SEAS-8414 Week 03 Student Lab Guide

SEAS-8414 Week 03 Student Lab Guide: Vulnerability Assessment, Correlation, and Prioritization

Start Here: Download the Student ZIP

Before running any lab commands, download the Week 03 student package from the course site:

https://8414.bwater.io/downloads/labs/packages/seas8414-blackboard-week-03-2026.05.0_4e76a52f_aws8414.zip

The recommended path is the package Makefile:

```
unzip seas8414-blackboard-week-03-2026.05.0_4e76a52f_aws8414.zip
cd seas8414-blackboard-week-03-2026.05.0_4e76a52f_aws8414
make week03
```

The Makefile calls `run-week03-lab.sh`, extracts the nested runtime ZIP, starts the lab, runs the Week 03 API workflow, saves evidence under `lab-results/week-03/evidence/`, generates `lab-results/week-03/index.html`, and cleans up containers at exit.

You can also download the runner directly from https://8414.bwater.io/downloads/labs/scripts/run-week03-lab.sh.

Manual extraction uses two ZIP layers. First extract the weekly Blackboard ZIP, then extract the nested runtime ZIP:

```
unzip seas8414-blackboard-week-03-2026.05.0_4e76a52f_aws8414.zip
cd seas8414-blackboard-week-03-2026.05.0_4e76a52f_aws8414
unzip runtime/seas8414-student-lab-2026.05.0+4e76a52f.aws8414.zip
cd seas8414-student-lab-2026.05.0+4e76a52f.aws8414/student-lab
```

Run the instructions in this guide from that `student-lab/` directory. The matching screencast and LLM prompt are published next to the ZIP on the labs page:

- Screencast MP4: https://8414.bwater.io/downloads/labs/screencasts/phase03-lab-screencast.mp4
- LLM Prompt: https://8414.bwater.io/downloads/labs/prompts/phase03-lab-llm-prompt.md
- Run Script: https://8414.bwater.io/downloads/labs/scripts/run-week03-lab.sh

---

Phase 3 Lab: Vulnerability Assessment, Correlation, and Prioritization

Phase 3 turns identity artifacts into vulnerability claims. The goal of this lab is not to run every scanner feature exposed by the platform. The goal is to decide which findings actually apply, how strong the evidence is, where contradiction remains, and what should be prioritized first.

Use the Chapter 3 textbook in this final package as the canonical source for the analytical boundary. This lab stays inside that boundary:

- CPE-to-CVE matching
- CVSS interpretation
- OpenVAS and Nuclei comparison
- default credential assessment
- false-positive triage
- cross-tool correlation
- prioritization under consequence

It does **not** treat exploit development, fuzzing, or protocol-digital-twin work as part of this phase.

Mapping This Lab to the Revised Chapter 3

Section 3.16 of the revised chapter (*Student Lab Workflow*) prescribes the exercise sequence below. This lab implements that sequence one-to-one:

Chapter §3.16 stage	This lab
Pre-commit before reading the output	Exercise 0
Compare broad vs. narrow CPE strategies	Exercise 1
Read CVSS vectors, not just scores	Exercise 2
Compare NVD, OpenVAS, and Nuclei on one host	Exercise 3
Diagnose a weak or empty CVE result	Exercise 4
Successful default credentials as direct evidence	Exercise 5
False-positive triage and validation	Exercise 6
Correlation and convergence without double-counting	Exercise 7

Chapter §3.16 stage	This lab
Build a prioritized remediation queue	Exercise 8

Four reminders before you start:

- **Distinct evidence classes.** NVD lookups, OpenVAS findings, Nuclei findings, default-credential outcomes, and firmware-CVE rules are *different kinds of evidence*. Do not collapse them into one count before the reader can see what each method observed.
- **Applicability, not severity, drives priority.** A high CVSS score is not an automatic first action. Priority depends on whether the finding *applies* to this device, how strong the evidence is, the device's role, and the consequence of abuse.
- **Zero results are ambiguous.** Few or no CVEs may mean the device is hardened, or it may mean the CPE was too weak, the scan was disabled, the network blocked active probes, or the database lacks coverage. The lab rewards naming the uncertainty, not declaring success.
- **Section numbering in the revised chapter.** Numbering skips (no §3.5, §3.11, or §3.14). Where this lab cites a section, follow the *topic*, not the number.

Each exercise output should look like an analyst memo with four explicit layers: tool output, reason the output appeared, evidence strength, and the next safe action. If those four collapse into one sentence, the answer is not ready for Chapter 4.

Learning Objectives

By the end of the lab, you should be able to:

1. Explain why a vulnerability result is an applicability claim, not just a tool output.
2. Compare broad and narrow CPE strategies and explain how they change precision and recall.
3. Interpret CVSS vectors in operational terms rather than relying on the numeric score alone.
4. Explain why NVD, OpenVAS, Nuclei, and credential testing produce different evidence.
5. Diagnose false positives caused by identity weakness, backporting, or active-scan overreach.
6. Build a small prioritized remediation queue that uses more than severity alone.

Working Rule for the Entire Lab

For every exercise, keep four things separate:

1. what the tool reported
2. why the tool reported it
3. what evidence makes the report stronger or weaker here
4. what action, if any, the finding justifies

If those four layers collapse into one sentence, your answer is probably too shallow.

Prerequisites

- Completed Phase 1 and Phase 2 labs
- Running student-lab stack
- `curl` and `jq` available

Re-establish your session variables if needed:

> The lab uses `student@example.com` because pydantic v2 (the API's request-body validator) rejects the `.local` TLD as a special-use name (RFC 6762). Any RFC 2606 example domain works; we standardize on `example.com`.

```
TOKEN=$(curl -s -X POST http://localhost:8100/v1/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"student@example.com","password":"SecurePass!2026"}' \
  | jq -r '.access_token')

SCAN_ID=$(curl -s "http://localhost:8100/v1/scanning/smart-
        scan/history?limit=1" \
  -H "Authorization: Bearer $TOKEN" \
  | jq -r '.scans[0].scan_id')

echo "Token: ${TOKEN:0:20}..."
echo "Scan ID: $SCAN_ID"
```

If `SCAN_ID` is empty or `null`, run a fresh scan and wait until usable identity evidence is visible before continuing. The API may remain in later vulnerability phases while NVD or template checks continue; this lab uses direct probes and explicit CVE lookup calls for the vulnerability evidence you must analyze.

```
if [ -z "$SCAN_ID" ] || [ "$SCAN_ID" = "null" ]; then
  SCAN_ID=$(curl -s -X POST http://localhost:8100/v1/scanning/smart-
        scan \
    -H "Authorization: Bearer $TOKEN" \
    -H "Content-Type: application/json" \
    -d '{"subnet":"172.30.0.0/24"}' \
    | jq -r '.scan_id')
  echo "launched scan $SCAN_ID; waiting for completion..."
  for _i in $(seq 1 120); do
    scan_payload=$(curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
      -H "Authorization: Bearer $TOKEN" || true)
    s=$(printf '%s' "$scan_payload" | jq -r '.status // "?"'
        2>/dev/null || echo "?")
    h=$(printf '%s' "$scan_payload" | jq -r '(.hosts // []) | length'
        2>/dev/null || echo 0)
    ready=$(printf '%s' "$scan_payload" | jq -r '
      any(.hosts[]?;
        .ip == "172.30.0.10" and
        (((.vendor // "") | ascii_downcase | contains("hikvision")) or
        ((.cpes // []) | length > 0))
```

```
      )' 2>/dev/null || echo false)
    case "$s" in completed|failed|cancelled) break ;; esac
    if [ "$ready" = "true" ]; then
      break
    fi
    sleep 5
  done
  echo "scan status: $s ($h hosts visible; identity ready: $ready)"
fi
```

Core Endpoints

You can complete the lab from a small surface:

```
GET  /v1/scanning/smart-scan/{scan_id}/results
POST /v1/cves/lookup
GET  /v1/cves/{cve_id}
GET  /v1/cves/search?keyword={kw}&limit={n}
```

The point is interpretation, not endpoint hunting.

Exercise 0: Pre-Commitment Before Reading the Findings

Purpose: force an initial hypothesis before the richer vulnerability outputs shape your thinking.

```
curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.hosts[] | {
      ip,
      vendor,
      device_type,
      cpes,
      services: [.services[]? | {port, service, product, version}]
    }]'
```

Tasks

1. Pick three hosts that appear to differ in device class or exposure.
2. For each host, state:
   - what vulnerabilities you expect are most likely to appear
   - what evidence would make you distrust those expectations
   - which tool in this phase you expect to be strongest for that host
3. Write your answer before inspecting the host's actual vulnerability outputs.

Deliverable: `Initial Vulnerability Hypotheses`

Exercise 1: Compare Broad and Narrow CPE Strategies

Purpose: see how identity precision and lookup-engine behavior together control CVE precision.

Run a lookup for one device using a broader and then a narrower description.

```
curl -s -X POST http://localhost:8100/v1/cves/lookup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"hosts": [{"ip": "broad-test", "vendor": "Hikvision", "model":
        "",
      "services": [{"port": 80, "product": "httpd", "state":
       "open"}]}]}' \
  | jq '.results[0] | {ip, total_cves, critical_count}'

curl -s -X POST http://localhost:8100/v1/cves/lookup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"hosts": [{"ip": "narrow-test", "vendor": "Hikvision", "model":
        "DS-2CD2032",
      "services": [{"port": 80, "product": "Hikvision httpd",
       "state": "open"}]}]}' \
  | jq '.results[0] | {ip, total_cves, critical_count}'
```

> **Reading the result.** Record both `total_cves` numbers exactly as the API returned them; do not assume a fixed ratio. The lab's `/v1/cves/lookup` performs a **keyword-expanded** NVD search rather than a strict CPE-string match — so adding more vendor / product / model tokens often *widens* the keyword set and can return more CVEs, not fewer. That is engine behavior, not a contradiction. A CPE-strict engine (vendor+product+version with explicit wildcards) would behave the opposite way.

Tasks

1. Record both observed `total_cves` and `critical_count` values. State which input produced more results.
2. Explain why a keyword-expanded engine can show that direction, and what a strict-CPE engine would have done with the same inputs.
3. Decide which engine model you would prefer for triage today, and which for an audit deliverable. They may differ.
4. State what additional evidence (firmware version, device class, vendor advisory) you would want before trusting *either* result as a complete answer.

Deliverable: `CPE Precision-Recall Memo`, 150-250 words. Cite both numbers you observed.

Exercise 2: Read CVSS as Structure, Not Score

Purpose: force a deeper interpretation of vulnerability severity using two canonical CVSS 3.1 vectors that students can decompose without depending on whatever happens to be in the scan results.

Compare these two vectors:

CVE	CVSS 3.1 Vector	Base Score
CVE-2021-36260	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`	9.8 (Critical)
CVE-2017-7921	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N`	7.5 (High)

(These are the two CVEs declared for the Hikvision camera in `student-lab/ground-truth.json` and discussed in Chapter 3 § "default credentials and information disclosure".)

Optionally, pull whatever your live scan attached to the same host for cross-reference:

```
curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.hosts[] | select(.ip == "172.30.0.10") | .cves[]? |
        {cve_id, cvss_score, cvss_vector, severity}] | .[:6]'
```

Tasks

1. Decompose each vector by hand: state what `AV`, `AC`, `PR`, `UI`, `S`, `C`, `I`, `A` mean in operational English.
2. Identify the *exact* metric difference between the two vectors above. (Hint: the third triplet differs.)
3. Despite a higher base score on CVE-2021-36260, name one operational scenario where you would prioritize CVE-2017-7921's information disclosure first, and why.
4. Explain why the same numeric score gap (9.8 vs 7.5) is not a sufficient triage signal on its own.

Deliverable: `CVSS Comparison Table` — three rows (per-vector decomposition + your priority decision).

Exercise 3: Compare Evidence Methods on One Host

Purpose: show that different tools produce different kinds of evidence — even when only a subset of methods is enabled in your stack.

```
curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.hosts[] | {
```

```
    ip,
    vendor,
    cve_total: (.cve_summary.total // 0),
    nuclei_findings: (.nuclei_findings // [] | length),
    openvas_findings: (.openvas_findings // [] | length),
    default_creds: (.default_creds // [])
  }]'
```

> **About the lab stack.** This lab enables NVD CVE lookup and the default-credentials check by default. **OpenVAS** is opt-in (it requires `BREAKWATER_OPENVAS_ENABLED=true` and a running gvmd) and **Nuclei** runs only when `BREAKWATER_NUCLEI_ENABLED=true`. If those fields are empty in your output, that is a *configuration* fact about your stack, not the absence of vulnerabilities. Exercise 11 has you run nuclei manually so you can compare a present-vs-absent template result against the scan-time evidence.

Tasks

1. Pick one host with at least one direct evidence class. The Hikvision at `172.30.0.10` is the running case for default-credential evidence; if scan-attached `cve_total` is `0`, use Exercise 1's `/v1/cves/lookup` output as the NVD-style evidence source and state that these are separate evidence paths.
2. Describe what each available evidence class is contributing.
3. State which classes are *absent* in your stack today, and what their absence means (no signal vs no probe).
4. Identify one place where the present classes converge, and one place where they cover different ground.
5. Explain why "more findings" is not the same as "better evidence."

Deliverable: `Multi-Tool Evidence Comparison`

Exercise 4: Diagnose a Weak or Empty CVE Result

Purpose: teach students not to confuse zero findings with safety.

Find a host with zero or very few CVEs.

```
curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.hosts[] | {
      ip,
      vendor,
      device_type,
      cpes,
      cve_total: (.cve_summary.total // 0),
      services: [.services[]? | {port, service, product, version}]
    }] | sort_by(.cve_total) | .[:6]'
```

Tasks

1. Explain at least two reasons a host can have zero CVEs without being safe.
2. Decide whether the weak result is more likely caused by:
   - sparse exposure
   - weak identity/CPE quality
   - outdated database coverage
   - or a genuinely quiet host
3. State what additional safe check would reduce uncertainty the most.

Deliverable: `Zero-Finding Diagnosis`

Exercise 5: Default Credentials as High-Confidence Evidence

Purpose: contrast high-confidence authentication findings with noisier database matches.

```
curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.hosts[] | {
      ip,
      vendor,
      default_creds: (.default_creds // []),
      cves: (.cves // [] | length)
    }] | map(select(.default_creds | length > 0))'
```

Tasks

1. Pick one host with a successful default credential finding if available.
2. Explain why this class of finding is usually stronger than a broad CPE-driven CVE list.
3. State what ethical and operational boundaries should govern this phase.
4. Explain why a single successful credential can still fail to tell the whole risk story of the host.

Deliverable: `Credential Finding Reflection`

Exercise 6: False Positive Triage

Purpose: turn "possible scanner noise" into explicit analytical categories.

Choose one finding you distrust or would want to verify.

Tasks

1. Assign it to one of three buckets:
   - identity/CPE weakness
   - backport or banner drift
   - active-scan interpretation error

2. Explain your classification.
3. State what evidence would most strengthen the finding.
4. State what evidence would most weaken it.

Deliverable: `False Positive Triage Card`

Exercise 7: Correlation and Convergence

Purpose: distinguish agreement from coincidence.

Build a simple per-host table:

Host	NVD	OpenVAS	Nuclei	Default Creds	Convergence Comment

Tasks

1. Fill the table for three hosts.
2. Mark one host where convergence meaningfully raises confidence.
3. Mark one host where single-tool uniqueness does **not** automatically make the finding weak.
4. Mark one host where contradiction survives and should be preserved in analyst notes.

Deliverable: `Tool Convergence Table`

Exercise 8: Build a Prioritized Remediation Queue

Purpose: force a real detective-to-operational handoff.

Pick five findings from across at least three hosts and rank them from highest to lowest action priority.

Tasks

1. Use all of the following in your ranking:
   - severity
   - asset criticality
   - exposure
   - evidence strength
   - convergence or contradiction
2. Do **not** rank only by CVSS.
3. For each item, write one sentence explaining why it sits where it does.
4. Include one example where a lower-severity issue outranks a higher-severity issue because of consequence or stronger applicability.

Deliverable: `Ranked Remediation Queue`

Hands-On Direct Probing

The previous exercises had you read findings the scanner already produced. The next exercises have you generate vulnerability evidence yourself — the same primitives Chapter 3 relies on (default-credentials probing, nuclei templating, raw exposed-endpoint reading, manual CVSS decomposition, manual queue construction).

The lab's API container is on the simulation network and has `nuclei`, `curl`, and `python3` available; you'll invoke them through `docker exec`.

Exercise 9: Default-Credentials Probe by Hand

Purpose: witness a default-credentials finding the way an analyst would: as a single HTTP exchange, not as a row in a database.

```
docker exec student-lab-api-1 curl -s -o /dev/null -w 'admin:12345 ->
        HTTP %{http_code}\n' \
  -u admin:12345 http://172.30.0.10/
docker exec student-lab-api-1 curl -s -o /dev/null -w 'admin:Hikvision
        -> HTTP %{http_code}\n' \
  -u admin:Hikvision http://172.30.0.10/
docker exec student-lab-api-1 curl -s -o /dev/null -w 'admin:wrong ->
        HTTP %{http_code}\n' \
  -u admin:wrong http://172.30.0.10/
```

Tasks

1. Record the HTTP status code for each credential pair.
2. Decide which response codes constitute a genuine default-credential finding and which do not. (`200` is not the only valid signal; some vendors return `301` to a session URL.)
3. **Classify** the finding under Chapter 3's evidence taxonomy: this is **direct authentication evidence** (a successful logon), not a CVE finding. State why direct-auth evidence is generally stronger than a CPE-driven CVE list, and whether you would *additionally* correlate this success against a published CVE (e.g. for the audit trail) — if so, which one and why.

Deliverable: `Default-Credential Probe Table`, three rows plus a one-sentence analyst verdict that names the evidence class first and the optional CVE correlation second.

Exercise 10: Probe a Known-Vulnerable Endpoint

Purpose: read a vulnerability's evidence directly from the device, not via a CVE feed.

Hikvision DS-2CD2143G2-I exposes its `/system/deviceInfo` endpoint without authentication on the simulator (modeling CVE-2017-7921's information-disclosure surface).

```
docker exec student-lab-api-1 curl -s
        http://172.30.0.10/system/deviceInfo
```

Tasks

1. Identify three pieces of vendor-internal information returned by the unauthenticated endpoint.
2. State why this information disclosure has security consequence beyond curiosity (think: targeted exploit selection, supply-chain footprinting, vendor-disclosed firmware version).
3. Decide whether you would file this finding under "default credentials," "information disclosure," or "both," and defend the choice.

Deliverable: `Information Disclosure Verdict`, one short paragraph.

Exercise 11: Run nuclei Against One Host

Purpose: see what a Chapter 3 templated scanner reports on a target you have already characterized, and assess template-target fit.

```
docker exec student-lab-api-1 timeout 45 nuclei \
  -u http://172.30.0.10 \
  -severity high,critical \
  -silent -j 2>/dev/null | head -20
```

Tasks

1. Record any high/critical findings nuclei produces. If nuclei produces zero findings, that is itself a finding — explain why.
2. Decide whether the absence of findings means "no vulnerability," "no template applicable," or "scanner-target mismatch." Chapter 3 calls these three states materially different.
3. Suggest one template family that *should* exist for this device class and would have applied if it were present.

Deliverable: `nuclei Coverage Note`, three short bullets.

Exercise 12: Decompose a CVSS Vector by Hand

Purpose: read a CVSS 3.x vector as structure rather than as a single number.

Take this vector for a representative finding:

`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`

Tasks

1. Translate each metric label-by-label without consulting an online calculator first. State what each `AV`, `AC`, `PR`, `UI`, `S`, `C`, `I`, `A` value means in operational terms.

2. Estimate the base score before looking it up; record your estimate.
3. Now look up the actual base score and identify which metric had the largest effect on it.
4. Reflect: would a Hikvision camera exposed only to a NAT'd LAN with no inbound port forwarding still warrant the same priority as the vector implies? Justify.

Deliverable: `CVSS Vector Read`, two paragraphs.

Exercise 13: Construct CPEs from Raw Evidence

Purpose: build broad and narrow CPE strategies yourself, then compare the resulting CVE counts.

Pick one host (default: the Hikvision at `172.30.0.10`).

1. From your enrichment evidence, write a **broad** CPE that lists vendor and product class but omits version (∗).
2. Write a **narrow** CPE that includes the firmware version observed.

Then run the lookup with both:

```
curl -s -X POST http://localhost:8100/v1/cves/lookup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"hosts": [{"ip": "broad-test", "vendor": "Hikvision", "model":
        "",
       "services": [{"port": 80, "product": "httpd", "state":
        "open"}]}]}' \
  | jq '.results[0] | {ip, total_cves, critical_count,
      cve_ids: ([.cves[]?.cve_id] | .[:15]),
      has_cve_2021_36260: ([.cves[]?.cve_id] | index("CVE-2021-36260")
        != null)}'

curl -s -X POST http://localhost:8100/v1/cves/lookup \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"hosts": [{"ip": "narrow-test", "vendor": "Hikvision", "model":
        "DS-2CD2143G2-I",
       "services": [{"port": 80, "product": "httpd", "version":
        "5.5.52", "state": "open"}]}]}' \
  | jq '.results[0] | {ip, total_cves, critical_count,
      cve_ids: ([.cves[]?.cve_id] | .[:15]),
      has_cve_2021_36260: ([.cves[]?.cve_id] | index("CVE-2021-36260")
        != null)}'
```

> **Reading the result.** Record both `total_cves` numbers exactly as the API returned them. As discussed in Exercise 1, this lookup engine performs keyword-expanded NVD search, so adding model and version tokens may *widen* the keyword set and return more CVEs than the bare vendor lookup. A strict-CPE engine would behave the opposite way. Use the observed direction as evidence about your engine, not as a contradiction of CPE theory.

Tasks

1. Record both observed `total_cves` and `critical_count` values.
2. State which strategy you would use for triage today, and which for an audit deliverable, and why they may differ.
3. Use `has_cve_2021_36260` and the displayed `cve_ids` sample to check whether the narrow result includes the canonical firmware-relevant CVE. If it is absent, decide whether that is an engine limitation, a query-token limitation, or a real exclusion.

Deliverable: `Broad vs Narrow CPE Trade-off`, one paragraph. Cite both numbers you observed.

Final Reflection

Write 200-300 words answering this question:

Why is vulnerability assessment intellectually harder than simply "running more scanners," especially in IoT and OT environments?

Your answer should touch at least three of the following:

- identity dependence
- applicability uncertainty
- false positives
- multi-tool disagreement
- prioritization under consequence

Cleanup

When you are done with the in-shell variables:

```
unset TOKEN
unset SCAN_ID
```

If you are finished with the lab environment entirely (i.e. not continuing to a later phase), tear the simulation stack down to free resources. **Do not run this if you intend to continue to other phases or homework that depend on the live scan data.**

```
# optional — full teardown including data volumes
( cd student-lab && docker compose down --remove-orphans -v )
```