 15 03:14:09 webservr01 sshd[28349]: Failed password for invalid user test from 198.51.100.23 port 44825 ssh2
Mar 15 03:14:10 webservr01 sshd[28351]: Failed password for invalid user ubuntu from 198.51.100.23 port 44826 ssh2
Mar 15 03:14:11 webservr01 sshd[28353]: Accepted password for deploy from 198.51.100.23 port 44827 ssh2
Mar 15 03:14:11 webservr01 sshd[28353]: pam_unix(sshd:session): session opened for user deploy
```

Key Indicators

Rapid succession of failures from the same IP, **username enumeration** (root, admin, test, ubuntu), followed by a **successful login** — the attacker found valid credentials for `deploy`.

Log Sources: Windows Event Log

Windows uses a structured binary event log system accessible via Event Viewer or `wevtutil`. Security-relevant logs live in dedicated channels.

Event ID	Channel	Meaning
4624	Security	Successful logon
4625	Security	Failed logon attempt
4648	Security	Logon using explicit credentials (pass-the-hash)
4672	Security	Special privileges assigned (admin logon)
4688	Security	New process created
4720	Security	User account created
7045	System	New service installed
1102	Security	Audit log cleared (anti-forensics!)

Notes: Event ID 1102 (log clearing) is one of the strongest indicators of compromise. An attacker who clears logs is actively covering tracks.

Real Log: Windows Failed & Successful Logon

A sequence of **Event 4625** followed by **Event 4624** from the same source is a classic brute-force-then-compromise pattern:

```
<!-- Event 4625: Failed Logon -->
<Event>
  <System>
    <EventID>4625</EventID>
    <TimeCreated SystemTime="2025-03-15T03:14:07.123Z" />
  </System>
  <EventData>
    <Data Name="TargetUserName">Administrator</Data>
    <Data Name="LogonType">10</Data>
    <Data Name="IpAddress">203.0.113.45</Data>
    <Data Name="FailureReason">%%2313</Data>
    <Data Name="SubStatus">0xC000006A</Data>
  </EventData>
</Event>
```

LogonType 10 = Remote Desktop (RDP). The attacker failed against `Administrator`, then succeeded with `svc_backup` — a service account often left with weak passwords.

Log Sources: Application & Web Server Logs

Application logs capture business-logic events. Web server access logs are a goldmine for detecting attacks at the HTTP layer.

Apache/Nginx Combined Log Format

```
198.51.100.77 - - [15/Mar/2025:04:22:11 +0000] "GET /admin/login HTTP/1.1" 200 3420 "-" "Mozilla/5.0"
198.51.100.77 - - [15/Mar/2025:04:22:12 +0000] "POST /admin/login HTTP/1.1" 401 182 "-" "python-reques
198.51.100.77 - - [15/Mar/2025:04:22:12 +0000] "POST /admin/login HTTP/1.1" 401 182 "-" "python-reques
198.51.100.77 - - [15/Mar/2025:04:22:13 +0000] "POST /admin/login HTTP/1.1" 200 8841 "-" "python-reques
198.51.100.77 - - [15/Mar/2025:04:22:14 +0000] "GET /admin/users/export?fmt=csv HTTP/1.1" 200 194820 "-
```

Attack Pattern Visible

Automated credential stuffing (python-requests user agent, rapid POST to /admin/login), followed by immediate **data exfiltration** (export of user database). The 200 response on login + large response on export confirms success.

Log Sources: Cloud Audit Trails

Cloud providers generate rich audit logs for every API call. These are critical for detecting misconfigurations, unauthorized access, and data exposure.

AWS CloudTrail

- Every AWS API call recorded
- Key events: `ConsoleLogin`, `AssumeRole`, `PutBucketPolicy`
- Watch for: IAM changes, S3 public access, unusual regions

Azure Activity Log

- Control plane operations
- Key events: role assignments, NSG changes, key vault access
- Watch for: new admin accounts, firewall rule modifications

GCP Audit Logs

- Admin Activity + Data Access logs

SaaS Audit Logs

- Google Workspace, M365, Okta, GitHub

- Key events: IAM policy changes, service account key creation

- Watch for: OAuth app grants, MFA resets, bulk downloads
-

SIEM: The Central Nervous System

Security Information and Event Management (SIEM) platforms collect, normalize, correlate, and alert on logs from across the entire environment.

Three Core SIEM Functions

1. **Aggregation** — Ingest logs from hundreds of sources into a single searchable data store.
Normalize different formats into a common schema.
2. **Correlation** — Link related events across time and systems. Example: failed SSH login on server A + successful login on server B + new service installed on server B = lateral movement chain.
3. **Alerting** — Trigger notifications when patterns match detection rules. Route alerts by severity to the right team with the right context.

Major SIEM Platforms

Splunk (enterprise leader) | **Elastic/ELK Stack** (open-source) | **Microsoft Sentinel** (cloud-native) | **Grafana Loki** (lightweight) | **Chronicle** (Google) | **QRadar** (IBM)

How AI Enhances Each SIEM Function

Function	Traditional Approach	AI-Enhanced Approach
Aggregation	Regex parsers, manual format mapping, grok patterns	LLMs auto-parse unknown log formats, extract fields from unstructured text, normalize schemas
Correlation	Static correlation rules (if X then Y within Z minutes)	ML models learn normal behavior baselines and detect multi-stage attacks that no single rule catches
Alerting	Threshold-based rules, high false-positive rates	AI classifies alert severity, suppresses false positives, enriches alerts with context and recommended actions

The AI Advantage

Traditional SIEM detects **known threats** via signatures. AI detects **unknown threats** via anomaly detection — the behavioral patterns that no rule anticipated.

Using LLMs to Write Detection Rules

Writing effective detection rules requires deep knowledge of log formats, attack techniques, and query languages. LLMs can accelerate this process dramatically.

The Workflow

1. **Describe the threat** in natural language (e.g., "Detect brute force SSH attacks")
2. **Specify the platform** (Splunk SPL, Elastic KQL, Sigma, YARA)
3. **LLM generates the rule** with proper syntax, field mappings, and thresholds
4. **Human reviews and tunes** — adjusts thresholds, adds exclusions, validates against test data

Critical Warning

Never deploy AI-generated detection rules to production without testing. LLMs can produce syntactically valid but logically flawed rules that either miss threats or generate thousands of false positives.

Detection Rule: Sigma Format

Sigma is the universal, platform-agnostic format for detection rules. Write once, convert to Splunk SPL, Elastic KQL, QRadar, and more.

```
title: SSH Brute Force Attempt
id: 5a3b4c2d-1e2f-4a5b-8c7d-9e0f1a2b3c4d
status: experimental
description: Detects multiple failed SSH login attempts from a single source
author: Blue Team AI Assistant
date: 2025/03/15
logsource:
  product: linux
  service: auth
detection:
  selection:
    syslog_message|contains: 'Failed password'
  timeframe: 5m
  condition: selection | count(src_ip) > 10
falsepositives:
```

Notes: The MITRE ATT&CK tag T1110.001 maps to "Brute Force: Password Guessing". Always map detection rules to ATT&CK for consistent taxonomy.

Detection Rule: Splunk SPL

Splunk SPL is the most widely used SIEM query language in enterprise environments.

Brute Force Detection

```
index=linux_auth sourcetype=syslog "Failed password"
| stats count as fail_count dc(user) as users_tried
  values(user) as usernames
  earliest(_time) as first_attempt
  latest(_time) as last_attempt
  by src_ip
| where fail_count > 10 AND users_tried > 3
| eval duration_sec = last_attempt - first_attempt
| where duration_sec < 300
| sort - fail_count
| table src_ip fail_count users_tried usernames duration_sec
```

Detecting Successful Login After Brute Force

```
index=linux_auth sourcetype=syslog
| eval status=if(searchmatch("Failed password"), "fail",
  if(searchmatch("Accepted"), "success", "other"))
```

```
| where status!="other"  
| stats count(eval(status="fail")) as failures  
|       count(eval(status="success")) as successes by src_ip  
| where failures > 10 AND successes > 0  
| table src_ip failures successes
```

Detection Rule: Elastic / ELK KQL

The **Elastic Stack** (Elasticsearch + Kibana) uses KQL for ad-hoc queries and EQL for event correlation.

KQL: Find Failed SSH Logins

```
event.action: "ssh_login" AND event.outcome: "failure" AND source.ip: *
```

EQL: Brute Force Followed by Success

```
sequence by source.ip with maxspan=5m
  [authentication where event.outcome == "failure"] with runs=10
  [authentication where event.outcome == "success"]
```

Elasticsearch Aggregation (Detection Rule)

```
{
  "query": {
    "bool": {
      "must": [
```

```
    { "match": { "event.action": "ssh_login" }},
    { "match": { "event.outcome": "failure" }},
    { "range": { "@timestamp": { "gte": "now-5m" }}}
  ]
}
},
"aggs": {
  "by_source": {
    "terms": { "field": "source.ip", "min_doc_count": 10 }
  }
}
}
```

Detection Rule: YARA for Log Content

YARA rules are traditionally used for malware detection, but they are also powerful for scanning log files for specific patterns and indicators.

```
rule Webshell_Access_In_Logs {
  meta:
    description = "Detects web shell access patterns in web server logs"
    author = "Blue Team CTF"
    severity = "high"
    mitre_attack = "T1505.003"

  strings:
    $shell1 = /GET\s+\[/[^\s]*\.(php|asp|aspx|jsp)\?cmd=/ nocase
    $shell2 = /POST\s+\[/[^\s]*(upload|shell|backdoor|c99|r57)/ nocase
    $shell3 = /eval\(base64_decode\(/ nocase
    $exec1  = /cmd\.exe|\/bin\/(sh|bash)|powershell/ nocase
    $b64cmd = /[A-Za-z0-9+\[/]{40,}={0,2}/

  condition:
```

Run against access logs: `yara webshell_rules.yar /var/log/apache2/access.log`

AI-Powered Anomaly Detection

Rule-based detection catches known patterns. **Anomaly detection** catches what rules cannot — the novel, the subtle, the "never seen before."

What AI Learns as "Normal"

- **Temporal patterns** — Login volumes by hour/day, typical working hours per user
- **Behavioral baselines** — Which users access which systems, from where, using what protocols
- **Volume baselines** — Normal request rates per endpoint, typical data transfer sizes
- **Sequence patterns** — Typical command sequences after login, normal process execution chains

Anomalies That Surface

- User logging in at 3 AM from a country they have never connected from
- Service account suddenly making API calls it has never made before
- DNS query volume spiking 10x for a single internal host (possible DNS exfiltration)

- Database query patterns shifting — new tables accessed, larger result sets
-

Anomaly Detection: Techniques

Technique	Best For	Example
Statistical (Z-score)	Volume-based anomalies	Login count 4 standard deviations above mean
Isolation Forest	Multi-dimensional outliers	Unusual combination of time + source + action
LSTM / Sequence Models	Temporal sequence anomalies	Unusual command sequence after SSH login
Clustering (DBSCAN)	Grouping similar events	Identifying coordinated scanning from a botnet
LLM-based Analysis	Unstructured log understanding	Explaining why a log sequence is suspicious

LLM Unique Advantage

LLMs can **explain** why something is anomalous in plain English — "This login is suspicious because the user has never connected from this ASN before, and the session immediately ran whoami followed by net user, which is consistent with credential compromise reconnaissance."

Real Log: Apache Attack Patterns

Can you spot the attacks hidden in normal traffic?

```
10.0.0.5 - - [15/Mar/2025:10:22:01 +0000] "GET / HTTP/1.1" 200 5124
10.0.0.5 - - [15/Mar/2025:10:22:02 +0000] "GET /style.css HTTP/1.1" 200 1842
192.168.1.100 - - [15/Mar/2025:10:22:03 +0000] "GET /api/users HTTP/1.1" 200 940
198.51.100.44 - - [15/Mar/2025:10:22:05 +0000] "GET ../../../../etc/passwd HTTP/1.1" 400 226
198.51.100.44 - - [15/Mar/2025:10:22:05 +0000] "GET /cgi-bin/../../etc/shadow HTTP/1.1" 403 199
10.0.0.5 - - [15/Mar/2025:10:22:06 +0000] "GET /images/logo.png HTTP/1.1" 200 14320
203.0.113.88 - - [15/Mar/2025:10:22:07 +0000] "GET /search?q=<script>alert(1)</script> HTTP/1.1" 200 3200
203.0.113.88 - - [15/Mar/2025:10:22:08 +0000] "GET /search?q=' OR 1=1-- HTTP/1.1" 200 3200
192.168.1.100 - - [15/Mar/2025:10:22:09 +0000] "POST /api/orders HTTP/1.1" 201 340
198.51.100.44 - - [15/Mar/2025:10:22:10 +0000] "POST /admin/upload.php HTTP/1.1" 200 89
```

Attacks Present

198.51.100.44: Path traversal (`../../../../etc/passwd`), web shell upload (`upload.php`).

203.0.113.88: XSS probe (`<script>alert(1)</script>`), SQL injection (`' OR 1=1--`). Normal traffic from 10.0.0.5 and 192.168.1.100 provides camouflage.

Building Alert Pipelines

Raw detection rules generate too many alerts. An effective pipeline **enriches, deduplicates, classifies, and routes** alerts before they reach an analyst.

The Alert Pipeline

1. **Detection** — Rule fires, raw alert generated
 2. **Enrichment** — Add context: GeoIP, threat intel (VirusTotal, AbuseIPDB), asset inventory, user profile
 3. **Deduplication** — Group 500 "failed login from 198.51.100.23" events into one alert
 4. **AI Classification** — Score alert severity (true positive probability), categorize attack type
 5. **Suppression** — Filter known false positives (e.g., vulnerability scanner IP ranges, expected cron jobs)
 6. **Routing** — Critical: page on-call. High: SOC queue. Medium: daily review. Low: weekly digest.
-

Reducing False Positives with AI

Alert fatigue is the number one reason SOC analysts miss real threats. AI classification can reduce false positive volume by 60–80%.

AI Classification Inputs

- Historical alert outcomes (TP/FP labels)
- Threat intel enrichment data
- Asset criticality scores
- User risk scores
- Time-of-day context
- Related alert clusters

LLM-Powered Triage

- Read raw log context around the alert
- Assess likelihood of true positive
- Generate human-readable summary
- Suggest investigation steps
- Link to similar past incidents
- Draft initial response playbook

Notes: The human analyst always makes the final call. AI triage is a decision-support tool, not a replacement for human judgment.

Prompt Engineering for Log Analysis

Effective prompts for log analysis follow a structured pattern. Here is a template that consistently produces high-quality results:

You are a senior SOC analyst performing threat hunting.

CONTEXT:

- Environment: Production web application (Ubuntu 22.04, Apache 2.4, PostgreSQL 15)
- Time window: Last 24 hours
- Alert trigger: Anomalous outbound traffic volume from web-server-03

TASK:

Analyze the following log entries and:

1. Identify all suspicious or malicious activity
 2. Classify each finding by MITRE ATT&CK technique
 3. Assess severity (Critical / High / Medium / Low)
 4. Recommend immediate response actions
 5. Identify any indicators of compromise (IOCs)
-

Prompt Patterns for Threat Hunting

Pattern 1: Baseline Comparison

"Compare these logs against the normal baseline for this system. Flag anything that deviates by more than 2 standard deviations in frequency, timing, or source."

Pattern 2: Kill Chain Mapping

"Map these events to the Cyber Kill Chain. Identify which stage(s) are represented and whether a complete attack sequence is present."

Pattern 3: IOC Extraction

"Extract all indicators of compromise: IP addresses, domains, file hashes, user agents, URLs, email addresses. Deduplicate and rate confidence."

Pattern 4: Timeline Reconstruction

"Reconstruct a minute-by-minute timeline of attacker activity. Correlate events across auth, web, and system logs."

Pro Tip: Few-Shot Examples

Include 1–2 examples of previous log analysis findings in your prompt. This dramatically improves the quality and consistency of LLM output for security analysis.

Tool Deep Dive: Splunk

Splunk is the enterprise SIEM market leader. Its Search Processing Language (SPL) is the query language every SOC analyst must know.

AI Integration Points

- **Splunk AI Assistant** — Natural language to SPL query conversion
- **MLTK (Machine Learning Toolkit)** — Built-in anomaly detection, clustering, prediction
- **Adaptive Thresholds** — Automatically adjust alert thresholds based on historical patterns
- **Custom LLM integration** — Send alert context to Claude/GPT API for triage recommendations

Example: AI-Assisted Query Generation

```
Prompt: "Show me all successful logins that occurred within  
5 minutes of 10+ failed attempts from the same IP"
```

Generated SPL:

```
index=auth sourcetype=linux_secure  
| transaction src_ip maxspan=5m  
| where eventcount > 10
```

```
| search "Accepted password"  
| table _time src_ip user eventcount duration
```

Tool Deep Dive: ELK Stack

Elasticsearch + Logstash + Kibana — the open-source SIEM alternative powering thousands of SOCs worldwide.

Component	Role	AI Integration
Elasticsearch	Search & storage engine	ML nodes for anomaly detection jobs, inference endpoints for LLM integration
Logstash	Log ingestion & parsing	LLM-assisted grok pattern generation for unknown log formats
Kibana	Visualization & dashboards	AI-powered alert explanations, natural language query bar
Elastic Agent	Endpoint data collection	Local ML models for endpoint anomaly detection

Elastic ML Anomaly Detection

```
{
  "analysis_config": {
    "bucket_span": "15m",
```

```
    "detectors": [{  
      "function": "high_count",  
      "partition_field_name": "source.ip",  
      "over_field_name": "destination.port"  
    }]  
  },  
  "data_description": { "time_field": "@timestamp" }  
}
```

Tool Deep Dive: Grafana Loki

Loki is a lightweight, cost-effective log aggregation system by Grafana Labs. It indexes only metadata (labels), not full text — making it dramatically cheaper to operate at scale.

LogQL: Loki's Query Language

```
# Count failed SSH logins per source IP in the last hour
{job="syslog", host="websvr01"} |= "Failed password"
| pattern `Failed password for` from port `
| count_over_time({src_ip!=""}[1h]) by (src_ip)
| sort desc
```

AI Integration with Grafana

- **Grafana AI/ML** — Built-in anomaly detection on metric and log data
- **Sift** — Automated root cause analysis across logs, metrics, and traces
- **Natural language querying** — "Show me error rates for the payment service in the last 3 hours"
- **Alert annotation** — LLM-generated alert summaries attached to Grafana alerts

Notes: Loki is ideal for CTF environments — lightweight, fast to deploy, and free. Pair with Promtail for log collection and Grafana for visualization.

Comparing the Tools

Feature	Splunk	ELK Stack	Grafana Loki
License	Commercial	Open Source / Commercial	Open Source
Query Language	SPL	KQL / EQL / Lucene	LogQL
Indexing	Full text	Full text (inverted index)	Labels only
Cost at Scale	High (GB/day pricing)	Medium (self-managed infra)	Low (minimal indexing)
Built-in ML	MLTK, Adaptive Thresholds	ML nodes, Anomaly Jobs	Grafana ML plugin
LLM Integration	Splunk AI Assistant	Elastic AI Assistant	Grafana AI features
Best For	Enterprise SOC	Mid-size / flexible	Cloud-native, cost-sensitive

End-to-End: AI Log Monitoring Pipeline

Putting it all together — from raw log to analyst action:

```
+-----+
|  Log Sources  |
| syslog, WinEvent, |
| cloud audit, apps |
+-----+
|
+-----v-----+
|  Collection &  |
|  Normalization  |
| (Logstash/Fluentd) |
+-----+
|
+-----v-----+
|  SIEM / Search  |
| (Splunk/Elastic/ |
```

Building an LLM-Powered Log Analyzer

Here is a minimal Python script that uses Claude to analyze log entries for security threats:

```
import anthropic

client = anthropic.Anthropic()

SYSTEM_PROMPT = """You are a senior SOC analyst. Analyze log entries for:
1. Security threats (classify by MITRE ATT&CK)
2. Anomalous patterns
3. Indicators of compromise (IOCs)
Respond with structured JSON."""

def analyze_logs(log_entries: str) -> str:
    response = client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=2048,
        system=SYSTEM_PROMPT,
```

Scaling Up: Streaming Log Analysis

For production environments, you need to process logs as a continuous stream rather than in batches:

```
import json
import time
from collections import defaultdict

class LogStreamAnalyzer:
    """Lightweight streaming log analyzer with AI escalation."""

    def __init__(self, threshold=10, window_sec=300):
        self.threshold = threshold
        self.window = window_sec
        self.failed_logins = defaultdict(list) # src_ip -> [timestamps]

    def process_line(self, line: str):
        """Process a single log line. Returns alert dict or None."""
        if "Failed password" in line:
```

Log Analysis Anti-Patterns

Common mistakes that undermine your log monitoring program:

Do NOT

- **Send all logs to AI** — Cost-prohibitive and slow. Use rules for known patterns, AI for the ambiguous.
- **Trust AI output blindly** — LLMs hallucinate IOCs, invent log fields, and misclassify events.
- **Ignore log rotation** — Missing logs = missing attacks. Ensure forwarding before rotation.
- **Skip normalization** — Different timestamp formats across sources break correlation.

DO

- **Layer detection** — Rules first (fast, cheap), ML second (pattern-based), LLM third (complex reasoning).
- **Validate AI rules** — Test every generated rule against labeled data before deployment.
- **Centralize time** — NTP sync all systems. Store logs in UTC.
- **Retain strategically** — Hot (7d), warm (30d), cold (1yr) tiers by log criticality.

Detection Engineering Workflow with AI

A mature detection engineering practice uses AI at every stage:

1. **Threat Intelligence Ingestion** — LLM summarizes new threat reports, extracts TTPs and IOCs
2. **Hypothesis Formation** — "If APT29 targets our sector with supply chain attacks, what log evidence would we see?"
3. **Rule Drafting** — LLM generates candidate Sigma/SPL rules from the hypothesis
4. **Backtesting** — Run candidate rules against 30–90 days of historical logs
5. **Tuning** — AI analyzes false positive patterns and suggests threshold adjustments
6. **Deployment** — Rule goes live with a burn-in period (alert-only, no auto-response)
7. **Continuous Review** — AI monitors rule effectiveness metrics (TP rate, FP rate, mean time to detect)

Metrics That Matter

MTTD (Mean Time to Detect) | **MTTR** (Mean Time to Respond) | **TP Rate** (True Positive %) |
Coverage (% of ATT&CK techniques with at least one detection rule)

MITRE ATT&CK Mapping for Log Detection

Every detection rule should map to at least one ATT&CK technique. Here are the techniques most commonly detected through log analysis:

Technique	ID	Log Source
Brute Force	T1110	auth.log, Windows 4625
Valid Accounts	T1078	auth.log, Windows 4624
Remote Services (RDP)	T1021.001	Windows 4624 (Type 10)
Web Shell	T1505.003	Web access logs, process logs
Command & Scripting Interpreter	T1059	Windows 4688, auditd
Indicator Removal (Log Clear)	T1070.001	Windows 1102
Exfiltration Over Web	T1041	Proxy logs, DNS logs

Technique	ID	Log Source
Account Creation	T1136	Windows 4720, auth.log

CTF Challenge: Find the Threats

Scenario

You are a SOC analyst at Acme Corp. Your SIEM has ingested 24 hours of logs from a web server, authentication service, and cloud audit trail. Somewhere in the noise, an attacker has compromised the environment.

Your Mission

1. **Analyze the provided log files** — auth.log, access.log, and cloudtrail.json
2. **Identify all malicious activity** — There are at least 5 distinct attack indicators hidden in the noise
3. **Build a timeline** — Reconstruct the attacker's actions in chronological order
4. **Write detection rules** — Create at least 2 Sigma rules that would have caught the attack
5. **Use AI-assisted analysis** — Craft prompts that help identify threats an analyst might miss

Scoring

- Each IOC identified: **10 points**
 - Complete attack timeline: **20 points**
 - Working detection rule: **15 points each**
 - AI prompt that reveals hidden threat: **10 points**
-

CTF Hint: What to Look For

Hidden in the log data you will find these attack stages:

Initial Access

- Credential stuffing against the web login
- SSH brute force from an unusual ASN

Persistence

- New user account created
- SSH key added to authorized_keys

Lateral Movement

- Pivot from web server to database server
- Credential reuse across services

Exfiltration

- Large data transfer to external IP
- Cloud storage bucket accessed from unknown region

Notes: The full CTF log dataset is provided in the lab materials. Use both manual analysis and AI-powered prompts to find everything.

Key Takeaways

- **Logs are the foundation** — Every defensive operation starts with visibility into what is happening across your environment.
- **Know your log sources** — Linux syslog, Windows Event Log, web server logs, and cloud audit trails each reveal different attack surfaces.
- **SIEM = aggregation + correlation + alerting** — AI enhances all three functions, but does not replace the need for solid architecture.
- **AI writes detection rules faster** — But human review and testing remain essential before production deployment.
- **Anomaly detection catches the unknown** — Layer ML-based anomaly detection on top of rule-based detection for defense in depth.
- **Alert pipelines reduce noise** — Enrich, deduplicate, classify, and route. AI classification cuts false positives by 60–80%.
- **Structured prompts produce better results** — Always provide context, specify output format, and include ATT&CK mapping requirements.

Next Lesson

Lesson 02 dives into **network traffic analysis** — packet captures, flow data, and using AI to detect command-and-control channels hiding in encrypted traffic.
