

Lab Guide

Four exercises across four log types: SSH brute force discovery, web-app reconnaissance, application debug leaks, and Windows event correlation — all AI-assisted.

Authorized Use Only

All log files are **synthetic**. The IPs, usernames, and hostnames are fictional. The patterns you learn to recognize here apply to real engagements only when you have explicit written authorization.

Prerequisites

- Completed the [Theory](#) reading
- Access to an AI assistant (Claude, ChatGPT, or local LLM)
- Downloaded all four files from the [Theory page resources](#) into a folder named `log-lab/`
- Command-line tools: `grep`, `awk`, `cut`, `sort`, `uniq`
- Optional: `python3` with the standard library (no extra packages needed)

Lab Scenario

You are continuing the AcmeCorp engagement from Lesson 01. Your initial access succeeded against a public-facing web server. You have copied off four log files from the compromised host. Your job in this lab is to use those logs to (a) understand what the defenders can already see about you, (b) discover other attackers' activity you can ride on, and (c) find the next system to pivot to.

Download Lab Files

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

↓ `auth-log.txt` ~7 KB

↓ `apache-access.log.txt` ~17 KB

↓ `app-debug.log.txt` ~5 KB

↓ `windows-events.txt` ~8 KB

Exercise 1 — SSH Brute Force & Lateral Movement

Open `auth-log.txt`. This is a Linux `/var/log/auth.log` excerpt covering several days.

Task 1.1 — Manual baseline (5 minutes)

Before you touch the AI, do this manually so you have something to compare against:

```
# How many failed password attempts total?
grep -c "Failed password" auth-log.txt

# Top 5 source IPs of failed attempts
grep "Failed password" auth-log.txt | awk '{print $11}' | sort | uniq -c

# Successful logins
grep "Accepted" auth-log.txt
```

Write down the numbers. You will use them to grade the AI in Task 1.2.

Task 1.2 — AI-assisted attack reconstruction

You are a red team analyst reading SSH auth.log from a compromised Linux web server. Analyze the log and produce:

1. Timeline of attacker activity (start time, end time, duration).
2. Top 5 source IPs ranked by attempt count.
3. For each source IP: list the usernames it tried, the techniques used (single-user spray, user enumeration, brute force), and whether ANY attempt succeeded.
4. Identify any successful logins. For each, list:
 - The user
 - The source IP
 - The authentication method (password vs. publickey)
 - Anything the user did after logging in (sudo, su, etc.)
5. Distinguish between:
 - External brute force (noisy attackers)
 - Legitimate-looking activity that may actually be the red team (publickey logins from internal IPs, sudo to root)
6. Recommend the next pivot: which credentials or trust relationship would a red team exploit from here?

[paste auth-log.txt]

✓ **Self-check:** You should have (a) a list of brute-force IPs and (b) at least one successful login that suggests valid credentials — check whether it matches the attack timeline or looks like legitimate admin activity.

Exercise 2 — Web Server Recon

Open `apache-access.log.txt`. This is an Apache combined-log-format access log.

Task 2.1 — Find the scanners and the real targets

Below is an Apache access log. Analyze it as a red team operator looking for:

1. Reconnaissance scanners (Nikto, Nuclei, dirb, gobuster, sqlmap, etc.). For each: what tool, what source IP, what time window, and what it found.
2. Probing for known vulnerabilities (CVE paths, common admin pages,

- .git/.env exposure, backup files).
- 3. Successful HTTP 200 responses to suspicious paths (these are the actual finds).
- 4. POST requests to authentication endpoints – were any successful?
- 5. Any "interesting" user agents (curl, wget, python-requests, Burp/ZAP) that suggest manual or scripted activity.
- 6. The single most exploitable finding in this log and why.

Output as a structured incident-style report.

[paste apache-access.log.txt]

✓ Self-check: You should be able to point to a specific URL that returned HTTP 200 to a scanner — that is your foothold or your target's existing vulnerability.

Exercise 3 — Application Debug-Log Mining

Open `app-debug.log.txt`. Application debug logs are a goldmine: they contain stack traces, SQL queries, environment dumps, and sometimes credentials.

Task 3.1 — Extract everything sensitive

Below is an application debug log. Extract:

1. Any credentials, API keys, tokens, or session IDs.
2. Any database connection strings or queries that reveal schema.
3. Any internal hostnames, IP addresses, or service URLs.
4. Any exception/stack trace that reveals:
 - Source-code file paths
 - Library or framework versions
 - Specific functions or business logic
5. Any "TODO", "FIXME", "DEBUG", or developer comments that hint at weaknesses.
6. Rank the findings by exploit value (most-useful first).

[paste app-debug.log.txt]

✓ Self-check: Debug logs almost always leak something. You should walk away with at least one credential, one piece of internal infrastructure info, and

one actionable vulnerability lead.

Exercise 4 — Windows Event-Log Correlation

Open `windows-events.txt`. This is a Windows Security/System event-log dump in text form.

Task 4.1 — Identify the attack chain

Below is a Windows event log dump. Identify any attack-chain artifacts and reconstruct what happened. Look for:

1. Event 4624 (successful logon) — note logon type:
2 = interactive, 3 = network, 10 = remote interactive (RDP),
11 = cached creds.
2. Event 4625 (failed logon) — bursts indicate brute force or spraying.
3. Event 4672 (special privileges assigned) — admin sessions.
4. Event 4688 (process creation) — what processes were spawned and by whom?
5. Event 4720/4722/4724 — user account creation, enable, password reset (persistence!).
6. Event 7045 (service installed) — common malware persistence.
7. Event 1102 (audit log cleared) — VERY suspicious.
8. PowerShell event IDs (4103/4104) — script-block content.

Build a chronological attack timeline. Identify:

- Initial access vector
- Persistence mechanism (if any)
- Privilege escalation (if any)
- Lateral movement attempts
- Defense evasion (log clearing, process renaming)

[paste windows-events.txt]

✓ **Self-check:** You should have a chronological narrative that reads like a mini incident-response report — "at 09:14, attacker tried X; at 09:17, they succeeded with Y; at 09:22, they created persistence via Z."

Final Exercise — Cross-Log Correlation

Task 5.1 — Connect the dots across all four logs

Take the outputs of Exercises 1–4 and feed them all back to the AI in a fresh conversation:

Here are four analyses of four log files from the same compromised environment. Identify:

1. Any IP, username, or timestamp that appears in MULTIPLE logs (these are the high-confidence indicators).
2. The most likely full attack chain across all four sources.
3. What the defenders could have detected (and likely will detect when their SOC reviews these logs tomorrow).
4. Three things a red team would do RIGHT NOW to maintain access before detection.
5. Three things a defender should do RIGHT NOW to evict the attacker.

Output: a 2-page incident summary written for both audiences.

✓ **Self-check: The cross-correlation should produce at least one finding that *none* of the individual exercises caught. That is the value of multi-log analysis.**

Reflection

- Which log type gave you the highest density of useful findings per kilobyte of text?
- How well did the AI handle ambiguity (e.g., "is this admin or attacker?")? When did it guess wrong?
- What is the biggest risk of trusting AI log analysis in a real engagement?
- If you only had time to read ONE log type during an active intrusion, which would you pick and why?

What's Next

Lesson 03 moves from text logs to **network traffic** — using AI to read pcaps, NetFlow, and DNS query logs to find C2 channels, data exfiltration, and lateral movement.