Contextual Evaluation of Industrial Control Actions: Hardware Validation and Learned Operational Grammar

Bruno Salmazo

Liscere

July 2026 · Liscere Technical Report · LTR-2026-03 · v1

Abstract

Industrial control protocols authenticate neither the sender nor the appropriateness of a command [1]; conventional defences add identity, access, and protocol validation, all of which answer whether an actor is permitted to act. That question is a weakening signal of safety, because a competent adversary arriving through legitimate means issues commands that are individually valid, authorised, and within range, and so indistinguishable, action by action, from a legitimate operator [3, 4]. This report develops and validates a contextual evaluation layer, Liscere, that judges not whether an action is permitted but whether it is appropriate: whether a given write, to a given control artefact, is coherent with the operational phase the process is currently in. The approach is passive and external, recovering process context from observed network traffic without modifying the plant, and it is structural rather than value-based, which distinguishes it from anomaly detection, from process interlocks, and from formal preventive enforcement.

The framework is validated on a physical testbed, a Siemens S7-1200 controlling a tank process over Modbus/TCP, with a strictly passive observer receiving only mirrored traffic, in three increments. The first shows that an identical protocol-valid write receives opposite verdicts, allowed or alerted, according to an operational phase the evaluator infers from observed process state rather than being told. The second makes that phase inference robust to the noise, pauses, and transitions of continuous operation, attaching to each verdict a confidence that separates an action incoherent with a known phase from one whose context is genuinely unsettled. The third replaces the hand-declared artefact-to-phase grammar with one learned from observation, and shows the learning to be genuine and artefact-specific: two control artefacts taught opposite grammars produce mirror-image verdicts, judged through graded rather than binary evaluation. Every result is backed by a decision log and a packet capture, published alongside this report.

The contribution is a validated proposition rather than a finished system. Its bounds are stated explicitly: the evaluator is still told which signal carries process state, learning assumes a clean and representative baseline, the inferred context is not resistant to an adversary who forges the observed telemetry, and validation rests on a single cyclic process, protocol, and controller. Within those bounds, the work establishes that contextual, structural evaluation of control actions is possible, that it occupies a gap left by existing approaches, and that its central dependencies, the operational context and the grammar of legitimate action, can both be lifted from human declaration toward autonomous observation.

Keywords: operational technology security, industrial control systems, contextual action evaluation, phase inference, learned operational grammar, passive monitoring, Modbus/TCP.

2. Motivation and positioning

2.1 The problem: permission is a weakening signal of safety

Industrial control systems were designed for environments assumed to be isolated and trusted. Their protocols, Modbus/TCP among the most widespread, carry no authentication and no notion of whether a command is appropriate; they carry only whether it is well formed [1, 2]. Security was historically layered

on top through perimeter, identity, and access controls, all of which answer a single question: is the actor permitted to act?

That question is becoming a weaker signal of safety. Attackers increasingly arrive through legitimate means, stolen credentials, hijacked engineering workstations, remote sessions across converged IT and OT networks. Once inside, a competent adversary does not send malformed traffic; it learns how the process runs and issues commands that are individually valid, authorised, and within protocol [3, 4]. At the level of any single action, operator and attacker become indistinguishable, because no identity, access, or protocol check asks whether the action makes sense for the physical process in its current state.

This is the gap the Liscere framework addresses. From the Latin *licere*, to be permitted, its premise is that in a live process *permitted is not the same as appropriate*: an action can satisfy every conventional check and still be wrong, because whether it belongs depends on the operational context in which it occurs, and that context is not something those checks observe.

2.2 The question Liscere asks

Liscere evaluates the action *in context* rather than in isolation. Given a control action that is well formed, authorised, and within range, it asks: does this action, on this artefact, belong in the operational phase the process is currently in?

Answering this requires a picture of what the process is doing, reconstructed from the network itself, without sensors and without modifying the plant. Against that picture, each action is weighed, not by who sent it or whether it is permitted, but by whether it is coherent with the process state and with how the target artefact is normally manipulated. This is a passive, external layer that sits beside existing controls rather than replacing them, and requires no change to the systems it observes.

2.3 Why this is not redundant with process interlocks

The most immediate objection is that a correctly engineered PLC should already prevent unsafe actions through interlocks, so any action a well-programmed controller accepts must be safe. The answer to this objection defines what Liscere contributes.

A process interlock evaluates the physical instant. It can clamp a value to a safe range, prevent incompatible actuators from engaging together, or hold a state until a physical precondition is met. Its scope is the current cycle and the current physical condition; it has no memory of how the process behaves over time, and no representation of which artefact is legitimately manipulated in which operational phase.

Consider a legal value written to a control parameter during a phase in which that parameter is not normally touched: altering an inlet setpoint while a tank is being drained, or an alarm threshold while a process is running rather than under maintenance. The value is physically acceptable, so no clamp fires, and a correctly programmed PLC accepts it. Yet the action makes no operational sense in that moment, and a competent attacker would use it precisely because it passes unremarked. Its illegitimacy is a property neither of the value nor of the instant, but of the action's coherence with the operational phase and the learned role of the artefact. Detecting it requires observed temporal context that an interlock structurally does not hold. A controller that did hold such context, comparing every write against a learned model of normal operation, would no longer be an interlock; it would be performing the evaluation Liscere performs.

This holds even for a perfectly programmed PLC. Liscere does not compensate for missing interlocks; it covers a class of illegitimacy outside what any interlock can evaluate, because interlocks reason about physical safety at an instant, while Liscere reasons about structural coherence across operational phases. That the installed base is dominated by devices too old or too critical to modify makes an external, observational layer practically urgent, but the necessity does not rest on poor engineering: it would hold in a plant of perfect controllers.

2.4 Why this is not anomaly detection

A second objection places Liscere among the process-aware and specification-based intrusion detection systems that already model process behaviour and flag deviations. The distinction is deliberate and load-bearing.

Anomaly detection asks whether a process value is unusual relative to a statistical baseline. This is vulnerable in two directions. It suffers chronic false positives, because a legitimate but uncommon value is statistically indistinguishable from an attack. And it is blind to actions that stay within learned ranges, which is precisely where competent OT attacks operate.

Liscere learns instead which artefact is written in which operational phase, a structural grammar rather than a value distribution. Its verdict depends not on whether a value is statistically rare, but on whether the action is coherent with the phase and the learned role of the artefact; the same valid value can be coherent in one phase and incoherent in another. This structural framing lets Liscere flag an action on its first occurrence, without a value baseline, and separate a legitimate uncommon action from a structurally incoherent one, a distinction a value baseline cannot make.

3. Related work

Liscere sits among a body of work on securing industrial control systems that already looks beyond identity and access to the behaviour of the process itself. This section situates the framework relative to that work and identifies the gap it addresses. The intent is not an exhaustive survey, but a positioning: to show which established ideas Liscere builds on, and where it departs from them.

3.1 Specification-based and process-aware intrusion detection

A substantial line of research detects attacks by modelling what a control system is supposed to do and flagging traffic or behaviour that departs from that model. Hadžiosmanović et al. [5] extract the semantics of process variables from control traffic and monitor them for behaviour inconsistent with a learned model, moving detection from the network layer to the process layer. Caselli et al. [6] model the expected *sequence* of control messages and detect sequence-based attacks, where individual messages are valid but their ordering is not.

This work shares Liscere's central intuition, that the process, not just the packet, is where meaning lives, and Liscere builds directly on it. The departure is in what is evaluated. Specification-based detection asks whether observed behaviour *conforms to a norm*: does this traffic, this sequence, this variable trajectory match what the specification or learned model permits? Liscere asks a different question of an action that already conforms: given that this write is well formed, in specification, and within the expected envelope, is it *appropriate* in the current operational phase? The action Liscere flags is not anomalous by a specification-conformance test; it is a legal action in an incoherent context. Sequence-based approaches [6] come closest, in that ordering is a form of context, but they model the order of messages rather than the coherence of an action with an inferred operational phase and the learned role of the target artefact.

3.2 Process-aware anomaly detection and physical invariants

A second line detects attacks by learning the normal physical behaviour of a process and identifying deviations. Carcano et al. [7] introduce the notion of *critical states*, monitoring the process for progressions toward states known or learned to be dangerous, rather than inspecting individual packets. Related approaches learn physical invariants or value baselines and alert when observed values fall outside them.

The distinction here is the one drawn in Section 2.4, now placed against this literature. These approaches are, in essence, value- and state-based: they learn which values, or which combinations of state, are normal or dangerous, and judge observations against that. Liscere does not learn a value baseline. It learns a *structural grammar*, which artefact is written in which operational phase, and judges an action by its

coherence with that grammar, independent of the value written. The consequences differ. A value baseline flags a legitimate but uncommon value as readily as an attack, and misses an attack that stays within learned ranges; a structural grammar flags an action that falls outside the learned artefact-phase association on its first occurrence, regardless of whether its value has been seen before. Critical-state analysis [7] is complementary rather than equivalent: it asks whether the process is approaching a dangerous state, while Liscere asks whether a given action belongs in the current phase, a question that does not require the state to be dangerous, or even unusual, for the action to be illegitimate.

3.3 Modbus/OT intrusion detection and evaluation infrastructure

A third line concerns the practical detection of attacks on OT protocols and the reproducible evaluation of such detection. Work on industrial protocol intrusion detection and shared evaluation, including the IPAL framework and associated datasets [8], provides common ground for comparing process-aware detection methods. Liscere relates to this line as a consumer of its methodological standards rather than as a competing detector: it is not an intrusion detection system in the classical sense of classifying traffic as malicious or benign, but a layer that evaluates the legitimacy of an action in context and surfaces incoherence for review. The evaluation discipline of this line, reproducible testbeds, documented traffic, and verifiable evidence, is one this report adopts directly: every claim is tied to a captured artefact a reader can inspect.

3.4 Formal, preventive approaches to composed-action attacks

One recent line shares Liscere's precise starting point but resolves it in the opposite direction. Amorim et al. [9] address attacks in which each action is individually innocuous yet the sequence drives the system into an unsafe state, the same class of composed, individually-valid actions that motivates Liscere. Their approach is formal and preventive: safe behaviour is specified as protocols written in a domain-specific language embedded in an interactive theorem prover, machine-checked proofs establish that protocol conformance implies safety, and a runtime attestation proxy blocks any non-conformant action before it executes.

The contrast defines Liscere by opposition. Amorim et al. *prevent* unsafe action sequences by requiring that the safe protocol be defined and proven in advance, and by placing an enforcing proxy inline in the control path; correctness is guaranteed, at the cost of the specification-and-proof effort and of an active component that can block. Liscere *evaluates* action legitimacy by learning the operational grammar from observation, with nothing declared or proven in advance, and remains passive and external, producing a verdict rather than blocking an action. The two occupy complementary points on a design axis: provable prevention with upfront formal effort at one end, and observational evaluation with no upfront specification at the other. Where formal protocols are available and the control path may be modified, an approach like Amorim et al.'s offers stronger guarantees; where systems cannot be touched and no formal specification exists, which describes the majority of the installed base, an external, learning, evaluative layer applies.

3.5 The gap

Across these lines, the process is modelled, sequences are checked, dangerous states are anticipated, and value baselines are learned. What none of them does is evaluate whether an action that is valid, authorised, in specification, and within range is *appropriate to the operational phase the process is currently in*, judged against a grammar of artefact-to-phase associations learned autonomously from observation. Specification-based detection would pass the action as conformant; anomaly detection would pass it as within range; a critical-state monitor would pass it as not approaching danger. The action is illegitimate on none of these axes, and illegitimate only on the axis Liscere introduces: coherence with the learned operational grammar of the process. That is the gap this work occupies.

4. Testbed and methodology

The framework was validated on a physical testbed built from industrial equipment rather than a simulation, so that the traffic evaluated is the traffic a real deployment would see. This section describes the testbed, the process under observation, and the evaluation method at the level needed to understand and trust the results. Full reproduction detail, device configuration, control logic, evaluator source, and the raw evidence for every result reported here, is published in the project repository [13] and referenced throughout; the report states what was done and why, and the repository holds the material required to repeat it.

4.1 Physical testbed

The testbed comprises five components on a single industrial Ethernet segment:

- A **Siemens S7-1200 PLC** (CPU 1212C, firmware 4.1) running the process control logic and exposing a Modbus/TCP server on port 502.
- A **Siemens TP700 Comfort HMI**, used for manual operation of the process. The HMI communicates with the PLC over Siemens' native S7 protocol, not Modbus, and is present to make the testbed a realistic control environment rather than a protocol-isolated bench.
- A **3onedata IES7110-2GS-2F managed switch**, which connects the components and, critically, provides port mirroring: traffic on the PLC and workstation ports is copied to a dedicated mirror port for observation.
- A **Windows workstation** running the engineering environment (TIA Portal, in a bridged virtual machine) and a Python supervisory script that plays the SCADA role, polling the process and issuing control writes over Modbus/TCP.
- A **passive Liscere observer**, a separate machine that receives mirrored traffic only and evaluates it.

The addressing and port-mirroring configuration are documented in the repository manifest [13].

4.2 Passive, external observation

The observation architecture is a deliberate materialisation of the framework's passive, external premise, not merely a convenient setup. The Liscere observer is a physically separate machine. It receives traffic exclusively through the switch's mirror port, a one-way copy of the control traffic, and it never establishes a connection to the PLC, the HMI, or the workstation. It has no Modbus session of its own, issues no requests, and cannot write to any device.

This has two consequences that matter for the framework's claims. First, the observer cannot affect the process even in principle: it is downstream of a mirror, electrically incapable of injecting traffic onto the control path. The evaluation is therefore non-intrusive by construction, not by policy. Second, the arrangement reflects how such a layer would deploy in a real plant, alongside systems that cannot be modified, drawing only a copy of the traffic they already produce, with no change to the controllers, the network logic, or the process. A single detail reinforces the point: the S7-1200's Modbus server accepts one connection, which the supervisory workstation holds; the observer could not connect even if it tried, and does not.

Traffic is captured on the observer with `tshark` on the mirror-facing interface. The evaluator reconstructs Modbus events from the captured stream, extracting for each transaction the function code, the register addressed, the value written, and the direction of the exchange. The reconstruction logic is adapted from the runtime monitor of OT Lab, an earlier OT/ICS simulation and monitoring framework from which this work also inherits its process model and the observation-rule nomenclature used in the decision logs [12]; its role in the wider programme is discussed in Section 9.2. The reconstruction logic itself is published in this report's repository [13].

4.3 Process under observation

The controlled process is a tank simulation, chosen because it exhibits the essential property the framework depends on: distinct, recurring operational phases. The PLC runs the tank model in Structured Control Language on a 100 ms cyclic interrupt, integrating an inlet flow and an outlet flow into a level that rises, holds, and falls over time. The level is clamped to a physical range within the control logic, so that the process behaves as a bounded physical system rather than an unbounded counter.

The process was programmed deliberately as an ordinary industrial process, with physical clamps but with no special logic about which register may be written when. No rule in the PLC restricts a control write by operational phase. This is essential to the validity of the results: all phase awareness resides in the external evaluator, none in the controlled system. The PLC accepts every well-formed, in-range write regardless of phase, exactly as a conventional controller would, and it falls to Liscere alone to judge whether a given write is coherent with the process state.

The Modbus register map exposes the control artefacts and the process variable to the supervisory workstation and the observer:

Address	Artefact	Role
0	PUMP_FLOW_SP	inlet flow setpoint (control)
1	VALVE_FLOW_SP	outlet flow setpoint (control)
2	ALARM_HI_SP	high-level alarm threshold (control)
3	ALARM_LO_SP	low-level alarm threshold (control)
5	LEVEL_AI	tank level (process variable, read)

Register addresses correspond directly to array indices in the PLC data block, with no offset, verified by reading the map from the observer. The two alarm-threshold registers are present in the map for completeness but were not exercised in the increments reported here; the validation focuses on the two flow setpoints and the level.

4.4 Evaluation method

The evaluation proceeds in the same terms across all three increments described in Section 6. From the mirrored read traffic, the evaluator observes the level trajectory and infers the current operational phase of the process. When it observes a control write, it evaluates that write against the inferred phase, according to the grammar in force, and emits a verdict together with the evidence behind it: the register written, the value, the inferred phase, the confidence in that inference, and the reason for the verdict.

Every verdict is recorded to a decision log, one line per evaluated write, capturing the full context of the decision. In parallel, the raw traffic is captured to a packet trace. The two are timestamped consistently, so that any verdict in the decision log can be located in the packet trace and independently verified at the byte level. This pairing, a human-readable decision log backed by a packet-level capture, is the evidentiary basis for every result in this report; both are published per increment in the repository [13]. Where a result depends on a specific value, that value is reported here and the corresponding log and trace entries are available for inspection.

The framework's evolution across the three increments concerns what the evaluator knows and how it decides: how robustly it infers phase, and whether the grammar it judges against is declared by hand or learned from observation. The observation architecture, the process, and the evidentiary method described in this section are common to all three.

5. The evaluation approach

This section describes how Liscere evaluates a control action. It sets out the model of evaluation, the workflow by which a verdict is reached, and the mechanism by which operational context is recovered from the network. The three validation increments in Section 6 are progressive realisations of this approach; the vocabulary established here is what those increments are built from.

5.1 What is evaluated: action, artefact, context, policy

Liscere evaluates a control action along four dimensions, which together determine whether the action is legitimate in the moment it occurs. This four-dimension model, protocol operation, automation artefact, operational context, and policy constraint, was set out in the framework definition of the Liscere series [11]; the present report validates it in hardware.

The **action** is the operation observed on the wire: a write, its function code, and the value it carries. This is what conventional protocol validation sees, and by itself it is insufficient, because a well-formed action carries no indication of whether it belongs.

The **artefact** is the target of the action: the specific control variable being written, identified by its register. Artefacts are not interchangeable. An inlet setpoint, an outlet setpoint, and an alarm threshold play different roles in the process, and each has its own relationship to the operational phases in which it is legitimately manipulated. Evaluating an action means knowing which artefact it touches.

The **context** is the operational state of the process at the moment of the action, the phase the process is in. Context is not declared by any party and is not present in the protocol; it must be recovered by observation. It is the dimension conventional defences lack, and the one that makes an action's legitimacy conditional rather than absolute.

The **policy** is the association between artefacts and the contexts in which they legitimately belong, the grammar against which coherence is judged. A policy may be declared by an engineer who knows the process, or, as Section 6.3 shows, learned from observation of normal operation. The distinction between a declared and a learned policy is a central axis of this work, but in both cases the policy plays the same role: it is the standard of coherence.

An action is evaluated by asking whether *this action*, on *this artefact*, is coherent with *this context*, according to *this policy*. The same action on the same artefact can be legitimate in one context and illegitimate in another, because legitimacy is a relation among the four dimensions, not a property of the action alone.

5.2 The evaluation workflow

Evaluation proceeds as a continuous two-track process over the observed traffic.

On one track, the evaluator maintains a running picture of context. It observes the process variable as it evolves, the tank level in this testbed, and from its trajectory infers the current operational phase, together with a measure of confidence in that inference. This picture is updated with every reading, independently of whether any action is taking place, so that context is always current when an action arrives.

On the other track, when the evaluator observes a control write, it evaluates that write against the current context. It identifies the artefact, reads the inferred phase and its confidence, consults the policy for the phases in which that artefact legitimately belongs, and reaches a verdict. The verdict is emitted with its justification: the artefact, the value, the inferred phase, the confidence, and the reason. Nothing about the verdict depends on the identity of the sender or on whether the action was permitted; it depends only on coherence in context.

The verdict is not a binary block-or-allow decision imposed on the process. Liscere is an evaluative layer, not an enforcing one: it produces a judgement and the evidence for it, and surfaces incoherence for

attention. This is a direct consequence of the passive, external architecture of Section 4. The framework's contribution is the judgement, not an intervention.

5.3 Recovering context: phase inference

Because context is neither declared nor carried in the protocol, the entire approach rests on recovering it from observation. This is the role of phase inference, and its treatment is what distinguishes a usable evaluation from a fragile one.

The principle is that a process with recurring operational phases writes those phases into the trajectory of its process variable. A tank that is filling shows a rising level; one that is draining shows a falling level; one that is holding shows a level that is steady. The phase is not stated anywhere, but it is legible in the movement of the variable, and it can be reconstructed by observing that movement over a window of time rather than at a single instant.

Reconstructing phase from a live signal raises problems that a single-instant reading does not, and the robustness of the inference determines the quality of every verdict built on it. A momentary flattening during a fill is not the same as the process settling into a steady phase; a brief fluctuation is not a reversal; the confidence the evaluator holds in its inferred phase should reflect how clearly the trajectory supports it. These problems, and the design decisions taken to address them, are the substance of the second validation increment (Section 6.2), and are examined there against the evidence that motivated each choice. What matters for the approach as a whole is that context is recovered, with a stated confidence, from observation alone, and that the quality of this recovery is not assumed but demonstrated.

5.4 From declared to learned policy

In the simplest realisation of the approach, the policy is declared: an engineer states which artefact belongs to which phase, and the evaluator judges coherence against that statement. This is enough to demonstrate that contextual evaluation works, and it is where validation begins (Section 6.1).

It is also a limitation. A declared policy must be written by someone who knows the process, for each process, and it fixes in advance a grammar that the real operation may exceed in ways the author did not anticipate. The more capable realisation, and the one that makes the approach applicable without per-process configuration, is a policy learned from observation: the evaluator watches normal operation and discovers, on its own, which artefact is written in which phase. This learned policy is structural, an association between artefacts and phases, and never a baseline of values, which is what keeps the approach distinct from anomaly detection even as it becomes autonomous. The progression from a declared to a learned policy is the arc of Section 6, and the point at which Liscere ceases to depend on a human statement of what belongs.

6. Incremental validation

The approach of Section 5 was validated in three increments, each building on the one before. This ordering is not merely expository: each increment depends on the validation of its predecessor. The first establishes that contextual evaluation works at all, using a declared policy and a simple phase rule. The second, taking the first as given, makes the phase inference robust enough to trust, since every later verdict is built on it. The third, taking robust inference as given, replaces the declared policy with one learned from observation. A weakness at any level would undermine the levels above it, so each was demonstrated and closed before the next was attempted.

All three share the testbed, the passive observation architecture, and the evidentiary method of Section 4: every verdict is recorded to a decision log and backed by a packet-level capture, timestamped consistently so that any result can be verified independently. The tables in this section are distilled to the representative cases; the complete decision logs and packet traces for each increment are published in the repository [13].

6.1 Step 1: observed context changes the verdict

What this step set out to prove. That a contextual evaluation layer can distinguish two identical control actions by the operational phase in which they occur, and that this distinction is structurally beyond what a process interlock provides (Section 2.3). Two writes identical in every conventional respect, same protocol, same function code, same target register, same valid value, should receive opposite verdicts when they occur in different phases, with the phase inferred from observed process state rather than declared by the sender.

How. A write to PUMP_FLOW_SP (register 0, value 90) was issued twice by the supervisory workstation: once while the tank was filling, once while it was draining. The policy in force was declared: a write to PUMP_FLOW_SP is coherent only during FILLING. The passive evaluator, receiving only mirrored traffic, inferred the phase from the observed level trajectory and evaluated each write against it. Both writes were placed well inside a stable phase, since handling of writes at a phase transition is deferred to Step 2. The PLC itself contained no rule restricting either write; it accepted both.

Results. The same write received opposite verdicts, distinguished only by the inferred phase:

Time (CEST)	Action	Register	Value	Observed level	Inferred phase	Decision
16:03:20	WRITE	PUMP_FLOW_SP (0)	90	8	FILLING	ALLOW
16:03:51	WRITE	PUMP_FLOW_SP (0)	90	1	DRAINING	ALERT

The value written was identical and legal in both cases; a per-action or per-instant check sees no difference between them, and the PLC accepted both. The only variable distinguishing the two verdicts is the operational phase, which the evaluator inferred on its own from the level trend. The sender never declared it. This is the framework's central proposition reduced to its simplest demonstrable form: the legitimacy of an action is a function of its context, and that context can be recovered by observation and made to change the verdict. The decision log and the packet capture record both writes with matching timestamps, verifiable to the millisecond [13].

What this step does not yet prove. Two limitations are left open by design, each addressed by a later increment. First, the phase-to-artefact policy is declared by hand, not learned; a human stated that PUMP_FLOW_SP belongs to FILLING. Autonomous learning of that grammar is the subject of Step 3. Second, the phase inference is a simple level-trend rule, adequate for writes placed deep inside a stable phase but not yet robust to the noise, plateaus, and transitions of continuous operation. Making the inference trustworthy is the subject of Step 2, and since every subsequent verdict rests on it, it is the necessary next step.

6.2 Step 2: robust phase inference

What this step set out to prove. That the phase inference underlying every verdict can be made robust enough for continuous operation, and that the evaluator can distinguish three kinds of response rather than a binary allow-or-alert: an action coherent with the phase, an action incoherent with a known phase, and an action during a genuinely uncertain phase. Step 1 placed its writes deep inside stable phases to avoid the hard cases; Step 2 confronts them, because every verdict built on phase inference is only as trustworthy as the inference itself.

How. The simple level-trend rule of Step 1 was replaced by an inference that reads the phase from a least-squares slope over a sliding window of level samples, rather than from an instantaneous difference. The evaluator maintains four states, FILLING, DRAINING, STABLE, and a transitional condition, and attaches to each inferred phase a confidence value reflecting how clearly the trajectory supports it. The window spans eight samples; slopes beyond ±0.25 level units per sample mark rising or falling movement;

a direction reversal must persist for three samples before the phase switches, and a flat trajectory must persist for six before the process is declared settled. The policy remained the declared one from Step 1.

The central design decision: phase inertia. The move from an instantaneous rule to a windowed one exposes a problem that the evidence made concrete. When confidence is tied purely to the instantaneous strength of the trend, a stable plateau, a tank sitting full and steady, is read as an absence of trend, and confidence decays toward zero. The evaluator would then treat a calm, well-understood plateau as a moment of uncertainty. This is the wrong reading, and consequentially so: a plateau is not doubt, it is knowledge. The process is clearly in a steady phase, and many legitimate operations occur precisely during these calm periods. An evaluator that raised uncertainty across every plateau would generate its loudest doubt exactly when the process is most clearly understood, eroding its own usefulness through false alarms.

The correction is to recognise stability as a high-confidence state in its own right, and this raises a design choice with two defensible answers. A short pause within a moving phase, a brief flattening during a fill, is ambiguous: it could be the beginning of a settle into STABLE, or merely a momentary hesitation within an ongoing fill. One approach, the conservative one, treats any flattening as incipient uncertainty and lowers confidence until the situation resolves. The other, phase inertia, holds the moving phase through a short pause and lowers confidence only when the trajectory actually reverses, that is, when movement appears in the opposite direction.

Phase inertia was chosen. The reasoning is operational. Momentary pauses are common during legitimate operation, an inlet flow modulates, a fill hesitates, and treating each as a reason for doubt would produce a stream of false uncertainty during entirely normal running. A pause within a fill is still, in every operational sense, part of filling; the process has not changed what it is doing, it has briefly slowed. Only a genuine reversal, movement in the opposite direction sustained long enough to confirm, represents a real change of phase and warrants a fall in confidence. Under this model a plateau reached and held becomes STABLE at full confidence, while a brief pause during a fill holds the FILLING phase rather than dissolving it into doubt. The conservative alternative is safe in the narrow sense of alerting more often, but it is unsafe in the sense that matters for a monitoring system: it trains its operators to ignore it.

Informed alert versus uncertain alert. Robust inference with confidence makes a conceptual distinction available that a binary scheme cannot express: the difference between two reasons to alert. An action may be incoherent with a phase the evaluator knows with high confidence, a write to an inlet setpoint while the tank is confidently draining, which is an alert by policy: the evaluator knows the phase, knows the artefact does not belong to it, and says so. Or an action may occur while the phase itself is unsettled, during a genuine transition where the evaluator does not yet know which phase holds, which is an alert by uncertainty: the evaluator is not claiming the action is wrong, only that it cannot vouch for the context. Separating these is more honest than a single undifferentiated alert. The first says "I know this does not belong"; the second says "I do not yet know where we are." Conflating them would attribute to the evaluator a certainty it does not have, and would waste an operator's attention by presenting genuine incoherence and mere ambiguity as the same signal.

Results. The same write to PUMP_FLOW_SP (register 0, value 90) was evaluated across the full range of cases. The representative verdicts:

Time (CEST)	Inferred phase	Confidence	Decision	Basis
17:21:16	FILLING	1.00	ALLOW	coherent with phase
17:21:26	FILLING (pause)	0.40	ALLOW	pause within fill held as FILLING
17:21:51	STABLE	1.00	ALERT	incoherent with known phase (by policy)
17:22:01	DRAINING	1.00	ALERT	incoherent with known phase (by policy)

Time (CEST)	Inferred phase	Confidence	Decision	Basis
17:23:24	transition	0.08	ALERT (uncertain)	phase unsettled (by uncertainty)

Two rows carry the weight of the step. The write at 17:21:26, issued during a brief pause in an ongoing fill, is allowed: phase inertia held the FILLING phase through the pause rather than dissolving it, so a legitimate operation during a momentary hesitation is correctly treated as coherent. The write at 17:23:24, issued during a genuine reversal, is surfaced as uncertain rather than as a policy violation: the evaluator declined to judge the action against a phase it could not yet confirm. Between them, these two rows demonstrate the robustness and the three-way distinction the step set out to establish, against the same declared policy as Step 1.

A note on capturing the transition case. The uncertain-transition window is narrow, typically two to three samples, because the modelled process reverses quickly once the outlet opens. This narrowness is itself a desirable property: the evaluator spends very little time uncertain, holding a confident phase for almost all of normal operation. It does, however, make the uncertain case difficult to capture by manually timing a write to fall inside it. To evidence the case reproducibly rather than by chance, a short programmatic burst of writes was issued immediately after a manual phase reversal, so that at least one write fell within the transition window. This is a matter of test instrumentation, not of the mechanism under test: the verdict itself is produced by the same evaluator logic as every other case. The captured uncertain verdicts are recorded in the decision log and packet trace [13].

What this step does not yet prove. The phase inference is now robust, and the evaluator distinguishes coherent, incoherent-in-known-phase, and uncertain. But the policy against which coherence is judged, the statement that PUMP_FLOW_SP belongs to FILLING, is still declared by hand. The evaluator applies a grammar given to it; it does not yet possess one of its own. Learning that grammar from observation, so that no human need declare what belongs where, is the subject of Step 3, and the parameters tuned by hand in this step, the window, the thresholds, the persistence counts, are themselves a motivation for it: a system that learns the process is a system that need not be hand-calibrated to it.

6.3 Step 3: learned grammar and graded evaluation

What this step set out to prove. That the artefact-to-phase policy, declared by hand in Steps 1 and 2, can instead be learned by the evaluator from observation of normal operation, with no manual configuration; that evaluation against the learned policy can be graded rather than binary; and, decisively, that the learning is genuine and specific to each artefact rather than embedded in advance. The scope is deliberately bounded: the state variable remains known to the evaluator (the level is what it reads to infer phase), and it is the grammar alone, which artefact belongs to which phase, that is moved from declaration to learning.

How. The evaluator was given a learning mode that observes normal operation without judging. Whenever it sees a control write, it records the pair of register and inferred phase, accumulating a distribution of the phases in which each artefact is written. It records only the structural association, which phase, and never the value written, and it learns only from writes seen under a confidently known, non-transition phase, so that the learned grammar is not contaminated by the ambiguous moments Step 2 identified. On completion, the learned grammar is exported to an inspectable file. The evaluator then judges live traffic against this learned grammar rather than a declared one.

Graded evaluation. With a distribution rather than a hand-declared rule, the verdict can reflect the degree of coherence rather than a binary membership. For the artefact written and the current phase, the evaluator consults the learned fraction of writes that occurred in that phase and grades the verdict accordingly: a phase well represented in the learned grammar yields COHERENT; a phase seen but rarely yields UNUSUAL, surfaced for review without a strong alarm; a phase never seen for that artefact yields INCOHERENT; and, inheriting the distinction from Step 2, an unsettled phase yields UNCERTAIN

regardless of the grammar. This grading captured a nuance that a hand-written rule would likely have missed. In an early learning session, the inlet setpoint was observed written predominantly during FILLING but once during a STABLE pause, a legitimate adjustment made while a fill was momentarily held. The learned grammar recorded FILLING as dominant and STABLE as rare rather than absent, and a later write during STABLE was accordingly graded UNUSUAL rather than INCOHERENT: a legitimate-but-uncommon operation, correctly distinguished from an incoherent one. A declared policy stating simply that the setpoint belongs to FILLING would have alerted on it.

The anti-memorisation proof. A learned grammar invites a sceptical question: how does one know the association was learned from observation, and not embedded in the evaluator in advance? The answer is to have the evaluator learn a grammar it was not given, and show that its verdicts follow. Two artefacts were taught opposite grammars from observation alone: the inlet setpoint PUMP_FLOW_SP written only during FILLING, and the outlet setpoint VALVE_FLOW_SP written only during DRAINING, each the natural phase for adjusting that flow. The evaluator learned both, from ten writes each, with no declaration. It was then tested across phases, and the two artefacts produced mirror-image verdicts:

Write	in FILLING	in DRAINING	in STABLE
PUMP_FLOW_SP (learned: FILLING)	COHERENT	INCOHERENT	INCOHERENT
VALVE_FLOW_SP (learned: DRAINING)	INCOHERENT	COHERENT	INCOHERENT

Each artefact is coherent only in its own learned phase and incoherent in the other's. The verdicts are opposite because the learned grammars are opposite, and the grammars are opposite because the observed operation was. No fixed rule embedded in the evaluator could produce this: a rule favouring FILLING would misjudge the outlet setpoint, which is coherent precisely in DRAINING. The mirror is the proof. The evaluator learns each artefact's own grammar from what it observes, and judges by what it learned [13].

Why this remains distinct from anomaly detection. The learned grammar is structural, not value-based, and this distinction (Section 2.4) is what the step preserves even as the policy becomes autonomous. The grammar records which phase each artefact is written in, never which values are normal. A verdict of INCOHERENT means the action falls outside the learned artefact-phase association, not that its value is statistically unusual. The same legal value, 90, is COHERENT or INCOHERENT for the same artefact depending solely on the phase and the learned grammar. An evaluator learning value baselines would be doing anomaly detection; this evaluator learns a grammar of structure, and judges coherence within it.

What this step does not yet prove. The grammar is now learned, but the evaluator is still told which signal is the state variable: it knows to read the level to infer phase. Discovering, from traffic alone, which registers are state variables and which are control artefacts, and thereby lifting the approach onto a process it has been told nothing about, is not demonstrated here and is the natural subject of further work (Section 9). So too is generalisation beyond a single cyclic process and a single protocol. What Step 3 establishes is narrower and firm: given an observable cyclic process, the association between control artefacts and the phases in which they are legitimately manipulated can be learned from observation, graded in evaluation, and shown to be genuinely learned rather than embedded.

7. Discussion

The three increments were presented in sequence, each closing a specific question. Seen together, they support observations that no single increment makes on its own. This section draws those out: the trajectory the three describe as a whole, what the approach revealed that was not evident at the outset, and the design tensions that run through the framework rather than through any one part of it.

7.1 The arc from declared to learned

Each increment removed a dependence on knowledge supplied by a human. The starting point of that arc predates this report: in OT Lab, the framework's central proposition was first demonstrated with an entirely declared semantic policy, mappings and rules stated by hand (Section 9.2). Against that baseline, the increments here trace a movement away from declaration. In Step 1, the phase was no longer declared by the sender but inferred from the observed process. In Step 2, that inference was made robust enough to be relied upon, so that the context feeding every verdict was not merely observed but trustworthy. In Step 3, the grammar itself, the association between artefact and phase, was no longer declared by an engineer but learned from observation. Read as a whole, the three describe a trajectory of decreasing reliance on what a human must state in advance, and increasing reliance on what the system can recover for itself.

This trajectory is the substantive claim of the work, beyond any individual verdict. A framework that required an engineer to declare the phase, hand-tune the inference, and specify the grammar for every process would be a configuration exercise, useful but not scalable, and only as good as the declarations it was given. Removing these dependencies one at a time, and demonstrating at each stage that the result still holds, is what distinguishes an approach that could apply to processes it has not been configured for from one that could not. The work does not complete that trajectory, the state variable is still identified for the evaluator, but it establishes the direction and shows that the two dependencies most central to contextual evaluation, the phase and the grammar, can both be lifted from declaration to observation.

7.2 What the evaluation revealed

One result was not anticipated at the outset and deserves to be drawn out, because it inverts an assumption. It is natural to assume that a policy declared by an engineer who knows the process is the ground truth, and that a learned policy is at best an approximation of it. The evidence suggests the opposite can hold. In the learning session that produced a grammar in which the inlet setpoint appeared during FILLING and, rarely, during a STABLE pause, the learned grammar recorded a legitimate operational reality, that an operator may adjust an inlet setpoint while a fill is briefly held, that a hand-declared rule stating simply "the inlet setpoint belongs to FILLING" would have omitted. The learned policy was, in this instance, more faithful to how the process is actually operated than the concise rule a human would likely have written.

This matters beyond the single case. It suggests that learning the grammar is not merely a convenience that saves configuration effort, but can produce a policy that captures operational nuance a declaration would miss, precisely because it is built from what the process does rather than from what an engineer remembers to specify. The graded evaluation is what makes this usable: without a middle verdict, the rare STABLE write would have to be either accepted silently or alerted on, whereas grading it as unusual surfaces it for a human to confirm or dismiss without treating it as an attack. The nuance and the grading are complementary, the first is only useful because the second can express it.

7.3 Design tensions that run through the framework

Two tensions are properties of the whole approach rather than of any single increment, and both are better acknowledged than resolved.

The first is the balance between sensitivity and noise. It appeared first as the choice of phase inertia over a conservative reading of plateaus, and again in the thresholds that separate a coherent from an unusual from an incoherent verdict. In both places the same tension operates: an evaluator tuned to miss nothing will alert on legitimate operation until its operators stop listening, while one tuned to stay quiet will let genuine incoherence pass. There is no setting that escapes this, only settings that place the balance well or badly for a given process. The framework's response is not to claim an optimal point but to make the balance explicit, through confidence values and graded verdicts that expose how firmly each judgement is held, so that the line between alerting and staying silent is visible and adjustable rather than hidden in a binary decision.

The second is the dependence of each layer on the one beneath it. The architecture's strength, that context is recovered, then trusted, then learned against, is also a chain along which error propagates. A verdict about grammar coherence is only as sound as the phase inference feeding it, and the phase inference is only as sound as the observed telemetry it reads. This layering is what makes the approach tractable, each stage solves one problem and hands a clean result to the next, but it means the quality of a high-level verdict cannot exceed the quality of the inference below it. The confidence values are, in part, a way of carrying this uncertainty upward honestly: a verdict issued under low phase confidence is marked as such, rather than presented with a certainty the underlying inference does not support.

8. Limitations

The results reported here are bounded, and stating those bounds precisely is part of the claim. The limitations below are not incidental gaps to be closed by more engineering; several are structural, inherent to the approach as demonstrated, and each defines a condition under which the present results do not extend. They are grouped by what they constrain: the threat model, the learning, the inference, and the scope of validation.

8.1 Threat model: observation is not forge-resistant

The evaluator reconstructs context from observed telemetry. It follows that an adversary who can manipulate that telemetry can manipulate the context the evaluator infers, and therefore its verdicts. The passive, external observation point of Section 4 protects against an adversary on the command path, one who issues writes, whether through stolen credentials, a hijacked workstation, or a remote session, because such an adversary is seen acting against a context reconstructed independently of them. It does not protect against an adversary who compromises the PLC itself, or who can forge the process telemetry the evaluator reads. Such an adversary could present a coherent-looking context while acting within it, and the evaluator, seeing only what it is shown, would judge against a falsified picture. Liscere raises the bar from "hold valid credentials" to "hold valid credentials and control the observed process state," but it does not place that bar out of reach of an adversary with deep access to the controller. The framework addresses the legitimacy of actions on the command path, not the integrity of the telemetry substrate beneath it.

8.2 Learning depends on clean, representative traffic

The learned grammar is only as good as the operation it is learned from, and this constrains it in two ways. First, learning assumes a clean baseline: an attack present during the learning window would be learned as normal, and thereafter judged coherent. The approach as demonstrated has no means of distinguishing legitimate operation from a well-formed attack during learning; it assumes the learning period is supervised or otherwise validated. Second, learning requires coverage. A phase in which an artefact is legitimately but infrequently written may be under-represented or absent in a short learning period, and would then be judged unusual or incoherent when it later occurs. The evidence showed this sensitivity directly: in an early session, a single write placed one phase into the grammar that a longer, more representative session would have weighted differently. A learned grammar inherits the gaps of the operation it observed, and absence of evidence during learning becomes, wrongly, evidence of absence during evaluation.

8.3 Verdicts inherit the limits of phase inference

Every verdict rests on the inferred phase, and the inference, though made robust in Step 2, is not infallible. It begins each session without a phase, requiring several samples before a trend is established, during which no contextual judgement is available. Its parameters, the window length, the slope thresholds, the persistence counts, were set empirically for the dynamics of the modelled process and would require recalibration for a process that fills, drains, or settles at materially different rates. And the uncertain-transition window, while deliberately narrow, is a period during which the evaluator cannot vouch for

context and can only surface actions for review rather than judge them. These are not faults in the inference so much as the price of reconstructing context from a signal rather than being told it, but they bound the conditions under which a confident verdict is available.

8.4 Scope of validation

The claims are validated on one process, one protocol, and one controller, and the boundaries of that validation should not be overstated. The process is a single cyclic tank; the demonstration that phases are legible in a process variable, and that a grammar can be learned over them, rests on the recurring structure of such a process and is not shown for non-cyclic or event-driven processes whose operational structure may not be recoverable from a single variable's trajectory. The protocol is Modbus/TCP; the reconstruction of actions is protocol-specific, and while the evaluation core is independent of it, that independence is asserted by design rather than demonstrated across protocols here. The controller is a single PLC exposing a single process; the interaction of multiple processes, multiple controllers, and richer register maps is outside what was tested. Most consequentially, the state variable was identified for the evaluator throughout: it was told to read the level to infer phase. The discovery of which signals constitute process state, without which the approach cannot be turned on an unknown process unaided, is not among the results of this report.

9. Future work

The limitations of Section 8 map directly onto the work that follows from this report. They fall into three horizons: completing the autonomy the approach began, integrating the validated evaluator into the broader platform it grew from, and extending the approach beyond the conditions demonstrated here.

9.1 Completing the autonomy: discovering process state

The most immediate research direction is the one Step 3 deliberately stopped short of. Throughout this work the evaluator was told which signal constitutes process state: it read the tank level to infer phase. The grammar became learned, but the identification of the state variable remained given. The natural next step, provisionally a fourth increment in the same incremental method, is to discover from traffic alone which registers are state variables, whose trajectories carry the process rhythm, and which are control artefacts written by operators. A process variable evolves with the structured continuity of a physical quantity; a setpoint changes discretely when acted upon. That difference is observable, and learning to separate the two would let the evaluator be turned on a process it has been told nothing about, lifting the last declared dependency identified in Section 8.4. This is a research problem, not an engineering one, and it is where the arc from declared to learned would reach its conclusion.

9.2 Integration with the OT Lab platform

This work did not originate in isolation. It grew from OT Lab, a Docker-based OT/ICS simulation and monitoring framework in which the central proposition of the Liscere framework, that a protocol-valid action is not automatically legitimate in process context, was first demonstrated in software with a declared semantic policy [12], with an initial Modbus/TCP feasibility baseline reported earlier in the series [10]. The present report inherits from OT Lab both its process model, the tank, its register semantics, and the observation-rule nomenclature, and advances beyond it on precisely the axes OT Lab records as unfinished: OT Lab's own documentation notes that process understanding there still depends on declared mappings rather than autonomous model extraction, and lists state-conditioned authorisation among its future paths. Those are the axes Steps 2 and 3 addressed.

There is also a methodological advance to fold back. OT Lab evaluates on a simulated process (OpenPLC) through an inline Modbus proxy on the data path; this work evaluated on physical hardware (a Siemens S7-1200) through strictly passive port-mirror observation, never inline. An immediate direction is therefore

to return the learned, robust evaluation validated here into the OT Lab platform, which provides the orchestration, the interface, the continuous capture, and the evidence pipeline that the deliberately lean standalone evaluator of this report does not. The platform would gain contextual evaluation that no longer depends on a declared policy; the evaluator would gain a home beyond a single-purpose script. The distinction between OT Lab's inline path and this work's passive observation would also need to be reconciled as a deployment choice, an enforcing inline mode against an observing passive mode, rather than a fixed property of the framework.

9.3 Extending beyond the demonstrated conditions

Several extensions widen the envelope in which the approach is known to hold. The first is protocol coverage: the evaluation core is protocol-independent by design, and adding a second protocol, the S7 traffic the same controller already speaks natively is an obvious candidate, would substantiate that independence rather than merely asserting it. The second is process class: the approach rests on recurring operational phases legible in a process variable, and its applicability to non-cyclic, event-driven, or multi-variable processes, whose structure may not be recoverable from a single trajectory, is open. The third concerns the telemetry-integrity limitation of Section 8.1: cross-checking the observed process state against independent signals, or reasoning about the mutual consistency of multiple observed variables, could raise the bar for an adversary who must now forge a coherent multi-signal picture rather than a single trajectory.

9.4 A longer horizon: from single process to process knowledge

Beyond these lies a longer direction, stated here as intent rather than result. Each process that Liscere learns produces a structural description of that process, a grammar of artefacts and phases. Accumulated across many processes, such descriptions would form a body of structural knowledge about how industrial processes of different kinds are legitimately operated. Whether that knowledge could support transfer, recognising a newly observed process as structurally similar to ones seen before, or a learned model that generalises across processes rather than being trained per process, is an open question and a substantial undertaking. It is noted here because the incremental method of this work builds toward it: every documented process, including the one in this report, is a first entry in that record. Any learned or model-based layer of that kind would remain subordinate to the structural, evidence-first principle established here, transparent grammars and verifiable verdicts, rather than an opaque classifier, so that the discipline of claims that governs this report continues to govern whatever is built on it.

10. Conclusion

This report set out to establish that the legitimacy of an industrial control action is not a property of the action alone, and that the missing dimension, the operational context in which the action occurs, can be recovered from observation and made to govern a verdict. The three increments demonstrate this on physical hardware, under strictly passive observation, with every result tied to evidence a reader can inspect.

What was shown is specific. The same protocol-valid, authorised, in-range write receives opposite verdicts according to the operational phase in which it occurs, a phase the evaluator infers rather than being told. That inference can be made robust to the noise, pauses, and transitions of continuous operation, and can carry a confidence that separates an action known to be incoherent from one whose context is merely unsettled. And the grammar against which coherence is judged, which artefact belongs to which phase, can be learned from observation rather than declared by hand, learned genuinely, as two artefacts taught opposite grammars and producing mirror-image verdicts confirm, and applied through graded rather than binary judgement. Across the three, the framework moves from a declared context to an inferred one, and from a declared grammar to a learned one, reducing at each stage what a human must supply.

What was not shown is stated as plainly. The evaluator is still told which signal carries process state; the learning assumes a clean and representative baseline; the inferred context is not resistant to an adversary who forges the telemetry it reads; and the demonstration rests on a single cyclic process, a single protocol, and a single controller. These bound the claim without diminishing it. The contribution is not a finished system but a validated proposition: that contextual, structural evaluation of control actions is possible, that it occupies a gap left by interlocks, by anomaly detection, and by formal prevention alike, and that its central dependencies can be lifted from human declaration toward autonomous observation. The evidence for each step is published alongside this report, and the direction it opens, from a single learned process toward a body of structural process knowledge, is the subject of the work that follows.

References

[1] Modbus Organization. *Modbus Application Protocol Specification V1.1b3*. 2012.

[2] Modbus Organization. *Modbus Messaging on TCP/IP Implementation Guide V1.0b*. 2006.

[3] N. Falliere, L. O. Murchu, and E. Chien. *W32.Stuxnet Dossier*, version 1.4. Symantec Security Response, 2011.

[4] R. Langner. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011.

[5] D. Hadžiosmanović, R. Sommer, E. Zambon, and P. H. Hartel. Through the Eye of the PLC: Semantic Security Monitoring for Industrial Processes. In *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC)*, 2014.

[6] M. Caselli, E. Zambon, and F. Kargl. Sequence-Aware Intrusion Detection in Industrial Control Systems. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security (CPSS '15)*, pages 13–24, 2015. DOI: 10.1145/2732198.2732200.

[7] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. Nai Fovino, and A. Trombetta. A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems. *IEEE Transactions on Industrial Informatics*, 7(2):179–186, 2011. DOI: 10.1109/TII.2010.2099234.

[8] K. Wolsing, E. Wagner, A. Saillard, and M. Henze. IPAL: Breaking up Silos of Protocol-dependent and Domain-specific Industrial Intrusion Detection Systems. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2022. DOI: 10.1145/3545948.3545968.

[9] A. Amorim, T. Kann, M. Taylor, and L. Joneckis. Towards Provable Security in Industrial Control Systems Via Dynamic Protocol Attestation. In *ICSS '24 Workshop*, 2024. arXiv:2412.14467.

[10] B. Salmazo. Context-Aware Evaluation of Industrial Control Actions: A Modbus/TCP Baseline in the OT Lab. *Liscere Technical Report LTR-2026-01*, 2026.

[11] B. Salmazo. Operational Legitimacy of Industrial Control Actions: The Liscere Framework. *Liscere Technical Report LTR-2026-02*, 2026.

[12] Liscere. OT Lab: a controlled laboratory for industrial action evaluation. https://github.com/LiscereSecurity/OT-Lab, 2026.

[13] Liscere. LTR-2026-03: evidence, evaluator source, and reproduction material. https://github.com/LiscereSecurity/Research, 2026.

Liscere Technical Report · LTR-2026-03 · v1 · liscere.com