SEAS-8414 Week 02 Student Lab Guide

SEAS-8414 Week 02 Student Lab Guide: Service Enrichment, Fingerprinting, and Identity Handoff

Start Here: Download the Student ZIP

Before running any lab commands, download the Week 02 student package from the course site:

https://8414.bwater.io/downloads/labs/packages/seas8414-blackboard-week-02-2026.05.0_4e76a52f_aws8414.zip

The recommended path is the package Makefile:

```
unzip seas8414-blackboard-week-02-2026.05.0_4e76a52f_aws8414.zip
cd seas8414-blackboard-week-02-2026.05.0_4e76a52f_aws8414
make week02
```

The Makefile calls `run-week02-lab.sh`, extracts the nested runtime ZIP, starts the lab, runs the Week 02 API workflow, saves evidence under `lab-results/week-02/evidence/`, generates `lab-results/week-02/index.html`, and cleans up containers at exit.

You can also download the runner directly from https://8414.bwater.io/downloads/labs/scripts/run-week02-lab.sh.

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

```
unzip seas8414-blackboard-week-02-2026.05.0_4e76a52f_aws8414.zip
cd seas8414-blackboard-week-02-2026.05.0_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/phase02-lab-screencast.mp4
- LLM Prompt: https://8414.bwater.io/downloads/labs/prompts/phase02-lab-llm-prompt.md
- Run Script: https://8414.bwater.io/downloads/labs/scripts/run-week02-lab.sh

---

Phase 2 Lab: Service Enrichment, Fingerprinting, and Identity Handoff

Phase 2 turns the discovery record from Chapter 1 into an identity artifact that later chapters may consume. The goal of this lab is not to "do vulnerability assessment early." The goal is to inspect how protocol evidence, confidence, provenance, ambiguity, conflict, and admissibility change what we can responsibly claim about a device.

Use the Chapter 2 textbook in this final package for the canonical explanation and identity-output contract.

Mapping This Lab to the Revised Chapter 2

Section 2.12 of the revised chapter (*Student Lab Bridge*) prescribes a four-step lab workflow. Each exercise here implements one or more of those steps:

1. **Authenticate to the local API** — Prerequisites and "The Only API Surface You Need" sections below.
2. **Select a completed scan** — Exercise 0 (pre-commitment) and Exercise 1 (inspect the enrichment record).
3. **Inspect the scan data with compact host views** — Exercises 2–4 (HTTP/TLS/RTSP/ONVIF/SSDP/mDNS comparison and the fingerprint cascade).
4. **Judge and record identity claims with evidence and unresolved uncertainty** — Exercises 5–6 (identity output contract and analyst handoff judgment).

Three reminders before you start:

- **Three layers, every time.** For every host, separate (a) what was directly observed on the wire, (b) what was inferred from those observations, and (c) what remains uncertain. The chapter calls collapsing these "a tidy label" and treats it as a weaker answer than an explicit, bounded identity claim.
- **Adapter silence is data.** Adapter failures, disabled probes, missing libraries, and protocol evidence the host did not return are part of the identity artifact, not lab noise. Record them.
- **Section numbering in the revised chapter.** Numbering skips (no §2.3, §2.10, or §2.11). Where this lab cites a section, follow the *topic*, not the number.

Your answer for each exercise should be defensible enough to hand to Chapter 3 with a CPE candidate, an admissibility decision, and a list of conflicts and ambiguities — not just a vendor and model.

Learning Objectives

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

1. Explain why different enrichment adapters fire for different hosts.
2. Compare stronger and weaker identity evidence across nmap, HTTP, TLS, RTSP, ONVIF, SSDP, and mDNS.
3. Distinguish confidence from provenance, ambiguity, and conflict.
4. Produce an identity output that is explicit about downstream admissibility.
5. Defend an "unknown" or "not yet admissible" result when the evidence does not support a stronger claim.

Working Rule for the Entire Lab

For every exercise, separate three things clearly:

1. what was directly observed
2. what was inferred from those observations
3. what still remains uncertain

If your answer collapses those three layers into one sentence, it is probably too weak.

Prerequisites

- Completed Phase 1 lab and a completed scan already present in the lab environment.
- Docker simulation stack running.
- `curl` and `jq` available in your shell.

If you need to re-establish your session variables:

> 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 after enrichment is ready; Phase 2 only needs the enriched host records.

```
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
```

The Only API Surface You Need

Everything in this lab can be completed from the scan-results endpoint:

`GET /v1/scanning/smart-scan/{scan_id}/results`

The point of the lab is not endpoint hunting. The point is evidence interpretation.

Exercise 0: Pre-Commitment Before Interpretation

Purpose: force yourself to make a bounded hypothesis before the richer evidence is interpreted.

Start by reviewing three candidate hosts from the scan output and writing down your best initial guess from minimal evidence alone.

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

Tasks

1. Choose three hosts that appear to represent different device classes.
2. For each host, write an initial bounded hypothesis using only:
   - IP address
   - OUI vendor hint
   - hostname, if any
   - visible service list
3. For each hypothesis, state:
   - what you think the host most likely is
   - what you definitely do **not** know yet
   - what evidence would most change your mind

Deliverable: `Initial Identity Hypotheses`.

Exercise 1: Inspect the Enrichment Record

Purpose: establish what the scanner actually knows per host after enrichment, without confusing raw evidence with interpreted output.

```
curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
  | jq '.hosts[0] | keys'
```

Now inspect a compact host view:

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

Tasks

1. Pick three hosts that appear to represent different device classes.
2. For each host, record what the scan knows before you interpret it:
   - IP address
   - vendor hint
   - device type
   - open services with product and version, where available
3. State which parts of that record are direct observations and which parts are already interpreted labels.
4. State one field that looks precise but still requires analytical caution.

Deliverable: a three-row table titled `Raw Enrichment Record`.

Exercise 2: Compare Protocol Surfaces on a Web-Managed Device

Purpose: show that multiple protocol surfaces reveal different slices of identity and that "web evidence" is not one thing.

Choose one host with HTTP or HTTPS services. A NAS or camera is usually a good candidate.

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

Tasks

1. Explain what nmap contributed for this host.
2. Identify which service entries appear stronger and which appear generic.
3. State what remains unresolved after the web-facing evidence alone.
4. Explain why a product string like `lighttpd`, `nginx`, or `OpenSSH` is useful but often insufficient for a product-level claim.
5. State whether the current web-facing evidence supports:
   - vendor-level identity
   - family-level identity
   - model-level identity
   - or no justified identity beyond "web-managed host"

Deliverable: a short memo titled `Web Surface Evidence`, 150-250 words.

Exercise 3: Camera Identity and Stronger Self-Report

Purpose: compare weaker media-surface evidence against stronger management-plane evidence and force an explicit ranking.

Find a host whose services suggest a camera-like profile, such as ports `80`, `443`, `554`, `8000`, or `37777`.

```
curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
  | jq '[.hosts[]
      | select(any(.services[]?; .port == 554 or .port == 8000 or
        .port == 37777))
      | {
          ip,
          vendor,
          device_type,
          services: [.services[]? | {port, service, product, version}]
        }]'
```

Tasks

1. Identify which fields look like weaker media-facing hints and which look like stronger device-identity evidence.
2. Explain why a structured self-report, when available, should usually outrank a generic server string.
3. State one way a camera can still produce conflicting evidence even when the final identity claim is strong.
4. State which version claim you would treat as:
   - exact
   - bounded
   - weak contextual hint if multiple version-like fields are present

Deliverable: a ranked evidence list titled `Camera Identity Hierarchy`, from strongest to weakest, with one sentence of justification per line.

Exercise 4: Fingerprint Cascade and Conflict Handling

Purpose: treat the fingerprint cascade as an inference system rather than a lookup table.

Use the textbook's conflict logic to reason about one ambiguous or partially identified host.

```
curl -s "http://localhost:8100/v1/scanning/smart-
        scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
```

```
| jq '[.hosts[]
    | {
        ip,
        vendor,
        device_type,
        services: [.services[]? | {port, service, product, version}]
      }]'
```

Tasks

1. Pick one host whose evidence would not justify immediate high confidence.
2. Write down at least two competing identity hypotheses.
3. Explain which hypothesis the fingerprint cascade should favor and why.
4. State what confidence penalty, ambiguity note, or conflict note should survive in the final record.
5. State one reason the weaker hypothesis is still worth preserving mentally, even if it does not survive as the primary claim.

Deliverable: a one-page `Conflict Resolution Note`.

Exercise 5: Build the Identity Output Contract

Purpose: formalize the endpoint of the identity stage as a machine-usable but human-auditable contract.

For one host, construct an explicit identity output record with the following fields:

- `identity_claim`
- `candidate_cpes`
- `source_provenance`
- `version_confidence_class`
- `ambiguity_flag`
- `conflict_note`
- `downstream_admissibility`

You may use JSON, YAML, or a clearly labeled table.

Rules

1. If the evidence only supports a family-level or wildcard candidate, say so.
2. If multiple sources disagree on version, preserve that disagreement.
3. If the output should not be used for unreviewed downstream CVE lookup, state that explicitly.
4. If two candidate CPEs arise from different evidence paths, do not collapse them just to make the record look cleaner.

Deliverable: `Identity Output Contract for Host X`.

Exercise 6: Analyst Handoff Judgment

Purpose: decide what the next analytical stage may and may not inherit.

Write a short analyst handoff for two hosts:

- one host whose identity output you judge **downstream-admissible**
- one host whose identity output you judge **not yet downstream-admissible**

For each host, state:

1. what the best current identity claim is
2. what evidence justifies that claim
3. what uncertainty still remains
4. whether an automated downstream CVE lookup would be acceptable, and why
5. what a human reviewer should inspect next if the artifact is not yet admissible

Deliverable: `Phase 2 Handoff Decision`, 250-400 words total.

Hands-On Direct Probing

The previous exercises had you read the scanner's enrichment output. The next exercises have you run the underlying protocol probes yourself, so you can compare what each protocol witness reveals against the integrated record. The lab's API container is on the simulation network and has `nmap`, `curl`, and `python3` available; you'll invoke them through `docker exec`.

Exercise 7: nmap Service Version Detection

Purpose: see what nmap's `-sV` probe reveals on a single host, and compare its evidence to the scanner's enrichment.

```
docker exec student-lab-api-1 nmap -sV -Pn -p 80,554,8000 172.30.0.10
```

> **Expected reading:** on the Hikvision camera you will likely see `aiohttp 3.13.5 (Python 3.11)` on port 80. That string identifies the *transport implementation* the simulator runs (Python's aiohttp), not the device vendor. Vendor identity for this host appears in the **HTTP `Server:` header** (`DNVRS-Webs`), the **`WWW-Authenticate: realm`** string (`Hikvision Digital Technology`), and the **RTSP `Server:` line** (`Hikvision Streaming Server 1.0`). Treat `aiohttp` as transport evidence, not identity evidence.

Tasks

1. Identify the per-port `Service`, `Version`, and any `extrainfo` nmap returns.
2. For each port, compare nmap's verdict to the API's `services[].product` and `services[].version` for the same host.
3. Note one piece of evidence nmap surfaces that the API does not expose, and one piece the API surfaces that nmap does not.

Deliverable: `nmap vs Enrichment — Host 172.30.0.10`, three short bullets.

Exercise 8: HTTP Banner Scraping by Hand

Purpose: identify vendor signals carried in HTTP response headers without relying on a higher-level fingerprint.

```
docker exec student-lab-api-1 curl -sI http://172.30.0.10/
docker exec student-lab-api-1 curl -sI http://172.30.0.11/
docker exec student-lab-api-1 curl -sI http://172.30.0.12/
```

Tasks

1. For each host, copy the `Server` header and any `WWW-Authenticate` realm string.
2. State which of those two headers carries the stronger vendor signal, and why.
3. If a header looks plausible but generic (e.g. `aiohttp`), explain whether you would treat it as identity evidence or as transport evidence.

Deliverable: `HTTP Banner Evidence Table`, three rows.

Exercise 9: RTSP Self-Report Probe

Purpose: retrieve the RTSP self-identification banner without a full streaming session — the same self-report a Chapter 2 enrichment cascade reads.

```
docker exec student-lab-api-1 python3 -c "
import socket
for ip in ('172.30.0.10', '172.30.0.11', '172.30.0.12'):
    s = socket.socket(); s.settimeout(3)
    try:
        s.connect((ip, 554))
        s.send(b'OPTIONS rtsp://' + ip.encode() + b'/
        RTSP/1.0\r\nCSeq: 1\r\n\r\n')
        data = s.recv(2048).decode(errors='replace')
        print(f'=== {ip} ===\n{data}')
    except Exception as e:
        print(f'=== {ip} === error: {e}')
    finally:
        s.close()
"
```

Tasks

1. For each host, identify the `Server:` header in the RTSP response.
2. Decide whether the RTSP self-report agrees with, conflicts with, or refines the HTTP-derived vendor claim from Exercise 8.
3. If a host returns `401 Unauthorized`, state explicitly whether the 401 is itself evidence of vendor identity, and why.

Deliverable: `RTSP Self-Report Comparison`, three short paragraphs.

Exercise 10: MAC OUI Verification by Hand

Purpose: verify a vendor claim against the OUI registry using only the hardware address.

```
docker inspect student-lab-hikvision-camera-1 -f '{{range
        .NetworkSettings.Networks}}{{.MacAddress}}{{end}}'
docker inspect student-lab-dahua-nvr-1 -f '{{range
        .NetworkSettings.Networks}}{{.MacAddress}}{{end}}'
docker inspect student-lab-axis-camera-1 -f '{{range
        .NetworkSettings.Networks}}{{.MacAddress}}{{end}}'
```

Tasks

1. For each MAC, record the OUI (the first three octets).
2. State the canonical vendor each OUI maps to (use any public OUI lookup; the IEEE registry is authoritative).
3. Compare the OUI vendor against the API's `vendor` field for the same host. If they disagree, state which witness you would prefer and why.

> **Expected reading:** the Hikvision MAC starts with `c0:56:e3` (Hikvision OUI). The Dahua and Axis simulator MACs may be fixed fixture prefixes or Docker-assigned locally administered addresses; if a public OUI lookup does not map them to the claimed vendor, record the OUI witness as silent rather than contradictory. A locally administered MAC has the second-least-significant bit of the first octet set; common Docker examples include prefixes such as `02`, `36`, `6a`, or `a2`. In those cases, rely on protocol self-reports instead.

Deliverable: `OUI Cross-Check`, three rows.

Exercise 11: Construct a CPE from Raw Evidence

Purpose: build a CPE 2.3 identifier yourself from observed evidence, then compare it to what the scanner produced.

Pick one host (the Hikvision camera at `172.30.0.10` is the running case). From the evidence you collected in exercises 7–10:

1. Decide a `vendor` token (lowercase, no spaces).
2. Decide a `product` token from the strongest service signal.
3. Decide a `version` token, or use `*` if no version is reliably observable.
4. Compose the CPE in 2.3 form:
   `cpe:2.3:o:vendor:product:version:*:*:*:*:*:*:*:*` for hardware/firmware-style products, or `cpe:2.3:a:vendor:product:version:*:*:*:*:*:*:*:*` for application-style products.

Then run the cross-check:

```
curl -s "http://localhost:8100/v1/scanning/smart-
         scan/$SCAN_ID/results" \
  -H "Authorization: Bearer $TOKEN" \
  | jq '.hosts[] | select(.ip == "172.30.0.10") | {vendor,
         device_type, cpes: (.cpes // [])}'
```

Tasks

1. Compare your hand-constructed CPE against the scanner's top-level `.cpes[]` array (the API does not expose an `.identity.cpe` scalar — host CPEs live in the `cpes` array on each host record).
2. The scanner often emits `cpe:2.3:a:hikvision:ip_camera:*:*:*:*:*:*:*:*` (part `a:` for application). If your hand-built CPE uses part `o:` (operating system / firmware) or `h:` (hardware), explain which evidence supports each part choice and which you would defend in a security-review conversation. The point is not to match the scanner — it is to defend your evidence.

Deliverable: `Hand-Constructed CPE vs Scanner CPE`, one short paragraph plus the two CPE strings.

Submission Requirements

Submit one document containing:

1. `Initial Identity Hypotheses`
2. `Raw Enrichment Record`
3. `Web Surface Evidence`
4. `Camera Identity Hierarchy`
5. `Conflict Resolution Note`
6. `Identity Output Contract for Host X`
7. `Phase 2 Handoff Decision`
8. `nmap vs Enrichment — Host 172.30.0.10`
9. `HTTP Banner Evidence Table`
10. `RTSP Self-Report Comparison`
11. `OUI Cross-Check`
12. `Hand-Constructed CPE vs Scanner CPE`

What This Lab Is Not

This lab is not:

- a disguised Chapter 3 vulnerability lab
- a software bill-of-materials lab
- a CRA compliance lab
- a vendor-trust ranking lab

Those topics may matter later in the course. They are not the intellectual endpoint of Chapter 2.

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 Phase 3), tear the simulation stack down to free resources. **Do not run this if you intend to continue to Phase 3 or homework that depends on the live scan data.**

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