RE-101 Binary-Diff Lab Cluster: Wireshark CVE Quartet (2026-05)

5,333 words · ~24 min read

2026-05-07 v2 (academy captures committed; placeholder substitutions complete; per-CVE §3.X.5 capture-loading discipline notes added; cyber-use citation forward-pointer collapsed to direct cite)

Course companion for: VCA-RE-101 (primary curriculum owner per D1) + VCA-ADV-101 (capstone application layer) + VCA-SEC-101 (defensive context cross-reference) **Scope:** Lab-cluster module scaffold; Week 9-10 binary-diff-methodology unit **Pairs with parent quartet handout:** `cve-lab-wireshark-rce-quartet-2026-05.md` (full walkthroughs of all 4 CVEs) **Cyber-use footnote:** Anthropic acceptable-use cyber-research exception on `munsonj2.0` per D7; canonical citation handout at `handouts/cross-chapter-anthropic-cyber-use-citation.md` **Version:** 2026-05-07 v2 (academy captures committed; placeholder substitutions complete; per-CVE §3.X.5 capture-loading discipline notes added; cyber-use citation forward-pointer collapsed to direct cite)

Overview

This handout is the RE-101 lab cluster for the four-CVE Wireshark binary-diff module per curriculum decision D1 (`docs/test-range-cve-quartet-2026-05-decisions.md`). RE-101 owns the per-CVE binary-diff lab modules and the assessment rubric; ADV-101 inherits the captures and recipes as an offensive-tradecraft application layer where one of the four CVEs becomes the capstone-arc primitive. The scaffold shipped first; the academy-original CVE-trigger captures were then committed to `pcap-tools/fixtures/cve-quartet-2026-05/`; this v2 round integrates the captures into the lab structure by substituting placeholder flags with concrete capture paths and adding per-CVE capture-loading discipline notes. The lab cluster is now fully usable.

The cluster slots into Week 9 + Week 10 of the RE-101 14-week schedule as the binary-diff-methodology unit. Per session-table accounting decision D6, each CVE expresses as one 45-minute lecture plus one 90-minute lab plus one 45-minute self-homework read; four CVEs total 12 hours of formal cohort time plus roughly 3-4 hours of student-side study time. The unit takes the cohort from an introduction to binary-diffing as RE methodology through four worked instances against four distinct bug-class shapes

(heap buffer overflow via integer truncation; heap buffer overflow via loop-accounting failure; heap buffer overflow via missing bounds check; logic bug via zip-slip plus auto-execute) and graduates them with the procedural fluency the ADV-101 capstone arc assumes is already in place.

`--authorized-by` discipline applies throughout. Every binary-diff exercise, every capture load, and every compiled-binary inspection runs against academy-owned, intentionally-vulnerable Wireshark builds inside the academy `cve-quartet-2026-05-range` Docker container per D5 (shared base image; per-CVE runtime profile selected via `CVE_ID` env var). Production Wireshark installations on student workstations are never the test target; the academy lab harness at `/opt/wireshark-{4.6.4,4.6.5,4.4.14,4.4.15}/bin/` is the only authorized loading surface for the malformed CVE captures.

The cyber-use citation footnote per D7: this handout's authoring discipline operates under Anthropic's acceptable-use cyber-research exception attached to the `munsonj2.0` account; the canonical academy-wide cyber-use citation handout at `handouts/cross-chapter-anthropic-cyber-use-citation.md` carries the per-handout footnote convention; this handout's §7 forward-points to the canonical citation directly.

---

§1: Why this lab cluster

Binary-diffing is the unifying methodology that RE-101 trains. A Belt-4 RE-101 graduate who can read a `git diff vN..vN+1` against a vulnerable open-source project's source code, predict the wire shape that drives the dissector or parser to the vulnerable code path, load a capture and confirm the prediction, and inspect the compiled-binary diff in their preferred reverser has the methodology fluency the ADV-101 capstone assumes. The methodology generalizes across language ecosystems, project sizes, and bug classes; the four-CVE quartet is a deliberately calibrated set of worked instances designed to exercise the methodology across distinct bug-class shapes.

The four CVEs cover four distinct bug-class shapes:

1. **CVE-2026-5402 TLS ECH integer-truncation** (heap buffer overflow driven by integer truncation; CWE-190 cascading to CWE-122). The dissector reads an attacker-controlled length field, truncates it through an integer-arithmetic path, allocates a buffer sized to the truncated value, and copies the full attacker-controlled payload. The bug-class shape is the integer-arithmetic failure that produces a size-mismatch primitive.

2. **CVE-2026-5403 SBC Codec frame-accounting** (heap buffer overflow driven by multi-frame loop accounting; CWE-122). The dissector iterates over a multi-frame SBC payload, accumulates per-frame size into an output buffer, and overflows when the accumulated size exceeds the buffer capacity. The bug-class shape is the loop-accounting failure that produces a cumulative-overflow primitive.
3. **CVE-2026-5405 RDP fastpath** (classic missing-bounds-check heap overflow; CWE-122). The dissector reads an attacker-controlled length field for an RDP fastpath update, copies the corresponding bytes into a fixed-size internal buffer, and never validates the length against the buffer capacity. The bug-class shape is the canonical missing-bounds-check that has shipped in dozens of CVEs across decades. This CVE is the recommended ADV-101 capstone candidate per curriculum decision D1 because the primitive is the cleanest of the four and the exploitation chain is the most teachable at the capstone register.
4. **CVE-2026-5656 Profile import zip-slip** (logic bug via path-traversal plus auto-execute; CWE-22). The Profile import handler extracts a ZIP archive without canonicalization-plus-boundary-check on entry names; an entry whose name traverses outside the extraction root lands a `.lua` file in the auto-loaded plugins directory; Wireshark's startup code executes the plugin. The bug-class shape is the logic bug that does not leave wire-level evidence; the lab structure for this CVE is substantively different from the other three because the academy artifact is a malicious `.zip` file, not a `.pcapng` capture (the loopback capture from a zip-slip exploitation may be empty or trivial; the pedagogical point is that not every CVE leaves wire-level evidence).

The lab cluster integrates across the academy's coordinated curriculum surfaces:

- **`vca-mini-wireshark-cves-2026-05` mini-page**: the public entry point that surfaces the quartet to prospective students and serves as the Recognize-tier on-ramp into the deeper RE-101 reproduce-tier work.
- **Parent quartet handout** `cve-lab-wireshark-rce-quartet-2026-05.md`: the full walkthroughs of all four CVEs with `§X.4 Code-level discussion` subsections per CVE that this lab cluster's self-homework reads anchor on.
- **CVE-class deep-dives** `cve-class-zip-slip-pattern.md` (shipped) plus `cve-class-asymmetric-validation-pattern.md` (companion handout in flight): class-shape deep-dives that contextualize each CVE's bug class within the broader CVE-class taxonomy at `cross-chapter-cve-class-vocabulary-reference.md`.

- **Detection-rule references** `cve-suricata-rules-reference-wireshark-quartet-2026-05.md` (shipped) plus the Snort 3 companion handout (in flight on spec-curriculum-sec-sonnet): network-layer detection-rule templates per CVE that compose with this lab cluster's binary-diff register as defense-in-depth.
- **Fundamentals walkthrough** `cross-chapter-pcap-walkthrough-fundamentals.md` (shipped as a companion handout): the academy-original baseline captures that students walk before they touch the malformed CVE captures; the baseline-vs-anomaly comparison anchor at vocabulary-fluency depth.
- **ADV-101 capstone arc** (in flight): the capstone-tier extension where one CVE is selected as the primitive for the full reproduction-and-tooling chain.

The lab cluster is the methodology bridge between the recognize-tier mini-page entry and the reproduce-tier ADV-101 capstone. Belt-4 RE-101 graduates who walk this cluster with the rubric's full point allocation are positioned to elect ADV-101 capstone work confidently; graduates who need additional methodology grounding return to the cluster's labs as practice surface before the capstone.

---

§2: Lab cluster session table (per D6)

The cluster runs across Week 9 and Week 10 of the RE-101 14-week schedule. Per session-table accounting decision D6, each CVE expresses as a triple of one 45-minute lecture plus one 90-minute lab plus one 45-minute self-homework read; the four CVEs are sequenced in CVE-number order (5402 → 5403 → 5405 → 5656) which also aligns with the bug-class shape progression (integer-truncation → loop-accounting → missing-bounds-check → logic-bug-plus-auto-execute).

Slot	Duration	Topic	Output artifact
Week 9 lecture 1	45 min	Pre/post-patch binary diff as RE methodology	Cohort note set; methodology framework reference
Week 9 lab 1	90 min	CVE-2026-5402 TLS ECH integer-truncation: source diff + capture analysis + compiled-binary diff	Per-student lab notebook entry covering all five rubric dimensions
Week 9 self-homework 1	45 min read	Parent quartet `§1.4 Code-level discussion` + cross-reference to `cve-class-vocabulary-reference` integer-truncation row	Reading completion checkpoint
Week 9 lecture 2	45 min	Loop-accounting bug class through the 5403 lens	Cohort note set extension
Week 9 lab 2	90 min	CVE-2026-5403 SBC frame-accounting: source diff + capture analysis + compiled-binary diff	Per-student lab notebook entry
Week 9 self-homework 2	45 min read	Parent quartet `§2.4 Code-level discussion`	Reading completion checkpoint
Week 10 lecture 1	45 min	Missing-bounds-check bug class through the 5405 lens	Cohort note set extension
Week 10 lab 1	90 min	CVE-2026-5405 RDP fastpath: source diff + capture analysis + compiled-binary diff (ADV-101 capstone-CVE candidate)	Per-student lab notebook entry; capstone-fit assessment
Week 10 self-homework 1	45 min read	Parent quartet `§3.4 Code-level discussion`	Reading completion checkpoint
Week 10 lecture 2	45 min	Logic bugs that do not leave wire evidence: zip-slip plus auto-execute	Cohort note set; logic-bug vs memory-corruption distinction
Week 10 lab 2	90 min	CVE-2026-5656 Profile import zip-slip: source diff + archive-entry construction + compiled-binary diff	Per-student lab notebook entry; archive analysis artifacts
Week 10 self-homework 2	45 min read	Parent quartet `§4.4 Code-level discussion` + `cve-class-zip-slip-pattern`	Reading completion checkpoint

Time totals: 4 × (45 + 90 + 45) = 4 × 180 = 720 minutes = 12 hours formal cohort time across the two weeks. Plus approximately 3-4 hours student-side study (4 × 45 min reading plus 30-60 min per-CVE additional review for capstone-fit candidates). The 6-hour-per-week formal load is consistent with RE-101's standing pacing of approximately 7-9 hours formal per week across the 14-week schedule.

Sequencing rationale: Week 9 covers the two memory-corruption bug-class shapes that are most pedagogically simple (integer-truncation drives a single-arithmetic-operation failure; loop-accounting drives a cumulative-iteration failure). Week 10 covers the most-cited memory-corruption shape (missing-bounds-check; the canonical CVE-class anchor; the ADV-101 capstone-CVE candidate) followed by the orthogonal logic-bug shape (zip-slip plus auto-execute) which expands the cohort's bug-class vocabulary beyond memory corruption. The Week 10 lecture 2 framing ("logic bugs that do not leave wire evidence") is the cluster's pedagogical capstone moment: students leave the cluster with the recognition that binary-diffing as methodology generalizes beyond memory-corruption-class bugs into logic-bug-class bugs that present differently at the wire layer but identically at the source-code-diff layer.

---

§3: Per-CVE lab module structure

Each subsection below specifies the lecture brief, the lab brief with cohort steps, the self-homework reading, and the assessment rubric for one CVE in the quartet. Subsections §3.1 through §3.3 follow a uniform structure (memory-corruption CVEs); subsection §3.4 deviates substantively (logic-bug-plus-auto-execute) and the deviations are called out explicitly.

§3.1 CVE-2026-5402 TLS ECH integer-truncation

§3.1.1 Lecture brief (45 min)

Frame the bug class: integer truncation in arithmetic that computes an allocation size, where the truncated value is small but the actual data length is large. Walk the canonical pattern: attacker-controlled length field reaches an arithmetic path that narrows the type (e.g., `uint16_t` cast from `uint32_t`), the narrowed value drives the allocation, the original full-width value drives the copy length. The cumulative effect is a heap buffer that is allocated for the small truncated size and written for the full attacker-controlled size; the bytes past the allocation corrupt heap metadata or adjacent allocations, producing the canonical heap buffer overflow primitive.

Situate the bug class in the broader binary-diffing-as-methodology framing. The pre-patch source code carries the integer-arithmetic path that produces the truncation; the post-patch source code adds either a saturating-cast (the narrowing operation produces the maximum representable value if the source value exceeds it) or an explicit bounds check before the narrowing happens. The diff between pre-patch and post-patch is small: a few lines of added validation. The bug class is detectable from the source diff alone; the wire shape that drives the dissector to the vulnerable code path is predictable from the source diff alone. This is the reproducible discipline the cluster trains.

Preview what the lab will discover: the specific arithmetic path in `epan/dissectors/packet-tls.c` that handles the TLS ECH (Encrypted Client Hello) extension, the integer cast that truncates the extension length, and the wire-shape construction that drives a dissector to the truncation path.

§3.1.2 Lab brief (90 min hands-on)

Cohort steps in sequence:

1. **Source-archive setup** (5 min). Confirm the academy support tree mounts at `/opt/cve-source-archives/`; the pre-patch tarball at `wireshark-4.6.4.tar.xz` and the post-patch tarball at `wireshark-4.6.5.tar.xz` are extracted. Alternative path: `git clone https://gitlab.com/wireshark/wireshark.git ~/wireshark` plus `git checkout v4.6.4` (the upstream Git history carries the version tags; the source-archive tree is the academy's offline-capable mirror per D4).
2. **Source-diff command** (5 min). Run the canonical binary-diff command:

   ```
   git diff v4.6.4..v4.6.5 -- epan/dissectors/packet-tls.c
   ```

   The diff is the raw input to every subsequent step. Expected output is a small set of changed lines covering the integer-arithmetic path that handled the TLS ECH extension length field plus a few comments and unrelated changes.
3. **Source-diff comprehension** (15 min). Read the diff. Identify the specific arithmetic operation that truncates the length field. Read the surrounding code to understand the data-flow: where the length field is read from the wire; where the truncated value

is used to compute an allocation; where the full-width value is used to drive the copy. Cohort discussion question: "What is the smallest wire shape that would drive the dissector through the truncation path?"

4. **Wire-shape prediction** (10 min). Each student writes a one-paragraph prediction of the malformed packet shape: TLS record type, handshake message type, ECH extension presence, ECH extension length-field value, ECH extension payload length. The prediction is committed to the lab notebook before the capture is loaded; this commit-then-confirm discipline trains the methodology habit of source-first prediction over capture-first inspection.
5. **Capture loading** (15 min). Load the academy capture at `pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5402-trigger-tls-ech-overflow.pcapng` (committed; sidecar metadata at `*.meta.json`; manifest entry on v4 at `cve-2026-5402-trigger-tls-ech-overflow.pcapng` with `cve_anchor=CVE-2026-5402` plus `range_fingerprint` plus `warning` field per D3) into the academy-isolated Wireshark build at `/opt/wireshark-4.6.4/bin/wireshark` (the pre-patch vulnerable build; required to reach the dissector path that the patch removed). Apply the canonical display filter `tls.handshake.extension.type == 0xfe0d` (TLS ECH extension type code) to surface the malformed packet.
6. **Prediction verification** (15 min). Compare the loaded packet's ECH extension fields against the pre-loading prediction. Identify any deviations between predicted and observed wire shape. Cohort discussion question: "Did the source diff fully predict the wire shape, or did the capture surface details that the diff alone did not?"
7. **Compiled-binary diff inspection** (20 min). Open both pre-patch and post-patch Wireshark binaries in the student's preferred reverser (radare2; ghidra; IDA Free with the Wireshark plugin; binary at `/opt/wireshark-{4.6.4,4.6.5}/bin/wireshark` or its `epan` shared library at `/opt/wireshark-{4.6.4,4.6.5}/lib/wireshark/`). Locate the dissector function for TLS ECH; visually diff the two compiled functions to identify the inserted validation. Note: ghidra's BinDiff plugin or radare2's `r2diff` provide automated diffing; manual inspection is acceptable at RE-101 register and is the practitioner-foundation skill the cluster trains.
8. **Lab-notebook commit** (5 min). Each student commits their lab-notebook entry covering all five rubric dimensions; the notebook entry is the assessment artifact for §3.1.4.

§3.1.3 Self-homework (45 min read)

Read `cve-lab-wireshark-rce-quartet-2026-05.md` §1.4 Code-level discussion (the parent quartet handout's full Code-level walkthrough of CVE-2026-5402). Cross-reference to `cross-chapter-cve-class-vocabulary-reference.md` §2 Family 1 (Memory-corruption + parser bugs) integer-truncation row (CWE-190 → CWE-122; the canonical bug-class anchor). Optional extension reading: the upstream Wireshark advisory at `wnpa-sec-2026-14` and the related issue tracker discussion if the student wants to see how the disclosure pipeline handled the bug. Reading completion checkpoint is a one-paragraph reflection in the lab notebook addressing the question: "How does CVE-2026-5402's bug-class shape compare to the canonical anchor CVE in the cve-class-vocabulary handout's integer-truncation row, and what does the comparison teach about the bug class?"

§3.1.4 Assessment rubric

5-point scale across 5 dimensions; 25 points maximum per CVE; 100 points maximum across the cluster.

Dimension	1 point	3 points	5 points
Source-diff comprehension	Identifies the changed file	Identifies the specific changed lines and the data-flow	Articulates the bug-class shape from the diff alone, including pre-patch failure mode and post-patch defensive structure
Wire-shape prediction accuracy	Names the protocol layer	Names the protocol fields and approximate length-field values	Predicts the malformed packet shape at byte-level granularity, with rationale grounded in the source diff
PCAP analysis depth	Loads the capture and identifies the malformed packet	Walks the layer stack of the malformed packet and confirms the prediction	Surfaces deviations between prediction and observation, articulates what the deviations reveal about the dissector's actual data flow
Compiled-binary diff inspection rigor	Opens the binaries in a reverser	Locates the dissector function and identifies the inserted validation	Walks the assembly-level diff of the validation insertion, articulates the optimizer-effect on the assembly delta
Cross-CVE class shape recognition	Names CVE-2026-5402's bug class	Compares to the parent CVE-class vocabulary anchor row	Articulates how the bug class generalizes beyond TLS-ECH into integer-truncation patterns in adjacent codebases (parsers; deserializers; framing dissectors)

§3.1.5 Capture-loading discipline note

Outside the §3.1.2 Step 5 pre-patch lab moment (where loading the capture in `/opt/wireshark-4.6.4/bin/wireshark` is the deliberate pedagogical action), students load the capture only with academy-isolated post-patch Wireshark at `/opt/wireshark-4.6.5/bin/wireshark` (or `/opt/wireshark-4.4.15/bin/wireshark` for the 4.4.x branch); host Wireshark is never the loading target because the host build may not carry the patch.

§3.2 CVE-2026-5403 SBC Codec frame-accounting

§3.2.1 Lecture brief (45 min)

Frame the bug class: loop-accounting failure where a multi-iteration loop accumulates per-iteration size into an output buffer without per-iteration validation against the buffer capacity. Walk the canonical pattern: the dissector iterates over a multi-frame payload,

computes per-frame size from a wire-encoded length field, accumulates the per-frame bytes into a fixed-size output buffer, and overflows the buffer when the accumulated total exceeds the capacity. The bug-class shape is the cumulative-overflow primitive; the bug is invisible at any single iteration because each iteration's per-frame size may be small.

Situate the bug class against the integer-truncation class (§3.1): both produce heap buffer overflows; both are size-mismatch primitives; the data-flow is structurally different (single-arithmetic-operation truncation vs cumulative-iteration accumulation). Belt-4 RE-101 graduates who recognize both shapes have a wider taxonomy of size-mismatch primitives than graduates who recognize only the simple-truncation shape.

Preview what the lab will discover: the specific iteration loop in `epan/dissectors/packet-sbc.c` that accumulates SBC audio frames into the output buffer without per-iteration bounds checking; the wire-shape construction that drives a multi-frame SBC payload through enough iterations to exceed the buffer capacity.

§3.2.2 Lab brief (90 min hands-on)

Same eight-step structure as §3.1.2, substituting the SBC-specific elements:

1. Source-archive setup (5 min).
2. Source-diff command: `git diff v4.6.4..v4.6.5 -- epan/dissectors/packet-sbc.c` (5 min).
3. Source-diff comprehension; identify the iteration loop and the per-frame accumulation arithmetic (15 min).
4. Wire-shape prediction; predict the multi-frame SBC payload encoded in RTP (10 min).
5. Capture loading from `pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5403-trigger-sbc-rtp-overflow.pcapng` (committed; sidecar metadata at `*.meta.json`; manifest entry on v4) into `/opt/wireshark-4.6.4/bin/wireshark`. Display filter `rtp.payload_type == sbc` or equivalent (15 min).
6. Prediction verification; compare the multi-frame payload's per-frame sizes against the predicted accumulation arithmetic (15 min).
7. Compiled-binary diff inspection; locate the SBC dissector loop in pre/post-patch binaries (20 min).
8. Lab-notebook commit (5 min).

§3.2.3 Self-homework (45 min read)

Read parent quartet handout §2.4 Code-level discussion (CVE-2026-5403 walkthrough). Cross-reference to `cross-chapter-cve-class-vocabulary-reference.md` §2 heap-overflow row. Reading completion checkpoint addresses: "How does loop-accounting differ from integer-truncation as a heap-overflow primitive class, and what does the difference imply for defensive validation patterns?"

§3.2.4 Assessment rubric

Same five-dimension five-point structure as §3.1.4; substitute SBC-specific anchors throughout. The cross-CVE class shape recognition dimension at 5 points requires articulating the loop-accounting shape and naming at least one adjacent codebase (audio codec dissectors in other tools; video codec dissectors; multi-frame protocol parsers in general) where the same shape would surface.

§3.2.5 Capture-loading discipline note

Outside the §3.2.2 Step 5 pre-patch lab moment, students load the capture only with academy-isolated post-patch Wireshark at `/opt/wireshark-4.6.5/bin/wireshark` (or `/opt/wireshark-4.4.15/bin/wireshark`); host Wireshark is never the loading target.

§3.3 CVE-2026-5405 RDP fastpath (ADV-101 capstone-CVE candidate)

§3.3.1 Lecture brief (45 min)

Frame the bug class: missing bounds check. The most-cited memory-corruption bug-class shape across decades of CVEs; the canonical CWE-122 anchor. Walk the canonical pattern: the dissector reads an attacker-controlled length field, copies the corresponding bytes from the wire into a fixed-size internal buffer, and never validates that the length field is within the buffer's capacity. The bug-class shape is the simplest of the four; the lecture pairs the simplicity with the observation that the bug class persists across the discipline because the simplicity itself produces blind spots in code review.

Situate the bug class as the ADV-101 capstone-CVE candidate per curriculum decision D1: the primitive is the cleanest of the four; the wire shape is the most teachable at the capstone register; the exploitation chain (length-field fabrication → fixed-size-buffer overflow → heap-corruption-to-control-flow) is the most pedagogically tractable. ADV-101 students who elect this CVE for their capstone arc work the chain end-to-end against an academy-isolated Wireshark build; the RE-101 lab cluster's binary-diff treatment is the methodology foundation the capstone arc assumes.

Preview what the lab will discover: the specific buffer-copy operation in `epan/dissectors/packet-rdp.c` that handles the RDP fastpath update without length-validation; the wire-shape construction that drives an oversized fastpath update through the dissector.

§3.3.2 Lab brief (90 min hands-on)

Same eight-step structure; substituting:

1. Source-archive setup (5 min).
2. Source-diff command: `git diff v4.6.4..v4.6.5 -- epan/dissectors/packet-rdp.c` (5 min).
3. Source-diff comprehension; identify the buffer-copy operation and the missing-validation site (15 min).
4. Wire-shape prediction; predict the RDP fastpath update with oversized length field (10 min).
5. Capture loading from `pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5405-trigger-rdp-zgfx-overflow.pcapng` (committed; sidecar metadata at `*.meta.json`; manifest entry on v4) into `/opt/wireshark-4.6.4/bin/wireshark`. Display filter `rdp` or `tcp.port == 3389` (15 min).
6. Prediction verification (15 min).
7. Compiled-binary diff inspection; the RDP dissector function in pre/post-patch binaries (20 min).
8. Lab-notebook commit (5 min). Capstone-fit assessment by the instructor: students who score above 20 of 25 on this lab's rubric are flagged as candidates for the ADV-101 capstone arc that uses this CVE as the primitive.

§3.3.3 Self-homework (45 min read)

Read parent quartet handout §3.4 Code-level discussion. Cross-reference to `cross-chapter-cve-class-vocabulary-reference.md` §2 heap buffer overflow / missing-bounds-check row (the canonical CWE-122 anchor). Reading completion checkpoint addresses: "Why has missing-bounds-check persisted as the most-cited memory-corruption bug-class shape across decades of CVEs, and what does the persistence imply about the discipline's defensive maturity?"

§3.3.4 Assessment rubric

Same five-dimension five-point structure. The cross-CVE class shape recognition dimension at 5 points requires articulating that missing-bounds-check is the canonical anchor against which the integer-truncation (§3.1) and loop-accounting (§3.2) shapes are recognized as variants; the bug-class taxonomy converges on missing-bounds-check as the parent class with truncation and accumulation as adjacent specializations.

§3.3.5 Capture-loading discipline note

Outside the §3.3.2 Step 5 pre-patch lab moment, students load the capture only with academy-isolated post-patch Wireshark at `/opt/wireshark-4.6.5/bin/wireshark` (or `/opt/wireshark-4.4.15/bin/wireshark`); host Wireshark is never the loading target. Capstone-fit candidates per §3.3.2 Step 8 carry this discipline forward into ADV-101 capstone-arc work where the same loading constraint applies during the primitive-to-tooling chain.

§3.4 CVE-2026-5656 Profile import zip-slip (logic bug; substantively different lab structure)

This subsection deviates from the §3.1-§3.3 uniform structure because the bug class is structurally different. The first three CVEs are memory-corruption bugs that present at the wire layer; the academy artifact is a `.pcapng` capture; the lab harness loads the capture into the vulnerable Wireshark build. CVE-2026-5656 is a logic bug that does not leave wire-level evidence; the academy artifact is a malicious `.zip` archive (the Wireshark profile); the lab harness imports the archive through the Wireshark GUI's "Manage Profiles → Import" dialog. The pedagogical point is that not every CVE leaves wire-level evidence, and the binary-diffing methodology generalizes beyond capture-driven analysis.

§3.4.1 Lecture brief (45 min)

Frame the bug class: logic bug via path-traversal plus auto-execute. The canonical zip-slip pattern (covered at class-shape depth in `cve-class-zip-slip-pattern.md`) chained to Wireshark's auto-execute behavior on `.lua` files in the plugin directory. Walk the canonical pattern: the Profile import handler iterates over archive entries, computes the destination path by concatenation, writes the entry's content to the resolved location; an entry whose name contains `../` traversal sequences resolves outside the intended extraction root; an entry whose resolved path lands a `.lua` file in the plugin directory becomes RCE on Wireshark startup.

Situate the bug class against the three memory-corruption CVEs: this CVE is a logic bug, not a memory-corruption bug. The bug-class shape is at the source-code-and-filesystem-semantics layer, not at the parser-and-allocation layer. The methodology generalization the cluster teaches: binary-diffing as RE methodology covers logic bugs equally well; the diff between pre-patch and post-patch Profile import code carries the same evidence the diff between pre-patch and post-patch TLS or RDP dissector code carries; the reading discipline transfers; the verification discipline transfers; the only difference is the academy artifact (`.zip` archive vs `.pcapng` capture).

Preview what the lab will discover: the specific extraction logic in `ui/qt/utils/wireshark_zip_helper.cpp` that handles profile-archive extraction without canonicalization-plus-boundary-check; the archive-entry construction that produces a zip-slip primitive landing a `.lua` file in the plugin directory.

§3.4.2 Lab brief (90 min hands-on)

Modified eight-step structure for the logic-bug shape:

1. Source-archive setup (5 min). Same as §3.1-§3.3.
2. Source-diff command (5 min). The relevant file is C++ rather than C: `git diff v4.6.4..v4.6.5 -- ui/qt/utils/wireshark_zip_helper.cpp`. The diff carries the addition of canonicalization-plus-boundary-check on each archive entry.
3. Source-diff comprehension (15 min). Read the diff. Identify the extraction loop and the missing-validation site that the patch fills. Cross-reference to `cve-class-zip-slip-pattern.md` §5.1 (canonicalization-plus-boundary-check defensive pattern) for the cross-language idiom; verify that the patch's C++ idiom matches the canonical pattern.
4. Archive-entry shape prediction (10 min). Each student writes a one-paragraph prediction of the malicious archive's entry name and content: traversal depth (number of `../` sequences); target path component (e.g., `wireshark/plugins/lab-marker.lua`); content (a minimal Lua file with a marker call). The prediction is committed to the lab notebook before any archive is constructed.
5. Archive loading (15 min). Load the academy-curated malicious archive at `pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5656-trigger-profile-zip-slip.zip` (committed; sidecar metadata at `*.meta.json`; manifest entry on v4) into the academy-isolated Wireshark build at `/opt/wireshark-4.6.4/bin/wireshark` via "Manage Profiles → Import". Observe the resulting `.lua` file in the plugin directory. The companion loopback packet capture at `pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5656-trigger-profile-zip-slip-loopback.pcapng` records the host-side activity around the import operation; load it in parallel and observe that the malformed-archive event leaves no malformed packet shape on the wire (the bug is at the filesystem-extraction layer, not at the network layer; this is the cluster's logic-bug-vs-memory-corruption pedagogical contrast). Optional bench-replication path: each student can additionally construct the malicious archive via Python's `zipfile` module per the construction recipe in `cve-class-zip-slip-pattern.md` §4 to confirm the structural shape, against the same academy-isolated build.

6. Prediction verification (15 min). Compare the actual archive's entry name and content against the pre-loading prediction. Cohort discussion question: "What is the wire shape that would surface this CVE in a packet capture, and why does that question not have an answer for this CVE?"
7. Compiled-binary diff inspection (20 min). Open the pre-patch and post-patch `wireshark_zip_helper.cpp` compiled output (the C++ helper compiles into the main `wireshark` binary or the appropriate shared library; locating the function in the binary may require reading the symbol table). Visually diff the function across the two builds.
8. Lab-notebook commit (5 min).

§3.4.3 Self-homework (45 min read)

Read parent quartet handout §4.4 Code-level discussion. Read `cve-class-zip-slip-pattern.md` end-to-end (the academy's class-shape deep-dive on the zip-slip pattern; this CVE is the academy's primary anchor for the class). Cross-reference to `cross-chapter-cve-class-vocabulary-reference.md` §4 path-traversal row (the parent CVE-class taxonomy). Reading completion checkpoint addresses: "How does the zip-slip class-shape generalize across language ecosystems, and what does the generalization teach about substrate-level vs language-level vulnerability classes?"

§3.4.4 Assessment rubric

Same five-dimension five-point structure with substrate substitutions: the source-diff dimension reads C++ rather than C; the wire-shape-prediction dimension becomes archive-entry-shape-prediction; the PCAP-analysis-depth dimension becomes archive-analysis-depth; the cross-CVE class shape recognition dimension at 5 points requires articulating the substrate-vs-language thesis from `cve-class-zip-slip-pattern.md` §2.3 (zip-slip persists across Rust/Go/Java/Python/JavaScript/.NET because the bug substrate is filesystem-plus-archive-format, not any specific language's memory model).

§3.4.5 Capture-loading discipline note

Outside the §3.4.2 Step 5 pre-patch lab moment, students load the malicious archive only with academy-isolated post-patch Wireshark at `/opt/wireshark-4.6.5/bin/wireshark` (or `/opt/wireshark-4.4.15/bin/wireshark`); host Wireshark is never the import target because a host build that has not absorbed the 4.6.5 / 4.4.15 patch will execute the auto-loaded Lua plugin on next startup. The companion loopback packet capture at `pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5656-trigger-profile-zip-slip-loopback.pcapng` carries no zip-slip primitive in its bytes and can be opened in any post-patch build for reference.

§4: Cross-track integration

The lab cluster integrates across four academy curriculum surfaces. Each row maps one CVE to its appearance across the surfaces.

Surface	CVE-2026-5402 (TLS)	CVE-2026-5403 (SBC)	CVE-2026-5405 (RDP)	CVE-2026-5656
`vca-mini-wireshark-cves-2026-05` mini-page (Recognize tier; public catalog entry point)	Lab 1 entry; Recognize-tier pillar	Lab 2 entry	Lab 3 entry	Lab 4 entry
`vca-re-101` lab cluster (Reproduce tier; this handout)	Week 9 lab 1 + lecture 1 + self-homework 1	Week 9 lab 2 + lecture 2 + self-homework 2	Week 10 lab 1 + lecture 1 + self-homework 1 (ADV-101 capstone-CVE candidate)	Week 10 lab 2 + + self-homework bug; substantive different structur
`vca-adv-101` capstone arc (Capstone tier; in flight)	Capstone option	Capstone option	Capstone primary candidate per D1	Capstone option variant; less prin
`vca-sec-101` defensive context (Defend tier; cross-references back from existing handouts)	Suricata rule template at `cve-suricata-rules-reference-...md` §1.4; Snort 3 companion handout sister §1.4	Suricata §2.4; Snort 3 sister §2.4	Suricata §3.4; Snort 3 sister §3.4	Suricata §4.4 (fil matching); Snort §4.4; cross-refere `cve-class-zip-pattern.md` §5.5 detection-rule pa

The four-tier register (Recognize → Reproduce → Capstone → Defend) is the academy's coordinated pedagogy across the cluster. A Belt-4 RE-101 graduate has touched the Recognize tier through the mini-page entry, the Reproduce tier through this lab cluster, and is positioned to elect the Capstone tier through ADV-101 capstone arc work. The Defend tier is owned by SEC-101 + the existing detection-rule references and is consulted as the defensive complement to the offensive register the cluster trains.

§5: Capture loading discipline + warning badges

Students MUST load CVE captures only with academy-isolated Wireshark builds at `/opt/wireshark-{4.6.4,4.6.5,4.4.14,4.4.15}/bin/wireshark` inside the academy `cve-quartet-2026-05-range` Docker container. **Production Wireshark installations on student workstations are NEVER the loading target.** The academy lab harness is the only authorized loading surface for the malformed CVE captures.

Rationale: defense-in-depth against unknown-unknowns. The CVE captures are crafted-malformed traffic that exercises pre-patch dissector code paths that have shipped publicly-disclosed memory-corruption vulnerabilities. The post-patch Wireshark build at `/opt/wireshark-4.6.5/bin/wireshark` carries the validation that closes the disclosed vulnerability; loading the capture in the post-patch build is safe (and is part of the prediction-verification step in each lab brief). The pre-patch build at `/opt/wireshark-4.6.4/bin/wireshark` carries the vulnerable code path; loading the capture in the pre-patch build is what reproduces the bug-class shape and is the lab's pedagogical target. Both builds are academy-isolated inside the Docker container; the host filesystem is mounted read-only; the network namespace is isolated; the container's lifecycle is bounded by the lab session.

Loading the capture in a student's host Wireshark installation is a violation of the cluster's `--authorized-by` discipline. Students who do so are operating outside the academy's authorization scope; the discipline is the discipline the academy wants graduates to internalize as professional habit before they encounter real-world malformed-traffic engagement work in their first paid pentest engagement.

The catalog manifest at `pcap-tools/fixtures/curated-12-manifest.json` (now at v4) carries a `warning` field per D3 on each CVE-trigger capture entry, alongside `cve_anchor` (the canonical CVE ID), `range_fingerprint` (the academy range build hash that produced the capture), and `capture_timestamp` (ISO 8601). The `pcap-selector.js` frontend renders the `warning` field as a visible badge on the `/pcap-tools/` catalog UI per D3 frontend convention. Belt-4 RE-101 students approaching the catalog see the warning badge before they click; the badge is the academy's discoverability-tier reminder of the loading discipline this section codifies.

§6: Forward-pointers

Resource	Path	Relationship to this cluster
Parent quartet handout	`handouts/cve-lab-wireshark-rce-quartet-2026-05.md`	Full walkthroughs of all 4 CVEs; per-CVE `§X.4 Code-level discussion` is the self-homework reading anchor
Mini-page (catalog entry point)	`website/vca-mini-wireshark-cves-2026-05.html`	Recognize-tier on-ramp; public catalog surface
Suricata rules reference	`handouts/cve-suricata-rules-reference-wireshark-quartet-2026-05.md`	Defensive-detection-rule layer; per-CVE template at §1.4 / §2.4 / §3.4 / §4.4
Snort 3 sister rules reference	`handouts/cve-snort3-rules-reference-wireshark-quartet-2026-05.md`	Snort 3 syntax equivalent of the Suricata reference; in flight on spec-curriculum-sec-sonnet
CVE-class deep-dive: zip-slip	`handouts/cve-class-zip-slip-pattern.md`	Class-shape deep-dive for CVE-2026-5656; §3.4 self-homework reading
CVE-class deep-dive: asymmetric-validation	`handouts/cve-class-asymmetric-validation-pattern.md`	Class-shape deep-dive for CVE-2026-5405 (companion handout in flight)
Parent CVE-class vocabulary reference	`handouts/cross-chapter-cve-class-vocabulary-reference.md`	Cross-references for each CVE's bug-class anchor row
Fundamentals walkthrough	`handouts/cross-chapter-pcap-walkthrough-fundamentals.md`	Baseline-vs-anomaly comparison anchor; students walk the fundamentals before the malformed CVE captures
ADV-101 capstone arc	TBD path; in development	Capstone-tier extension; CVE-2026-5405 is the recommended capstone primitive
FSM Workbench Mode-3 vuln-overlay	TBD path; in development	Forward-stretch visualization layer; consumes the captures + sidecars
Decisions log	`docs/test-range-cve-quartet-2026-05-decisions.md`	D1 RE-101 primary; D6 session-table accounting; D7 cyber-use citation; D8 Mode-3 timing
Range build artifact	Lab environment Docker (instructor-provided)	

Resource	Path	Relationship to this cluster
		Docker compose + Dockerfile.range + entrypoint + 4 trigger scripts; shared base image with per-CVE runtime profile
Source archives	Pre-patch + post-patch Wireshark source archives (instructor-provided)	Mounts read-only into the range container

§7: `--authorized-by` discipline and cyber-use citation footnote

This handout's authoring and the lab cluster's exercise both operate under the academy's `--authorized-by` discipline established by PEN-101 Lab 1 (Statement of Work; Rules of Engagement; legal sign-off from the client's authorized representative; the cohort discipline that ADV-101 capstone work assumes). Every binary-diff exercise, every capture load, and every compiled-binary inspection runs against academy-owned, intentionally-vulnerable Wireshark builds inside the academy `cve-quartet-2026-05-range` Docker container. The discipline is professional habit; the lab cluster trains it.

Per D7 cyber-use citation convention: the academy's authoring of cyber-research-class material operates under Anthropic's acceptable-use cyber-research exception attached to the `munsonj2.0` account. The canonical academy-wide cyber-use citation handout at `handouts/cross-chapter-anthropic-cyber-use-citation.md` (LIVE; shipped as a companion handout) carries the per-handout footnote convention; this handout's footnote cites the canonical citation directly. Students reading this handout encounter the citation pointer at first read; instructors deploying the lab cluster carry the citation in their per-cohort instructor onboarding materials.