

Answers

Reference answers for Lesson 02 lab exercises and homework tasks.

All outputs are non-deterministic. These answers describe the expected *findings*, not exact wording.

Part 1 — Lab Answers

Exercise 1 — SSH Brute Force & Lateral Movement

Expected findings

- **Two distinct attacker profiles:** a noisy external brute-forcer hitting `root` and common usernames from one or two IPs, and a separate quiet session with a valid credential from a different source.
- **The successful login:** usually a non-root account via publickey or password from an internal-looking IP. This is the red team — or a legitimate admin you cannot yet distinguish from one.
- **Post-auth actions:** `sudo` elevation, `su` to root, or scp to another host — indicating intent beyond idle login.
- **Next pivot:** the credentials or trust relationship that the successful session used — often an SSH key that could unlock other hosts.

Exercise 2 — Web Server Recon

Expected findings

- At least two automated scanners identifiable by User-Agent (`Nikto` , `sqlmap` , `python-requests` , `gobuster`).
- A successful HTTP 200 to a path that was probed: most often a backup file (`.zip` , `.bak`), a `.git/` directory, an `.env` , or an admin panel.
- A POST to an authentication endpoint that returned a 302 redirect — the successful credential try.
- One "manual" request (from a browser-like UA) that closely follows a scanner hit — attacker pivoting from automated to manual exploitation.

The exploitable finding

Whichever 200-response was against a path that should not exist in production.

Exercise 3 — Application Debug-Log Mining

Expected findings

- At least one credential, API key, or token (often a JWT or database password) printed to the log.
- A stack trace revealing source paths, framework version, and at least one internal class name.
- A database query that leaks schema (table and column names).
- A `TODO` or `FIXME` comment that hints at an unfixed vulnerability.

Top exploit value

The credential is almost always #1 because it is instantly reusable.

Exercise 4 — Windows Event-Log Correlation

Expected attack chain

1. 4625 burst (failed logons) — brute force or spray begins.

2. 4624 type 3 or type 10 — successful remote logon.
3. 4672 — special privileges assigned (admin context).
4. 4688 — PowerShell or cmd child process spawned.
5. 7045 or 4720 — persistence via service install or user creation.
6. 1102 — audit log cleared (defense evasion).

A good analysis places these events on a unified timeline with timestamps.

Final Exercise — Cross-Log Correlation

Expected cross-source finding

An IP, username, or timestamp cluster that appears in at least two of the four logs. Typical example: the SSH successful-login IP also appears as the source of the Apache 200 on the backdoor path, and the debug log shows an exception around the same timestamp.

Part 2 — Homework Answers

Task 1 — Detection Rule Pack

Expected rules

- **Sigma 1: SSH brute force** — Linux auth.log, detect N failed passwords from one source within T seconds.
- **Sigma 2: Audit log cleared** — Windows Security, event 1102. Near-zero false positives outside legitimate log rotation.
- **Sigma 3: Sensitive data in app logs** — match on `password=`, `api[_]?key=`, `authorization: bearer` in application log streams.

Good rules include a `falsepositives` section that lists realistic benign triggers (e.g., "SRE rotating logs manually").

Task 2 — Blind-Spot Analysis

Typical blind spots

- **DNS tunneling** — no DNS logs in the set → requires a DNS source.
- **In-memory malware** — no EDR telemetry → requires sysmon or EDR.
- **Cloud API abuse** — no cloudtrail / audit log → requires cloud provider audit.
- **Kerberoasting** — no ticket-request events → requires 4769 events + detection tuning.
- **Encrypted C2 over HTTPS** — Apache access log sees the host but not body → requires TLS inspection or NetFlow.

Task 3 — Correlation Script

Expected shape

A ~100-line Python script with three regex pools (IPv4, usernames, ISO timestamps), a dictionary counting which files each token appears in, and a final print of any token with file-count ≥ 2 . The bonus timeline should use simple dict bucketing by hour and print a bar of `#` characters per bucket.