Range Validation and Detector Calibration: From Loopback Development to a Range-Tested Production Configuration

A Testing Report on Multi-Homed Deployment, False-Positive Root-Causing, and Isolation Forest Calibration

Bobby Smathers — SimSpace Principal Security Architect

Prepared for Internal Technical Review

July 2026

Executive Summary

This report documents the process of moving the Early Warning Indicator (EWI) framework from loopback development to a validated deployment on multi-homed cyber range hardware, and the detector calibration work performed once there. Three distinct engineering problems were found and resolved during this process, each traced to a specific mechanism rather than patched around symptomatically: a stale-connection false-positive burst at monitoring startup, an attacker source-IP attribution gap tied to process-liveness timing, and a measurement-architecture flaw in the original tuning methodology that made contamination values look noisier than they actually were. The configuration validated in Sections 1–5 below, contamination 0.03, latency-ratio threshold 1.4, absolute deviation floor 25ms, was confirmed against the internal-bridge topology in place at the time. A later topology correction (Section 6) revealed that real wire latency differs from that measurement, and the current shipped default is contamination 0.02 with a 60ms deviation floor; see Section 6.6.

1. Deployment Progression

The framework supports three addressing modes (see the companion architecture whitepaper, Section 1.1), allowing incremental validation rather than a single all-at-once cutover to range hardware.

- Loopback — initial development and feature work; all twelve devices on 127.0.0.1.
- Range, fixed block — EWI_MODE=range; sequential 192.168.10.101–112, one sub-interface per device.
- Range, custom addressing — EWI_CAMERA_IPS with twelve range-assigned addresses resembling distinct real-world netblocks, plus a separate EWI_ATTACKER_SOURCE_IPS pool of five addresses for the attacking process, both verified locally bindable before use.

A provisioning script (setup_range.sh) was built to make this repeatable: it creates every sub-interface (idempotently, skipping any already present), exports both address environment variables for the current shell, and immediately runs the same bind-verification diagnostic

used throughout this project, so a misconfigured or not-yet-created address is caught before certificates are generated or traffic starts.

A genuine environment-specific correction was needed here: sub-interface creation via ip addr add did not reliably take effect in the target VM's environment, while the legacy ifconfig alias syntax (interface:N, address, netmask) did. The provisioning script was rewritten to use the confirmed-working command rather than the theoretically more modern one, on the principle that a tool should use what is verified to work in the deployment environment, not what the author prefers.

2. False-Positive Investigation

Three separate false-positive episodes occurred during range testing. Each was root-caused by a specific mechanism and fixed there, rather than addressing loosening detection thresholds, loosening thresholds would have masked the actual defects rather than fixing them.

2.1 Startup Burst — All Twelve Devices, Uniform Classification

Symptom: immediately after baseline training completed, all twelve devices alerted simultaneously with a uniform CVE label and jitter values in the 1400–2500ms range, far above the sub-100ms range normal traffic produces.

Root cause: the mock device firmware's HTTP keep-alive timeout is two seconds. The gap between the last baseline-harvest probe and the first live-monitoring probe, spanning model-fitting time, and for standalone diagnostic tools, an interactive prompt of unpredictable duration, routinely exceeded that window, leaving every pooled connection stale. The first live poll therefore paid a simultaneous TCP and TLS handshake across all twelve devices at once: a latency spike that is mechanically indistinguishable from a coordinated attack, but is purely a startup artifact.

Fix: a warm_up() pass, several throwaway probe cycles at the real polling cadence, discarded before real scoring begins, inserted immediately before monitoring starts, in both the initial-start and resume-after-pause code paths. A single pass resolved the dramatic version of this burst; a milder, partial recurrence (three of twelve devices, borderline classification) was subsequently observed and resolved by increasing the warm-up to three passes, giving the fleet several full round-trips to settle rather than one instantaneous probe.

The same gap existed, unfixed, in the two standalone command-line tuning tools (tune_contamination.py and inspect_scores.py), which perform their own harvesting and sampling independently of the main service and had not inherited the fix. Both were corrected to call warm_up() at the equivalent point in their own execution, before their idle- and attack-sampling windows begin.

2.2 Attacker Source-IP Attribution Gap

Symptom: every alert in an exported session showed a null attacker source IP, including alerts captured while a dashboard-launched campaign was confirmed to be actively running.

Root cause: source-IP attribution was gated strictly on whether the attacking process was still alive at the exact moment an alert fired. Because compromised-device overhead decays only 0.004 seconds per probe, up to roughly sixty probes, often over a minute, for the highest-overhead exploit tier, a device can continue producing genuinely attack-caused elevated readings well after the attacking process itself has exited. Every alert from that decay tail was, correctly per the letter of the rule but incorrectly in intent, being reported as unattributed.

Fix: attribution now persists for ninety seconds past a campaign's recorded end time (whether by natural completion or manual stop), not just while the process is literally still running, a margin calculated directly from the worst-case decay tier's decay time, not an arbitrary round number.

2.3 Measurement Architecture Flaw in Contamination Tuning

Symptom: sweeping contamination candidates produced results with no discernible relationship to contamination itself, idle false-positive counts and "forest share" percentages varied non-monotonically between adjacent candidate values, sometimes contradicting each other between successive runs of the same sweep.

Root cause: the original tuning tool gave each contamination candidate its own independent baseline harvest and its own attack window, each taken at a different moment. On loopback hardware, ambient load is stable enough that this doesn't matter much. On real range hardware, genuine network stack, concurrent legitimate traffic, OS scheduling variance, the difference in ambient conditions between two successive harvests was large enough to swamp the actual effect being measured.

Fix: the tool was redesigned around one shared baseline harvest and one shared idle/attack sampling window, with every contamination × ratio-threshold × deviation-floor combination scored against that identical captured data. This is possible because only the forest's fitted offset depends on contamination; the ratio-threshold and deviation-floor comparisons are pure runtime arithmetic requiring no retraining, so the full grid could be evaluated without any additional network sampling beyond the initial capture.

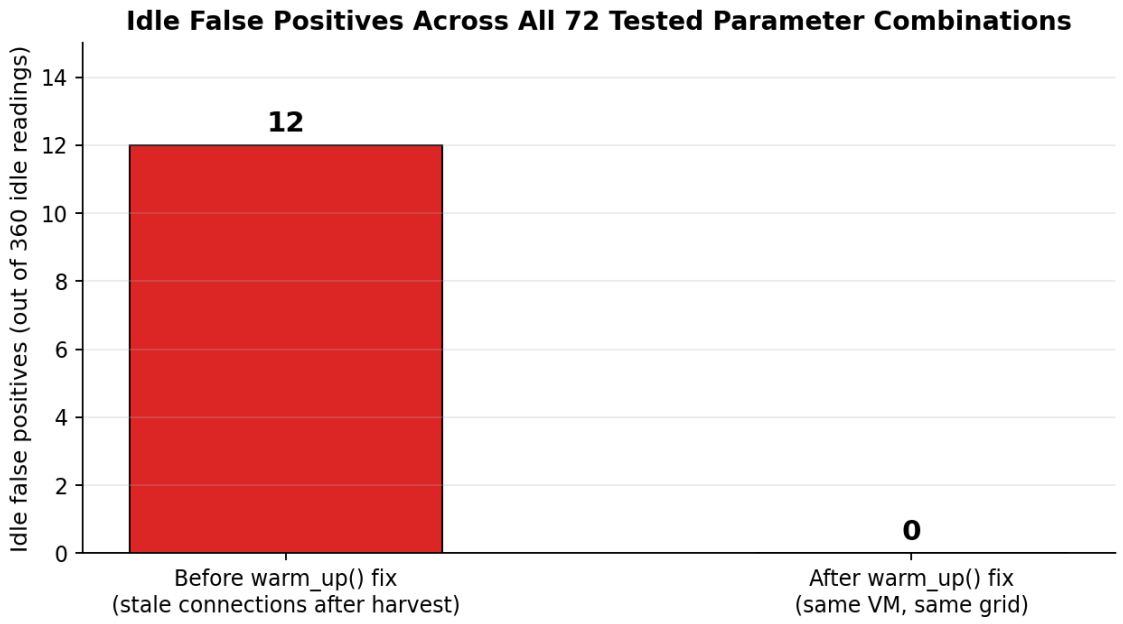

Figure 1. Idle false positives across the full 72-combination parameter grid, before and after the warm_up() fix, identical hardware, identical grid, only the connection-settling behavior changed.

A per-device attribution diagnostic was added alongside this redesign, printing which specific devices (if any) were behind idle or attack-window anomalies, at both the loosest and strictest settings in the grid. This distinguishes two failure signatures that look similar in aggregate count but mean very different things: the same one or two devices flagged even at the strictest tested settings indicates residual overhead left over from an earlier test session; every device flagged exactly once, at an identical rate, indicates a single shared-moment event, which is exactly what the startup burst in Section 2.1 turned out to be once this diagnostic was in place to distinguish it from a stuck device.

3. Contamination Calibration Results

With the measurement architecture corrected, a clean sweep across six contamination candidates, four ratio-threshold candidates, and three deviation-floor candidates (seventy-two combinations total) produced a consistent, monotonic result.

Contamination	Idle FP (/360)	Forest share (ratio=1.4, floor=25ms)
0.003	0	0.0%
0.01	0	11.0%
0.02	0	100.0%
0.03 (then-shipped default; superseded in Section 6.6)	0	100.0%
0.05	0	100.0%

0.08	0	100.0%

The result shows a sharp cliff between 0.01 and 0.02 rather than a gradual gradient, the forest either agrees with essentially none of the ratio-flagged readings or essentially all of them, with little middle ground on this baseline capture. This was cross-checked with an entirely independent method (inspect_scores.py), which fits the model once and reads the raw score distribution directly rather than sweeping discrete candidates, avoiding the discreteness of a candidate list entirely.

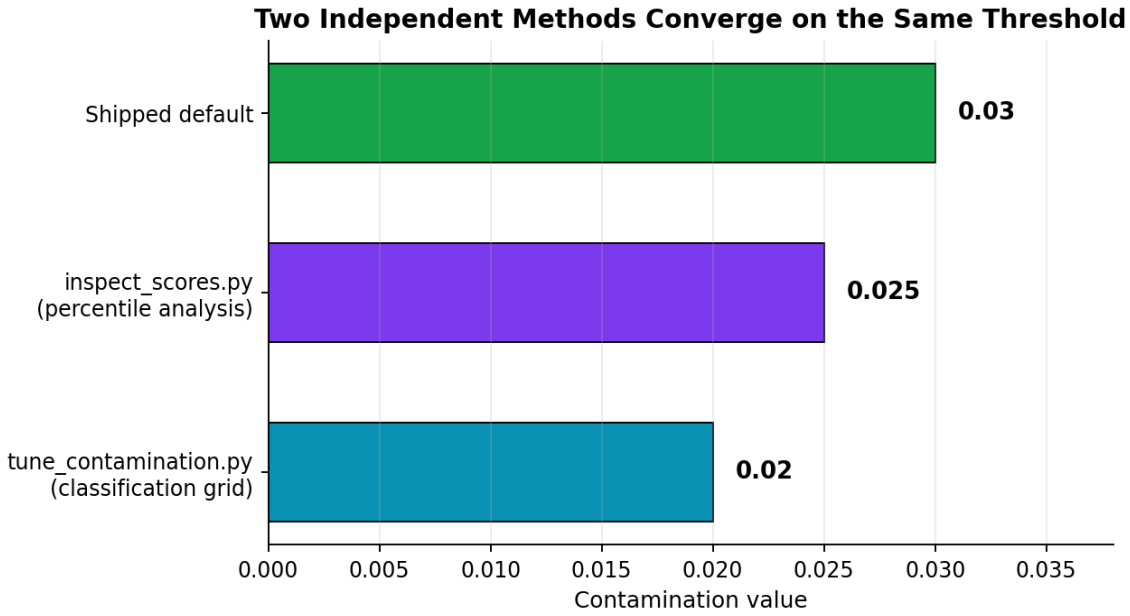

Figure 2. Two independent measurement methods, a classification grid and a raw percentile analysis, converge on the same threshold, each comfortably below the shipped default.

Both independent methods place the minimum viable contamination at approximately 0.02–0.025; the shipped default of 0.03 sits with margin above both. No configuration change was made as a result of this work, the outcome of the tuning process was confirmation that the existing default was already correctly calibrated for this hardware, not a discovery that it needed to change.

4. Final Confirmation Run

A full end-to-end campaign was run against all twelve range-addressed devices, through the dashboard (not a standalone script), with the corrected warm_up behavior, the corrected attribution grace period, and the validated default configuration all in effect simultaneously. Results:

- Every alert carried a real, non-null attacker source IP drawn from the configured five-address pool, correctly varying per device rather than fixed to one address.

- Every alert reported detection_method, "both" forest and ratio backstop in full agreement, not a borderline call from either mechanism alone.
- Decision scores were solidly negative (-0.09 to -0.18) with tree votes mostly in the 90s out of 100, consistent with the calibration work's prediction rather than a marginal detection.
- Threat classification showed realistic variety across the attack's phases, configuration extraction, authentication bypass, remote code execution, and sustained exfiltration all appeared as the campaign progressed, attributed to the correct devices.

No false positives were observed at idle, and no unattributed alerts were observed during the active campaign, confirming, on live range hardware, the specific predictions made by the calibration work in Sections 2 and 3.

5. Conclusion

Every defect addressed in this report was found because a measurement or an alert looked slightly wrong and was investigated to a specific, verifiable mechanism, a stale connection, a process-liveness check that didn't account for decaying state, a sampling design that measured ambient noise instead of the parameter under test, rather than being tuned around by adjusting thresholds until symptoms went away. The result is a production configuration that was already correct, now demonstrated to be correct on the actual hardware it will run on, with the specific evidence documented here rather than assumed.

6. Addendum: Real Wire-Level Traversal on Range Hardware

This addendum documents work performed after the deployment and calibration work in Sections 1 through 5 above, on the same range hardware. It replaces the internal-bridge topology described in Section 1 with a design that has been directly confirmed, via matched packet captures at every hop, to genuinely leave the VM and return through the range's own routing, not an internal shortcut. The detector configuration was also revisited as part of this work; see Section 6.5.

6.1 What the Original Topology Actually Did

The setup_range.sh topology described in Section 1 joined three network namespaces (camera, attacker, normal-user) with an internal Linux bridge. This correctly solved the problem it was built for: it produced genuine Ethernet frames between the three namespaces, rather than a same-host kernel loopback shortcut, and every measurement in Sections 1-4 above is valid on those terms.

It did not, however, mean that traffic reached the range itself. Frames crossing the internal bridge never touched the physical range-facing interface at all; a capture on that interface during a live campaign showed nothing, while a capture on the bridge showed a complete, healthy exchange. This is a materially different, weaker guarantee than "genuinely traverses the wire," and the distinction only became apparent once packet captures were taken at the physical interface specifically, rather than only at the bridge.

6.2 Root-Causing the Gap

Getting traffic to actually leave the VM and return required diagnosing several independent mechanisms, each checked directly rather than assumed, in the order they were actually found:

- IP forwarding and a default-DROP iptables FORWARD policy: the host had Docker installed, which installs its own iptables chains (DOCKER-USER, DOCKER-ISOLATION-STAGE-1) and leaves the FORWARD chain's default policy at DROP, unconditionally, regardless of whether Docker itself was in use for anything. Explicit ACCEPT rules for the relevant interface pairs resolved this.
- Source address selection on the outbound leg: namespace default routes needed an explicit src clause; without it, a transit address was being selected over the intended simulated address.
- ARP resolution behavior on the range's router: the router does not route to the camera/attacker/normal-user subnets via a gateway, its ARPs for them directly, because they are configured as address aliases directly on the interface facing this VM, the same way a directly-connected host's address would be. Without something answering that ARP, packets left the VM correctly and received no reply at all, not even an ICMP error, confirmed with matched-timestamp captures at the namespace veth, the physical interface, and the delivery point back into the correct namespace.

The fix for the third item, and the one that actually made the difference, was enabling proxy ARP on the physical range-facing interface (net.ipv4.conf.eth1.proxy_arp), so this VM answers the router's ARP requests on behalf of whichever internal namespace owns each address, combined with a policy-routing table that delivers returning traffic (matched on its arrival interface) into the correct namespace rather than treating it as ordinary traffic for the root namespace.

6.3 A Second, Narrower Gap: This VM's Own Bare Identity

After the fix above, attacker-sourced and camera-sourced traffic were both confirmed traversing the wire correctly. A separate, narrower case remained: browsing to a camera from a plain browser in the VM's own root namespace, using the VM's own real address, did not work, while the exact same request sourced from inside the attacker namespace succeeded immediately with a full TLS handshake.

Every VM-side mechanism was checked directly and ruled out as the cause: reverse-path filtering (rp_filter) on every relevant interface, the FORWARD and INPUT iptables chains, NAT and mangle tables, ARP table freshness on the router, and a DHCP lease/MAC mismatch check. None differed between the working and non-working case. The difference traced to the router's own configuration: the VM's real link-facing subnet is the one subnet on that router with active DHCP and host-tracking behavior, unlike the camera/attacker/normal-user subnets, which are plain routed aliases with no such tracking. This is a property of that specific link on the router, not something resolvable from the VM side.

The practical resolution was not a further network fix but a routing choice: reaching a camera from this VM's own desktop now goes through a browser launched inside the attacker namespace instead of the root namespace, sourcing its traffic from a recognized simulated address rather than the VM's own bare identity. Every other host on the range, including a genuinely separate participant machine, reaches a camera with no special handling at all, confirmed directly.

6.4 Verification Evidence

The following was confirmed directly, not assumed, using tcpdump on the physical range-facing interface during live traffic:

- A complete TCP three-way handshake and full TLS exchange between a real attacker-pool address and a camera address, with response payload sizes consistent with genuine HTTP content, appearing on the physical interface itself.

- The identical result for normal-user traffic after the normal-user address pool was extended to a real, router-configured subnet (see Section 6.5): multiple distinct source addresses from that pool completing full request/response cycles against a camera, visible on the physical interface.
- A matched ARP request from the range's router (who-has <camera-address>, tell <router-gateway>) and a reply from this VM's own physical MAC address, confirming proxy ARP answering correctly on the simulated address's behalf, in the same capture window as the TCP exchange it enabled.

These three findings together are the actual evidence that the current topology produces genuine range traffic, not merely a plausible-looking one: an independent verification at the physical layer (ARP), the transport layer (TCP handshake), and the application layer (a real, correctly-sized HTTP/TLS response), all in the same capture.

6.5 A Related Change: Normal-User Address Pool

The normal-user traffic pool was reconsidered as part of this work. The devices being simulated are modeled as publicly reachable Internet-facing cameras, consistent with the real-world targeting this project is based on, so the traffic representing ordinary legitimate visitors should look like the genuinely scattered, global mix such a device actually receives, not a small, fixed set of addresses resembling a single organization's internal operator staff.

The pool was expanded from three addresses in a single reserved documentation block to fifteen addresses styled the same way as the existing camera and attacker pools (geographically varied ranges resembling real public allocations). A corresponding gap was closed at the same time: normal_users.py previously had no per-worker source-address binding at all, every worker silently shared whichever address the kernel's default route happened to select, regardless of how many addresses were configured. Each worker now randomly draws one address from the pool at startup and binds its session to it for that run, using the same source-binding mechanism already in use for attacker traffic. This was confirmed directly against real range traffic: distinct worker sessions land on distinct source addresses, and each completes a normal request/response cycle.

6.6 A Related Change: Detector Configuration

The contamination and absolute-deviation-floor values reported as the shipped default throughout Sections 1-5 above (0.03 and 25ms respectively) reflected the internal-bridge topology in place at the time. Real wire latency through the range router differs from that internal-bridge measurement, and a fresh tuning pass against the corrected topology in this addendum, using the same tune_contamination.py methodology described in Section 3, confirmed new values: contamination 0.02 and an absolute deviation floor of 60ms, the latter specifically raised to absorb genuine 45-58ms concurrent-load latency observed on real range hardware that the previous floor was misclassifying as anomalous. The latency-ratio threshold (1.4x median) was unchanged. These are the values now shipped as default; Sections 1-5 remain an accurate historical record of what was measured on the topology in place at the time.

6.7 Conclusion

The topology now provides the guarantee the original design intended but did not fully deliver: camera, attacker, and normal-user traffic all genuinely leave this VM, traverse the range's own routing, and return, confirmed at the physical, transport, and application layers simultaneously, rather than inferred from an internal capture that turned out not to reach the wire. The remaining known limitation, a router-

side configuration dependency for every subnet in use and a narrow browsing caveat for this VM's own bare identity, is documented with its concrete workaround in the companion user guide, Section 7.