ICS/OT Security Series · Inputs Lie · Part 4 of 4

Detecting Lies at the Signal Layer

Norris Cornell

IAM Analyst II / Engineer · CornellSecurity.com

May 2026

The relay tripped at the wrong time.

Nobody touched it.

No firmware was modified. No unauthorized commands crossed the network. No attacker was inside the perimeter.

The PLC evaluated its inputs, reached a conclusion, and acted on it with complete confidence.

The inputs were wrong.

This is where the first three parts of this series have been building.

Part 1 established that industrial systems trust external signals they were never designed to question. Part 2 showed how GNSS, NTP, and RF channels enter ICS environments as assumed-valid infrastructure. Part 3 demonstrated what happens inside the controller when those signals lie—forced state transitions, logic pollution, timing desynchronization, processes that fail silently and mathematically while the HMI shows green.

Now the question becomes: how do you detect it?

And then: does the same architecture of misplaced trust exist at the application layer?

It does. I built a lab to prove it.

Why Standard Monitoring Misses Signal-Layer Attacks

Most ICS security monitoring watches two layers: the network and the controller.

It looks for:

- Protocol anomalies
- Unauthorized command sequences
- Unexpected firmware changes
- Rogue connections
- Known malware signatures

These are meaningful controls. They've improved considerably as the industry has matured.

But none of them catch a spoofed sensor value.

A well-crafted lie arrives on the right protocol, from the right address, within the expected engineering range, at a plausible time. It passes every format check. It satisfies every network filter. It reaches the input buffer looking exactly like truth.

The controller doesn't hesitate. It evaluates.

Standard monitoring has no opinion about physical plausibility. It only has opinions about format.

This is the gap—not a gap in tooling, but a gap in what defenders have instrumented themselves to see.

I discovered the same gap while building Iron Gate, a Proxmox-hosted detection lab running crAPI—a deliberately vulnerable API platform—alongside Loki log aggregation and Grafana alerting. The parallel is exact. What follows are five detection techniques, each grounded in lab evidence.

Five Detection Techniques

These aren't theoretical countermeasures. They're approaches that give defenders visibility into the truth layer—the gap between what the controller believes and what the process actually is.

1. Physics-Based Anomaly Detection

A temperature sensor reading 40°F in a vessel that's been running at 180°F for six hours isn't a measurement. It's a lie. The log doesn't know that. The PLC doesn't know that. They process the value because it arrived.

Detection requires encoding what the process actually does:

- Rate-of-change limits
- Thermodynamic relationships
- Safe operating bounds
- Causally coherent sequences across multiple sensors

The question isn't "is this value in range?" That's trivially easy to spoof.

The question is: "could this value have arrived through legitimate physics given everything else we know about this process right now?"

That's a harder question. It requires domain knowledge. But it's the only question that catches the lie before consequence.

Iron Gate Lab · Session 3

BOLA enumeration against GET /workshop/api/shop/orders/{id} produced 200 sequential requests: 5 HTTP 200s buried inside 195 HTTP 500s. A detector watching only for successful responses sees 5 hits—below any reasonable alert threshold. The monitoring system had a format opinion about each response. It had no physical plausibility opinion about what 195 identical server errors in two minutes actually means. A patient attacker spacing requests to one per hour returns 5 exfiltrated records and generates zero alerts. The sequence is the anomaly. No single request is. In both environments, the detection gap isn't a missing tool. It's a monitoring assumption that failed against the actual data.

---

2. Sensor Redundancy and Cross-Correlation

Two independent sensors measuring the same process variable should agree within a defined tolerance.

When they diverge, one of three things is true:

- A sensor has failed
- A genuine process anomaly is occurring
- An attacker is manipulating one stream while the other still tells the truth

Redundancy doesn't prevent the lie. It makes the lie detectable.

The key word is *independent*. Redundant sensors sharing a common power source, a common signal path, or a common engineering assumption inherit the same vulnerability. True independence means:

- Separate physical sensing elements
- Separate transmission paths

- Separate validation logic

In practice, most industrial environments don't achieve this. Which is itself worth knowing.

3. Timing Drift Detection

GNSS and NTP timing references don't drift randomly.

Legitimate drift is slow, consistent, and correlated with known physical causes—oscillator aging, temperature variation, atmospheric conditions. It moves in one direction over time and stays within predictable bounds.

Spoofed timing jumps. It reverses. It shifts discontinuously. It moves faster than any legitimate source could account for.

Monitoring the drift *rate*—not just the time value—surfaces manipulation that the timestamp itself conceals.

A clock that suddenly advances 47 milliseconds and then continues advancing normally is not a clock experiencing legitimate drift. It's a clock that was told to lie.

Most ICS environments log timestamps. Very few monitor the second derivative of those timestamps. That's the gap.

Iron Gate Lab · Session 4

The initial alert rules evaluated on a 60-second cycle against a 5-minute detection window. The BOLA scan completed in approximately 2 minutes—well within range. But a patient attacker spacing requests to one every 4 minutes enumerates all 5 vulnerable records across 20 minutes and stays entirely dark. The timing assumption built into the detection window is itself an attack surface. The rate matters more than the value. Most ICS environments log timestamps. Very few monitor the second derivative of those timestamps. The alert window has the same blind spot.

---

4. State-Machine Integrity Monitoring

Every industrial process follows a defined sequence of states.

- Pressure equalizes before vessels connect
- Valves open before pumps start
- Temperature stabilizes before reactions initiate
- Safety permissives confirm before sequences advance

A controller that attempts a state transition out of sequence—even if every individual input value appears valid—has been fed a lie somewhere upstream.

The transition itself is the anomaly.

This is fundamentally different from monitoring individual sensor values. It monitors the *sequence*, which is harder to forge convincingly. An attacker manipulating a single sensor can produce a plausible-looking value. Producing a complete, causally coherent sequence of falsified values across multiple sensors and timing references simultaneously is a much harder problem.

State-machine integrity monitoring exploits that difficulty.

Iron Gate Lab · Sessions 4–5

The broken-auth Grafana rule was logically correct for six full sessions. The LogQL query was right. The threshold was right. It never fired. The reason: the Django debug log format used by crAPI's identity service didn't contain HTTP status codes. Loki captured all traffic. The attack was in the stream. The alert system evaluated faithfully—and produced nothing, because the format didn't contain what the rule needed. The sequence was invisible. The fix was deploying an nginx proxy to produce JSON access logs with status, remote_addr, method, and path as indexable Loki labels. After that change, 10 failed login attempts produced an alert payload containing remote_addr: 10.0.0.101—attacker IP, captured, attributed. The sequence became visible the moment we instrumented the right layer.

---

5. Scan-Cycle Stability Monitoring

PLC scan cycles run at deterministic intervals. That determinism is by design—it's what makes industrial control systems predictable and safe.

When scan cycles start showing jitter—running slightly long, varying in ways that don't correlate with legitimate process load—something is wrong:

- Inputs are taking longer to evaluate than expected
- Resources are being contested
- The controller's determinism is degrading

This is subtle. The symptoms look like ordinary performance variation.

But scan-cycle instability is a measurable signal that something upstream of the controller is not behaving as designed.

Most historians log process values. Very few log scan-cycle timing with enough resolution to detect this pattern. That's a data collection problem with a straightforward fix.

The Same Lie, Different Layer

The physics changes. The trust assumption doesn't.

In the ICS environment, the controller doesn't know it was lied to because it was never designed to verify its inputs. In the API environment, the Grafana alert system didn't know it was lied to because it was never instrumented to see the right layer. Both failed for the same structural reason: implicit trust in the completeness of what arrived.

The Django debug log was there. Loki was ingesting it. The attack traffic was in the stream. The alert system evaluated faithfully and produced nothing—not because the attacker evaded detection, but because the monitoring layer had a format opinion and no plausibility opinion about what it was seeing. That's the same sentence as Part 1.

BOLA at the API layer is a state-machine violation—accessing account B's resources from account A's authenticated session is a sequence the system was never designed to permit. Schema validation confirms the request is well-formed. It has no opinion on whether the object ID belongs to the requester. Broken authentication is timing drift—a token appearing at an unexpected time, from an unexpected origin, requesting a sequence inconsistent with prior session behavior isn't necessarily expired. It may be a clock that was told to lie.

The OWASP API Top 10 describes attack patterns. The "Inputs Lie" framework describes why those patterns work: every layer in the stack trusts the one below it, and nobody has instrumented for the difference between valid format and legitimate intent.

What This Actually Requires

These techniques don't come pre-packaged.

Physics-based anomaly detection requires encoding the process—and that means security engineers and process engineers working together to define what the physics actually are:

- What rate of temperature change is physically possible in this vessel?
- What pressure differential is acceptable across this valve?
- What timing relationship between these two sensors is thermodynamically coherent?

Those questions have answers. The answers exist in the heads of the engineers who designed and operate the process. They are almost never written down in a form that a monitoring system can act

on.

The Iron Gate lab exposed the same structural problem at the API layer. Loki captured all traffic from session one. Three sessions passed before alert rules existed. One more before those rules actually fired. The data was there the entire time. The gap wasn't a missing tool—it was that nobody had defined what anomalous behavior looks like in terms the monitoring system could evaluate.

In most OT organizations, security and engineering operate in parallel. They share a site but not a threat model. In most API organizations, security and development operate in parallel. They share a codebase but not a behavioral baseline.

Until that changes, the signal layer—whether it's a sensor bus or an API endpoint—remains the easiest, most cost-effective attack surface in any environment. An attacker who can lie convincingly to an input doesn't need to touch the controller, the application, or the business logic.

The system will do everything they want on its own.

The controller doesn't know it was lied to. The alert rule didn't know either.

Neither can know. That's not what they're designed to do. They're designed to evaluate inputs and execute logic. They will do that faithfully, correctly, and without hesitation—even when the inputs are wrong.

Detection is not the controller's responsibility.

It's yours.

This concludes the Inputs Lie series. The same trust failure examined here—systems accepting unvalidated input as ground truth—runs through every domain of cybersecurity. A future series will explore how it surfaces in software, APIs, and identity systems, and what detection looks like when the physical world isn't the consequence.

Norris Cornell is an IAM Analyst II / Engineer specializing in ICS/OT security and the convergence of cyber-physical systems. He presented "When Cyber Meets the Spectrum" at BSides Delaware 2025. Iron Gate: github.com/samson-2369/iron-gate