ADV-101 Capstone Arc: CVE-2026-5402 TLS ECH Integer-Truncation

3,960 words · ~18 min read 2026-05-07 v1

Course: VCA-ADV-101 (Adversarial Techniques: CVE-to-Tool); Belt-5 capstone application layer **Scope:** Capstone-arc application; engagement-tradecraft depth **Pairs with parent quartet handout:** `cve-lab-wireshark-rce-quartet-2026-05.md` §1 (full CVE-2026-5402 walkthrough) **Prerequisite lab cluster:** `re-101-cve-quartet-binary-diff-lab-cluster.md` §3.1 (RE-101 binary-diff lab on this CVE) **Cyber-use policy:** Anthropic acceptable-use cyber-research exception on `munsonj2.0` per D7 **Version:** 2026-05-07 v1

[Authorized under Anthropic acceptable-use cyber-research exception; see handouts/cross-chapter-anthropic-cyber-use-citation.md for policy details and academy provenance.]

Overview

Field	Value
Tier	capstone-arc application
Course	VCA-ADV-101, Adversarial Techniques: CVE-to-Tool
CVE	CVE-2026-5402, TLS ECH integer-truncation (heap buffer overflow)
Advisory	wnpa-sec-2026-14
Bug class	CWE-190 (integer truncation) cascading to CWE-122 (heap buffer overflow)
Lab harness	`fwlab` / `cve-quartet-2026-05-range` Docker container; academy-isolated
Prerequisite	RE-101 §3.1 binary-diff lab completed; ADV-101 modules 1-N engagement baseline; written cohort cyber-use authorization signed
Engagement depth	Reproduction + primitive characterization + framing; NO weaponized exploit code
ATT&CK techniques	T1190, T1203, T1499 (vocabulary; see §5)
Capture path	`pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5402-trigger-tls-ech-overflow.pcapng` ([PLACEHOLDER] until academy captures are committed)
Range path	Lab environment Docker setup (instructor-provided)
Syntax validated	Manual; Snort 3 / Suricata rule syntax confirmed against sibling handouts

---

§1 Why this capstone arc

CVE-2026-5402 is the primitive-richest of the four-CVE Wireshark quartet disclosed in May 2026, which is what makes it the ADV-101 capstone-arc candidate per curriculum decision D1. The other three quartet members each have one primary primitive shape:

CVE-2026-5403 is a loop-accounting failure; CVE-2026-5405 is a missing-bounds-check; CVE-2026-5656 is a logic bug via path traversal. CVE-2026-5402 chains three compounding defects into one exploit path:

1. **16-bit integer truncation at `extensions_end`.** The TLS dissector reads the ECH extension's length into a `uint16_t` local. Values above 65,535 bytes wrap to a near-zero truncated value. An attacker who controls the ECH extension length field controls this truncated value.
2. **16-bit integer truncation at `outer_offset`.** A second `uint16_t` local holds the offset into the outer ClientHello. The same wrapping applies; a wrapped offset can resolve to a position behind the current buffer pointer, pointing the subsequent copy into adjacent heap memory.
3. **Unsigned underflow in the `hello_length` bounds check.** The check that should refuse to copy bytes past the buffer end computes the remaining space in unsigned arithmetic. When the inner operands would produce a negative difference, unsigned subtraction wraps to `UINT_MAX`, making the check always-true. The dissector then writes attacker-controlled bytes past the allocated transcript buffer.

The three defects compound: the truncation produces a small allocation; the underflow disables the guard that would have caught the small allocation; the copy writes the full attacker-controlled payload. This is not one bug; it is a bug class cascade. Teaching a capstone arc on a three-fault cascade is pedagogically denser than teaching one on a single missing bounds check.

The second reason CVE-2026-5402 is the right capstone-arc choice is its **protocol context**. The bug lives in the TLS dissector, the parser that handles the cryptographic trust anchor for nearly every modern network communication. Students completing this arc learn not only the integer-arithmetic primitive but also the broader principle: trust-anchor components -- TLS handshake parsers, certificate verifiers, session-key derivation paths -- are themselves an attack surface, and the attacker who finds a parsing bug in a trust-anchor component bypasses the cryptographic layer entirely without needing to attack the cryptography. This protocol-context lesson is more valuable than the integer-arithmetic mechanics alone.

The capstone arc sits at **engagement-tradecraft depth**, not advanced worked-PoC level. RE-101 §3.1 (prerequisite) already trained the source-diff fluency and binary-diff methodology. This arc applies that fluency in the engagement-simulation context ADV-101 owns: cohort operationalizes the reproduction, characterizes the primitive,

frames the exploitation path at vocabulary depth, reads the patch, and writes the engagement report. The assessable output is the engagement report in §3.6, not the exploit itself. The no-weaponization discipline in §3.4 is explicit and enforced.

---

§2 Prerequisite knowledge

Students entering this capstone arc are expected to have completed the following prerequisites. The arc does not re-teach them; it applies them.

RE-101 §3.1: CVE-2026-5402 binary-diff lab. Students must have completed the 90-minute RE-101 lab on this CVE: source diff between Wireshark 4.6.4 and 4.6.5 at `epan/dissectors/packet-tls.c`, capture loading and prediction verification, and compiled-binary diff inspection in their preferred reverser. The arc presumes source-diff fluency with the three defects named in §1. Students who completed the RE-101 lab but want a quick refresher should re-read `re-101-cve-quartet-binary-diff-lab-cluster.md` §3.1 before the §3.2 reproduction step.

SEC-101 Module 4 vocabulary: integer-arithmetic class. Students must have the vocabulary for integer-overflow / truncation / underflow at the level of `cross-chapter-cve-class-vocabulary-reference.md` §2 (CWE-190 entry). The arc uses this vocabulary throughout without defining it.

ADV-101 modules 1-N engagement-tradecraft baseline. Students must have completed the engagement-lifecycle framing (Week 1), the authorization-gated tool engineering sequence (Weeks 4-7), the CVSS scoring lab (Week 8), and the CVD disclosure draft (Week 9). The §3.6 engagement-report deliverable applies the ADV-101 §7 report shape directly; students who have not internalized that shape will struggle with §3.6.

Current-cohort cyber-use authorization signed. Every student in the cohort must have a signed current-cohort cyber-use authorization on file before any step in §3 begins. The authorization covers academy test-range use against academy-owned Wireshark 4.6.4 / 4.4.14 instances in the `cve-quartet-2026-05-range` Docker container. The authorization document cites this handout by filename. See §6 for the `--authorized-by` discipline applied throughout.

---

§3 The capstone arc

The arc runs as six graded steps across two to three cohort lab sessions (~30-60 minutes each, 90 minutes for §3.3 and §3.6). Steps are sequential; each step's output becomes the input for the next. The assessable output is the engagement report in §3.6; the earlier steps are the evidence base for that report.

Cohort lab environment. All steps run against academy-owned Wireshark builds in the `cve-quartet-2026-05-range` Docker container per D5. The pre-patch vulnerable build is at `/opt/wireshark-4.6.4/bin/wireshark` (and its `epan` shared library). The post-patch build is at `/opt/wireshark-4.6.5/bin/wireshark`. The container is the authorized lab surface; student workstation Wireshark installations are never the test target.

§3.1 Trigger discovery (~30 min)

Goal. Cohort identifies the malformed ECH extension on the wire and correlates wire bytes to the dissector source path.

Steps.

1. Retrieve the academy-captured PCAP at the canonical path: `pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5402-trigger-tls-ech-overflow.pcapng`. [PLACEHOLDER: this path is the academy capture; until it is committed, use the upstream-disclosure-attached capture if the Wireshark advisory at `wnpa-sec-2026-14` carries one, or defer this step until the capture is available.]
2. Load the capture in the pre-patch Wireshark build (`/opt/wireshark-4.6.4/bin/wireshark`). Apply the display filter `tls.handshake.extension.type == 0xfe0d` to surface the ECH extension (TLS extension type code `0xFE 0x0D`). Confirm the malformed packet appears.
3. Walk the TLS record stack in the packet: record-layer header, handshake message header, ClientHello, extensions list, ECH extension. Note the ECH extension's declared length fields. Record the specific field values that will drive the truncation.
4. Correlate the observed ECH extension length fields back to the source diff from RE-101 §3.1. Identify which field value maps to `extensions_end` truncation and which maps to `outer_offset` truncation. Write a one-paragraph correlation statement in the lab notebook: "The wire field at offset X in the ECH extension drives the `extensions_end` truncation because..."

Output. Lab-notebook correlation statement mapping wire bytes to dissector source path. This is the evidence basis for §3.6 "reproduction steps."

§3.2 Reproduction (~45 min)

Goal. Cohort spins the academy test range and reproduces the crash using the trigger capture, then varies the truncation parameters to observe per-variant behavior.

Steps.

1. Spin the `cve-quartet-2026-05-range` Docker container per the lab environment setup guide (instructor-provided). Confirm the pre-patch Wireshark build loads and the range container is isolated from production network segments. Record the per-session authorization statement in the lab notebook before any traffic is generated or loaded.
2. Load the trigger capture in the pre-patch build. Confirm the dissector reaches the vulnerable code path: either a crash (SIGABRT / SIGSEGV from heap corruption), a silent heap-state corruption detectable via AddressSanitizer output if the range build carries ASAN, or an anomalous dissection result. Record the observed behavior.
3. Modify the trigger to vary the ECH extension length field values. The variation space has three dimensions: (a) `extensions_end` field values above and below the 65,535 truncation boundary; (b) `outer_offset` field values in the same range; (c) `hello_length` field values that interact with the bounds-check underflow. For each variant, observe and record: does the dissector crash? Does it produce an anomalous dissection? Does it dissect cleanly? A 3x3 grid of variants (low / at-boundary / high for two of the three parameters) is a sufficient minimum for this step.
4. Synthesize the per-variant results into a "trigger parameter space" table in the lab notebook: which combinations of field values produce a crash, which produce silent corruption, and which dissect cleanly. This table is the primary technical evidence for §3.6.

Output. Per-session authorization record, trigger PCAP + variant set, behavioral observation table.

§3.3 Primitive characterization (~60-90 min)

Goal. Cohort characterizes the heap-overwrite primitive: how many bytes can be written; at what heap offset; what data is attacker-controlled vs constrained; what defensive layers are in scope.

Steps.

1. **Write volume.** From the §3.2 behavior observations and the source diff, determine the theoretical maximum overwrite volume. The dissector's copy loop writes until the `hello_length` underflow check passes (which it always does post-underflow) or until

the TLS record runs out of bytes. In practice, the attacker controls the ECH extension payload length up to the outer TLS record length limit. Characterize: "An attacker who controls the ECH extension payload up to N bytes can write up to M bytes past the transcript buffer boundary."

2. **Write offset.** The overwrite begins at the transcript buffer's allocated end. From the source diff, the transcript buffer is allocated from the ECH extension's declared `hello_length` (the truncated value). The overwrite offset is therefore `allocated_size` bytes past the buffer start, which is `truncated_hello_length`. Characterize: "The overwrite begins at heap offset equal to the attacker-controlled truncated `hello_length` value."
3. **Attacker-controlled vs constrained bytes.** The bytes written in the overwrite come from the ECH extension's inner ClientHello payload. These bytes are fully attacker-controlled within the constraints of TLS ClientHello grammar -- the dissector parses them before the overwrite occurs, so they must be syntactically valid TLS extension data. Characterize the degree of control: which bytes are fixed (TLS handshake header, record-layer framing), which are partially constrained (extension type codes, length fields the dissector validates before the vulnerable path), and which are fully attacker-controlled (extension payload bytes past the point of truncation).
4. **Defensive layers in scope.** For the academy `cve-quartet-2026-05-range` build (Linux, x86-64, standard GCC + glibc), the heap metadata layout is glibc `ptmalloc2`. Relevant defensive layers: ASLR (base addresses randomized; overwrite must target a deterministic or predictable offset); stack canaries (the overwrite is on the heap, not the stack; canaries do not protect heap metadata); heap-coalescing (adjacent free-chunk metadata can be corrupted if the overwrite reaches the next chunk boundary). Name each layer, whether it applies to this primitive, and whether it presents a bypass requirement.

Output. Primitive characterization table: write volume / write offset / attacker-controlled bytes / defensive-layer analysis. This is the "technical findings" section of the §3.6 engagement report.

§3.4 Exploitation framing (NO weaponization) (~30 min)

Goal. Cohort frames how an attacker would chain the heap-overwrite primitive into RCE on a real target, at vocabulary depth only. No publishable exploit code is produced.

Explicit constraint. This step produces **no exploit code, no shellcode, no ROP chain, no weaponized PCAP that delivers a payload beyond academic crash-reproduction, and no tooling that could be used against a non-academy target.** The framing is discussion-register engagement: cohort names the exploitation strategy, walks the vocabulary of how heap primitives are chained, and identifies which published exploitation technique classes apply. The deliverable is a written framing paragraph, not working exploit code.

Framing discussion (instructor-led).

The heap-overwrite primitive produced by CVE-2026-5402 is a classical "write-past-buffer-end on the heap" shape. Modern exploitation of such primitives on Linux x86-64 against glibc `ptmalloc2` follows one of several well-documented technique classes, all covered at vocabulary depth in academic exploit-development literature:

- **Heap metadata corruption.** If the overwrite reaches the next free chunk's `fd` / `bk` forward-and-backward-link pointers, a subsequent `malloc()` call can be directed to an attacker-controlled address, achieving an arbitrary write primitive. The technique is described in academic and conference literature under "unlink attack" / "house of" technique families. Mitigations: glibc safe-unlink checks (verify `fd->bk == chunk` and `bk->fd == chunk`); tcache double-free detection; heap hardening patches in hardened distributions.
- **Vtable / function-pointer overwrite.** If the adjacent heap allocation after the transcript buffer holds a C++ object with a vtable pointer, or a function-pointer struct (Wireshark uses callback-heavy plugin architecture), and the overwrite reaches that pointer, a subsequent virtual-dispatch or callback invocation jumps to attacker-controlled code. The technique presumes ASLR bypass (or a partial-overwrite with predictable bits) and a controlled object layout.
- **Heap grooming.** The attacker sends multiple TLS sessions to arrange heap allocations in a predictable layout before delivering the overwrite. Grooming is required when the target allocation is not adjacent to the transcript buffer in the default allocator layout. Grooming techniques for ptmalloc2 are described in academic literature; the specific technique for this primitive depends on Wireshark's per-session allocation pattern, which RE-201-level RE of the allocator behavior would characterize.

Cohort deliverable. A one-paragraph framing statement: "An attacker who controls the write-past-buffer primitive from CVE-2026-5402 would attempt [technique class] to chain it into RCE. This requires [preconditions]. The defensive layers that block this

chain are [list]. The technique class is named in published literature as [citation-class name]." This framing at vocabulary depth is the "exploitation path" section of the §3.6 engagement report.

No-weaponization sign-off. Each student's lab-notebook entry for §3.4 includes an explicit one-line sign-off: "No exploit code, shellcode, or weaponized payload was produced in this step. This entry is framing at vocabulary depth only." The instructor countersigns before the cohort advances to §3.5.

§3.5 Mitigation analysis (~30 min)

Goal. Cohort reads the Wireshark 4.6.4 -> 4.6.5 patch, explains why it closes the primitive, and identifies compile-time and runtime defense-in-depth layers.

Steps.

1. **Patch analysis.** Run the source diff at `epan/dissectors/packet-tls.c`:

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

   (RE-101 §3.1 already covered this diff; revisit it with the primitive-characterization context from §3.3.) Identify the three specific changes: (a) the `uint16_t` locals for `extensions_end` and `outer_offset` replaced with `unsigned int` / `size_t`; (b) the `hello_length` bounds check rewritten in a width that cannot underflow; (c) any added early-exit that refuses to enter the copy loop if the size arithmetic produced an impossible value. For each change, write one sentence: "Change X closes Defect Y because..."

2. **Defense-in-depth layer 1: compile-time.** Identify one compile-time tool or flag that would have caught the bug before shipping. Options: `-Wconversion` (GCC/clang warn on implicit narrowing conversions, including `uint32_t` to `uint16_t` without explicit cast); static analyzers (Coverity, CodeQL, Clang Static Analyzer can detect truncation in size-arithmetic paths); sanitizers (`-fsanitize=undefined` would fire an UBSan report on the `uint16_t` wrapping in debug builds). Document which of these the Wireshark project does or does not currently run in CI, if public CI configuration is available in the repository.

3. **Defense-in-depth layer 2: runtime.** Identify one runtime mitigation that would have reduced the exploitability of the primitive even without the patch. Options: AddressSanitizer in the build (intercepts the out-of-bounds write immediately; not practical in production but appropriate in fuzz-test builds); `_FORTIFY_SOURCE=2` (hardens certain glibc functions called in the copy path); running Wireshark in a sandbox (seccomp profile, AppArmor policy, or `bubblewrap`-based sandboxing that restricts syscalls and filesystem access would limit the blast radius of a successful heap corruption). Document which option a production SOC would most realistically deploy and why.

Output. Patch-analysis writeup covering the three changes and their closure rationale; defense-in-depth layer 1 and layer 2 selections with justification. This is the "remediation and mitigation" section of the §3.6 engagement report.

§3.6 Engagement report drafting (~60 min)

Goal. Cohort writes a 2-page engagement report in the ADV-101 §7 report shape, synthesizing the evidence from §3.1 through §3.5 into a deliverable that a professional analyst could hand to a client.

Report structure (ADV-101 §7 shape).

The engagement report follows the same four-section shape as the CERT/CC CVD submission template:

Section 1: Executive summary. One paragraph. State the vulnerability (CVE-2026-5402, TLS dissector, Wireshark 4.6.0-4.6.4), the severity (CVSS 8.8, heap buffer overflow, potential RCE), the attack vector (malformed PCAP opened offline or malicious TLS handshake packet captured live), and the recommended action (upgrade to Wireshark 4.6.5 / 4.4.15; apply the mitigation layers named in Section 4). The executive summary is for a technically literate manager who will not read the rest of the report; it must stand alone.

Section 2: Technical findings. Two to three paragraphs. Describe the three compounding defects (§1, §3.3 evidence), the trigger conditions (§3.1 evidence: ECH extension length field values that drive truncation), the primitive characterization (§3.3 output: write volume, write offset, attacker-controlled bytes, defensive layers in scope), and the exploitation framing (§3.4 paragraph: technique class, preconditions, blocking defensive layers). Do not reproduce exploit code. The report section for findings should read like a coordinated-disclosure technical advisory: precise enough that an independent researcher can reproduce the finding, not so precise that it constitutes a step-by-step exploitation tutorial.

Section 3: Reproduction steps. Numbered list of 5-8 steps. A qualified engineer reading only this section, working in an authorized environment, should be able to confirm the finding. Reference the academy test range (`cve-quartet-2026-05-range/README.md`), the trigger PCAP path (`pcap-tools/fixtures/cve-quartet-2026-05/cve-2026-5402-trigger-tls-ech-overflow.pcapng`), and the display filter (`tls.handshake.extension.type == 0xfe0d`). Include the behavioral observation (crash / ASAN alert / dissection anomaly). Note the `--authorized-by` prerequisite: these steps must only be performed against an authorized lab target.

Section 4: Recommendations. Short list. Minimum: (1) upgrade to 4.6.5 / 4.4.15; (2) deploy compile-time static analysis (§3.5 layer 1); (3) runtime mitigation (§3.5 layer 2). Optional: (4) capture-file sandboxing for analyst workstations opening unknown `.pcapng` files; (5) SOC tshark-first workflow for triage; (6) consider disabling ECH decoding in Wireshark preferences if analysts do not routinely review ECH-using traffic.

Section 5: References. CVE record, Wireshark advisory (wnpa-sec-2026-14), issue tracker (https://gitlab.com/wireshark/wireshark/-/issues/21090), Wireshark 4.6.5 release notes, and the three academy handouts cross-referenced in this arc (parent quartet handout, RE-101 lab cluster, this handout).

document overview section for the report. The report's own header section carries a `--authorized-by` citation block:

```
--authorized-by: VCA-ADV-101 capstone cohort [cohort ID]; [instructor name]; [date]
Scope: CVE-2026-5402 TLS ECH integer-truncation; academy cve-quartet-2026-05-range
container;
       Wireshark 4.6.4 (pre-patch vulnerable build) at /opt/wireshark-4.6.4/bin/
wireshark
Policy: handouts/cross-chapter-anthropic-cyber-use-citation.md
```

Assessment rubric for §3.6. 25 points; evaluated on five dimensions:

- Executive summary stands alone (5 pts): severity, vector, and recommended action clear to a non-expert reader; no undefined jargon.
- Technical findings are precise and non-weaponizing (5 pts): three defects correctly attributed; primitive characterized with write-volume and offset; no exploit code.
- Reproduction steps are independently executable (5 pts): a qualified engineer could confirm the finding from only the steps list; authorization prerequisite stated.

- Recommendations are actionable (5 pts): each recommendation is specific, not generic ("apply patches" is generic; "upgrade to Wireshark 4.6.5 and verify with `wireshark --version`" is specific).
- Register and discipline (5 pts): report reads at engagement-tradecraft depth (not student writeup register); `--authorized-by` block present; references section complete.

---

§4 Cross-track integration

Course / surface	CVE-2026-5402 appearance	Register
`vca-mini-wireshark-cves-2026-05` mini-page	Recognize-tier entry: one-paragraph CVE summary; link to parent quartet handout	Catalog-register; vocabulary-fluency
VCA-RE-101 §3.1	Binary-diff lab: source diff at `packet-tls.c`; capture loading; compiled-binary diff in reverser; assessment rubric 5D x 25 pts	Methodology-tier; binary-diff register; prerequisite for this arc
VCA-ADV-101 (this handout)	Capstone arc: trigger discovery, reproduction, primitive characterization, exploitation framing (no weaponization), mitigation analysis, engagement report	Engagement-tradecraft register; assessable deliverable = the report
VCA-SEC-101	Sidebar in Module 4 on "trusting wire-format length fields"; integer-truncation class anchor; detection-rule shape (see Suricata + Snort 3 rule sibling handouts)	Defensive-fluency register; vocabulary + detection primitives

---

§5 ATT&CK technique IDs surfaced

The following ATT&CK Enterprise technique IDs appear in the exploitation framing context of this arc. All are at vocabulary depth. T1190 and T1203 are named explicitly in `cross-chapter-engagement-attck-vocabulary-reference.md` §2.1 and §2.2; T1499 is in the Impact tactic (TA0040) covered at recognize-only register in §2.12 of that handout. Technique IDs follow MITRE ATT&CK Enterprise matrix v15.x.

Technique ID	Name	Tactic	Relevance to CVE-2026-5402
T1190	Exploit Public-Facing Application	Initial Access (TA0001)	If the target is a `tshark` instance processing live-captured TLS traffic from an internet-facing tap, a malicious TLS ClientHello from a remote attacker constitutes exploitation of a public-facing application. The attacker does not need local access; the malformed packet reaches the vulnerable dissector via the analyst's capture path.
T1203	Exploitation for Client Execution	Execution (TA0002)	If the delivery vector is a malicious `.pcapng` file that an analyst opens in Wireshark on their workstation (the "more dangerous delivery vector" per the parent handout §1.3), the technique maps to T1203: the malicious file exploits the client-side Wireshark process to execute attacker-controlled code in the analyst's session. The analyst is the victim; the PCAP file is the malicious document.
T1499	Endpoint Denial of Service	Impact (TA0040)	A crash-only exploitation (no RCE achieved) that kills the Wireshark or `tshark` process constitutes T1499.004 (Application Exhaustion Flood sub-technique is adjacent; the direct crash shape maps to the DoS impact). An attacker who cannot achieve RCE can still use CVE-2026-5402 to crash analyst tooling during an active incident response, degrading the analyst's visibility into concurrent attacker activity.

Vocabulary-register note. Naming technique IDs is not the same as reproducing the techniques. The three IDs above describe the attack *shape* that CVE-2026-5402 enables; the cohort's capstone arc reproduces the vulnerability itself (not an operational attack) against an authorized academy lab target. The technique-ID vocabulary is present so the §3.6 engagement report can cite the relevant ATT&CK context, which is standard practice in professional vulnerability advisory reports.

---

§6 `--authorized-by` discipline

Every step in §3 is academy-authorized offensive-reproduction work conducted under the cyber-research exception documented at `handouts/cross-chapter-anthropic-cyber-use-citation.md`. The discipline is not optional and is not waived at any step.

Cohort-level authorization. Each student has a signed current-cohort cyber-use authorization on file before §3 begins (prerequisite per §2). The authorization names the academy test range, the pre-patch Wireshark builds, and the specific CVEs in scope. Reproduction against any target not named in the authorization is out of scope.

Per-session authorization log. Each lab session that includes steps from §3 opens with a one-line authorization record in the lab notebook: the date, the cohort ID, the instructor name, the step being performed, and the target (the `cve-quartet-2026-05-range` container). This record is the per-session half of the `--authorized-by` discipline; it pairs with the signed cohort authorization to provide both standing authorization and per-session invocation.

`--authorized-by` annotation in the engagement report. The §3.6 engagement report carries the `--authorized-by` block shown in §3.6's overview-section example. The report is the capstone deliverable; an engagement report without an authorization block fails the engagement-report rubric's "format and discipline" dimension.

No-weaponization perimeter. The ADV-101 capstone arc explicitly prohibits: publishable exploit code, shellcode, ROP chains, weaponized PCAPs that deliver payloads beyond academic crash-reproduction, and any tooling that could be used against a non-academy target. The §3.4 no-weaponization sign-off is countersigned by the instructor before §3.5 proceeds. If a student produces output that approaches this boundary, the instructor intervenes and the student revises; out-of-scope production does not count toward the capstone grade and must be deleted from the lab notebook.

---

§7 Forward-pointers

Resource	Path	Role in this arc
Parent quartet handout	`handouts/cve-lab-wireshark-rce-quartet-2026-05.md` §1	Primary technical walkthrough: background, vulnerability details, trigger conditions, code-level discussion, defensive analysis
RE-101 binary-diff lab cluster	`handouts/re-101-cve-quartet-binary-diff-lab-cluster.md` §3.1	Prerequisite lab: source-diff fluency, capture loading, compiled-binary diff inspection, 5D assessment rubric
CVE-class vocabulary reference	`handouts/cross-chapter-cve-class-vocabulary-reference.md` §2	Integer-truncation class anchor (CWE-190 -> CWE-122 row); self-homework reading
Suricata rules reference	`handouts/cve-suricata-rules-reference-wireshark-quartet-2026-05.md` §1	Detection-rule template for CVE-2026-5402; §3.6 report recommendations
Snort 3 rules reference	`handouts/cve-snort3-rules-reference-wireshark-quartet-2026-05.md` §3.1	Snort 3 sister rule template; §3.6 report recommendations
Engagement + ATT&CK vocabulary reference	`handouts/cross-chapter-engagement-attck-vocabulary-reference.md` §2	T1190 / T1203 / T1499 technique ID descriptions; engagement-report vocabulary
Cyber-use citation	`handouts/cross-chapter-anthropic-cyber-use-citation.md`	Policy framework for all arc steps; footnote forward-pointer per D7
Mini catalog page	`/vca-mini-wireshark-cves-2026-05`	Public-facing entry point; vocabulary-tier on-ramp
FSM Workbench Mode-3 vuln-overlay	`/workbench/fsm/` (forward-stretch)	TLS state-machine visualization with pre-patch vs post-patch dissector state comparison; enables side-by-side FSM diff that illustrates which states the patch removes; not yet shipped as of 2026-05-07
Per-CVE LMS-side walkthroughs	Future infrastructure	Advanced worked-PoC walkthroughs for each quartet CVE; not yet shipped; this arc is the foundational bridge to that future surface