CVE Suricata Rules Reference: Wireshark RCE Quartet (2026-05)

3,883 words · ~18 min read 2026-05-07 v2 (2026-05-07: cyber-use footnote per D7)

Course companion for: vca-mini-wireshark-cves-2026-05 (catalog page) + ADV-101 / SEC-101 / NET-101 **Scope:** Defensive: rule templates, rationale, false-positive considerations, tuning notes **Parent handout:** `cve-lab-wireshark-rce-quartet-2026-05.md` (full CVE walkthroughs) **Version:** 2026-05-07 v2 (2026-05-07: cyber-use footnote per D7)

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

Overview

This handout materializes the detection-rule layer for the four Wireshark CVEs disclosed in May 2026 and covered in full in `cve-lab-wireshark-rce-quartet-2026-05.md`. Each section below gives a rule template for one CVE in Suricata 7.x syntax, a detection rationale that walks every line of the rule, false-positive considerations for a SOC deployment, and tuning notes.

This is a rule-reference handout for teaching, not a production rule set. Every rule below is a **template that teaches the shape** of detection against the corresponding bug class. The templates use Suricata's standard variable macros (`$HOME_NET`, `$EXTERNAL_NET`), carry placeholder SIDs with a note to replace them, and target the structural pattern of each vulnerability rather than byte-exact production signatures. Students reading this handout should leave understanding why each rule is shaped the way it is; students adapting rules for production SOC work own their own tuning and change-management discipline beyond what this handout describes.

The `--authorized-by` discipline that ADV-101 enforces applies unchanged: rules deploy against lab-owned, intentionally-vulnerable Wireshark instances inside the academy `fwlab` container or against pre-recorded `.pcap` / `.pcapng` files supplied by the instructor. Production analyst workstations are never the test target (see §7).

§0.5: What This Rule Reference Covers

A foundational Suricata rule reference teaches the structural elements of a detection rule matched to a specific bug-class shape, not the syntax of Suricata's rule language in the abstract. The five structural elements this handout addresses for each CVE are:

1. **Protocol family, direction, port.** Which transport carries the attack? Which direction does the malicious payload travel? Is there a canonical port (RDP on 3389; TLS on any port; RTP on any UDP port)?
2. **Dissector-shape pattern.** What byte-level invariant distinguishes a malicious packet from a legitimate one? For the quartet's three heap-overflow CVEs, the invariant is a length-field or frame-count value that exceeds the dissector's output-buffer capacity. For the zip-slip CVE, the invariant is a `../` path-traversal sequence in a ZIP entry filename.
3. **Suricata primitives.** Which Suricata keyword matches the invariant? `byte_test` for a length-field consistency check; `content` + `threshold` for a volumetric anomaly; `file.data` + `content` for an archive-path traversal in a file transfer. Each primitive is explained in the detection rationale section.
4. **Metadata and classtype discipline.** Every template carries `metadata: cve CVE-XXXX-YYYYY, ref wnpa-sec-XXXX-XX` and a canonical Suricata `classtype`. The SID is a placeholder in the `1000XXXX` range; site-specific SID allocation follows the Suricata community SID convention.
5. **False-positive surface.** Every rule template will fire on some legitimate traffic. The false-positive section for each CVE names where, and the tuning section names the SOC strategies that maintain precision.

Production rule authoring (writing rules from scratch against packet captures and dissector source patches) is taught in ADV-101 capstone work and RE-101 binary-diffing exercises.

§1: CVE-2026-5402, TLS Dissector ECH Integer Truncation

§1.1 CVE Recap

Field	Value
CVE	CVE-2026-5402
Advisory	wnpa-sec-2026-14
Bug class	Heap buffer overflow via integer truncation (CWE-190 -> CWE-122)
Affected	Wireshark 4.6.0 through 4.6.4; 4.7.0rc0 development builds
Fixed in	Wireshark 4.6.5
CVSS 3.1	8.8 High
Discoverer	Duc Anh Nguyen

Full walkthrough: parent handout §1.

§1.2 Bug-Class Shape

The TLS dissector's ECH transcript-reconstruction loop reads the ECH extension's length fields into `uint16_t` locals. Values above 32 767 wrap when used in signed comparisons; values above 65 535 wrap to near-zero when held in unsigned 16-bit arithmetic. The attacker's weapon is a TLS ClientHello whose ECH extension (IETF extension type `0xFE 0x0D`) carries inner length fields crafted to exploit one of three distinct integer-width failures: `extensions_end` truncation, `outer_offset` truncation, and a `hello_length` underflow in the bounds check. After the truncation, the dissector's copy loop writes attacker-controlled bytes past the allocated transcript buffer on the heap. The delivery vector is either a live TLS handshake captured on the wire or a crafted `.pcapng` file opened offline.

§1.3 Suricata Rule Template

```
alert tls $EXTERNAL_NET any -> $HOME_NET any (
    msg:"VCA-ADV CVE-2026-5402 TLS ECH extension oversized length field integer-truncation shape wnpa-sec-2026-14";
    flow:to_server,established;
    tls.handshake.type:1;
    content:"|fe 0d|";
    byte_test:2,>,16383,0,relative,big-endian;
    metadata: cve CVE-2026-5402, ref wnpa-sec-2026-14;
    classtype:protocol-command-decode;
    # replace SID with site-allocated range per Suricata SID convention
    sid:10001402; rev:1;
)
```

§1.4 Detection Rationale

`alert tls ... -> $HOME_NET any` scopes the rule to TLS traffic arriving at any port on the monitored network. TLS is not port-bound in the modern deployment landscape (HTTPS on 443, DNS-over-TLS on 853, IMAP-over-TLS on 993, custom application ports), so `any` is correct here rather than a port restriction.

`flow:to_server,established` limits the rule to the client-to-server direction on an established TCP session. A TLS ClientHello is always client-to-server; filtering on direction halves the rule's evaluation surface.

`tls.handshake.type:1` uses Suricata's built-in TLS parser to match only TLS handshake messages of type 1 (ClientHello). This is more precise than a raw content match for the ClientHello handshake byte (0x01) and avoids false matches on TLS application data records that happen to contain 0x01.

`content:"|fe 0d|"` matches the ECH extension type bytes (0xFE 0x0D = decimal 65037) in the extensions list of the ClientHello. After this match, Suricata's cursor sits at the end of the matched bytes.

`byte_test:2,>,16383,0,relative,big-endian` reads the 2 bytes at offset 0 from the current cursor (the extension_length field immediately following the extension_type) and tests whether the value exceeds 16 383. Values above 16 383 are candidates for 16-bit arithmetic overflow when combined with typical offset values inside the ECH processing loop. The threshold is conservative; a production rule would narrow it after baseline-profiling legitimate ECH extension sizes on the monitored network.

`metadata: cve CVE-2026-5402, ref wnpa-sec-2026-14` provides the upstream references for alert triage.

`classtype:protocol-command-decode` is the canonical Suricata classtype for protocol-parsing vulnerabilities (decoding a protocol command caused the alert).

§1.5 False-Positive Considerations

Legitimate ECH extensions carry substantial inner payloads: the encrypted inner ClientHello is typically 128-512 bytes, and implementations may pad to power-of-two boundaries. An extension_length value above 16 383 is uncommon in legitimate ECH traffic but is possible in edge cases (very long inner ClientHello with many extensions; padding-aggressive implementations). The primary false-positive surface is any TLS client that sends legitimately large ECH payloads.

A SOC deploying this rule should first baseline ECH extension lengths seen in production TLS traffic. If the 99th-percentile legitimate ECH extension_length is well below 8 192, the `byte_test` threshold can be raised to 8 191 for lower false-positive rates. If ECH traffic is uncommon on the monitored network, the rule can be narrowed to specific monitored hosts (replace `$HOME_NET any` with the sensor placement's relevant CIDR).

§1.6 Tuning Notes

This rule runs most effectively in a monitor-mode IDS placement that sees TLS handshake traffic before decryption. An inline IPS placement would require careful false-positive analysis before enabling drop mode, because blocking a legitimate ECH handshake denies TLS service to the client.

Recommended deployment: monitor mode, `alert` action, with a `threshold: type limit, track by_src, count 1, seconds 60` suppression to reduce alert volume from scanning sources. `metadata: signature_severity High; metadata: priority 2;` is appropriate given the 8.8 CVSS score and the heap-overflow impact.

---

§2: CVE-2026-5403, SBC Codec Loop Accounting Overflow

§2.1 CVE Recap

Field	Value
CVE	CVE-2026-5403
Advisory	wnpa-sec-2026-16
Bug class	Heap buffer overflow via multi-frame decode accounting failure (CWE-122 / CWE-787)
Affected	Wireshark 4.6.0 through 4.6.4; 4.4.0 through 4.4.14
Fixed in	Wireshark 4.6.5; 4.4.15
Discoverer	Duc Anh Nguyen

Full walkthrough: parent handout §2.

§2.2 Bug-Class Shape

The SBC codec plugin's `codec_sbc_decode()` function decodes Bluetooth audio frames into a fixed 8 192-byte PCM output buffer. The decode loop advances input and output pointers on each iteration but never subtracts consumed bytes from a running remaining-capacity counter and never checks that capacity before allowing the next decode call to write. An attacker supplies an SBC stream containing enough frames that the cumulative decoded PCM output exceeds 8 192 bytes. The loop continues past the end of the heap-allocated output buffer, writing PCM samples into adjacent heap memory. The delivery vector is an RTP-over-Bluetooth capture file opened for audio playback in Wireshark, or a synthetic RTP payload crafted in a `.pcapng`.

§2.3 Suricata Rule Template

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (
    msg:"VCA-ADV CVE-2026-5403 anomalous Bluetooth-audio RTP stream with SBC frame
burst loop-accounting-overflow shape wnpa-sec-2026-16";
    flow:to_server,established;
    content:"|9c|";
    threshold: type both, track by_src, count 20, seconds 1;
    metadata: cve CVE-2026-5403, ref wnpa-sec-2026-16;
    classtype:protocol-command-decode;
    # replace SID with site-allocated range per Suricata SID convention
    sid:10001403; rev:1;
)
```

§2.4 Detection Rationale

`alert udp $EXTERNAL_NET any -> $HOME_NET any` scopes to UDP in either direction. RTP over Bluetooth audio does not have a canonical reserved port; the outer protocol is UDP and the payload carries RTP-framed SBC audio. The direction `to_server` in Suricata's UDP flow tracking matches the sending direction for the capture.

`flow:to_server,established` requires an established UDP flow. This requires Suricata's UDP flow tracking to be active (which is Suricata's default when `--runmode` is set to `workers` or `autofp`).

`content:"|9c|"` matches the SBC sync byte (0x9C). Every SBC audio frame begins with 0x9C; its presence in a UDP payload is a reliable indicator of SBC-encoded audio. The sync byte is a single distinctive byte; a false-positive match on non-SBC traffic containing 0x9C is possible but is narrowed significantly by the threshold below.

`threshold: type both, track by_src, count 20, seconds 1` is the shape that makes this rule detect the overflow-triggering condition rather than any single SBC packet. The vulnerability requires enough SBC frames to accumulate more than 8 192 bytes of decoded PCM output. A burst of 20+ UDP packets containing SBC sync bytes from the same source within a 1-second window is anomalous; legitimate Bluetooth audio traffic at A2DP frame rates does not typically produce this burst pattern in a network capture. `type both` fires once per threshold crossing per direction rather than suppressing after first fire.

The detection rationale here is volumetric anomaly detection: SBC-over-RTP is uncommon on enterprise networks, and a burst large enough to trigger the accounting failure is doubly suspicious.

§2.5 False-Positive Considerations

Enterprise environments that deploy Bluetooth-audio-to-network bridges (some VoIP conference-room hardware routes Bluetooth audio as RTP over Wi-Fi) will see legitimate SBC sync bytes in UDP traffic. The threshold count of 20 is calibrated for the specific overflow condition; in a high-density A2DP deployment, legitimate audio frames will cross this threshold during normal operation.

SOCs in environments with legitimate A2DP traffic should raise the threshold significantly (200+ per second) or apply a suppression list for known A2DP bridge MAC/IP addresses. In environments where Bluetooth audio over RTP is not expected at all, the threshold can be lowered to 1 per second (effectively alerting on any SBC sync byte in UDP traffic), turning this into a strict anomaly-detection rule.

§2.6 Tuning Notes

This rule is most useful as a monitoring rule in environments where SBC-over-RTP is entirely unexpected. In those environments, even a single match warrants investigation. For environments where Bluetooth audio traverses the monitored network segment, coordinate with the network operations team to identify the source IP ranges for known A2DP bridges and apply Suricata `suppress` entries for those ranges.

`metadata: signature_severity High; metadata: priority 2;` appropriate given heap-overflow impact. In offline-pcap-replay mode (replaying a captured file through Suricata), this rule will fire on any SBC-containing capture regardless of session history; adjust threshold accordingly for pcap-replay workflows.

---

§3: CVE-2026-5405, RDP ZGFX Uncompressed-Path Missing Bounds Check

§3.1 CVE Recap

Field	Value
CVE	CVE-2026-5405
Advisory	wnpa-sec-2026-17
Bug class	Heap buffer overflow via missing bounds check on uncompressed path (CWE-122 + CWE-120)
Affected	Wireshark 4.6.0 through 4.6.4; 4.4.0 through 4.4.14
Fixed in	Wireshark 4.6.5; 4.4.15
CVSS 3.1	8.8 High
Discoverer	Duc Anh Nguyen
Analogous CVEs	FreeRDP CVE-2022-39316, CVE-2022-39320 (same ZGFX uncompressed-path shape)

Full walkthrough: parent handout §3.

§3.2 Bug-Class Shape

The RDP dissector's `rdp8_decompress_segment()` function dispatches on a ZGFX segment descriptor byte. Compressed segments pass through helper functions that individually enforce output-buffer bounds checks against a 65 536-byte fixed `outputSegment` buffer. The uncompressed path is a single `tvb_memcpy()` call with no length check: if the ZGFX segment's declared payload length exceeds 65 536, the copy runs past the end of the output buffer. The asymmetry is the vulnerability: the complex compressed path has careful bounds discipline; the simple uncompressed path has none. The delivery vector is a crafted RDP capture containing a ZGFX segment with the uncompressed flag set and a payload-length field above 65 536.

§3.3 Suricata Rule Template

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 3389 (
    msg:"VCA-ADV CVE-2026-5405 RDP ZGFX uncompressed-path oversized payload
missing-bounds-check shape wnpa-sec-2026-17";
    flow:to_server,established;
    content:"|03 00|"; depth:2;
    content:"|00|"; within:128;
    byte_test:4,>,65535,1,relative,big-endian;
    metadata: cve CVE-2026-5405, ref wnpa-sec-2026-17;
    classtype:protocol-command-decode;
    # replace SID with site-allocated range per Suricata SID convention
    sid:10001405; rev:1;
)
```

§3.4 Detection Rationale

`alert tcp ... -> $HOME_NET 3389` scopes to TCP traffic destined for port 3389, the IANA-assigned RDP port. Standard RDP deployments listen on 3389; environments that run RDP on non-standard ports require adjusting the destination port.

`flow:to_server,established` matches client-to-server on an established TCP session. A malicious RDP ZGFX segment travels from the attacker (client) to the RDP server; in the analyst-exploitation scenario, the attacker crafts an `.pcap` that the analyst opens in Wireshark, but the network IDS would see the same traffic on the wire.

`content:"|03 00|"; depth:2;` matches the TPKT header preamble: version byte 0x03, reserved byte 0x00. Every TPKT-framed RDP packet begins with these two bytes; this anchor scopes the rule to TPKT-over-TCP traffic rather than matching all TCP port-3389 traffic.

`content:"|00|"; within:128;` matches a null byte (0x00) within the first 128 bytes following the TPKT anchor. The ZGFX segment descriptor byte for the uncompressed path is 0x00 (uncompressed-segment flag). The `within:128` constraint limits the match window; a production rule would use a more precise offset derived from the TPKT length field via `byte_jump`.

`byte_test:4,>,65535,1,relative,big-endian` reads the 4-byte field at offset 1 from the cursor (immediately following the uncompressed flag byte) and tests whether it exceeds 65 535 (0x0000FFFF). Any payload-length value above this threshold exceeds the 65 536-byte output buffer and is the direct trigger for the out-of-bounds copy.

Precision note (simplified for teaching). The `content:"|00|"; within:128` match is intentionally simplified for teaching. In a production rule, the ZGFX segment header offset within a live RDP session is not a fixed distance from the TPKT preamble; `byte_jump` or the Suricata RDP parser (when the SOC enables it) is required for precise offset computation. The simplified match teaches the key structural elements: uncompressed flag + oversized length field.

§3.5 False-Positive Considerations

Any zero byte appearing within 128 bytes of the TPKT preamble (which is common in RDP protocol framing, especially in keyboard/mouse input PDUs that contain many null-padded fields) will trigger the first content match. The `byte_test` on the 4-byte field following the null byte then checks whether that field reads above 65 535; most protocol-field values in RDP headers are much smaller, so the compound rule produces relatively few false positives in practice. The largest false-positive risk is RDP traffic that happens to have a null byte followed by a large 4-byte field for unrelated reasons (e.g., certain RDP channel data PDUs with large payload lengths).

A SOC deploying this rule should run it in monitor mode for 48-72 hours on a baseline RDP traffic sample and inspect all alert events before enabling suppression or drop mode.

§3.6 Tuning Notes

This rule is most valuable during the patch-rollout window for CVE-2026-5405 in environments where analysts regularly capture RDP traffic for forensic review. Once 4.6.5 / 4.4.15 is deployed across all analyst workstations, this rule can be demoted to the archive rule set.

For SOCs that have disabled the Suricata RDP parser for CPU-overhead reasons, re-enabling it specifically for this rule period is a reasonable temporary configuration. `metadata: signature_severity High; metadata: priority 2;` appropriate. For analogous ZGFX CVEs in FreeRDP (CVE-2022-39316, CVE-2022-39320), the same structural rule shape applies; the rule template generalizes to any RDP implementation's ZGFX uncompressed path.

---

§4: CVE-2026-5656, Wireshark Profile ZIP Path Traversal

§4.1 CVE Recap

Field	Value
CVE	CVE-2026-5656
Advisory	wnpa-sec-2026-21
Bug class	Path traversal / Zip-Slip (CWE-22) chained with Lua auto-execution
Affected	Wireshark 4.6.0 through 4.6.4; 4.4.0 through 4.4.14
Fixed in	Wireshark 4.6.5; 4.4.15
Discoverers	Joohyun Park, Hyuk Kwon, Yonghwa Lee, Taisic Yun, Sangjun Song (Theori), with Xint

Full walkthrough: parent handout §4.

§4.2 Bug-Class Shape

Wireshark's profile import unzips a user-supplied archive into the per-profile configuration directory without validating that each entry's resolved absolute path stays within the extraction directory. An archive entry named `../../../plugins/auto-evil.lua` escapes the profile directory and lands in the Lua plugins directory. On the next Wireshark startup, the plugin auto-executes. The two-step chain is: (1) zip-slip places the attacker's Lua file outside the intended extraction boundary; (2) Wireshark's startup code auto-loads any `.lua` file in the plugins directory, converting a path-traversal primitive into arbitrary code execution. This is a logic-bug family (CWE-22) rather than a memory-corruption bug; no binary-exploitation primitives are required.

§4.3 Suricata Rule Template

```
alert http $EXTERNAL_NET any -> $HOME_NET any (
    msg:"VCA-ADV CVE-2026-5656 Wireshark profile ZIP with path-traversal entry
name zip-slip shape wnpa-sec-2026-21";
    flow:to_server,established;
    file.data;
    content:"PK|03 04|";
    content:"../"; within:512; nocase;
    metadata: cve CVE-2026-5656, ref wnpa-sec-2026-21;
    classtype:policy-violation;
    # replace SID with site-allocated range per Suricata SID convention
    sid:10001656; rev:1;
)
```

§4.4 Detection Rationale

`alert http ... -> $HOME_NET any` scopes to HTTP traffic inbound to the monitored network. The primary delivery vector for a malicious Wireshark profile archive is an HTTP download (from an attacker-controlled server, a compromised vendor distribution page, or a file-sharing link in a phishing email), so HTTP is the correct protocol anchor. The rule can be replicated for SMTP (change protocol to `smtp` and use `mime.data`) to cover email-attachment delivery.

`flow:to_server,established` matches the HTTP request direction on an established TCP session, scoping to the file-upload or download request rather than the HTTP response.

`file.data` is Suricata's sticky buffer for matching against file content extracted from protocol parsers. When the HTTP parser identifies a file being transferred (upload or download), `file.data` applies subsequent `content` matches to the file's bytes rather than the raw TCP stream. This is the correct primitive for matching inside a ZIP archive being transferred over HTTP.

`content:"PK|03 04|"` matches the ZIP local file header magic bytes: ASCII `PK` (0x50 0x4B) followed by the local file header signature bytes 0x03 0x04. Every ZIP archive begins with a local file header at offset 0; this anchor scopes the `file.data` match to ZIP-format files.

`content:"../"; within:512; nocase;` matches the path-traversal sequence `../` within 512 bytes of the ZIP magic. In a ZIP local file header, the filename field begins at offset 30 (after a 30-byte fixed header). `within:512` allows for the fixed header plus a short

filename before the traversal sequence. `nocase` covers `../` and upper-case variants; on Windows targets, `..\\` (backslash traversal) is a separate variant that requires a second content match.

`classtype:policy-violation` is appropriate for a file-level policy rule rather than a protocol parsing vulnerability; the attack is about file-system policy (no archive extraction should escape the extraction directory), not protocol-layer corruption.

§4.5 False-Positive Considerations

Any legitimate ZIP archive whose entry names happen to contain `../` as a substring (outside of a path-traversal context) will match. This is uncommon in well-formed archives; the PKWARE specification does not permit `../` sequences in entry names for conforming archives. However:

1. Developers who zip source trees containing relative symlinks may produce archives with `../` in entry names on some platforms.
2. ZIP archives that contain documentation files with `../` in their filenames (not paths) would match.
3. False positives are most likely in high-volume HTTP environments where many ZIP files transfer per hour.

The `within:512` constraint limits false matches to ZIP files where `../` appears near the start of the file (near the first local file header's filename field). Archives whose first entry has a long filename before any `../` sequence may escape the match; a production rule should iterate over multiple local file headers using `byte_jump` on the filename-length and extra-field-length fields.

§4.6 Tuning Notes

This rule is best deployed as a content-inspection rule in a network security monitoring (NSM) tier that reassembles TCP streams and performs file-extraction. Suricata's file extraction engine (enabled via `file-store:` output in `suricata.yaml`) can preserve a copy of flagged ZIP files for manual inspection.

For SMTP coverage, mirror the rule with `smtp` protocol and `mime.data` sticky buffer. For SMB file transfer coverage, a separate `smb` rule using `file.data` is appropriate for file-server environments. `metadata: signature_severity Medium; metadata: priority 3;` reflects that the attack requires user interaction (the analyst must import the profile) and is not a passive network exploitation.

---

§5: Cross-Bug-Class Shape Comparison

Reading the four CVEs as a detection set surfaces the structural differences in the Suricata primitives each bug class requires.

CVE	Bug-class shape	Delivery vector	Suricata primitive	Detection difficulty	Genera
CVE-2026-5402 (TLS/ECH)	Integer-truncation -> heap overflow	Live network capture or crafted `.pcapng`	`byte_test` on length-field consistency	Medium (ECH extension length is inspectable; threshold calibration needed)	Any TLS extensio inconsis inner an length fi recurs a TLS-diss CVEs
CVE-2026-5403 (SBC codec)	Loop accounting failure -> heap overflow	RTP payload in a Bluetooth-audio capture	`content` + `threshold` for volumetric anomaly	Low precision without accurate A2DP baseline; high recall	Any fixe buffer c dissecto accumul decode can be anomalo large
CVE-2026-5405 (RDP/ZGFX)	Missing bounds check on fast path -> heap overflow	ZGFX segment in RDP capture or live RDP traffic	`content` anchor + `byte_test` for oversized length field	Medium (the uncompressed-flag match is simplified; production rule needs ZGFX parser depth)	Any prot parser w fast/slow asymme ZGFX re across R impleme (FreeRDI CVE-202
CVE-2026-5656 (Profile/ZIP)	Path traversal + auto-execute logic bug	HTTP/SMTP file transfer (social engineering)	`file.data` + `content` for `../` in ZIP entry names	Low false-positive rate in enterprise environments where ZIP traversal is policy-violating; requires stream reassembly	Zip-slip i applicati combine extracti auto-exe extracte content

Narrative. The three heap-overflow CVEs (5402, 5403, 5405) all require length-field or accumulation-count anomalies for detection; `byte_test` is the primary Suricata primitive across all three. The zip-slip CVE (5656) is structurally distinct: the invariant is a string pattern in a file name rather than a numeric field value, so `content` against `file.data` is the appropriate primitive. This structural split in detection primitives mirrors the structural split in the bug classes: memory-corruption bugs produce numeric anomalies in length fields; logic bugs produce policy violations in string fields.

The hardest rule to tune is CVE-2026-5403 because the overflow is driven by accumulated frame count rather than any single frame's field value. Suricata's `threshold` keyword approximates the accumulation check volumetrically, but a production-grade detection for SBC loop accounting would require a Zeek script or custom Suricata protocol plugin that tracks per-session decoded byte counts.

---

§6: Cross-Track Linkage Table

Course	CVE-2026-5402 (TLS/ECH)	CVE-2026-5403 (SBC codec)	CVE-2026-5405 (RDP/ZGFX)	CV
SEC-101 Module 4	Introduce rule template as reference for integer-truncation detection; students read the rule and match each line to the bug-class shape	Introduce as an anomaly-detection example; the SBC sync-byte + threshold pattern teaches volumetric detection vs. field-value detection	Introduce as reference for asymmetric-validation detection; the fast-path/slow-path asymmetry is a named CVE class students carry forward	Int slip `fi` pat sha tra
ADV-101 Belt-5	Students read, critique, and compare the template against their own rules written from packet captures; capstone work produces a refined production-candidate variant	Capstone optional track: students write a Zeek script that counts per-session SBC decoded bytes and alerts when accumulated output exceeds 8 192; compare to the Suricata volumetric approximation	Students replicate the rule against a live-traffic lab, tune the uncompressed-flag match using `byte_jump`, and compare against the FreeRDP analogous CVE rules	Stu ver mu usi len cor SM
NET-101 Wk 8 + Wk 11	Week 8 (TLS module): rule template introduced as 2026-currency anchor for TLS-extension field-parsing rules; students trace ECH extension framing from RFC draft to Suricata primitive	Week 11 (RTP/VoIP module): rule template introduced as the Bluetooth-audio anomaly-detection pattern; students contrast with G.711 / G.722 rule shapes	Not a primary NET-101 anchor; referenced in the cross-CVE comparison table for completeness	No net ref cor to mo

---

§7: Authorization and Safe-Use Reminder

All hands-on rule testing in this module runs against lab-owned, intentionally-vulnerable Wireshark 4.6.4 or 4.4.14 instances inside the academy `fwlab` container, or against pre-recorded `.pcap` / `.pcapng` files supplied by the instructor. Production analyst workstations are never the test target.

The `--authorized-by` discipline that ADV-101 enforces for the SB6141 capstone applies unchanged here: a written authorization for the specific lab target, a controlled capture-file provenance chain, and explicit refusal to use these techniques against systems the academy does not own.

SOC teams adapting these rule templates for production deployment are operating outside the academy curriculum's classroom-defined scope. Production tuning, change-management approval, and regression testing against production baselines are the SOC team's own discipline, not a curriculum deliverable.
